

NodeJS and Express Worksheet: Request and Serve

In this worksheet you will learn about NodeJS and use the Express module to serve up a static HTML website.

Links:

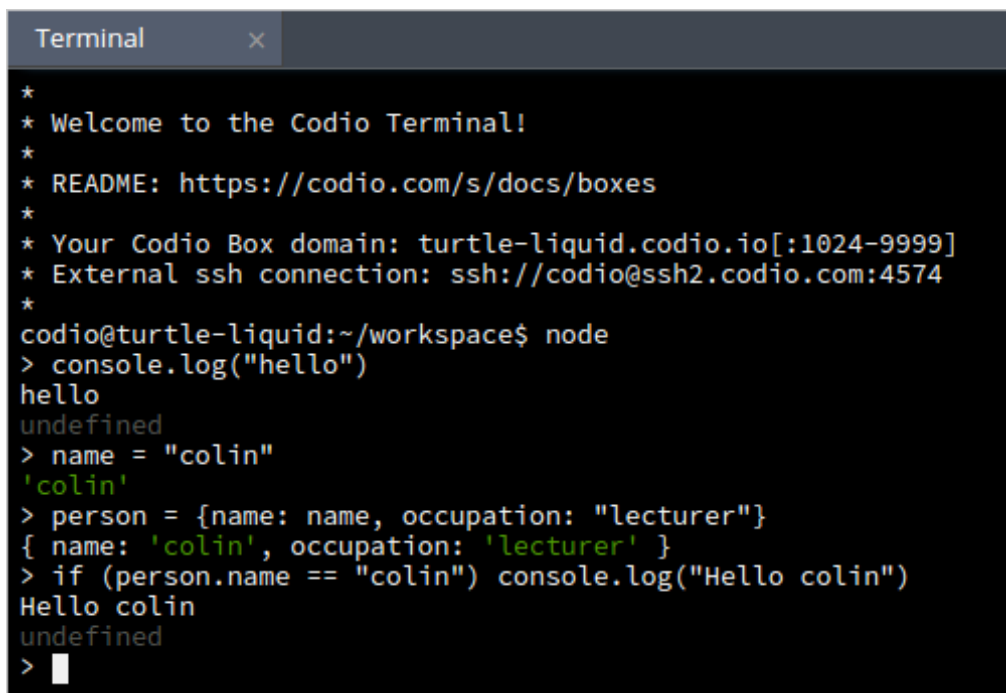
- [NodeJS Tutorials](http://nodeschool.io/#workshoppers) - <http://nodeschool.io/#workshoppers>
 - [ExpressJS](http://expressjs.com/) - <http://expressjs.com/>
-

NodeJS

Node is a server-side runtime engine for JavaScript. Basically, it is a container that lets you run JavaScript without a browser. With it, you can run programs written entirely in JavaScript to run the “server side” of web applications. This can be useful, since often the web client will be written in JavaScript too, so there is less to learn. But Node is not limited to web servers, it can do anything any other language can do (e.g. numerical modelling, statistics, image processing, etc.).

Node is installed on Codio, so you can begin using it right away.

Start a node console by typing `node` in a Codio terminal. Then try out some JavaScript:



```
Terminal x
*
* Welcome to the Codio Terminal!
*
* README: https://codio.com/s/docs/boxes
*
* Your Codio Box domain: turtle-liquid.codio.io[:1024-9999]
* External ssh connection: ssh://codio@ssh2.codio.com:4574
*
codio@turtle-liquid:~/workspace$ node
> console.log("hello")
hello
undefined
> name = "colin"
'colin'
> person = {name: name, occupation: "lecturer"}
{ name: 'colin', occupation: 'lecturer' }
> if (person.name == "colin") console.log("Hello colin")
Hello colin
undefined
> 
```

Figure 1: Node console running on Codio

Modules

Like any good language, JavaScript has lots of useful libraries and frameworks available. Many of these are specifically designed to work with Node on a server:

- Express: an HTTP server written in JavaScript, for serving websites
- Request: an HTTP client written in JavaScript, for accessing web resources such as APIs
- Nano: an Object Relational Mapper (database wrapper) for CouchDB
- FS: a module giving direct access to the host file system, for reading and writing files
- and *many* more

To install, manage, and uninstall any packages that you would like to use, Node ships with NPM: the Node Package Manager.

