Server-side processing with Node
Week 4, S1 2020

As with all lab exercises, this exercise is not directly assessed. It is intended to complement the material presented in lectures, and may turn up in the exam. You are encouraged to complete it in your own time if you do not finish it during the lab hour.

The aim this week is to become familiar with (i) the use of Javascript and Node.js for server-side processing; and (ii) client-side user input with server-side processing in Javascript. Specifically:

- There are basic examples of Javascript coding patterns to help you learn Javascript quickly. These can be found on Studres in CS2003/Examples/web/node/web2/node-01.
- An extended example for a simple application – a CS2003 'class homepage checker' – is given, which brings together many of the basic examples. This is built up in stages so that you can see how the application is developed from the raw data source. The code can be found on studres in CS2003/Examples/web/node/web2/node-02/.
- A simple example of a 'user registration form', to show the use of HTML5 <form> and <input> tags, along with Javascript processing at the server-side. The code can be found on Studres in CS2003/Examples/web/node/web2/node-03/.

These examples present simple versions of what might be used in industry, for example, but allow you the chance to see, discuss, and experiment with the source code. However, even with more complex code and libraries, many of the underlying concepts are similar.

You should copy all of the files into your own file space for executing and editing, as you see fit. For web access, you will, of course, need to copy the files to your own web file space e.g.:

~/node/cs2003/web2/

Then, for example, script *01-hello_world_html.js* will be accessible via your virtual host as:
https://<userid>.host.cs.st-andrews.ac.uk/cs2003/web2/01-hello_world_html.js

Please note that anything in the web filespace is publicly visible via a web browser if the URL is known.

You need to look through all the examples yourself to make sure you understand how they work. Please make your own notes in your copy of the code by adding comments in the Javascript and/or HTML markup.

1. Background on running Node.js

Node is installed on the Linux systems in the lab.

```
$ node --version
v10.16.3
```

Please make use of the documentation for at https://nodejs.org/en/docs/. The API documentation is at: https://nodejs.org/dist/latest-v10.x/docs/api/. The API documentation has many examples of how to use the API.

You can use node interactively:

```
$ node
> console.log( "you can use node interactively to try things out" )
you can use node interactively to try things out
undefined
> let a = 5
undefined
> console.log( "value of a is " + a )
value of a is 5
undefined
control-d to quit
```

You can also use node to run regular node.js programs like this:

```
$ cd node/cs2003/web/web2/node-01
$ node 01-hello_world.js
Hello world
```

Finally you can use node to run a web server like this:

```
$ node 02-hello_world_html.js
Server running
```

*As always, there is much to be gained in undertaking these exercises together with one or two other people from the class, discussing what you observe, as well as attempting the tasks and exercises below. Please **run**, **examine**, and **discuss** the code in the lab, and work on exercises and questions with others.*

2. Node, Nginx and addresses

By default in the School all your web URL requests are sent to nginx (https://www.nginx.com) which is an open source high performance web server. If you make a request for some content for example:

https://ozgur.host.cs.st-andrews.ac.uk/cs2003/web1/cathedral.jpg

This will be mapped to the host server for ozgur (actually a machine called Lyrane) and a path ~/nginx_default/cs2003/web1/cathedral.jpg where ~ is a (Unix) shorthand for the home directory of al (/cs/home/ozgur/).

Separately and in parallel with this infrastructure you can run a node.js server which can serve web content. A node server program like the ones you have seen in lectures listen on a particular port on some host. For example, the following program will bind to the port **port** on localhost (127.0.0.1).

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(port, '127.0.0.1');
console.log('Server running');
```

Clearly, if everyone used the same port this would end in disaster. We therefore recommend that in development mode you use your unique Unix ID to identify the port on which your programs run. On Unix machines you can use the command id to find your own uid and use this in a program like the one above. For example, if I type id into a shell the following happens:

```
$ id
uid=16831(ozgur) gid=16831(ozgur) groups=16831(ozgur),10001…
```

showing that my uid is 16831.

You can do the same within node by running the following Javascript command:

```
let port = process.getuid()
```

Thus if I run the above program on a lab machine and launch a browser on the same machine I can read the output of the file by going to the address http://localhost:16831/.

There are many more things you can do (**but most of you will have no need or interest to do so**) and these can be found in the systems wiki at the following address:

https://systems.wiki.cs.st-andrews.ac.uk/index.php/New_Linux_Web_Service

in particular section 5.5 describes the node.js configuration. As I said most of you will not need to look at this but may be find it interesting to see how the School's systems are configured.

If you would like to install node.js on your own machines the installer may be found here: https://nodejs.org/en/. **Note that if you do this your assessment is marked on the School systems and anything that does not work on them may be marked as being wrong.**

3. Simple Examples – Node

These examples are provided as separate, stand-alone scripts, each of which can be executed from the command line. They are not to be accessed from a Web browser in the manner described above for e.g.:

```
$ node 01-hello_world.js
```

Things to try / think about / discuss:

1. Script *02-variables.js* does not produce output. Modify it so that all the variables that are defined in the script are displayed.
2. Script *06-functions.js* has output that might be considered 'surprising', for the lines marked "(look at the code!)". What is surprising? Explain what is happening in the code, and why it is a useful feature, potentially, as well as why you should be cautious in its use. How could the code be changed so that the 'correct' answer is given?
3. Script *09-csv_files.js* has a function called *compare_uid()*. Describe how this function works with the *sort()* function. How would you change the function to reverse the sort order? How would you change the code to sort according to the surname?
4. Script *10-server_info.js* can be accessed remotely via a web browser. Look at what is does. Use two different web browsers to access the page delivered by it and compare results. Use the *curl* command, e.g.

   ```
   $ curl -ivs <userid>.host.cs.st-andrews.ac.uk -o blob.txt
   $ less blob.txt
   ```

   and compare the result with what you see in the web browser(s).

4. Extended Example: CS2003 Class List

The examples for this part are in: StudRes/Examples/web/node/web2/node-02.

The files here build up incrementally a simple application enabled by node. The steps in the development are:

- *csclass-a.js*: Read the CSV file and display its contents. The first step is to ensure that you are able to read the data source with your Javascript code.
- *csclass-b.js*: Now that we know we can see the data in our own code, we can add some basic HTML structure, for example a table.
- *csclass-c.js*: At this point, some artistic license is employed to demonstrate the use of CSS3, which we introduced previously – however, colours and formatting are not so important. More importantly, custom (semantic) tags (i.e. not defined in HTML5) are used to structure the data in the web-page, and CSS3 is used to control how the page looks.
- *csclass-d.js*: Now, we introduce a mechanism to check the homepage URL of each student, to see if a web-page exists. An additional column is added to the tabular output (some of this code is complex and we will revisit this later in the course, if you do not follow it all do not worry too much)

Please examine and execute the code and its output, both on the command line, and via a web-browser.

Things to try / think about / discuss:

1. What could be the advantage for the use of the semantic tags in *csclass-c.js*?
2. Examine the file *csclass.css*. Several of the tag names also have *selectors* that modify their behaviour. Explain in a single sentence the operation of each CSS3 definition with a selector.
3. Make a new version of the Javascript file, called *csclass-e.js* (by copying *csclass-d.js*), as well as a new version of the CSS file, *csclass-e.css* (by copying *csclass.css*). Now, modify *csclass-e.js* and *csclass-e.js* so that:

• the homepage column no longer appears in the HTML.
• if a homepage URL is valid (i.e. *checkURI ()* returns *true* for that URL), the script links the homepage URL to the value in the uid column.
• the uid appears with a *lightgreen* background if a URL is linked to it.

5. Client-side input from the user with HTML5 forms

The examples for this part are in: StudRes/Examples/web/node/web2/node-03.

As shown in the lecture, we will be using HTML forms with the HTTP POST command in order to allow input from the user to be sent to the server for processing. In the folder node-03 you will find two versions of a form generating a form:

- *registration-a.js*: Which serves a form (read from a file – look at how that works!) but does not process any submission.
- *Registration-check.js*: Which serves the same form but also processes the POST and returns an confirmation page to the client.

Things to try / think about / discuss:

1. Why is validation of user input required at both client-side and server-side?
2. Modify *registration-check.js* so the user is asked if the details are correct (client-side) before being confirmed. If the user says the details are incorrect, then the user is directed back to the original page, only if they are the details are sent to the server.

Hint: Check the HTML5 documentation at https: //developer.mozilla.org/en-US/docs/Web/HTML for the attributes *value* and *type="hidden"* for the tag <input>.