CWEB280 -wk6 – LO3 -Cookies and Sessions – State Management

# Cookies

Cookies are small strings (less than 4KB) that contain key-value pairs of information.

* Cookies are sent from the webserver to the browser to store some information about the user.
* Cookies are saved by the browser locally on the client.
* All following requests made by the browser will quietly send the cookies as part of the request to the server
* The server can immediate change update the content based on the cookies contents
* A cookie is HTTP generated; thus, called an HTTP cookie.

## Cookie Options

Cookies have some properties that can be set when they are created.

* Just like JWTs one important setting is the max age or expiry when the cookie is deleted from local storage the browser
* Unlike JWTs the sever can update/change the cookie’s expiry to either extend or shorten its life.
* Domain and path settings tell the browser to only send the cookie when making a request to a certain website and path otherwise do not send that cookie
* HttpOnly setting – tells the browser to make the cookie hidden (true) or shown (false) to javascript .
  + This will become important later when we need to send cookies to API web apps

Learn More about using cookies with Express : <https://www.section.io/engineering-education/what-are-cookies-nodejs/>

## cookie-parser

* cookie-parser looks at the headers in between the client request and the server response and parses out the cookies being sent.
* cookie-parser will help create and manage cookies.
* cookie-parser exposes a new property in the request object - ***req.cookies***

Learn more about cookie parser: <https://www.npmjs.com/package/cookie-parser>

# Make a Form that will set Cookies with Different Values and Options

OPTIONAL: In the terminal run (cookie parser should already be included in our project)

npm i cookie-parser

**\app.js - Ensure cookie parser is already be in the app**

const createError = require('http-errors');  
const ***express*** = require('express');  
const ***path*** = require('path');  
const cookieParser = require('cookie-parser');

## a new route file

**Create a new router called state.js**

**\routes\state.js**

const ***express*** = require('express');  
const ***router*** = ***express***.Router();  
/\* eslint-disable max-len \*/  
  
*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* GET Set Cookie  
 \*/****router***.get('/cookie', (req, res, next)=>{  
 res.render('set-cookie', {  
 title: 'GET - Set Cookie',  
 activeCookies: req.cookies, // cookies sent to the server  
 });  
});  
  
  
*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* POST Set Cookie  
 \*/****router***.post('/cookie', (req, res, next)=>{  
 // set some standard cookie options  
 const cookieOptions = {  
 path: req.baseUrl,  
 sameSite: 'lax',  
 httpOnly: req.body.hide && req.body.hide ==='yes', // setting httpOnly to true hides the cookie from Javascript in the browser  
 };  
  
 // did the user click the Clear all Cookies button  
 if (req.body.clear && req.body.clear ==='clear') {  
 ***console***.log(req.cookies);  
 // loop thorough all the cookies and TRY to remove all cookies  
 for (const cookieName in req.cookies) {  
 // clearCookie is very strict - will only clear the cookie if name AND options are the same  
 // it seems to ignore the expiry date when checking the cookie options  
 res.clearCookie(cookieName, cookieOptions);  
 }  
 } else {

cookieOptions.maxAge = 1000 \* req.body.expiry; // value in milliseconds

// set the cookie with the name the value and the options  
 // server sends a Set-cookie header instructing the browser save the cookie locally  
 res.cookie(req.body.name, req.body.value, cookieOptions);  
 }  
  
 res.render('set-cookie', {  
 title: 'POST - Set Cookie',  
 activeCookies: req.cookies, // cookies sent to the server  
 postedValues: req.body,  
 });  
});

Add the route and path in app.js

**\app.js - add the code in violet**

const ***indexRouter*** = require('./routes/index');  
const ***usersRouter*** = require('./routes/users');  
const ***examplesRouter*** = require('./routes/examples');  
const ***secureRouter*** = require('./routes/secure');  
const ***stateRouter*** = require('./routes/state');

.

.

.

***app***.use('/examples', ***examplesRouter***);  
***app***.use('/secure', ***secureRouter***);  
***app***.use('/state', ***stateRouter***);

