Table of Contents

[USEFUL LINKS 1](#_Toc494559178)

[BASICS - NEED TO KNOW 1](#_Toc494559179)

[INSTALLING 2](#_Toc494559180)

[SETUPING IT UP FOR USE 2](#_Toc494559181)

[USING SESSIONS IN THE SERVER 4](#_Toc494559182)

[USING COOKIES IN THE CLIENT SIDE 5](#_Toc494559183)

[IMPLEMENTING “REMEMBER ME” FUNCTINALITY FOR LOGIN/REGISTER TO KEEP USERS LOGGED IN OR NOT 7](#_Toc494559184)

\*This file shows how you can use sessions and store them in mongoDB. Some of the bullets below are simplified ignoring things you would already need/install in a full mean app (such as mongoose)

# USEFUL LINKS

* connect-mongo : <https://github.com/jdesboeufs/connect-mongo>
* Express sessions - <http://expressjs-book.com/index.html%3Fp=128.html>
* connect sessions - <http://www.senchalabs.org/connect/session.html>
* Cookies and sessions, the basics - <http://www.lassosoft.com/Tutorial-Understanding-Cookies-and-Sessions>
* stack overflow, how to set individual session maxAge in express - <https://stackoverflow.com/questions/9714785/how-to-set-individual-session-maxage-in-express>
* stack overflow, how to set expiration date for cookie in angularjs - <https://stackoverflow.com/questions/12624181/how-to-set-expiration-date-for-cookie-in-angularjs>
* w3Schools, javascript object dates - <https://www.w3schools.com/jsref/jsref_obj_date.asp>

# BASICS - NEED TO KNOW

* The method detailed below uses session (with cookies in them) on the server side and cookies in angularjs

# INSTALLING

* **SERVER SIDE**: You need to install express (which you already have to), express-session to use sessions, and connect-mongo to save your sessions in mongoDB, the commands are below:

**npm install --save express**

**npm install --save express-session**

**npm install --save connect-mongo**

* **CLIENT SIDE:** You need to install angular and angular-cookies, with the commands shown below. May have to add “**sudo**” before the commands, and “**--allow-root**” after.

**bower install --save angular**

**bower install --save angular-cookies**

# SETUPING IT UP FOR USE

* **SERVER SIDE - required**: In your server.js file, you need to require express, express-session, and connect-mongo, as shown below:

**var express = require('express'),**

**app = express(),**

**session = require('express-session')**

**MongoStore = require('connect-mongo')(session)**

* **SERVER SIDE - SETTING UP SESSION USE AND MONGOSTORE**: Also in your server.js file, after the required files and variables above, you include the below code, which initializes the session and sets it up for use. The proper format is “**app.use( session({code here…}) )**”. Now to explain each part of the object, the "secret:" is a required option and used to sign the session ID cookie, this can either be a single string or an array of multiple secrets. "resave:" forces the session to be saved back to the session store, even if the session was never modified during the request. The default is true, typically you want it to be false. If "saveUninitialized:" is set to true, it forces a session that is uninitialized to be saved to the store. A session is uninitialized when it is new but not modified. Setting this to false is useful for implementing login sessions, reducing server storage usage, or complying with laws that require permission before setting a cookie. See the code below:

app.use(session({

secret: 'testingoutsessions',

resave: false,

saveUninitialized: false,

store: new MongoStore({url: 'mongodb://localhost/session\_experiments'})

}))

* **SERVER SIDE - EXPLAINING MONGOSTORE**: Continuing with the bullet point above, we’ll talk about “store: new MongoStore({})”. this creates a new connection with mongoDB to store the session, typically I make the last part of the url the same as the name of the database for that project, “session\_experiments” in this case. Anything after the url in the MongoStore object are advanced options, we’ll talk about one below, “ttl”.
* **CLIENT SIDE**: to use angular-cookies on the front end with angular, you need to include the two green scripts below in index.html, add “ngRoute” to you list of injected dependent modules in your angular module, and add $cookies to your controller and factories, as shown below:

INDEX.HTML BELOW-

<script src="angular/angular.js" charset="utf-8"></script>

<script src = 'angular-cookies/angular-cookies.js'></script>

APP.JS FILE BELOW-

var app = angular.module('myApp', ['ngCookies'])

FACTORY BELOW-

app.factory('loginRegFactory', [ '$cookies', function(cookie){

var factory = {};

return factory;

}])

CONTROLLER BELOW-

app.controller('loginRegController', ['loginRegFactory', ‘$cookies’, function(fact, cookie){

var self = this

}])

# USING SESSIONS IN THE SERVER

* **BASICS - SETTING, ACCESSING, AND REMOVING**: After setting everything up in server.js, the process to work with a session is simple. We’ll work with the standard way I format my methods in my server controllers for this example, say I have the method “register”, it would look like “**register: function(req, res) {}”**, in order to anything with express sessions, we use “req.session”, basically put the first parameter for the function ahead of “.session”. An express session is just a javascript object, so we treat it like one. A basic session before we do anything to it looks like:

Session {

cookie:

{ path: '/',

\_expires: null,

originalMaxAge: null,

httpOnly: true } }

* **STILL BASICS - SETTING, ACCESSING, AND REMOVING**: Now, say we wanted to plug the user’s info into the session, we could do “**req.session.user = user**” or “**req.session.user = req.body**”, both simply create a key in the session and make it equal to something, in this case another object. See how it would look after doing that below, but lastly, we destroy all sessions with “**req.session.destroy()**”, now see what req.session.user would look like:

Session {

cookie:

{ path: '/',

\_expires: null,

originalMaxAge: null,

httpOnly: true },

user:

{ first\_name: 'last',

last\_name: 'try',

email: 'l@try.com',

password: 'dddddddd',

confirm: 'dddddddd',

remembered: true,

\_id: '59cec2d8e11976b3d8b151e3' } }

* **SETTING THE SESSION TO EXPIRE, METHOD 1, MAXAGE (PREFERRED METHOD):** Let’s say we don’t want to session to live forever, which it would normally, then we want to make it expire or be deleted after a set amount of time. The currently preferred way of doing this is setting the “maxAge” or “originalMaxAge” of the cookie within the session. Every express session has a cookie, so setting it to expire after a certain amount of time will cause the entire session to disappear. We do this like so: “**req.session.cookie.maxAge = 60000**”, this would cause the session to be deleted after one minute. We set maxAge with milliseconds, and 1000 milliseconds equals on second, so 60,000 milliseconds is equal to 60 seconds or one minute. You can set maxAge to anything you want, but it needs to be in milliseconds, so I’ve included some other increments below, do the math to make any changes, just take out the commas:

30 min = 1,800,000 ms

1 hour = 3,600,000 ms

1 day = 86,400,000 ms

1 week = 604,800,000 ms

* **SETTING THE SESSION TO EXPIRE, METHOD 2, TTL:** The second method is setting a universal “time to live” or “ttl” for the MongoStore in the server.js file, see the code below. ttl is one of the advanced options for Mongostore, the ttl determines how long the session will live or persist in our database, once the determined amount of time has passes, the session is removed. This type of thing is what causes you to have to log back in after not signing in for a few days. If the user interacts with the server, the expiration date is refreshed. The format is somewhat complicated and difficult to google, but ttl is listed in seconds. Basically the below code is saying 14 days times 24 hours times 60 minutes times 60 seconds, or 14 days. If we say, changed it to "2 \* 24 \*60 \* 60", then it would expire after 2 days exactly. See the example code below:

app.use(session({

secret: 'testingoutsessions',

resave: false,

saveUninitialized: false,

store: new MongoStore({url: 'mongodb://localhost/session\_experiments',

ttl: 14 \* 24 \* 60 \* 60

})

}))