Installing modules

To install a module, use the command `npm install module_name` with the name of the module you would like to install.

Task

- On the Codio terminal install the modules called `express` and `request`.
- Note that a new directory called `node_modules` is created storing the module code
- Check that the modules are installed correctly by running a quick console test. Run `node` then use the following commands:
 - `var request = require('request');`
 - `var response = request.get('http://www.example.com');`
- Also test that `express` is available with a similar `require` statement.

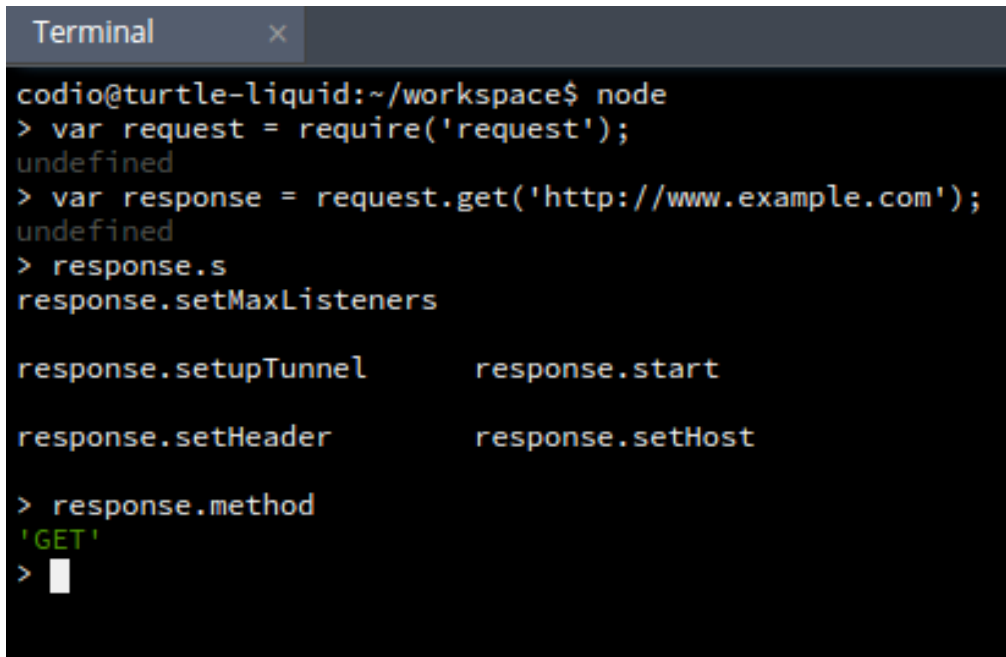
Your First NodeJS Script

We will now use the `request` module to do something very simple: call a weather API and display weather data on the terminal. Save the following code as `request_weather.js`.

```
// import the HTTP client module
var request = require('request');

// set up the base URL
var url = 'http://api.openweathermap.org/data/2.5/weather'

// define a function to send a GET request with the city desired
function checkWeather(city, callback) {
  var query_string = {q: city};
  request.get({url: url, qs: query_string}, callback);
}
```

