Learning Outcome 7: Manage state information and security

HTTP/HTTPS is what is called a “stateless” protocol. This means that the web server doesn’t need to know what web pages were visited, or what was done on those web pages, before arriving at the current page.

Advantages:

* Web server is much simpler.
  + Server doesn’t need to remember what pages the visitor visited before.
  + Server doesn’t need to decide when to throw away outdated information (visitor hasn’t moved a new page for a long time, they must have gone away)

Although HTTP is a stateless protocol, there are methods that we can use on top of HTTP to add state information (think of it like another layer on top of HTTP).

# Learning Step 7.1: Save state information using hidden form fields

HTML provides a “hidden” input element that we can use in forms. For example, if we have the following element in a form:

<input type="hidden" id="custId" name="custId" value="123" />

This element won’t be displayed in the form, but the name and value will be transmitted to the server when the page is submitted.

**Practice:** Starting with **hiddenfield.html** in the OneDrive folder, add a hidden field to the form containing a name of “custId” and a value of “123”. Submit the form and verify that the customer ID would be transmitted to the server.

**Extra practice:** Add JavaScript code that will create a hidden field whose value will be the URL of the page that the form is submitted from.

Limitations:

* Can only pass state information from web pages that have forms on them
* It would be easy for a user to fake the information (just type a different parameter in the URL in the address bar for GET, slightly more work for POST) and thus impersonate someone else

# Learning Step 7.2: Save state information with query strings

Idea: for every link on a page, we have the server re-write the links to add in a name/value pair at the end of the link. Example: Suppose we have a web site with:

* <http://www.example.com/computers>
* <http://www.example.com/electronics>

If someone is logged in, the web server could change those links to:

* <http://www.example.com/computers?custid=123>
* <http://www.example.com/computers?custid=123>

**Practice:** Starting with the file **rewriteurl.html**, you are to add a click event handler to the “Rewrite URLs” button. In this handler, you are to change the href property of all <a> elements on the page to include a customer id parameter with the value of 123.

Note: if you use jQuery, you will need to use the **each()** method. Use Google to find out how the each() method works and how to use it (it takes in a function that is run on every element in a jQuery object).

Limitations:

* Easy to fake – just add a different customer id to the link
* Will be stored as part of a bookmark to the page
* Can only be used when user is following links

# Learning Step 7.3: Save state information with cookies

A cookie is a small file written to your file system by your web browser that contains a number of name/value pairs. These pairs can then be used to maintain states across browser sessions. A website can tell the browser to write these pairs to disk (or to a database on the disk); then when the browser visits the site again, the browser sends those pairs back to the website that originally sent them.

Cookies are special:

* The browser can only write cookies to a specific location on the disk (location varies by browser and operating system)
* The size of the cookies is limited (max 4096 bytes)
* The total number of cookies stored in your browser is limited to 3000
* The total number of cookies allowed per web site is limited to 50

Other restrictions:

* A cookie will only be sent to the website that set it (arbitrary websites can’t just read any cookie)
* The browser decides what cookies to send to the server; web servers can’t request a cookie – it’s all under the browser’s control

We can specify an expiry time for cookies; the cookie will last until that expiry time, unless the user decided to clear their cookies. If we don’t specify an expiry time, the cookie is called a “session cookie” and will only last until the browser is closed. Cookies that have an expiration time are called “persistent cookies”; they persist across browser sessions.

See also <https://en.wikipedia.org/wiki/HTTP_cookie>

**Activity:** Copy the **quickphp\_webserver** from the **lo7 > GroupA > bin** folder on **OneDrive** to a folder named **CWEB190** on your **C:** drive. Copy the file **cookiedemo.php** to **C:\CWEB190\html\**. Start **QuickPHP.exe** and use a Binding Address of “127.0.0.1”, a Port of 8080, and **C:\CWEB190\html** as your root folder. Click Start. Open the page [**http://127.0.0.1:8080/cookiedemo.php**](http://127.0.0.1:8080/cookiedemo.php) both with and without a query string setting the username at the end of the address. Examine how the developer tools show the difference in the requests and the cookie settings for this page. *(F12 for Developer tools, Network tab and click on the page requested (may need to reload); Application tab and click on Cookies)*

**Practice:** Go to a website that you visit frequently. Check to see if the website sets any cookies on your computer. Answer the following:

* What is the website?
* What is the name of one of the cookies?
* What is the value of the cookie?
* Is the cookie a session cookie or a persistent cookie?
* If the cookie is a persistent cookie, when does it expire?
* Find an example of a session cookie and a persistent cookie. You may need to visit more than one site.

**Creating cookies with JavaScript**

The document object has a property called “cookie”. We can create a cookie by assigning a name/value pair to the cookie like so:

document.cookie = "custid=123";

You can add multiple name/value pairs to the cookie by having successive assignments to the document.cookie property. For example, we can add another name/value pair to the cookie with:

document.cookie = "city=Saskatoon"; // adds another cookie – does not overwrite other cookies

document.cookie isn’t just a write-only property. If we want to get all the name/value pairs for this web page’s cookies, we can read from the property as well. For instance, to display all the name/value pairs in an alert box, we could execute:

window.alert( document.cookie );

When we just specify a cookie, it is a session cookie, and only last as long as the current browsing session is open.

