Basic Authentication for json-server

Mitch Dresdner

Table of Contents

Simple approach for adding authentication to your requests

Summary

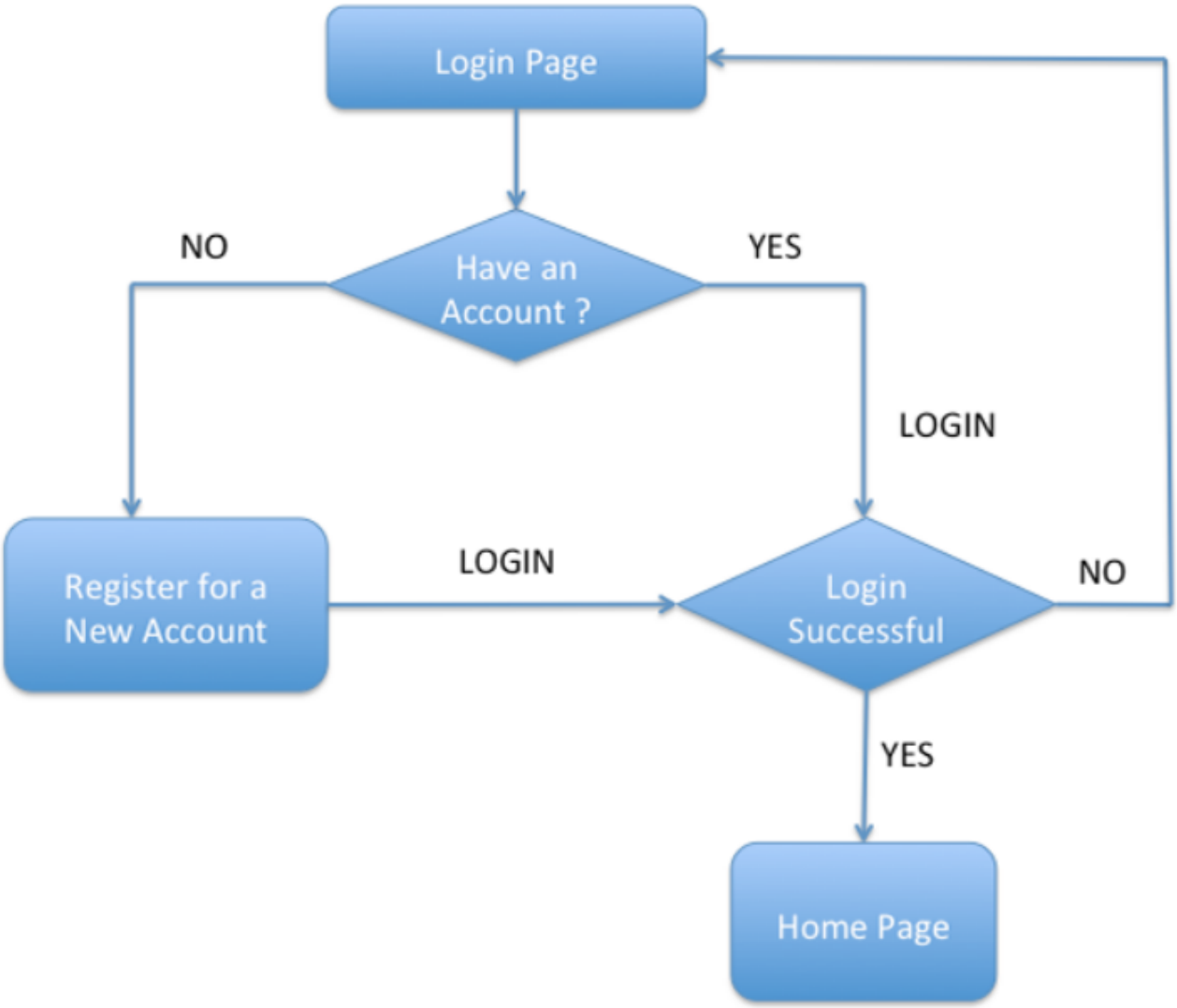
In my last DZone article on Zero Code REST with json-server I showed you how to standup a REST server with minimal effort and *no coding*.

