Session Dependancies

Task 0: Getting Started

1. As always, start and connect to your Azure Labs' virtual machine (VM) by visiting this link: https://labs.azure.com/virtualmachines.
2. Connect to the VM:
    1. Toggle the button to start – it might take a while.
    2. Once it changes to Running, click on the monitor icon.
    3. A file will be downloaded – click on it to run it, and you will be prompted to enter the password you created last time. IMPORTANT: the username must be `labuser` (remove the ~/).
    4. Remember to click on the following icon to make the window resize appropriately

Task 0.1: Setting an Environment Variable

Since, for this part of the module, we are going to start to develop more substantial programs, we need to level up our development environment.

Task 1

- Let's make our first Node web application.
- Create a folder in your `code` directory.

- Within your lab_8 folder create the file `sample_1.js`.
- Type in the code below (see, `A simple NodeJS Application`)
- Run you application by pointing your IDEs terminal to your lab_8 folder and input the command `node sample_1.js`.
- Now visit your browser, and navigate to `http://localhost:8000` see what happens! You should see "hello world". Next, take a look at your terminal, this is where your `console.log()` statements will be be output.

```
1  const http = require("http");
2  const server = http.createServer();
3
4  server.on("request", (req, res) => {
5    const { method, url, headers } = req;
6
7    console.log(method);
8    console.log(url);
9    console.log(headers);
10
11   res.end("hello world");
12 });
13
14 server.listen(8080);
```

> A simple NodeJS Application

Do not worry if the code above seems somewhat alien to you; much of it may be new. In short, we are setting up web server that is listening on local host port 8080. When a client connects, we simply return hello world.

To achieve this, we first must pull in Node's http module (l.1). Next, we create a new server instance (line 2).

Then we set up an HTTP request handler (l.4). When an HTTP request is received the function, which is the second argument passed into server.on(), is fired. Two objects are passed into this handler function (res, req). As you can see we can use these objects to inspect the type of request (l.7 - 8) and write a response to the client (l.11).

Finally, we tell the server to listen for incoming requests on port 8080 (l.14).

We can now stop our application from running. To do this, navigate to the terminal window used to run your application and press `control + C`. If you forget to do this, you'll get a `Error: listen EADDRINUSE: address already in use :::8080` error when you try and run the application again!

Take a deep breath, and pat yourself on the back - you've just made your first NodeJS application!

Task Two

Let's do something a little more interesting, and see if we can use Node.js to create construct a simple web application!

Above,we used Node's inbuilt HTTP module to create a basic web server and respond to HTTP requests.

In this task, we will look to simplify managing HTTP requests using, the node package, express.

According to the the express documentation, “express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications”. In other words, it provides a lightweight framework to assist in the development of web applications.

An example using express

```
1  const express = require("express");
2  const app = express();
3  const port = 3000;
4
5  app.get("/", (req, res) => {
6    res.send("Hello World!");
7  });
8
9  app.listen(port, () => {
10   console.log(`Example app listening at http://localhost:${port}`);
11 });
```

The above example uses the express package to create a new web-server that listens on port 3000 (l.9). We can then set up listeners to respond to any given HTTP request. Above, we listen for a get request to the base URL of our server (l.5).

Steps

- Create a new folder called `week_2`
- Open VSCode pointing to the location of your `week_2` folder
- Within `week_2`, create a file called `index.js`
- Run the command `npm init -y`
- We can now install express, `npm install express --save`
- To make our lives a little easier, we are also going to install nodemon. Run `npm install nodemon --save-dev`. Nodedemon is a useful tool that recompiles our applications in realtime. This means we don’t need to keep running `node 'our application name'` to reflect our latest changes.
- To use nodedemon add the following the scripts section of your **`package.json`** file:

```
1  "scripts": {
2     "dev": "nodemon index.js"
3   }
```