A terminal window titled "Terminal" with a close button. It shows a Node.js REPL session. The user enters 'node' at the prompt. Then they enter 'var request = require('request');' and get 'undefined'. Then they enter 'var response = request.get('http://www.example.com');' and get 'undefined'. Then they enter 'response.s' and see 'response.setMaxListeners'. Then they enter 'response.setupTunnel' and see 'response.start'. Then they enter 'response.setHeader' and see 'response.setHost'. Then they enter 'response.method' and see 'GET'. Finally, they enter '>' and see a blank line.

```
Terminal
codio@turtle-liquid:~/workspace$ node
> var request = require('request');
undefined
> var response = request.get('http://www.example.com');
undefined
> response.s
response.setMaxListeners

response.setupTunnel      response.start

response.setHeader        response.setHost

> response.method
'GET'
>
```

Figure 2: Test that you can `require` an installed module.

```
// async callback to process what comes back from the request
function logWeather(error, response, body){
  if (error) {
    console.log(error);
  } else {
    console.log(response.statusCode, body);
  }
}
```

```
// call the function to check the weather!
checkWeather("coventry,uk", logWeather);
```

Now from the terminal run `node request_weather.js`, take a look out of the window, and check whether the data displayed on the terminal is correct.

Task

- Modify your weather program to find *historical* weather data for New York.
- Use an appropriate API URL you find from the openweathermap website: <http://openweathermap.org/api>

Your First NodeJS Module

It is good practice to encapsulate your programs as modules for reuse in other contexts. We will now do this by converting `request_weather.js` to a module that you can `require` in another program.

Node looks for a global variable called `module` in your scripts. If it finds one, then it makes the `exports` attribute of that object available in any other script that `requires` the first script. To see this in action do the following.

- Copy the entire code of `request_weather.js` in to a new file called `weather_module.js`
- Replace these lines:

```
// call the function to check the weather!  
checkWeather("coventry,uk", logWeather);
```

with these lines:

```
// make the weather checker available in other scripts  
module.exports.checkWeather = checkWeather;  
module.exports.logWeather = logWeather;
```

- this change makes `weather_module.js` in to a CommonJS module, just like `request`, so you can use it elsewhere.
- Create a new file called `my_weather_app.js` and put the following code in to it.

```
// import the new module  
var weather = require('./weather_module');  
  
// access the exported attribute that we need  
var checkWeather = weather.checkWeather;  
var logWeather = weather.logWeather;  
  
// use its functionality  
checkWeather("coventry,uk", logWeather);  
checkWeather("london,uk", logWeather);
```

- Now from the terminal run `node my_weather_app` and you should see the weather for the two cities.

Task

- Modify your weather module by adding a new function that gets historical weather data, if you didn't already in the previous task.
- Export the new function from the module.
- Access the new attribute in your `my_weather_app` so that it can find historical data too.

ExpressJS

The next part of the worksheet shows you how to run a NodeJS server using the `express` module, which is a framework allowing your JavaScript code to respond to requests from remote HTTP clients (i.e. the other side of what the `request` module does!).

Get started with the following code, in a file called `hello_server.js`:

```

var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

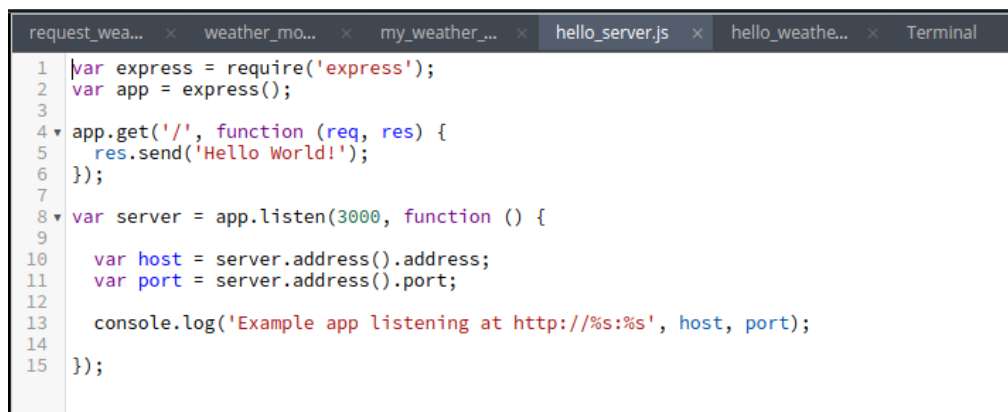
var server = app.listen(3000, function () {

  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);

});