While this approach may work well in many situations, you often find that you'll to need extend capabilities, so you'll soon roll up you sleves and start bending some code. The fun stuff!

This article will share an approach for adding authentication to json-server using a simple, unobtrusive Express middleware component called Passport.

Architecture

Most sites require users to register and then use their credentials for access. Passport provides the layers necessary for authenticating claims made by users logging in to the site.



Basic Authentication is the simplest form of authenticating users, consisting of a user name and a secret password. Being the simplest, it's arguably also the weakest form and vulnerable to intercept and exploitation when not used with secure transports.

The intent of this article is to share how we can start moving from plain text unauthenticated methods of communicating with our json-server to more robust solutions. Passport provides an extensible set of plugins known as *strategies* which will help you to do just that. At the time of this writing there are nearly 500 authentication strategies to choose from including:

- Basic Authentication
- Facebook
- Twitter
- Google
- LinkedIn

Strategies can range from verifying username and password credentials, delegated authentication using OAuth (using Facebook or Twitter), or federated authentication using OpenID.

Passport Strategy

Passport strategies are generally of the following form:

Example Passport Strategy

```
passport.use(new Strategy(
  function(username, password, cb) { ①
    db.users.findByUsername(username, function(err, user) { ②
      if (err) { return cb(err); } ③
      if (!user) { return cb(null, false); } ④
      if (user.password != password) { return cb(null, false); } ⑤
      return cb(null, user);
    });
  }));
```

① Authentication credentials

② Callback to verify access

③ Error handling

④ User Verification

⑤ Password authentication

We'll now integrate passport authentication with the json-server we created in the last article.

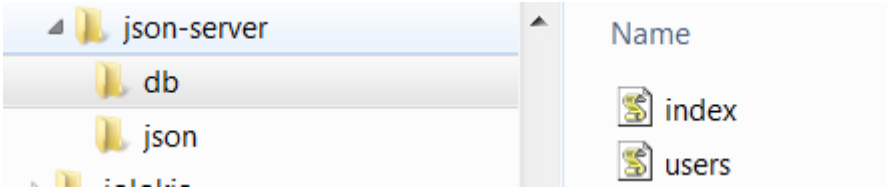
For the sake of brevity and focusing on authentication, this example doesn't use TLS as any real world application should.

Configuring User Authentication

Passport author Jared Hanson, provides us with many a good examples of Strategy implementation, as good OO practitioners we'll follow the mantra to *reuse* rather than *reinvent*, leveraging his Basic Authentication example.

Our new folder hierarchy now includes a *db* folder, simulating a database containing authentication credentials.

Passport authentication support



index.js Listing

index.js

```
exports.users = require('./users');
```

users.js Listing

users.js

```
var records = [
    { id: 1, username: 'jack', password: 'secret', displayName: 'Jack', emails: [ {
value: 'jack@example.com' } ] }
  , { id: 2, username: 'jill', password: 'birthday', displayName: 'Jill', emails: [ {
value: 'jill@example.com' } ] }
];

exports.findByUsername = function(username, cb) {
  process.nextTick(function() {
    for (var i = 0, len = records.length; i < len; i++) {
      var record = records[i];
      if (record.username === username) {
        return cb(null, record);
      }
    }
    return cb(null, null);
  });
}
```

Configuring NPM packaging

In our earlier Zero Code example, we were able to start the json-server with static assets in our project hierarchy. As we add custom code we'll need a way of leting NPM know about our project dependencies. This is accomplished by adding *package.json*.

