COMP 484
Web Engineering I

Instructor: ~kaplan (adam.kaplan@csun.edu)

Lecture #5: 8/3/2020

Cookies, HTML5 Storage, and JSON

Cookies

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Yes
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	No
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	No
sneaky.com	/	UserID=3627239101	31-12-12 23:59	No

- A means for web sites to identify previous visitors
  - <= 4 KB string stored on client machine as a file
    - Presented to the site it came from when that site is re-browsed
- Contents:
  - Domain
    - Which domain the cookie originates from

Cookies (cont)

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Yes
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	No
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	No
sneaky.com	/	UserID=3627239101	31-12-12 23:59	No

- Path
  - Which directories in server file-tree that may use this cookie
  - "/" means "all directories"
- Content
  - Name=value pairs
- Expires
  - Timestamp of expiration date
  - If empty, expires when browser closes
- Secure
  - If "Yes," client will only present cookie to secure site

More on Cookies

- Before HTML5, websites could store only small amounts of text-based information on a client computer using cookies
  - <= 4KB per cookie
  - Browsers typically allow 50+ cookies per domain

Problems with Cookies

- Extremely limited in size
- Cookies cannot store entire documents
- If the user browses the same site from multiple tabs, all of the site's cookies are shared by the pages in each tab
  - This could be problematic in web applications that allow the user to purchase items

Cookie Example

```
// determine whether there is a cookie
if ( document.cookie )
{
   // convert escape characters in the cookie string to their
   // English notation
   var myCookie = unescape( document.cookie );

   // split the cookie into tokens using = as delimiter
   var cookieTokens = myCookie.split( "=" );

   // set name to the part of the cookie that follows the = sign
   name = cookieTokens[ 1 ];
} // end if
else
{
   // if there was no cookie, ask the user to input a name
   name = window.prompt( "Please enter your name", "Paul" );

   // escape special characters in the name string
   // and add name to the cookie
   document.cookie = "name=" + escape( name );
} // end else
```

Does cookie exist on client computer?

In this example, cookieTokens will contain 2 values… [“name”, name]

NOTE: multi name/value pairs would be separated by ‘;’

Creating the cookie, if it does not yet exist

localStorage and sessionStorage

- HTML5 gives us two new mechanisms for storing key/value pairs...
  - window object's **`localStorage` property**
    - Stores up to several megabytes of key/value-pair string data on the user's computer
    - Data can be accessed across browsing sessions and browser tabs
  - window object's **`sessionStorage` property**
    - Data accessible *only* during a browsing session
      - Does not persist across sessions
    - Data is separate among multiple tabs browsing the same site
      - Separate `sessionStorage` object for every tab

Storing Favorite Twitter Searches

- This app allows users to save their favorite (possibly lengthy) Twitter search strings
  - The user's favorite searches are saved using `localStorage`, so they're immediately available each time the user browses the app's web page
    - Associate search string with user-chosen tag names
  - The app uses `sessionStorage` to determine whether the user has visited the page previously during the current browsing session
    - If not, the app displays a welcome message

Storing Favorite Twitter Searches

a) **Favorite Twitter Searches** app when it's loaded for the first time in this browsing session and there are no tagged searches

Storing Favorite Twitter Searches

b) App with several saved searches and the user saving a new search

Storing Favorite Twitter Searches

c) App after new search is saved—the user is about to click the Deitel search

Storing Favorite Twitter Searches

d) Results of touching the **Deitel** link

Storing Favorite Twitter Searches

First, let's have a glance over the code

Then we will have a deeper look and see how it all functions…

```
var tags; // array of tags for queries

// loads previously saved searches and displays them in the page
function loadSearches()
{
   if ( !sessionStorage.getItem( "herePreviously" ) )
   {
      sessionStorage.setItem( "herePreviously", "true" );
      document.getElementById( "welcomeMessage" ).innerHTML =
         "Welcome to the Favorite Twitter Searches App";
   } // end if

   var length = localStorage.length; // number of key/value pairs
   tags = []; // create empty array

   // load all keys
   for (var i = 0; i < length; ++i)
   {
      tags[i] = localStorage.key(i);
   } // end for

   tags.sort(); // sort the keys

   var markup = "<ul>"; // used to store search link markup
   var url = "http://search.twitter.com/search?q=";

   // build list of links
   for (var tag in tags)
   {
      var query = url + localStorage.getItem(tags[tag]);
      markup += "<li><span><a href = '" + query + "'>" + tags[tag] +
         "</a></span>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Edit' onclick = 'editTag(id)'>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Delete' onclick = 'deleteTag(id)'>";
   } // end for

   markup += "</ul>";
   document.getElementById("searches").innerHTML = markup;
} // end function loadSearches
```

```
// deletes all key/value pairs from localStorage
function clearAllSearches()
{
   localStorage.clear();
   loadSearches(); // reload searches
} // end function clearAllSearches

// saves a newly tagged search into localStorage
function saveSearch()
{
   var query = document.getElementById("query");
   var tag = document.getElementById("tag");
   localStorage.setItem(tag.value, query.value);
   tag.value = ""; // clear tag input
   query.value = ""; // clear query input
   loadSearches(); // reload searches
} // end function saveSearch

// deletes a specific key/value pair from localStorage
function deleteTag( tag )
{
   localStorage.removeItem( tag );
   loadSearches(); // reload searches
} // end function deleteTag


// display existing tagged query for editing
function editTag( tag )
{
   document.getElementById("query").value = localStorage[ tag ];
   document.getElementById("tag").value = tag;
   loadSearches(); // reload searches
} // end function editTag
```

