Web Development with PHP
Practical - 5
PHP Form Handling and Cookie Management

HTML Forms are used to collect information submitted by the user, such as a user's name, email address, or any other essential information, and are sent to the server for processing the data. Forms in HTML contain a checkbox, radio button, input text fields, password fields etc., which can be seen on any website registration or Login/signup pages or other pages.

Forms in HTML are required to collect information or data from the user. For example, registration form, form for user's order details, inquiry forms, etc. Let's say you want to make an account on any social media platform; there will be an option for Sign-up, which redirects the user to a page where the user has to input certain details like Name, Email, Date of Birth, Passwords etc., that can be sent to the server.

To create a form, we use the <form> tag. Below are the attributes for the form tag.

- accept-charset
    - The character encodings to be utilized for form submission are specified by the accept-charset attribute.
    - Syntax
      <form accept-charset = character_set>
      //In HTML5, the default character set is UTF8.
    - Browsers supported by accept-charset
        - Chrome
        - Safari
        - Microsoft Edge
        - Firefox.
- action
    - After the submission of the form, the action attribute specifies the action to be taken, it specifies where the information should proceed. For example, it can be any URL that you prefer, like .php, .asp, etc. If it is left empty, then by default, it will be processed on the same page.
    - Syntax
      <form action = “file.html” method = “post”>

- method
    - The method attribute in the form instructs the browser on how to send data from the form to a web server. That is possible in two ways:
        - GET method
        - POST method
    - GET Method It is not secure as data is displayed in the URL. If the method is not specified, the value is set to "get" as default.
    - Syntax
      <form action="file.html" method="get">
    - POST Method POST Methos is used to process sensitive data as the data is not displayed in the URL. Form data is sent as per the HTTP post-transaction.
    - Syntax
      <form action="file.html" method="post">

- autocomplete
    - If we want the user to enter the data manually or autocomplete the data by itself from the past entries, it can be done by Autocomplete on/off.
    - ON - It is set as default, if not specified. If autocomplete is on, then the browser completes the data from the entries made by the user in the past.
    - Example

      ```
      <form action="/data.php" autocomplete="on">
        <label for="pname">Name:</label>
        <input type="text" id="pname" name="pname"><br><br>
        <label for="email">Email:</label>
        <input type="text" id="email" name="email"><br><br>
        <input type="submit">
      </form>
      ```

    - OFF - Users will have to complete every field by themselves. The browser will not fill it automatically.

    - Example

      ```
      <form action = “file.php” method = “post” autocomplete = “off”>
         <input type = “text” id = “name” value = “name”>     <br><br>
         <input type = “password” id= “pass” value = “Password”><br><br>
         <input type= “submit”>
      </form>
      ```

- enctype
    - Only when the method is "post" can we use enctype. With enctype, we can select the format in which the form's data should be sent to the server.
    - This can be submitted in 3 values -
        - application/x-www-form-urlencoded: It is set as default, if enctype is not specified. The data is encoded before the form is submitted.
        - multipart/form-data: If the form contains any file uploads, then it is used, and also note that no character is encoded by it.
        - text/plain (HTML5): It is used for debugging purposes. There are no changes made; therefore, it is not recommended; only the spaces are converted to +.
    - Syntax

      ```
      <form method="post" enctype="multipart/form-data"></form>
      ```

- name
    - The name attribute is used to reference form data after submission or to refer to items in JavaScript. It is also used to specify the name of the form.

- novalidate
    - It defines that the form data should not be validated if entered value is wrong or does not match the field. For example, entering a mobile number instead of email or leaving the required field empty. It is a boolean attribute added in HTML5.