## Create New Handlebar template

**\view\set-cookie.hbs**

<form action="/state/cookie" method="post">  
 <h1>{{ title }}</h1>  
 <div class="mb-2">  
 <label for="name" class="form-label">Cookie Name:</label>  
 <input type="text" class="form-control "  
 placeholder="Enter cookie name" name="name" id="name" value="{{postedValues.name}}" required/>  
 </div>  
 <div class="mb-2">  
 <label for="value" class="form-label">Cookie Value:</label>  
 <input type="text" class="form-control "  
 placeholder="Enter cookie value" name="value" id="value" value="{{postedValues.value}}" required/>  
 </div>  
 <div class="mb-2">  
 <label for="expiry" class="form-label">  
 Cookie Expiry: (Integer in seconds, -1 to expire cookie)  
 </label>  
 <input type="number" class="form-control "  
 placeholder="Enter cookie expiry in seconds" name="expiry" id="expiry" value="{{postedValues.expiry}}" required/>  
 </div>  
 <div class="form-check form-switch">  
 <input class="form-check-input" type="checkbox" id="hide" name="hide" value="yes"  
 {{#if postedValues.hide}}checked{{/if}}>  
 <label class="form-check-label" for="hide">Hide cookie from javascript</label>  
 </div>  
 <button type="submit" class="btn btn-primary">Submit</button>  
</form>  
<form action="/state/cookie" method="post">  
<button type="submit" name="clear" value="clear" class="btn btn-primary">Clear all Cookies</button>  
</form>  
<div class="card p-2 mb-2">  
 <h2>Posted values</h2>  
 {{#each postedValues}}  
 <p class="col-md-8 fs-4">{{@key}}: {{this}}</p>  
 {{/each}}  
</div>  
<div class="card p-2">  
 <h2>Cookies Sent to Server</h2>  
 {{#each activeCookies}}  
 <p class="col-md-8 fs-4">{{@key}}: {{this}}</p>  
 {{/each}}  
</div>  
<div class="card p-2">  
 <h2>Cookies Available to Javascript</h2>  
 <script type="text/javascript">  
 let ***keyVals*** = ***document***.cookie.split(';');  
 ***keyVals***.forEach( item=>{  
 let line = item.split('=');  
 if(line.length===2) {  
 ***document***.writeln(`<p class="col-md-8 fs-4"> ${line[0]}: ${line[1]}</p>`);  
 }  
 });  
 </script>  
</div>

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/state/cookies>

Exercise

What happens when you first post the form?  
Do you see the cookie you created listed in the Cookies Sent to Server section?   
What happens if you leave and come back to this page?

What happens when you make turn on the “Hide cookie from javascript” option and post the form?   
Do you see the cookie in the “Cookies Available to Javascript” section?  
What happens if you leave and come back to this page?

# Sessions

Session are way to store information about a user’s previous interactions with the website.

* Session information is usually stored in the server memory for quick access.
* Session information is usually backed up into a file or a database on the server depending on server resources
* Sessions can store entire objects or arrays not just key value pairs like cookies
* A property created in the session object is often referred to as a ***session variable***
* Sessions have a unique identifier for each user to know what data is associated to the user
* Sessions make use of a cookie to store the unique identifier on the user’s browser
* Sessions are more secure because no information is stored on the client machine other than the ID
* Sessions do use up server resources and require well thought out strategies as to which information to store

## express-session

* express-session will make I easier to set and store session variables.
* express-session can store the sessions data in many different places depending on configuration used/
* express-session requires a secret be used to encode the id in a cookie
  + cookie-parser must use the same secret or we may run into problems with the session id cookie

Example Configuration Options:  
const sessOptions = {  
 // set secret from environment variable used to encode the session cookie  
 secret: ***process***.env.SECRET, // should be the same for cookie-parser  
 name: 'session-id', // name of the session cookie default: connect.sid  
 resave: false, // store session after every request  
 saveUninitialized: false, // if true sets a cookie even if no session info is stored  
 cookie: {httpOnly: false, maxAge: 1000\*60\*60}, // cookie options - see cookie-parser docs  
 unset: 'destroy', // instruction for stored session when unset  
 store: new SqliteStore({ // a location to store session besides the memory  
 client: new Sqlite('sessions.db', {verbose: ***console***.log}),  
 expired: {clear: true, intervalMs: 1000\*60\*15},  
 }),  
};

Learn more about Express Session: <https://expressjs.com/en/resources/middleware/session.html>

# Make a Form that will set Session Variables with Different Values and Options

In the terminal run express session and Sqlite packages to store the sessions in a Sqlite database

npm i express-session better-sqlite3 better-sqlite3-session-store

**\app.js - Ensure sessions are required and configured**

// require packages to implement Sessions in express  
const session = require('express-session');  
const Sqlite = require('better-sqlite3'); // sqlite database driver  
const SqliteStore = require('better-sqlite3-session-store')(session); // helps store session in an sqlite database  
const sessOptions = {  
 // set secret from environment variable used to encode the session cookie  
 secret: 'shhhhhhh\_this-is+SECret' , // should be the same for cookie-parser  
 name: 'session-id', // name of the session cookie default: connect.sid  
 resave: false, // store session after every request  
 saveUninitialized: false, // if true sets a cookie even if no session info is stored  
 cookie: {httpOnly: false, maxAge: 1000\*60\*60}, // cookie options - see cookie-parser docs  
 unset: 'destroy', // instruction for stored session when unset  
 store: new SqliteStore({ // a location to store session besides the memory  
 client: new Sqlite('sessions.db', {verbose: ***console***.log}),  
 expired: {clear: true, intervalMs: 1000\*60\*15},  
 }),  
};

.

