Maintaining User State with Cookies

User State As We Know It

- Right now, our apps tend to lose all state upon refresh or visiting a new page
- Sometimes we store things in the database, but we can only track those by keeping query params in the URL, or showing all of the contents and having the user choose it
- What we need is to have the user keep some data in between page loads, or visits to our site

Introducing the Cookie

- It's probable you've heard the term "cookie" before as it relates to your web browser
- Cookies are a dead simple key / value store that lives in your user's browser
- Think of it as like having one Javascript object that doesn't go away between page loads
- Using this, we can record things about the user, and use them again later
- We can also access cookies both on the client, AND the server, to keep the behaviour consistent in both places

Cookie Use Cases

- Keep any client side state across page loads, such as expanding a sidebar, or changing your site's theme
- Show "tutorials" for how to use your site, for first time visitors
- Keep a user logged in by giving them a token that your server can authenticate them with
- Track where a user has gone on your website

Using Cookies Client-Side

```
document.cookie = "key1=value1; key2=value2;";
```

- Cookies are unfortunately a little annoying to use by default
- They accept a string of key value pairs, not an object, separated by semicolons
    - This means that cookies can only be Strings, not complex data types
- Cookie values also can contain information about how long they should stick around
- In addition to these oddities, not all browsers support cookies in the same way, and users can disable cookies altogether

Using Cookies Client-Side with `js-cookie`

- Because of all the weirdness from before, a library known as `js-cookie` was made to make dealing with cookies much easier
- It provides a `Cookies` singleton that comes with a few simple functions:
  - `set(key, value, options)` - Set a cookie key to a particular value
  - `get(key)` - Get a cookie at a key, or leave out key to get all cookies as an object
  - `getJSON(key)` - Same as `get`, but attempt to convert its type
  - `remove(key)` - Delete the value of a cookie at a key
- Despite

Simple `js-cookie` Example

```
var Sidebar = {
    // Grab sidebar element, open state from cookie
    // isOpen defaults to false if there is no cookie
    init: function() {
        this.$sidebar = $("#sidebar");
        this.isOpen = Cookies.getJSON("sidebar-open") || false;
    },

    // Toggle isOpen from true to false, or vice versa
    // Save isOpen to the cookie, once toggled
    toggle: function() {
        this.isOpen = !this.isOpen;
        Cookies.set("sidebar-open", this.isOpen);
        this.render();
    },

    // Add / remove isOpen class depending on sidebar state
    render: function() {
        this.$sidebar.toggleClass("isOpen", this.isOpen);
    },
}
```

Cookie.set Options

```
Cookie.set('key', 'value', {
    // How long the cookie should stick around, in number
    // of days, or a Date object for when it should expire.
    // Defaults to expiring when the user closes their browser.
    expires: 365,

    // Make the cookie only accessible for certain paths. Useful
    // for if you want the same cookie to behave differently
    // on certain pages. Defaults to none, accessible everywhere.
    path: "/some/site/path",

    // Indicate the cookie should only be available over HTTPS.
    // Useful for sensitive data, but not something we can do
    // right now, since our servers are HTTP.
    secure: true,
});
```

Reading Cookies Server Side

```
# In termianl
npm install --save cookie-parser


// app.js
const cookieParser = require("cookie-parser");
app.use(cookieParser());

app.get("/", function(req, res) {
    console.log(req.cookie.isOpen); // "false"
})
```

- To read Cookies in express, we'll need a new Middleware called `cookie-parser`
- This adds a new key to the Request, `req.cookies`
- This is an object that has the same keys and values as in `document.cookie`

Writing Cookies Server Side

```
res.cookie("key", "value", {
    // One hour from now
    expires: new Date(Date.now() + 3600000),
    // A little easier, also expires one hour from now
    // 60 minutes * 60 seconds * 1000 milliseconds
    maxAge: 60 * 60 * 1000,
});
```

- Out of the box (No modules needed) the Response object has a function at `res.cookie(name, val, options)`
- The function signature is nearly identical to `js-cookie`'s `Cookie.set`
- However, the `expires` option *must* be a Javascript `Date` object
- Instead, you can use `maxAge` to set how many milliseconds the cookie should live for

Cookie Caveats: JSON and Types

- In the previous example, we saw `req.cookie.isOpen` *was the string* **`"false"`**, *not the boolean* **`false`**
- This is because cookies are treated as a string
- `js-cookie` and `cookie-parser` both do JSON encoding, but they're not compatible with each other
- Object / array / number / boolean types are saved between `Cookie.set` and `Cookie.get`, but not in `req.cookie`
- Because of this, you're best off keeping cookies as simple as possible
  - However, you can get around this by manually `JSON.stringify` and `JSON.parse`ing your cookies wherever you use them, but this is not advised

Cookie Caveats: Security

- Cookies live on the client's computer
- So much like client-side javascript, cookies are **not secure**
- Users can read and alter cookies any way they want, so you should not store sensitive data in there like passwords
- You also should not rely on regular old cookies for *security*, only for *configuration*
    - e.g. if we had a `"username"` cookie that logged someone in, anyone could set the value to anyone else's username and have full access to their account

Assignment: Required Cookie Middleware

- We're going to build a website that requires the user to identify themselves before accessing our website
- In order to do that, we'll build a `nameCookie.js` middleware
- It will check for the existence of of a `name` cookie, and if it doesn't exist, present a `<form>` to the user that sets the cookie.
  - You can set the cookie in JS by intercepting the form `submit` event and setting the cookie using `js-cookie`'s `Cookie.set()` function
  - OR you can set the cookie in the POST request using Express' `res.cookie()` function
- If they have a `name` cookie, present them with the page they asked for
  - Your page should use this cookie to greet the user
- Once you've set the cookie one way (client or server), try to set it using the other way

Hint: This middleware is VERY similar to the **`passwordForm.js`** *middleware*

Additional Reading

- Cookies in Javascript - WAY more in depth on Cookies using document.cookie, rather than a module. Probably more than you need to know, but cool if you're curious.
- Cookie Management in Express - Very simple, no frills examples of how to use cookies in Express. Good reference.
- `js-cookie`'s Docs
- `cookie-parser`'s Docs