    - Syntax

      ```
      <form novalidate>
      ```

- required
    - When we want the user to enter a required field, we use the required attribute. If that person does not enter it, the form will be unable to be submitted. It applies to input and select items.
    - Example
      <input type = “password” name = “Password” value = “password” required>

- Rel
    - The rel attribute provides a connection between the current and linked documents. The rel attribute stands for relationship. To link the CSS file to your HTML page, we use the rel attribute.
    - The value it takes is -
        - External - It means that they both are not related or it is not a part of the current document.
        - Help - It will link to the help document.
        - License - to link to the copyright information document.
        - Next - select the next document.
        - Nofollow - Links to a document that is not endorsed, such as a paid advertisement. Google uses the Nofollow to signify that the search spider should not follow that link.
        - Noopener - here, you're effectively telling the browser to open a link in a new tab without giving it context access to the webpage that opened the link. This is accomplished by returning a null value for the window.opener attribute.
        - Noreferrer - If the user clicks on the hyperlink, the browser will not send an HTTP referer header.
        - Prev - To select the previous document.
        - Search - For the document, there are links to a search tool.

    - Syntax
      <form rel = “any_value”>

- Target
    - The target attribute specifies where the response should be opened after the form has been submitted.
    - It can be opened in two forms -
        - _self - The current page will display the response.
        - _blank - A new page will display the response.
        - _parent - The parent frame will display the response.
        - _top - The full page will display the response.
    - If there is no parent in _parent and _top they behave the same as _self.
    - Syntax
      <form action = “file.php” target = “_self”>

Get and Post Methods in PHP

The Get and Post methods in php are two methods that are commonly used to submit data from a web form to a server-side script for processing. The GET method is used to retrieve data from the server and display it in the browser, while the POST method is used to send data to the server for processing. GET sends data in the URL as a query string, while POST sends data in the request body. GET is useful for retrieving data, while POST is more secure and used for submitting sensitive information like login credentials.

GET is used to retrieve data from a web server. It is a simple method that sends data as part of the URL query string, which is visible to the user. GET requests can be bookmarked, cached, and shared, making it a good option for retrieving data that does not need to be kept private.

POST, on the other hand, is used to submit data to a web server. It is a more secure method that sends data in the body of the HTTP request, which is not visible to the user. POST requests are not bookmarked or cached, making it a good option for submitting sensitive data, such as passwords or credit card information.

Get and post methods in php requests are handled through the $_GET and $_POST super global variables, respectively. These variables can be used to retrieve and manipulate data sent through the corresponding HTTP method.

Example:

Suppose we have a PHP script called "example.php" that retrieves information about a product. We can pass the ID of the product we want to retrieve using the GET method as follows:

```
http://example.com/example.php?id=1234
```

In this example, the value "1234" is the ID of the product we want to retrieve. The PHP script can then retrieve this value using the $_GET superglobal array, as follows:

```
$id = $_GET['id'];
```

Example:

Consider a simple example of a login form that uses the POST method to submit data to a PHP script:

```
<form method="POST" action="login.php">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username"><br>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password"><br>

  <input type="submit" value="Login">
</form>
```

GET Method

Advantages

- Simplicity:
  GET requests are simple and easy to use. They can be used to retrieve data quickly and easily without the need for complex coding.
- Caching:
  GET requests can be cached by browsers, allowing them to be served faster and reducing the load on the server.
- Bookmarking:
  GET requests can be bookmarked by users, allowing them to quickly return to the same data without having to navigate through a website.
- Idempotent:
  GET requests are considered idempotent, meaning that multiple identical requests will produce the same result without causing any side effects. This property is useful when retrieving data or performing read-only operations.
- Browser history navigation:
  GET parameters are included in the URL, which allows users to navigate backward and forward through their browser history. This feature can enhance the user experience by allowing them to easily revisit a specific state of a page.
- Integration with external services:
  GET requests are commonly used when interacting with external services or APIs. Many APIs require passing parameters through the URL for data retrieval or making specific API calls, making the GET method a suitable choice for integration purposes.

