Edinburgh Napier University

SET081010/SET08702 - Web Tech

Lab 9 - Data & Persistence

Dr Simon Wells

Aims

At the end of the practical portion of this topic you will be able to:

- Persist client side data using HTTP Cookies
- Persist client side key:value data using Web Storage
- Persist data to the filesystem

1 Data Representation & JSON

We are going to use JSON to represent our data. We could use another *data representation language* like eXtensible Markup Language (XML) but JSON happens to work very well with JavaScript[1]. JSON can best be considered a simple and easy to use language for describing data. You can get more information about the language itself from the JSON language webpage at `http://www.json.org`. JSON is easy to write using a text editor but there are also additional tools that help us to check with we have the syntax correct as, like all programming languages, it likes things to be written precisely and correctly. We can use JSON Lint `http://jsonlint.com/` to automatically check whether a given JSON document is corrent. It is also a useful web-based JSON editor for fairly short documents.

Here is a simple example of a JSON document:

```
{
  "firstName": "Jebediah",
  "lastName": "Springfield",
  "isAlive": true,
  "age": 125,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "Springfield",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
}, {
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Try typing that into jsonlint and playing around with it, letting the tool tell you when you've broken things. Use the railroad diagrams from json.org to help you understand how a JSON document is structured. JSON is quite straightforward. The basic *unit* of JSON is the document which can contain an object or a list. In turn an object or list can contain any number of encapsulated objects, lists, or key:value pairs.

Because JSON originally stems directly from JavaScript, it was developed as a way to serialise JS objects as text, hence JavaScript *Object Notation*, it is straightforward in JS to move between a string that contains JSON and an object that instantiates the variables that the JSON string described. The only thing to be aware of is making sure not to get string and object representations of JSON mixed up and attempting to treat one as the other.

[1]in fact the JS of JSON actually stands for JavaScript.

This fragment of JS will give you an idea of how to turn a JSON string, such as the data that you might supply in an API call, or might be sent as the result of calling a third party API:

```
var j_str = '{"firstname":"simon"}';
var j_obj = JSON.parse(j_str);
console.log(j_obj);
console.log(j_str)
```

In our first line we have JSON encoded as a string and stored in j_str. We then use the JSON.parse() method to create a JS object hierarchy which we store in j_obj. The parse method simply converts each element that JSON describes in the string into objects, arrays, or basic datatypes so that they can be used for computation by JS. Note that we print out the value of j_obj and j_str so that you can compare what the string looks likes and the printed object.

Once JSON is parsed from a string into an object we can use it in our code, for example, we can add new or remove existing elements or perform computations with the data. For example, adding a new key to j_obj:

```
j_obj['lastname'] = "wells";
```

Finally we can convert our JS objects into JSON strings, for example:

```
var s = JSON.stringify(j_obj);
console.log(s);
```

This JSON could then be written to a file, persisted in a datastore, sent to a remote API, or returned to our user as the response to an HTTP request. For our purposes, we will probably store our JSON data within the browser using a *client side storage* method.

2 Client Side Storage

Storing data on the client, in the browser, is a useful way to perist data between requests. Because HTTP is *stateless* it doesn't care, at the protocol level, about past or future requests and responses. This leads to a really robust and simple architecture that underpins the web as we know it. However, sometimes we want to do things that require information from previous requests to be available to future requests, for example, information about login status or credentials, or in a game, perhaps an accumulation of points or experience across multiple pages. There are a variety of mechanims for storing data on the client side, starting with cookies as the simplest but most restricted mechanism for very small amounts of data, through Web Storage for larger volumes of key:value structured data. There are also mechanims for browsers to store an unlimited[2] data for implementing browser based applications, such as email clients, that require large amounts of data and performant access to that data. However this final approach is a little complex and is outside of the scope of this module. We'll stick with cookies and web storage for now.

If we want a website to be dynamic, to let user's interact with it beyond just reading static data, then there is a good chance that we'll want to save some of that data. We have a number of strategies to achieve this. We'll take two main approaches to persisting our data, firstly on the client side, and then on the server side.

For the client side we'll look at ways to store data using the browser; cookies and web-storage

Data storage is a huge topic so this lab only really scratches the surface. However, you should, by the end, feel comfortable with persisting data on the server or client, and should be building some skills in deciding which approach to take depending upon the problems that you are presented with.

[2]obviously limited by available drive space

Cookies

We can create cookies from JavaScript. It's not the only way, but it is a good method to get started. You can do the following exercises from the Javascript console in your browser, first ensuring that you have cookies enabled. Cookies have a really simple JavaScript API. For the most part just using a call to document.cookie is sufficient to create, retrieve, update, and delete a cookie.

Enter the following into the console, replacing "your name" with your actual name.