To specify an expiration date for a cookie, we add a semicolon, then a space, then "expires=" followed by a date. The date is in a very specific format, which is the format returned by the Date.toUTCString() method.

The following statement will create a persistent cookie that expires on June 30, 2021 at noon UTC:

document.cookie = "program=CST; expires=Wed, 30 Jun 2021 12:00:00 GMT";

**Practice:** Create a web page that allows us to set and display cookies. Start with the file **cookies.html** from the OneDrive folder. You are to do two things:

1. Add a click event handler to the “Display cookies” button that displays the current value of all cookies for this web page in the span element with the “cookieValues” id.
2. Add a click event handler to the “Set cookie” button that sets a cookie on the current web page. The name of the cookie will come from the text field with the nameField id, and the value will come from the “valueField” id. In addition, if the “session” radio button is checked, create the cookie as a session cookie; if the “persistent” radio button is checked, create the cookie as a persistent cookie with an expiry date of some time in the future. *Bonus practice:* make the expiry date exactly 7 days from the time that the button was clicked.

Remember, for cookies to work, the file must be served from a web server.

If you are using jQuery, the following selector will determine which radio button is selected:

$("[name=cookieType]:checked")

You can call the val() method on the return value of this selector to get the value attribute of the radio button:

$("[name=cookieType]:checked").val()

In our case, this will return “persistent” if the persistent radio button is selected, or “session” if the session radio button is selected.

**Some additional notes about cookies**

***Deleting cookies***

We can delete a cookie by setting an expiration date in the past. You can use any time that is in the past, but often we’ll do this with the time that is the start of the Unix epoch, like so:

document.cookie = "custid=123; expires=Thu, 1 Jan 1970 00:00:00 UTC";

This will delete the cookie with the name “custid” (the value doesn’t matter and is ignored).

***Cookies with special characters***

Typically, before setting the cookie, we pass both the name and the value through a function called encodeURIComponent – this will encode the name and value into something that doesn’t mess up the cookie. For example:

encodeURIComponent("Santa;Claus")

returns the string "Santa%3BClaus".

When we read the cookie, we’ll need to unencode the name and the value; we can do that with decodeURIComponent. For example:

decodeURIComponent("Santa%3BClaus")

returns the string "Santa;Claus".

***Accessing cookies across many pages on a website***

Cookies are typically tied to a particular domain/folder on a website. However, we can specify that a cookie applies to any page under a particular folder on a website. We can do this by specifying a “path” for a cookie. For example, if we do the following:

document.cookie = "custid=123; path=/electronics";

This cookie will only be available for any pages underneath the “electronics” folder on the website. To make a cookie available to all pages on the website, specify a path option of "path=/".

**Cookie utility functions**

See **scripts/cookieutils.js** on the OneDrive folder (under **lo7/GroupA**). This file will be available on the final exam. It contains two functions:

The first function, setCookie, will create a cookie. It takes up to four arguments:

* The name of the cookie
* The value of the cookie
* (optional) the number of days before the cookie expires
* (optional) the path

This function will look after encoding the name and value, and it will do the calculation and formatting of the expiration date.

The second function, readCookie, will read a particular cookie from the cookie string. It takes a single argument, the name of the cookie (such as "custid") and returns either:

* the cookie value associated with that particular cookie name, if the cookie name exists; or
* undefined if the cookie name doesn’t exist

It will look after decoding the name and value for you.

**Practice:** Update your previous practice to use setCookie function. You don’t need to use readCookie because we’re displaying ALL the cookies.

**EXTENDED PRACTICE #1** – game.html (from the OneDrive folder)

Set up the game so that the wins this session and the wins all-time are tracked using cookies, and the values are loaded in at the start of the game (if the cookies exist) and are updated whenever the player wins. When the Reset Stats button is clicked, delete any cookies and reset the wins to 0.

**EXTENDED PRACTICE #2** – home.html, login.html (from the OneDrive folder)

On the login.html page, when the user enters their name and presses the login button, the page will store a persistent cookie with the user’s name, then will redirect the user back to the first page. You can set the window.location property to go back to the first page.

When we’re in the Known User state, instead of the generic greeting, we welcome the user by name. Then, rather than including a link to the login page, we provide a link that allows the user to tell the website to forget who they are. If the user clicks this link, the cookie is deleted and the page is refreshed (you can call window.location.reload() to refresh the page).

*Hint:* In the home.html page, if the cookie exists, the web page should display one thing, otherwise it should display something else.

**Alternative to cookies for client-side storage: Web Storage API**

Web storage – simple client-side data that stays intact through page loads

Stored separately for each domain (other domains cannot access a domain’s storage)

It includes two new objects:

* sessionStorage – stores data just for the current session
* localStorage – stores data indefinitely

Methods on these objects are:

* .setItem(*keyString, valueString*) – creates new values or updates existing values
* .getItem(*keyString*) – returns the *valueString* associated with the item or null if the item does not exist
* .removeItem(*keyString*) – removes the *keyString/valueString* pair from storage

See also <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage>

Or <https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API>

# Learning Step 7.4: Discuss secure coding with JavaScript

There are a number of security considerations that you need to take into account when you are writing JavaScript programs that run in the web browser. Some of these include:

* You cannot execute operating system commands
* You cannot perform arbitrary local file I/O
* You can only perform very limited network communication
* Anyone can see the data that you pass over the network
* Anyone with access to your computer can access your cookies/local storage