Disadvantages

- Limited data size:
  The GET method appends the form data to the URL, which has a limited length. Different browsers and servers have different limitations, but generally, URLs longer than 2,048 characters may cause issues. Therefore, GET is not suitable for transferring large amounts of data.
- Security concerns:
  When using the GET method, the form data is visible in the URL, which can be easily bookmarked, cached, or shared. This makes it susceptible to being intercepted and viewed by unauthorized users. Sensitive information like passwords or personal data should not be sent via the GET method.
- Data exposure in logs:
  Since the form data is part of the URL, it may get logged in various places such as server logs, browser history, or referrer logs. This exposes the data to potential privacy concerns.
- Limited data types:
  GET is primarily designed for simple data retrieval, and it has limitations when it comes to handling binary data or complex data structures.

Best Practices:

- Use GET for safe and idempotent operations:
  The GET method should be used for safe operations (read-only) and idempotent (repeating the same request multiple times produces the same result). GET requests should not have any side effects on the server or the application's state.
- Limit sensitive information in URLs:
  Avoid including sensitive information such as passwords, API keys, or personally identifiable information (PII) in the URL parameters. GET requests to append the parameters in the URL, which can be easily visible and cached, posing a security risk. If sensitive information needs to be transmitted, consider using the POST method with appropriate security measures.
- Validate and sanitize input parameters:
  Since the parameters of a GET request are directly visible in the URL, it's crucial to validate and sanitize the input to prevent malicious or unexpected data. Use server-side validation techniques like regular expressions, type-checking, and parameter filtering to ensure the data received is valid and safe to use.
- Avoid long or complex URLs:
  Long URLs can cause issues with certain servers, proxies, or browsers, as there might be limitations on URL length. Keep your URLs concise and readable, focusing on providing essential parameters and avoiding unnecessary complexity.
- Use URL encoding for special characters:
  When passing data in the URL, make sure to properly encode special characters using URL encoding (also known as percent encoding). This ensures that the characters are correctly interpreted by the server and prevents issues related to reserved characters, spaces, or non-ASCII characters.

POST Method

Advantages

- Security:
  POST requests are more secure than GET requests because the data is not visible in the URL query string.
- Data Size:
  POST requests can handle a larger amount of data as the data is sent in the body of the HTTP request.
- Flexibility:
  POST requests can handle different types of data, such as binary data or files, which cannot be sent using GET requests.
- Security with sensitive data:
  Since POST parameters are sent in the body of the request, they are not visible in the URL or browser history. This enhances the security of sensitive data such as passwords, credit card information, or any other confidential data that should not be exposed.
- Data integrity:
  POST requests can include a CSRF (Cross-Site Request Forgery) token as an additional security measure. This token ensures that the request originated from the intended source and protects against malicious attacks attempting to manipulate or forge requests.
- Compliance with web standards: I
  n cases where a large amount of data needs to be sent to the server, the POST method is more compliant with web standards. For example, when submitting HTML forms with a significant number of input fields, the POST method is recommended to avoid exceeding URL length limitations imposed by browsers and servers.

Disadvantages

- Performance:
  POST requests may have slightly slower performance than GET requests because they require additional steps to send and receive data.
- Cacheability:
  POST requests are not cacheable, which can result in slower performance if the same data needs to be accessed multiple times.
- Complexity:
  POST requests require more code to handle server-side data validation and processing, which can be complex and time-consuming.
- Reloading warnings:
  When using the POST method and a user tries to reload the page after submitting a form, the browser may display a warning message about re-submitting the form data. This can be confusing for users and may disrupt the user experience.
- Potential security risks:
  Although the POST method is more secure than GET because the form data is not visible in the URL, it is still vulnerable to certain security risks such as cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks. Proper security measures, such as input validation and CSRF tokens, need to be implemented to mitigate these risks.