…and, finally, the initializer function…named **start** in this code

```
// register event handlers then load searches
function start()
{
   var saveButton = document.getElementById( "saveButton" );
   saveButton.addEventListener( "click", saveSearch, false );
   var clearButton = document.getElementById( "clearButton" );
   clearButton.addEventListener( "click", clearAllSearches, false );
   loadSearches(); // load the previously saved searches
} // end function start

window.addEventListener( "load", start, false );
```

```
// register event handlers then load searches
function start()
{
   var saveButton = document.getElementById( "saveButton" );
   saveButton.addEventListener( "click", saveSearch, false );
   var clearButton = document.getElementById( "clearButton" );
   clearButton.addEventListener( "click", clearAllSearches, false );
   loadSearches(); // load the previously saved searches
} // end function start

window.addEventListener( "load", start, false );
```

- From function start we can begin to assemble a mapping of events to their listeners…

event	listener/function-call
saveButton clicked	`saveSearch`
clearButton clicked	`clearAllSearches`
window loads	`loadSearches`

```
var tags; // array of tags for queries

// loads previously saved searches and displays them in the page
function loadSearches()
{
   if ( !sessionStorage.getItem( "herePreviously" ) )
   {
      sessionStorage.setItem( "herePreviously", "true" );
      document.getElementById( "welcomeMessage" ).innerHTML =
         "Welcome to the Favorite Twitter Searches App";
   } // end if

   var length = localStorage.length; // number of key/value pairs
   tags = []; // create empty array

   // load all keys
   for (var i = 0; i < length; ++i)
   {
      tags[i] = localStorage.key(i);
   } // end for

   tags.sort(); // sort the keys

   var markup = "<ul>"; // used to store search link markup
   var url = "http://search.twitter.com/search?q=";

   // build list of links
   for (var tag in tags)
   {
      var query = url + localStorage.getItem(tags[tag]);
      markup += "<li><span><a href = '" + query + "'>" + tags[tag] +
         "</a></span>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Edit' onclick = 'editTag(id)'>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Delete' onclick = 'deleteTag(id)'>";
   } // end for

   markup += "</ul>";
   document.getElementById("searches").innerHTML = markup;
} // end function loadSearches
```

event	listener/function-call
saveButton clicked	`saveSearch`
clearButton clicked	`clearAllSearches`
window loads	`loadSearches`
edit button clicked on item with id x	`editTag(x)`
delete button clicked on item with id x	`deleteTag(x)`

```
// deletes all key/value pairs from localStorage
function clearAllSearches()
{
   localStorage.clear();
   loadSearches(); // reload searches
} // end function clearAllSearches

// saves a newly tagged search into localStorage
function saveSearch()
{
   var query = document.getElementById("query");
   var tag = document.getElementById("tag");
   localStorage.setItem(tag.value, query.value);
   tag.value = ""; // clear tag input
   query.value = ""; // clear query input
   loadSearches(); // reload searches
} // end function saveSearch

// deletes a specific key/value pair from localStor
function deleteTag( tag )
{
   localStorage.removeItem( tag );
   loadSearches(); // reload searches
} // end function deleteTag


// display existing tagged query for editing
function editTag( tag )
{
   document.getElementById("query").value = localStorage[ tag ];
   document.getElementById("tag").value = tag;
   loadSearches(); // reload searches
} // end function editTag
```

event	listener/function-call
saveButton clicked	`saveSearch`
clearButton clicked	`clearAllSearches`
window loads or clearAllSearches or saveSearch or deleteTag or editTag	`loadSearches`
edit button clicked on item with id x	`editTag(x)`
delete button clicked on item with id x	`deleteTag(x)`