package.json

```
{
  "name": "custom-json-server",
  "version": "1.0.0",
  "description": "Custom json-server",
  "main": "server.js",
  "dependencies": {
    "json-server": "^0.12.1",     ①
    "passport": "^0.2.2",         ②
    "passport-http": "^0.3.0"     <
  },
  "devDependencies": {
    "json-server": "^0.12.1"

  },
  "scripts": {
    "serve": "node server.js --watch ./json/db.json",   ③
    "test":       "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "json",
    "api"
  ],
  "author": "M. Dresdner",
  "license": "MIT"
}
```

① json-server dependencies

② Passport dependencies

③ Script for starting our server

After creating the *package.json* we'll need to install the required packages. This is accomplished by running the *npm install* command or the short form *npm i*.

npm install

```
npm i
```

You should see npm pull down all the required packages and save them into folder *node_modules*.

With json-server and passport packages installed we turn our attention toward creating our custom code *json-server*.

Configuring Server.json

Our customized rendition of json-server allows for additional middleware products in our json-server implementation. To add the *Basic Authentication* support, we start with the sample code described in the *Module* section of the json-server website and add our references to Passport.

server.json

```
const jsonServer = require('json-server')
const server = jsonServer.create()
const path = require('path')
const router = jsonServer.router(path.join(__dirname, './json/db.json'))

const middlewares = jsonServer.defaults()

var passport = require('passport');   ①
var Strategy = require('passport-http').BasicStrategy;
var db = require('./db/index');


// Configure the Basic strategy for use by Passport.    ②
passport.use(new Strategy(
  function(username, password, cb) {
    db.users.findByUsername(username, function(err, user) {
      if (err) { return cb(err); }
      if (!user) { return cb(null, false); }
      if (user.password != password) { return cb(null, false); }
      return cb(null, user);
    });
  }));

// http -a jack:secret localhost:3000/email     ③
server.get('/email',
  passport.authenticate('basic', { session: false }),
  function(req, res) {
    res.json({ username: req.user.username, email: req.user.emails[0].value });
  });


// Create a new Express application.
//var app = express();

server.use(middlewares)


// http localhost:3000/wines
server.use(router)
server.listen(3000, () => {
  console.log('JSON Server is running on 3000')
})
```

① Bring in Passport support

② configure our Basic Authentication strategy

③ Add a new Express handler for authenticating a new request

With the integration of Passport with json-server complete, Passport should now be configured to validate a new route request to */email*.

Let's start up the json server and see if we broke anything in our latest refactorings.

start json-server

```
npm run serve
```

Since we now have a custom json-server implementation, we start it using the script (above) that we added earlier in our *package.json*.

When the json-server starts you see the message *JSON Server is running on 3000*.

We can now send some test messages with *httpie*.

Regression testing

```
http localhost:3000/wines
```

If all went well you should see the familiar response to *all wines* from json-server.

Authentication test

```
http -a jack:secret localhost:3000/email
```

If all went well you should see the following response -

Authentication response

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 55
Content-Type: application/json; charset=utf-8
Date: Mon, 30 Apr 2018 18:06:27 GMT
ETag: W/"37-lew270BaMoufxDP6PMJe/Pp2Pys"
X-Powered-By: Express

{
    "email": "jack@example.com",
    "username": "jack"
}
```

Next, well mess with the password and try again.

Authentication failure

```
http -a jack:Xsecret localhost:3000/email
```

As expected we get an authentication failure.

Authentication failure response

```
HTTP/1.1 401 Unauthorized
Connection: keep-alive
Content-Length: 12
Date: Mon, 30 Apr 2018 18:08:45 GMT
WWW-Authenticate: Basic realm="Users"
X-Powered-By: Express

Unauthorized
```

That was useful and a bit of fun, for your next steps I expect you may want to try another *Passport Strategy* like *Google* or *LinkedIn* authentication!

Passport's pluggable authentication greatly simplifies a complex, arduous and oft times error prone process.

I hope you've enjoyed reading this article as much as I have writing it and am looking forward to your feedback.

About the Author:

Mitch Dresdner is a Senior Mule Consultant at TerraThink