- Both GET and POST create an array (e.g. array(key1 => value, key2 => value2, key3 => value3, ...)).
- This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST (super globals).
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.
- You can use the “isset()” function on any variable to determine if it has been set or not.

e.g

```
<html>
    <body>

    <form action = "test.php" method = "GET">
        Name: <input type = "text" name = "name" />
        Age: <input type = "text" name = "age" />
        <input type = "submit" />
    </form>

    </body>
</html>

//File Test.php
<?php
    if( $_GET['name'] || $_GET['age'] ) {
        echo "Welcome ". $_GET['name']. "<br />";
        echo "You are ". $_GET['age']. " years old.";
    }

?>
```

PHP Form Handling

When we develop a website or a web application, we often have to create forms to take input from users, like a Login form or a Registration form. Creating a form on the webpage is accomplished using HTML, while PHP serves as a transport for those values from the web page to the server and then in further processing those values. PHP provides two superglobals $_GET and $_POST for collecting form-data for processing.

Form validation is done to ensure that the user has provided the relevant information. Basic validation can be done using HTML elements. For example, the email address text box has a type value as “email”, which prevents the user from entering the incorrect value for an email. Every form field in the above script is followed by a required attribute, which will tell the user not to leave any field empty before submitting the form. PHP methods and arrays used in form processing are:

- isset()/empty(): This function is used to determine whether the variable or a form control is having a value or not.
- $_GET[]: It is used to retrieve the information from the form control through the parameters sent in the URL. It takes the attribute given in the url as the parameter.
- $_POST[]: It is used to retrieve the information from the form control through the HTTP POST method. IT takes the name attribute of corresponding form control as the parameter.
- $_REQUEST[]: It is used to retrieve information.
- htmlspecialchars() function - converts special characters to 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.
- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
    - The $_SERVER["PHP_SELF"] variable can be used by hackers!
    - If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.
    - Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side scripts into Web pages viewed by other users.
    - $_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

Cookie Management

Cookies are small text files that are stored on a user's computer by a web server. In PHP, cookies can be used to store and retrieve information about the user's interactions with a website. By setting a cookie, the server can remember user preferences and track their activities. PHP provides functions like setcookie() to create a cookie and $_COOKIE superglobal array to access its values. Cookies can be used for various purposes such as session management, personalization, and tracking.

Cookies in PHP are a fundamental mechanism for storing and retrieving small pieces of data on the client side. They play a crucial role in web development by allowing web servers to maintain stateful interactions with web browsers. Cookies are essentially text files that are stored on the client's device and sent back to the server with subsequent requests. In PHP, cookies are commonly used to track user preferences, maintain user sessions, personalize content, and implement features like "Remember Me" functionality. They provide a way to store data that persists across multiple page visits or sessions.

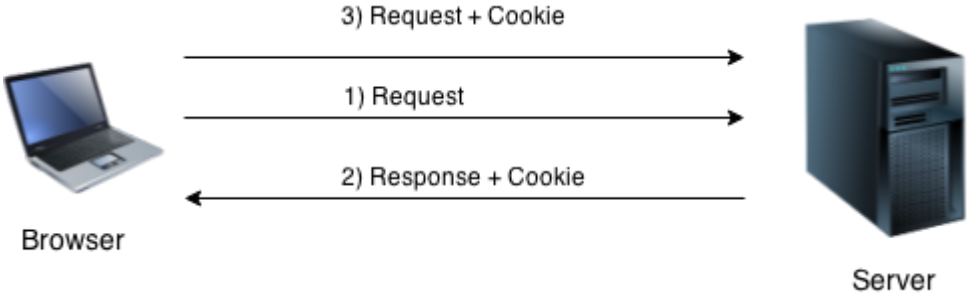
Using PHP's cookie functions, developers can set cookies by specifying a name, value, expiration time, path, and domain. The server sends these cookies to the client's browser, which stores them locally. On subsequent requests, the browser automatically sends the cookies back to the server, allowing the server to retrieve and utilize the stored data.

Cookies are essential for creating personalized and dynamic web experiences. They enable websites to remember user preferences, such as language settings or theme choices. They also play a crucial role in session management, allowing servers to identify and maintain user sessions across multiple page visits.