event	listener/function-call
saveButton clicked	`saveSearch`
clearButton clicked	`clearAllSearches`
window loads or clearAllSearches or saveSearch or deleteTag or editTag	`loadSearches`
edit button clicked on item with id x	`editTag(x)`
delete button clicked on item with id x	`deleteTag(x)`

```
var tags; // array of tags for queries

// loads previously saved searches and displays them in the page
function loadSearches()
{
   if ( !sessionStorage.getItem( "herePreviously" ) )
   {
      sessionStorage.setItem( "herePreviously", "true" );
      document.getElementById( "welcomeMessage" ).innerHTML =
         "Welcome to the Favorite Twitter Searches App";
   } // end if

   var length = localStorage.length; // number of key/value pairs
   tags = []; // create empty array

   // load all keys
   for (var i = 0; i < length; ++i)
   {
      tags[i] = localStorage.key(i);
   } // end for

   tags.sort(); // sort the keys

   var markup = "<ul>"; // used to store search link markup
   var url = "http://search.twitter.com/search?q=";

   // build list of links
   for (var tag in tags)
   {
      var query = url + localStorage.getItem(tags[tag]);
      markup += "<li><span><a href = '" + query + "'>" + tags[tag] +
         "</a></span>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Edit' onclick = 'editTag(id)'>" +
         "<input id = '" + tags[tag] + "' type = 'button' " +
            "value = 'Delete' onclick = 'deleteTag(id)'>";
   } // end for

   markup += "</ul>";
   document.getElementById("searches").innerHTML = markup;
} // end function loadSearches
```

`getItem` method given a name of a key as argument
- if key exists, method returns string value
- else, null

`setItem` associates a string key with a string value

`localStorage.length` gives # of key/value pairs
- Can access i^{th} key with .key(i)
- Can use getItem method with a key to get its value

```
// deletes all key/value pairs from localStorage
function clearAllSearches()
{
   localStorage.clear();
   loadSearches(); // reload searches
} // end function clearAllSearches

// saves a newly tagged search into localStorage
function saveSearch()
{
   var query = document.getElementById("query");
   var tag = document.getElementById("tag");
   localStorage.setItem(tag.value, query.value);
   tag.value = ""; // clear tag input
   query.value = ""; // clear query input
   loadSearches(); // reload searches
} // end function saveSearch

// deletes a specific key/value pair from localStorage
function deleteTag( tag )
{
   localStorage.removeItem( tag );
   loadSearches(); // reload searches
} // end function deleteTag

// display existing tagged query for editing
function editTag( tag )
{
   document.getElementById("query").value = localStorage[ tag ];
   document.getElementById("tag").value = tag;
   loadSearches(); // reload searches
} // end function editTag
```

`clear` removes all key/value pairs from object

`setItem` adds a new key/value pair to localStorage (or updates value if the key already exists)

`removeItem` removes a key (and its corresponding value) from the localStorage

`Note:` each of these functions ends with a call to loadSearches() ... why?

…and, once again, the initializer function…

```
// register event handlers then load searches
function start()
{
   var saveButton = document.getElementById( "saveButton" );
   saveButton.addEventListener( "click", saveSearch, false );
   var clearButton = document.getElementById( "clearButton" );
   clearButton.addEventListener( "click", clearAllSearches, false );
   loadSearches(); // load the previously saved searches
} // end function start

window.addEventListener( "load", start, false );
```

Using JSON to Represent Objects

- **JSON (JavaScript Object Notation)**
  - A simple way to represent JavaScript objects as strings
  - introduced as an alternative to XML as a data-exchange technique

- Each JSON object is represented as a list of property names and values contained in curly braces, in the following format:
  { *propertyName1 : value1, propertyName2 : value2* }

- Arrays are represented in JSON with square brackets in the following format:
  [*value0, value1, value2*]

- Each value can be a string, a number, a JSON object, `true`, `false` or `null`.

How Do We Parse This JSON?

```
[ { first: 'Cheryl', last: 'Black' },
  { first: 'James', last: 'Blue' },
  { first: 'Mike', last: 'Brown' },
  { first: 'Meg', last: 'Gold' } ]
```

Describe The Following JSON…

```
{"filemenu": {
      "id": "file",
      "name": "File",
      "popup": {
            "menuitem":
            [
              { "value": "New", "whenclicked": "CreateNewDoc" },
              { "value": "Open", "whenclicked": "OpenDoc" },
              { "value": "Close", "whenclicked": "CloseDoc" }
            ]
      }
} }
```

/api/6Br82uLH6al2MiA3furyCaTiqQCrpzKHFtRYTZ4s/lights/1

GET PUT POST DELETE

Message Body:

```
{"on": true, "sat": 100, "bri": 255, "hue": 45000}
```

Command Response:


```
{
        "state": {
                "on": true,
                "bri": 77,
                "hue": 42387,
                "sat": 252,
                "effect": "none",
                "xy": [
                        0.2213,
                        0.1467
                ],
                "ct": 153,
                "alert": "select",
                "colormode": "xy",
                "reachable": true
        },
        "type": "Extended color light",
        "name": "Hue color lamp 1",
```