.

.

***app***.use(logger('dev'));  
***app***.use(***express***.***json***());  
***app***.use(***express***.***urlencoded***({extended: false}));  
***app***.use(cookieParser(sessOptions.secret));// should have the same secret as the session cookie  
***app***.use(***express***.***static***(***path***.join(\_\_dirname, 'public')));  
// configure passport to create the req.currentUser property with the token payload  
// in order to use passport with express we need to initialize passport  
***app***.use(***passport***.initialize({userProperty: 'currentUser'}));  
// configure session session options  
***app***.use(session(sessOptions));

## Add session Handlers to the Router

**\routes\state.js - add the following to the bottom of the file ABOVE module.exports = router;**

*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* GET Set Session  
 \*/****router***.get('/session', (req, res, next)=>{  
 res.render('set-session', {  
 title: 'GET - Set Session',  
 sessionID: req.sessionID,  
 activeSession: ***JSON***.stringify(req.session, null, 4), // session nested object structure  
 });  
});  
  
*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* POST Set Session  
 \*/****router***.post('/session', (req, res, next)=>{  
 // declare callback function for session actions  
 const callback = (err)=>{  
 if (err) throw err;  
 };  
  
 // determine the purpose of the post  
 switch (req.body.purpose) {  
 case 'regenerate':

// you should call regen when you login a user – to get a new session id  
 req.session.regenerate(callback);  
 break;  
 case 'destroy':

// call destroy when you want logout a user  
 req.session.destroy(callback);  
 break;  
 case 'reload':  
 req.session.reload(callback);  
 break;  
 default:  
 // if the session does not contain the posted category - then initialize to an empty object  
 if (req.body.category && !(req.session.hasOwnProperty(req.body.category))) {  
 req.session[req.body.category] = {};  
 }  
 // determine id new session name and value are part of a category or root session object  
 const sess = req.body.category? req.session[req.body.category] : req.session;  
 sess[req.body.name] = req.body.value;  
 }  
  
 res.render('set-session', {  
 title: 'POST - Set Session',  
 sessionID: req.sessionID, // the session id used to uniquely id the current user  
 activeSession: ***JSON***.stringify(req.session, null, 4), // session nested object structure  
 postedValues: req.body, // remember what was posted  
 });  
});

module.exports = ***router***;

## Create New Handlebar template

**\view\set-session.hbs**

<form action="/state/session" method="post">  
 <h1>{{ title }}</h1>  
 <div class="mb-2">  
 <label for="name" class="form-label">Session Name:</label>  
 <input type="text" class="form-control" value="{{postedValues.name}}"  
 placeholder="Enter Session name" name="name" id="name" required/>  
 </div>  
 <div class="mb-2">  
 <label for="value" class="form-label">Session Value:</label>  
 <input type="text" class="form-control " value="{{postedValues.value}}"  
 placeholder="Enter Session value" name="value" id="value" required/>  
 </div>  
 <div class="mb-2">  
 <label for="expiry" class="form-label">Session Category: (Optional) </label>  
 <input type="text" class="form-control " value="{{postedValues.category}}"  
 placeholder="Enter Session Category" name="category" id="category" />  
 </div>  
 <button type="submit" name="purpose" value="add" class="btn btn-primary">Submit</button>  
</form>  
<form action="/state/session" method="post">  
 <button type="submit" name="purpose" value="regenerate" class="btn btn-primary">Regenerate Session</button>  
 <button type="submit" name="purpose" value="destroy" class="btn btn-primary">Destroy Session</button>  
</form>  
<div class="card p-2">  
 <h2>Session: {{sessionID}}</h2>  
 <code><pre>{{activeSession}}</pre></code>  
</div>  
<div class="card p-2 mb-2">  
 <h2>Posted values</h2>  
 {{#each postedValues}}  
 <p class="col-md-8 fs-4">{{@key}}: {{this}}</p>  
 {{/each}}  
</div>

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/state/session>

Exercise

What happens when you post the form with no category?  
What happens when you post the form with a category?  
Can you keep adding more properties to the category?  
What happens when you specify a new value to an existing property (aka session variable)?  
What happens when you call Destroy or Regenerate?