However, it's important to note that cookies have certain limitations and considerations. They have size limitations, and sensitive information should be avoided or properly encrypted. Additionally, users have the option to disable or delete cookies, so applications should be designed to handle scenarios where cookies are not available.

Cookies in PHP provide a simple and effective method for storing and retrieving data on the client side. They allow web servers to maintain stateful interactions and personalize web experiences. Understanding how to work with cookies in PHP is essential for developing dynamic and user-friendly web applications.

Cookies are created at the server side and saved to the client browser. Each time when a client sends a request to the server, a cookie is embedded with the request. Such way, cookies can be received at the server side.



Understanding a cookie in PHP involves knowing its components and how they interact.

- Name:
  A cookie has a name that serves as its identifier. The name should be unique and meaningful to identify the specific cookie.
- Value:
  The value represents the data stored within the cookie. It can be any string or data that needs to be saved and retrieved later.
- Expiration:
  A cookie can have an expiration time, after which it becomes invalid. This allows developers to set cookies with different lifetimes, ranging from a session cookie that expires when the browser is closed to persistent cookies that remain on the user's computer for an extended period.
- Domain:
  The domain specifies the scope of the cookie. It defines the websites within which the cookie is valid. By default, the cookie is valid for the current domain that sets it.
- Path:
  The path determines the URL paths for which the cookie is valid. If set to "/", the cookie will be valid for the entire website. Otherwise, it will be limited to the specified path or its subdirectories.
- Secure and HttpOnly:
  These flags indicate whether the cookie should be sent over secure connections (HTTPS) and restrict access to server-side scripting only, respectively. Enabling the Secure flag ensures that the cookie is transmitted securely, while HttpOnly prevents client-side scripts from accessing the cookie, reducing the risk of cross-site scripting (XSS) attacks.

Understanding a cookie in PHP allows developers to control the behavior, lifespan, and security features of cookies. By manipulating these components, PHP applications can store and retrieve data on the client side, maintain user sessions, personalize content, and enhance the overall user experience.

Uses of Cookie

Cookies have various uses and can be employed to enhance the functionality and user experience of a website. Here are some common uses of cookies:

- Session Management:
  Cookies are commonly used for session management. When a user logs into a website, a unique session identifier can be stored in a cookie. This allows the server to recognize and authenticate the user across multiple page requests without requiring them to log in again.
- Personalization:
  Cookies enable websites to remember user preferences and personalize the browsing experience. For example, a website can store the user's language preference, theme selection, or display settings in a cookie, ensuring that the website is customized to their preferences on subsequent visits.
- Shopping Carts:
  E-commerce websites often use cookies to track and maintain shopping cart information. By storing the contents of a user's shopping cart in a cookie, the website can retain the selected items even if the user navigates to other pages or closes the browser.
- Tracking and Analytics:
  Cookies can be used for tracking user behavior and gathering analytics data. They allow websites to collect information such as the number of visits, pages visited, and referral sources. This data can be utilized to analyze user patterns, improve website performance, and provide targeted advertising.
- Remember Me Functionality:
  Many websites offer a "Remember Me" feature for user convenience. By using cookies, a website can store a persistent identifier that allows users to bypass the login process on subsequent visits, providing a seamless user experience.

Cookies provide a versatile mechanism for storing small pieces of data on the client side, enabling personalized experiences, session management, user tracking, and various other functionalities that enhance web applications.

In PHP, setting a cookie is a straightforward process that involves using the setcookie() function. This function allows developers to define the properties of the cookie, such as its name, value, expiration time, and optional parameters.

