Worksheet

Introduction to Node

In this worksheet you are going to start writing server-side JavaScript. You will be working on the Codio platform which already has Node installed.

Prerequisites

You will need to have hands-on familiarity with a number of key JavaScript concepts before attempting this worksheet.

callbacks
promises

Install, packages (global v local), readme, package file, running server, https, response object.
frisby testing (simple then CRUD operations), nvm node version manager
Summary
This worksheet covers the basics involved in setting up Node and writing scripts and tests. Specifically it covers:

1. installation and upgrading
2. installing global packages
3. creating a configuration file
4. writing acceptance tests
5. creating a simple web server
6. starting and stopping the server

Setup

There are a number of steps you need to take before writing any code. This section takes you through this process.

Verifying Node is Installed

Lets start by running the Node VM (virtual machine). Open the terminal window and type in node. This will display the Node prompt. Press ctrl-C twice to exit to the shell.

Installing Node

You will be working on the Codio platform which already has Node installed. if you are working on a different platform you may need to install it yourself. Be aware that you will need to install a recent version for your chosen platform. Binaries are available for Mac and Windows on the official Node.js website[1] as well as the source code (which requires Python to install). Check that you are running the latest version using `node -v`. As of writing the current version is **v0.10.3x**. If you are running an old version you can update using the npm (Node Package Manager) command together with the Node Binary Management Module 'n'.

terminal

```
sudo npm cache clean -f
sudo npm install -g n
npm -v
```

[1] http://nodejs.org/download/

Installing Global Packages

Node can be extended using packages, these are available through NPM[2] (Node Package Manager) and installed using the npm command-line tool. Packages can be installed either locally or globally. A general rule of thumb is packages that install tools (such as test runners) should be installed globally whilst those that provide libraries installed locally[3].

In this worksheet we will be developing a simple API using a REST api testing framework called frisby.js which is available through npm. This uses a cli implementation of jasmine which is likewise available through npm.

Installation is straightforward using the npm tool, note the -g flag which indicated the package should be installed globally.

terminal

```
npm install -g jasmine-node
npm install -g frisby
```

This will install the packages together with any dependencies.

Creating a Script File

We will be writing our code in a .js file so lets go ahead and create an empty file called `server.js` in the root project folder. Create a second file called `notes.spec.js` where we will create our tests.

Initialising the Project

Every project needs to have a configuration file. This is used when installing your project as well as by task runners such as Grunt[4] and Continous Integration tools like TravisCI[5]. The `npm init` command runs a wizard that asks for key information and uses this to build the configuration file for you. Most of the questions are self explanatory but under the run and test sections you should enter:

```
start:      node server.js
test:       jasmine-node --noStack notes.spec.js
```

These create aliases to start your project and to run the test suite. By adding these to the project you abstract away details and allow any developer to run your app using `npm start` and run all the tests using `npm test`, very handy. As you can see we use the `node` command to run the app and the `jasmine-node` command to run our test suite, the --verbose long flag tells the jasmine-node to print out details of all the tests.

If you check your project folder you should find a new file called `package.json` which contains your project settings. This can be updated automatically or you can make changes manually. Note it automatically detects your repository settings.

package.json

```
{
  "name": "Notes",
  "version": "1.0.0",
  "description": "Simple API for storing notes",
  "main": "server.js",
  "dependencies": {},
```

[2] https://www.npmjs.com/
[3] http://blog.nodejs.org/2011/03/23/npm-1-0-global-vs-local-installation
[4] http://gruntjs.com/
[5] https://travis-ci.org/

```
  "devDependencies": {},
  "scripts": {
    "test": "jasmine-node --noStack notes.spec.js",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git@gitlab.com:marktyers/notesapi.git"
  },
  "keywords": [
    "restful",
    "notes",
    "api"
  ],
  "author": "Mark J Tyers",
  "license": "ISC"
}
```

Simple Web Server

We are going to build a simple web server to show how to get a Node project up and running. This will return a simple JSON string when we make a request to our server. The response will be:

browser

```
{
    message: "Hello World!"
}
```

Writing Our First Test

In the spirit of TDD we start by writing tests to define our spec. Specifically these are that when we make a GET request to our server root:

1. the response code should be `200`.
2. the response content-type should be `application/json`.
3. the value in the message key should be a string
4. the value in the message key should match the string `"Hello World!"`.