```
    document.cookie = "username=your name";
```

This will add the double-quoted content to the "cookie string" for the current page.

To retrieve our cookie we need to create a variable and then call document.cookie which returns the cookie string. In the next example we will do so then print out the variable using console.log()

```
    var x = document.cookie;
    console.log(x);
```

We can also set an expiry date for a cookie, for example:

```
document.cookie = "password=secret; expires=Thu, 18 Dec 2019 12:00:00 UTC";
```

After the expiry date, that particular key & value pair will be deleted.

If we repeat the retrieval of our cooke from above, we'll notice that our password key has been appended to hte previous string so both username and password are stored in the same cookie. This is because there is just one cookie string for each page. So to retrieve a particular key we need to pull apart the cookie using JavaScript to retrieve the specific key that we are interested in.

For example, to retrieve just the value for the username we could create a function that searches the cookie string for the supplied cookie_name, e.g.

```
function getCookie(cookie_name)
{
    var name = cookie_name + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i <ca.length; i++)
    {
        var c = ca[i];
        while (c.charAt(0) == ' ')
        {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0)
        {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}
```

We can then use our function to retrieve a specific value from the cookie, for example, the username:

```
console.log(getCookie("username"));
```

Deleting a cookie is achieved by setting an expiry date that is in the past, for example:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC;";
```

To update a value stored in a cookie you merely replace the existing key with the new kvalue, for example, here we just set a new username with an updated value which will replace the existing value:

```
document.cookie = "username=Carol Danvers;";
```

Your cookies should probably not live forever, it is good practise to set an expiry date which is a reasonable length of time into the future. To make it easier to program, a nice way to do this is to use a function, for example, this one will create a cookie using the supplied key and value and will set an expiry date based upon the supplied value for the number of days.

```
function setCookie(name, value, expiry)
{
    var d = new Date();
    d.setTime(d.getTime() + (expiry*24*60*60*1000));
    var expires = "expires="+ d.toUTCString();
    document.cookie = name + "=" + value + ";" + expires;
}
```

Now create a simple site using HTML, CSS, & JS that uses cookies to persist some data. A good place to start is to create a simple form that asks some questions about the user. stores the responses in a cookie, then uses the data in the cookie to personalise a message to the user. For example, asking a user what their name is, then whether they are happy or sad and respond with something appropriate that is personalised to them.

Web Storage

This option provides more space for your site to store data on the client. The amount of space is browser specific, and the API is only supported by newer browsers, so you'll have to check first, in code, whether web storage is supported.

The basic idea with web-storage is that you can store keys and values. Both keys and values are always strings, so you may have to do some type conversion for some of your variables. If you have JSON objects, then you can store these by converting them to a string using the JSON.stringify function. Thera are two types of web-storage available, localStorage and sessionStorage. The main difference is that sessionStorage is only available whilst the current window is open and is deleted afterwards, i.e. once your current session is over. However data stored in local storage will be available for ever unless explicitly deleted, for example, by your webapp, or by the user clearing all local data, reformatting their drive, etc.

```
function storageAvailable(type)
{
    try
    {
        var storage = window[type],
            x = '__storage_test__';
        storage.setItem(x, x);
        storage.removeItem(x);
        return true;
    }
    catch(e)
    {
        return e instanceof DOMException && (
            e.code === 22 ||
            e.code === 1014 ||
            e.name === 'QuotaExceededError' ||
            e.name === 'NS_ERROR_DOM_QUOTA_REACHED') &&
            storage.length !== 0;
    }
}
```

In this we attempt to use the storage API and then catch any exceptions that are thrown. Even if storage is available, there are reason why it might not be usable, either due to local settings, such as privacy settings, the way that private browsing works (if you are in a private browsing window then you might have local storage available but the amount of space to store anything in is set to zero), or because you've already filled it up.

We can call our storageAvailable function with the value "localStorage" or "sessionstorage" to see which is available to use.

```
if (storageAvailable('localStorage'))
{
    // We can do something with the localStorage...
}
else
{
    // Have to come up with an alternative way to store data
}
```

We can then actually add data to the storage using the setItem function, for example,

```
localStorage.setItem('name','simon');
```

Checking whether something has been set can be quite useful. We can achieve that like this using the getItem function (we assume that we, as programmers, will know which keys our web pages will have set):

```
if(localStorage.getItem('name')) {console.log("set");}
```

We can also retrieve that data using the getItem function, for example, let's retrieve what we just stored and display it:

```
var myname = localStorage.getItem('name');
console.log(myname);
```

3 Challenges

You should consider how to incorporate appropriate data storage into the sites that we've already developed during previous mini-projects and challenges.

For example, if you've been working on the dungeon-crawler game then cookies are a really good, light-weight method of storing a small amount of data about the current player. For example, the players health, what they are carrying, which locations they've visited. Obviously, you have a small amount of space to store your data in a cookie, but this should be more than sufficient for simple player specific data. If you are generating a larger dungeon, then you might want to consider using one of the other types of client-side storage. The nice thing about this approach is that each time your player revisits the site they should be able to pick up at exactly the place that they left off. This is a nice way to persist a game whilst maintaining a very lightweight, casual approach for your player.