```



```

1 | var express = require('express');
2 | var app = express();
3 |
4 | app.get('/', function (req, res) {
5 |   res.send('Hello World!');
6 | });
7 |
8 | var server = app.listen(3000, function () {
9 |
10 |   var host = server.address().address;
11 |   var port = server.address().port;
12 |
13 |   console.log('Example app listening at http://%s:%s', host, port);
14 |
15 | });

```

Figure 3: First express server

Here the key lines are the `app.get` ones. They do the opposite of what `request.get` does: they **listen** for GET requests from a remote client. The `res` callback then *sends back a response*.

- Run your new server with `node hello_server.js` in the terminal.
- In a new tab browse to `http://subdomain.codio.io:3000` and you should see Hello World.

Task

- First add a new route to your express app with another `app.get` block. However, instead of `/` use `/weather` as the first argument:

```
app.get('/weather', function...
```

- Inside the function you just added, send the response `"<h1>Weather Data</h1>
"`
- Now browse to `http://subdomain.codio.io:3000/weather` and check that your new response is shown.

- If that works, then adapt the `/weather` route as follows: ““ `app.get('/weather', function(req, res) { checkWeather("coventry,uk", getWeatherDescription);`

```
// define a new callback to get the weather description function getWeatherDescription(error, response, body) { if (error) { console.log(error); res.end(error); } else { console.log(response.statusCode, body); res.end("
```

Weather Data

" + body); } } }); “* Test that this returns the weather data to the browser when you go to `http://subdomain.codio.io:3000/weather`‘.

The screenshot shows a code editor with the following JavaScript code:

```
1 var express = require('express');
2 var app = express();
3
4 // import your own module
5 var weather = require('./weather_module');
6 // access the exported attribute that we need
7 var checkWeather = weather.checkWeather;
8
9 app.get('/', function (req, res) {
10   res.send('Hello World!');
11 });
12
13 app.get('/weather', function(req, res) {
14   checkWeather("coventry,uk", getWeatherDescription);
15
16   // define a new callback to get the weather description
17   function getWeatherDescription(error, response, body) {
18     if (error) {
19       console.log(error);
20       res.end(error);
21     } else {
22       console.log(response.statusCode, body);
23       res.end("<h1>Weather Data</h1><br>" + body);
24     }
25   }
26 });
27
28 var server = app.listen(3000, function () {
29
30   var host = server.address().address;
31   var port = server.address().port;
32
33   console.log('Example app listening at http://%s:%s', host, port);
34
35 });
```

Below the code editor, a browser window titled "rondo-ohio.codio.io:3000/weather - Google Chrome" is shown. The address bar displays "rondo-ohio.codio.io:3000/weather". The browser content area displays the text "Weather Data".

Weather Data

```
{ "coord": { "lon": -1.51, "lat": 52.41 }, "sys": { "message": 4.4616, "country": "GB", "sunrise": 1429592073, "sunset": 1429643
{"temp": 288.17, "temp_min": 288.17, "temp_max": 288.17, "pressure": 1036.92, "sea_level": 1049.73, "grnd_level": 1036.9
{"all": 0, "dt": 1429616474, "id": 2652221, "name": "Coventry", "cod": 200 }
```

Challenge

You have now seen how to:

1. query APIs using the `request` Node module
 2. write your own modules using `module.exports` and `require`
 3. run a simple HTTP server using the `express` Node module
- Now write a `hello_movies.js` Node server script that uses all of these techniques to query the movies API at `http://www.omdbapi.com/`.
 - Write your script so that when a user browses to `http://subdomain.codio.io:3000/bestmovie` the data describing your favourite movie is displayed on the browser page.