Open the notes.spec.js file and lets define our first assertion.

notes.spec.js

```
var frisby = require('frisby');

frisby.create('testing a simple API')
      .get('http://cricket-proton.codio.io:8080/')
      .expectStatus(200)
.toss();
```

We start by importing the frisby module and use this to create a new `test()`, passing it a description. After creating this we make the API call by chaining the `get()` function and passing the URL (remember to substitute your own subdomain).

Then we chain the `expectStatus()` function and pass it the expected value. Finally we call the `toss()` function that passes the result to the `jasmine-spec` tool to run the tests and display the results.

Writing a Simple Server

The next task is to create a simple web server to display a message in the browser. Enter the following into the server.js file.

server.js

```
var http = require('http');

http.createServer(function(req, res) {
      console.log('incoming request being handled');
      res.end();
}).listen(8080);
console.log('listening on port 8080');
```

Lets go through this code to understand what is happening. The first line imports the http library which is a low level API to handle streams and messages sent over http. Next we use this library to create an http server which takes a single callback which gets run every time the server receives a request over http. The callback takes two objects as parameters, the first contains all the data passed in the request and the second represents the response data that will be returned to the client.

After the server object has been created we chain the listen function which starts it listening out for requests from clients. The optional parameter defines the port that will be used. We need to use a port higher than 1024 otherwise the server would need to run with root privileges which is not desirable from a security standpoint. Later you will be shown how to modify the routing tables on your server to pass any requests coming from port 80 to our higher numbered port.

Once a request has been detected we send a message to the log and tell the server the response has ended.

Starting the Server

Since we have already defined a script to start the server in our configuration file we should be able to start the server using the alias.

terminal

```
npm start

> Notes@1.0.0 start /home/codio/workspace
> node server.js

listening on port 8080
```

You should be able to access this page in a web browser. Nothing gets displayed in the browser but you should see a log message to tell you the request has been received by the server.

Running The Tests

Its time to run the tests. Since we already have a script alias in the configuration file we can run this. Open a new browser tab.

terminal

```
npm test

> Notes@1.0.0 test /home/codio/workspace
> jasmine-node --noStack notes.spec.js
```

```
.

Finished in 0.146 seconds
1 test, 1 assertion, 0 failures, 0 skipped
```

Adding The Other Tests

As we can see, the first assertion is true which means the first test passes. The next step is to add the additional three tests as defined earlier in the lab:

1. the response code should be `200`.
2. the response content-type should be `application/json`.
3. the value in the message key should be a string
4. the value in the message key should match the string `"Hello World!"`.

Lets add three assertions into our test.

notes.spec.js

```
var frisby = require('frisby');

frisby.create('testing a simple API')
    .get('http://cricket-proton.codio.io:8080/')
    .expectStatus(200)
    .expectHeader('Content-Type', 'application/json')
    .expectJSONTypes({
        message: String
    })
    .expectJSON({
        message: 'Hello World!'
    })
.toss();
```

If we run this test we should see it fail.

terminal

```
npm test

> Notes@1.0.0 test /home/codio/workspace
> jasmine-node --noStack notes.spec.js

F

Failures:

  1) Frisby Test: testing a simple API
       [ GET http://cricket-proton.codio.io:8080/ ]
   Message:
     Error: Header 'content-type' not present in HTTP response
```

Passing the Tests

All we need to do now is pass the tests. We will take this one test at a time. The first error indicates there is no content-type header in the response.

To fix this we need to write to the response header. According to the Node documentation[6] there is a class called http.ServerResponse which contains a function called response.setHeader which takes two parameters, a key and a value. The second parameter **res** is an instance of this class and so we call the writeHead() function on it.

We already know from the error message that we need to set the content-type. So lets set it to text/plain. This should generate an error and the test should fail.

server.js

```
var http = require('http');

http.createServer(function(req, res) {
      console.log('incoming request being handled');
      res.setHeader('content-type', 'text/plain');
      res.end();
}).listen(8080);
console.log('listening on port 8080');
```

If we run our test we can see that the error still indicates that the content-type is still not being sent in the header. What is going on?