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

In PHP retrieving a cookie is also very simple, Here's an example of how to retrieve a cookie in PHP:

```
if (isset($_COOKIE['username'])) {
    $username = $_COOKIE['username'];
    echo "Welcome back, $username!";
} else {
    echo "Cookie not found.";
}
```

Here's an example of how to update a cookie in PHP:

```
// Retrieve the existing value of the cookie
$existingValue = $_COOKIE['my_cookie'];

// Modify the value or any other parameters as needed
$newValue = $existingValue . " Updated";
$newExpiration = time() + 3600; // Add 1 hour to the
current time

// Update the cookie using setcookie()
setcookie("my_cookie", $newValue, $newExpiration, "/");

// Display a message to confirm the update
echo "Cookie updated successfully.";
```

Here's an example of how to remove a cookie in PHP:

```
// Set the cookie expiration time to a past date
setcookie("my_cookie", "", time() - 3600, "/");

// Display a message to confirm the removal
echo "Cookie removed successfully.";
```

Use Cases

Session Management:

```
// Start the session
session_start();

// Set session variables
$_SESSION['username'] = 'JohnDoe';
$_SESSION['role'] = 'admin';

// Store session ID in a cookie
setcookie('session_id',  session_id(),  time()  +  3600,
'/');

// Retrieve session data on subsequent requests
session_start();
echo 'Welcome back, '. $_SESSION['username'];
```

Here, a session is started using session_start(), and session variables like username and role are set. The session_id is stored in a cookie using setcookie(). On subsequent requests, the session is resumed with session_start(), and the stored session data can be accessed. Run the above code in your editor for a better and clear explanation.

Personalization:

```
// Set a personalized theme preference
$theme = 'dark';

// Store the theme preference in a cookie
setcookie('theme', $theme, time() + (86400 * 30), '/');

// Retrieve the theme preference and apply it
$selectedTheme      =      isset($_COOKIE['theme'])      ?
$_COOKIE['theme'] : 'light';
echo 'Using '. $selectedTheme . ' theme';
```

Here, a user's personalized theme preference is stored in a cookie. When the user visits the website again, the cookie is retrieved, and the selected theme is applied to provide a personalized experience. Run the above code in your editor for a better and clear explanation.

Tracking:

```
$page = 'home';
// Store the visited page in a cookie
if (!isset($_COOKIE['visited_pages'])) {
    $visitedPages = [];
} else {
                                    $visitedPages        =
unserialize($_COOKIE['visited_pages']);
}
$visitedPages[] = $page;
setcookie('visited_pages',      serialize($visitedPages),
time() + (86400 * 30), '/');
```

```
// Retrieve and display the visited pages
if (isset($_COOKIE['visited_pages'])) {
                                    $visitedPages        =
unserialize($_COOKIE['visited_pages']);
    echo 'Visited pages: '. implode(', ', $visitedPages);
}
```

Here, each time a user visits a page, the visited page is stored in a cookie. The visited pages are tracked and stored in an array. On subsequent requests, the stored cookie data is retrieved and displayed, allowing tracking of the user's navigation history. Run the above code in your editor for a better and clear explanation.