# USING COOKIES IN THE CLIENT SIDE

* **BASICS - CREATING, ACCESSING, AND REMOVING:** Using cookies in the controller or factory after setting it up is simple. You create it with “**$cookies.put(‘key’, value)**”, then access it with “**$cookies.get(‘key’)**”, and remove it with “**$cookies.remove(‘key’)**. For example, create it with “**$cookies.put(‘id’: response.data.id)**”, access it with “**$cookies.get(‘id’)**”, and remove it with “**$cookies.remove(‘id’)**”.
* **SETTING COOKIES TO EXPIRE:** You set the expiration date for a cookie with the optional “**{ ‘expires’: date\_here}**”, which you set after the value in “$cookies.put(‘key’, value)”. So, we start by creating a datetime variable with "var today = new Date()", if we console.log it, it looks like: “Sat Sep 30 2017 10:47:26 GMT-0500 (CDT)” . Now, for these examples we'll be using get/setSeconds(), but there are other methods we'll touch on below. The way we determine the date to expire, is after creating the first date, we create another date, using the first one as a parameter for Date(), see below. We use the first date as a parameter so that they will be the exact same datetime, otherwise a small amount of time could be between them, which wouldn't be a big deal normally. Now that we have both date variables, we use setSeconds() to set the seconds, basically if we did: "test.setSeconds(15)", it wouldn't increase the seconds by 15, or it would set the time part of the date to the 15 second mark. However, if we go over, or go over 60 seconds, then it should get added. But, we also use it in conjunction with "(today.getSeconds() + 120)", basically, this says to get today's date, grab the seconds, and add 120 seconds to it, and use that to set the seconds for the test variable. So, when we first set "test" it would look like this: "Sat Sep 30 2017 11:14:21 GMT-0500 (CDT)" but after adding 120 seconds (2 minutes) with the below code, it looks like: "Sat Sep 30 2017 11:16:21 GMT-0500 (CDT)", exactly 2 minutes later. Once we determine the date we want to the cookie to expire on, when we assign/create the cookie with "$cookies.put()", we add the optional parameter "{'expires': }", and set expires to the expiration date, 'test' in this case. After doing this, the cookie will die, or disappear after 2 minutes have passed. The other date methods we can use are detailed below, after the example code:

var today = new Date()

var test = new Date(today)

test.setSeconds(today.getSeconds() + 120)

cookie.put('id', response.data.id, {'expires': test})

* **OTHER DATE METHODS YOU CAN USE:** We can use the setSeconds and getSeconds combo to set the cookie to die after so many seconds, but for larger amounts, it's better to use a more appropriate method, some examples are listed below. All the relevant info and methos are located here: <https://www.w3schools.com/jsref/jsref_obj_date.asp>

To set the time to expire to 2 minutes ahead via seconds:

var today = new Date()

var test = new Date(today)

test.setSeconds(today.getSeconds() + 120)

To set the time to expire to 2 minutes ahead via minutes:

var today = new Date()

var test = new Date(today)

test.setMinutes(today.getMinutes() + 2)

To set the time to expire to 2 hours ahead:

var today = new Date()

var test = new Date(today)

test.setHours(today.getHours() + 2)

To set the time to expire to 2 months ahead:

var today = new Date()

var test = new Date(today)

test.setMonth(today.getMonth() + 120)

# IMPLEMENTING “REMEMBER ME” FUNCTINALITY FOR LOGIN/REGISTER TO KEEP USERS LOGGED IN OR NOT

* A simple way to implement this by adding a checkbox input inside a register/log in form, like so:

<form>

Email: <input type="text" ng-model='lrc.log.email'>

Password: <input type="password" ng-model='lrc.log.password'>

Do you want to be remembered: <input type="checkbox" ng-model="lrc.log.remembered">

<input type="submit" ng-click='lrc.login()' value="Login">

</form>

* Next, inside the server, after registering/logging in, when you create the server session, you could do this, where “req.body.remembered” would equal true if they clicked the checkbox:

if (req.body.remembered) {

req.session.cookie.expires = null

} else {

req.session.cookie.maxAge = 60000 //This is milliseconds, equal to one minute, 1000 milliseconds equals one second

}

* Lastly, you basically do the same thing in the factory when you create the cookie, after you create the session and get a response from the server, like so:

if (data.remembered) {

cookie.put('id', response.data.id)

} else {

var today = new Date()

var date\_to\_expire = new Date(today)

date\_to\_expire.setSeconds(today.getSeconds()+60)

cookie.put('id', response.data.id, {'expires': date\_to\_expire})

}

END