Node.js Twitter Lab

Introduction

In this lab we'll make a simple node app that retrieves tweets containing a keyword. We'll start with a version running in the terminal, after that we create a webserver around it.

Terminal version

Step 1: Setup and dependencies

Let's get started. Assuming that you have installed node.js (if not, go do so now and come back here), create a directory for this lab and create the first two files we are going to need. One is the app itself `twitter.js` and the other one is used for dependency management (this is going to be enterprise ready so we'll work like pros here).

```
touch package.json
touch twitter.js
```

We'll use the `ntwitter` module to connect with twitter, therefore we add a dependency in the `package.json` file:

```
{
  "name": "twitter",
  "dependencies": {
    "ntwitter": "0.5.x"
  }
}
```

After that we can install the dependencies by running the node package manager npm:

```
npm install
```

Step 2: Client API keys

Some more hassle before we can get going. We need some client keys from twitter for this app. You can get them at `https://dev.twitter.com`. We set them as environment variables to avoid having to hardcode them.

```
export TWITTER_CONSUMER_KEY=xxxxxxxxxxxx
export TWITTER_CONSUMER_SECRET=xxxxxxxxxxxx
export TWITTER_ACCESS_TOKEN_KEY=xxxxxxxxxxxx
export TWITTER_TOKEN_SECRET=xxxxxxxxxxxx
```

Step 3: using the API

Ok, now we can get going. Open `twitter.js` in your favorite editor and start typing.

We want to use the installed ntwitter module so we start by telling node that we require it. And then we configure it.

```
var twitter = require('ntwitter');

var twit = new twitter({
  consumer_key: process.env.TWITTER_CONSUMER_KEY,
  consumer_secret: process.env.TWITTER_CONSUMER_SECRET,
  access_token_key: process.env.TWITTER_ACCESS_TOKEN_KEY,
  access_token_secret: process.env.TWITTER_TOKEN_SECRET
});
```

Remember to test often so you can already run this program in your terminal. Nothing should happen.
To execute a twitter query, we are going to write a function that asks twitter for statuses filtered by a keyword. If it receives a tweet, it will emmit a 'data' event to which we subscribe. We get a parameter named 'data' each time that event happens which is a structure with a lot of information about the tweet. We use the screen name and the actual content of the tweet.

```
function queryTwitter(q) {
  twit.stream('statuses/filter', {'track':q}, function(stream) {
    stream.on('data', function (data) {
      if (data != undefined && data.user != undefined) {
        console.log('tweet', data.user.screen_name+' tweets: '+data.
            text);
      }
    });
  });
};
```

We should be good to go now, all we need to do is call this new function:

```
queryTwitter("train");
```

Now we can run it ans see what happens:

```
node twitter.js
```

You should see tweets appearing in your terminal (if you have internet connection and someone actually tweets about ohm2013). If you want to see a lot of tweets, try 'now playing' instead of ohm2013.

Enterprise Web App Edition

Step 4: create a webserver

Ok, thats all very nice, but it would be much more fun if we could see all those incredible tweets on a website. So let's create a webserver that updates the page every time a new tweet comes in.

If we want to create a nice website for this, we'll probably end up serving some static content like css files, images and javascript files for the client. We are going to use `node-static` for that. To install that module we'll update the `package.json` file to include this new dependency.

```
{
  "name": "twitter",
  "dependencies": {
    "ntwitter": "0.5.x",
    "node-static":"0.7.x"
  }
}
```

After that we can run npm again to install it:

```
npm install
```

Now we create the webserver in our twitter app by adding these lines:

```
var http = require('http');
var static = require('node-static');

// http server
var app = http.createServer(handler);
app.listen(1337);
var file = new static.Server('./public');
function handler(req, res) {
  file.serve(req, res);
}
```

We've configured the webserver to serve static content from the public folder, so we'll have to create that folder and some files to serve:

```
mkdir public
touch pubic/index.html
touch public/client.js
```

in that folder we can put all our static content like a HTML file:

```
<!DOCTYPE html>
<html>
  <head>
    <script src='/socket.io/socket.io.js'></script>
    <script src='/client.js'></script>
    <link rel="stylesheet" href="style.css">
    <title>Twitter Lab</title>
  </head>
  <body>
  </body>
</html>
```

If we run this now we can see tweets in the terminal and if we visit the website we see a hello there page.

Step 5: Send new tweets to the browser

Now we need to find a way to send new tweets to the webpage when they arrive. We are going to use socket.io for that.

Once again we'll need to add that module to our dependencies the `package.json` file:

```
{
  "name": "twitter",
  "dependencies": {
    "ntwitter": "0.5.x",
    "node-static":"0.7.x",
    "socket.io": "0.9.x"
  }
}
```

After that we can run npm once again to install it:

```
npm install
```

and edit the top of our `twitter.js` file to use socket.io:

```
var sio = require('socket.io');
```

We need to tell socket.io where our webapp is so that it can handle communication with the clients. We do this after we created the webserver:

```
var io = sio.listen(app);
```

Now, if we receive a new tweet, we'll broadcast it to all the connected clients. For this we change the queryTwitter function we already have:

```
function queryTwitter(q) {
  twit.stream('statuses/filter', {'track':q}, function(stream) {
    stream.on('data', function (data) {
      if (data != undefined && data.user != undefined) {
        io.sockets.emit('tweet', data.user.screen_name+' tweets: '+
            data.text);
      }
    });
  });
};
```

We name our broadcast event 'tweet'. This is important to remember to handle the incoming tweets in the browser.

Step 6: Handle new tweets in the browser

One of the nice things of socket.io is the use of almost identical code on the server and in the browser. We'll need to add another link to a javascript file to our HTML like so:

```
<!DOCTYPE html>
<html>
  <head>
    <script src='/socket.io/socket.io.js'></script>
    <script src='/client.js'></script>
    <link rel="stylesheet" href="style.css">
    <title>Twitter Lab</title>
  </head>
  <body>
  </body>
</html>
```

Please note that also the message line is removed in the HTML.

And the client side javascript:

```
var socket = io.connect();

socket.on('tweet', function (tweet) {
});
```

Now we receive tweets but we don't do anything with them. Let's append them to a unordered list on the page:

```
var socket = io.connect();

var tweetList;
window.onload = function() {
  tweetList = document.createElement('ul');
  document.body.appendChild(tweetList);
}

socket.on('tweet', function (tweet) {
  var tweetListItem = document.createElement('li');
  tweetListItem.innerHTML = tweet;
  tweetList.insertBefore(tweetListItem, tweetList.firstChild);
});
```

Step 7: Test it, deploy it, sell it

Now we have our basic realtime web app with Node.js ready! Run it and test it with your browser! You should see tweets arriving in (almost) realtime.

It is not looking very nice yet, but what we have is all ingredients to create a webserver, connect with a external datasource, receive realtime data and send it to all connected clients.

I hope you enjoyed this lab and I hope you learned something about Node.js and that you well be able to use it in one of your (future) projects!