Best Practices for Setting and retrieving cookies in PHP

- Set appropriate cookie parameters:
  When setting a cookie, use the setcookie() function and provide necessary parameters like the cookie name, value, expiration time, path, domain, and secure flag. Set the expiration time appropriately based on your application's requirements.
- Encrypt sensitive cookie data:
  If you need to store sensitive information in a cookie, consider encrypting the data before setting the cookie. This adds an extra layer of security and ensures that even if the cookie is intercepted, the data remains encrypted and unreadable.
- Limit cookie data size:
  Cookies have a limited size (typically 4KB), so avoid storing excessive data in cookies. Keep the data stored in cookies to a minimum and consider alternative methods like session storage or database storage for larger data sets.
- Sanitize cookie data:
  Before using cookie data in your application, sanitize and validate it to prevent any potential security vulnerabilities. Use appropriate sanitization functions or validation techniques depending on the specific data being stored in the cookie.
- Use secure and HTTP-only cookies:
  When dealing with sensitive information, consider setting secure and HTTP-only flags for your cookies. The secure flag ensures that the cookie is only transmitted over HTTPS, while the HTTP-only flag prevents client-side JavaScript access to the cookie, mitigating cross-site scripting (XSS) attacks.
- Set cookie expiration wisely:
  Set the expiration time of the cookie based on your application's requirements. Avoid excessively long expiration times, as it may lead to persistent cookies that remain on the user's device for an extended period. Consider using session cookies that expire when the user closes the browser for sensitive data.
- Validate and authenticate cookies:
  Before using cookie data for authentication or authorization purposes, validate and authenticate the cookie. Verify that the cookie is genuine, has not been tampered with, and is associated with an active user session. Implement appropriate mechanisms like session identifiers or digital signatures for cookie authentication.
- Handle expired or invalid cookies:
  Check for expired or invalid cookies and handle them gracefully. When a cookie is expired or invalid, take appropriate actions such as redirecting the user to a login page or displaying a message indicating the need to authenticate again.
- Use secure session management:
  When using cookies for session management, implement secure session handling practices. Generate secure session IDs, store them securely, and ensure that session data is not exposed or vulnerable to session hijacking or session fixation attacks.
- Regularly review and update cookie usage:
  Periodically review the cookies used in your application and assess their necessity. Remove any unnecessary cookies to minimize the amount of data stored on the user's device and reduce potential security risks.

Security

The security implications of using cookies in PHP are crucial to consider to protect user privacy and prevent potential vulnerabilities. While cookies serve important purposes in web development, it is essential to be aware of their security implications and take appropriate measures to mitigate any risks.

- Data Privacy:
  Cookies can store sensitive information, such as user IDs, session tokens, or personal preferences. It is important to be mindful of the data being stored in cookies and ensure that any sensitive data is properly encrypted or obscured to prevent unauthorized access.
- Cross-Site Scripting (XSS) Attacks:
  XSS attacks occur when an attacker injects malicious scripts into a website, which can then access and manipulate cookies. To mitigate this risk, it is crucial to properly sanitize and validate user input and employ output encoding techniques when interacting with cookie data.
- Cross-Site Request Forgery (CSRF) Attacks:
  CSRF attacks exploit the trust a website has in a user's browser by tricking them into performing unintended actions. Using secure tokens and implementing measures such as the SameSite attribute can help prevent CSRF attacks by validating the origin of requests.
- Session Hijacking and Fixation:
  Cookies that store session identifiers are potential targets for session hijacking or fixation attacks. To prevent these attacks, it is advisable to regenerate session IDs after authentication, use secure session management techniques, and regularly expire and refresh session cookies.
- Secure Flag and HttpOnly Flag:
  Setting the Secure flag ensures that cookies are only transmitted over secure HTTPS connections, mitigating the risk of data interception. The HttpOnly flag prevents client-side scripts from accessing cookies, reducing the possibility of XSS attacks.
- Cookie Expiration and Cleanup:
  It is important to set appropriate expiration times for cookies and regularly clean up expired cookies. This helps prevent the accumulation of unnecessary and potentially sensitive data on the client side.
- Cookie Consent and Privacy Policies:
  Complying with applicable privacy laws and regulations, such as the General Data Protection Regulation (GDPR), may require obtaining user consent for cookie usage and providing clear information about how cookies are used and managed.

Exercise

1. Use HTML form created in practical -1 . Enter respective data into each and using the post method pass the data and display the data to the data.php page on click on submit button.
2. Create a HTML form that reads 2 numbers and has 4 buttons for operation:- Add, Difference, Product. Based on the button clicked by the user the result should be displayed as output.
3. Create an HTML form which takes 10 numbers (comma separated) and redirect control to findsum.php create a php script which stores the numbers into an array and displays the total of the array
4. Create a page signin.php , which takes the username, age ,city from user on click of submit button set the cookies values for each and display values on next page view.php
5. Create a php page which takes the student's name and marks of three subjects from the user. It stores all information as cookies and displays all information on the next page.