Stopping the Server

Each time we make changes to our code we need to **restart Node** for the changes to take effect. Return to the first terminal tab (with the Node log messages) and press `ctrl-c` to stop the server. Restart it using `npm start` (this will be in your shell history). Now return to the second terminal tab and re-run the tests.

terminal

```
Failures:

  1) Frisby Test: testing a simple API
        [ GET http://cricket-proton.codio.io:8080/ ]
   Message:
     Expected 'text/plain' to equal 'application/json'.
```

There are two errors returned but lets focus on the first one. The message is very clear. The test was expecting ‘application/json’ but it found ‘text/plain’. Fixing this is trivial so see if you can resolve this error. Run the tests until you get a different error. Remember to restart the server after you make changes to your code.

terminal

```
2) Frisby Test: testing a simple API
        [ GET http://cricket-proton.codio.io:8080/ ]
   Message:
     Error: Error parsing JSON string: Unexpected end of input
        Given:
```

[6] http://nodejs.org/api/http.html

This error is because we are not returning anything in the response body. To fix it lets return a simple message ‘hello world’

server.js

```
res.setHeader('content-type', 'application/json');
res.write('Hello World!');
res.end();
```

terminal

```
1) Frisby Test: testing a simple API
        [ GET http://cricket-proton.codio.io:8080/ ]
   Message:
     Error: Error parsing JSON string: Unexpected token H
        Given: Hello World!
```

Your task now is to complete the server code so that it passes all four tests. Hint: you need to pass a JSON formatted string.

Running Supervisor

As you have seen, each time you make changes to your code you need to first stop then start the node server. In a big project this can become quite a chore. Supervisor is a module that monitors files in your project and if it detects any of them have changed, automatically kills and restarts the server.

The first step is to install the supervisor module globally (see the previous instructions). Next we use it to run the project instead of the node command. Remember we run the node server using an alias in the configuration file so you should update this entry.

package.json

```
"scripts": {
  "test": "jasmine-node --noStack notes.spec.js",
  "start": "supervisor server.js"
},
```

Now when we use this alias to run our server we are running supervisor[7].

terminal

```
npm start

> Notes@1.0.0 start /home/codio/workspace
> supervisor server.js


Running node-supervisor with
  program 'server.js'
  --watch '.'
  --extensions 'node,js'
```

[7] One problem is that if you are using Codio for development the files auto-save which means supervisor will try to run your script after each character typed. There was an option that you can switch between manual save and auto save, but now this option is not available, see here https://codio.com/blog/2015/04/changing-how-we-save/

```
  --exec 'node'

Starting child process with 'node server.js'
Watching directory '/home/codio/workspace' for changes.
listening on port 8080
```

Make a change to your code, save the changes and see what happens in the terminal window.

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Essentially Express.js simplifies a lot of settings in node.js.

Install

Install Express is straightforward:

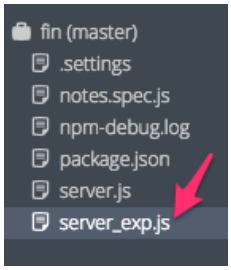
```
npm install -g express
```

Configuration

Next, in your terminal issue the following command to create a new file called server_exp.js

```
touch server_exp.js
```

Click on the file in Filetree within Codio IDE to open this file, and input the following codes:




```
var express = require('express');
var app = express();
app.get('/', function(request, response) {
    response.send("This would be some HTML");
});
app.listen(8080);
```

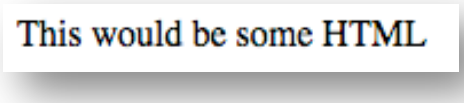
Next, open package.json file and change the following line

```
    "start": "supervisor server.js"
```

to

```
    "start": "supervisor server_exp.js"
```

Now if you run the server and open a browser tab for it, what you'll see is the following:



Congratulations! You have just finished your first Express.js application.

Next Steps

After completing this worksheet you should start to learn more about node. Nodeschool[8] have a range of interactive tutorials packaged up as node modules. Install the learnyounode module and run it to start the tutorial. Aim to complete the tutorial before the next lab session.

```
npm install -g learnyounode
...
learnyounode
```

If you are still struggling with the core JavaScript skills they also offer a module covering these skills called **javascripting** and another one covering git called **git-it**.

Further Reading

There are a number of useful online resources you may find useful when learning the topics in this worksheet.

Installing Node

http://theholmesoffice.com/node-js-fundamentals-how-to-upgrade-the-node-js-version/

Learning Node

http://nodeschool.io/

Node Promises

http://stackoverflow.com/questions/21564993/native-support-for-promises-in-node-js
Need to install a version of node that supports promises

Frisby

http://www.younictechnolabs.com/automated-api-testing-using-frisby-js-node-js-and-jasmine-part-2-execution/
http://cacodaemon.de/index.php?id=58

NPM

https://docs.npmjs.com/

Express

http://expressjs.com/

[8] http://nodeschool.io/