Creating and importing modules

Welcome to this Lab activity

In this lab activity you will be exploring the concept of Node modules.
The code will be very similar to your first web server exercise but, to keep things organised, you will work on another subfolder inside *topic2* for this exercise.

What is a module

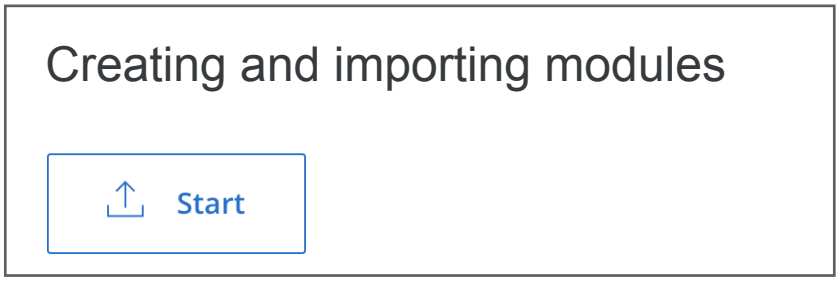
- A module is essentially a set of functions that you want to include in your application. Think of it as a library; a set of functions or code written by you or someone else which you can import and export across your project.
- There are certain modules which are built in inside Node. For example, do you recall the line *require('http')* in the previous lab? Well, **http** is a module which lives inside Node and available for you to use.
- Other modules, on the other hand, are not part of Node and they need to be included inside your project if you want to use them. We will look into including external modules in the next lab when we will introduce **Express**.

For now, let's see how we can create our own modules, export them and successively use them in our web server application.

Start the Lab environment application

It is simple to launch a lab exercise. You only need to click on the button "Start" below the activity title to enter a lab environment.

Let's explore this lab activity. Go ahead and click on the "Start" button!



Task 1: Serve a web page and Access it via http

The folder structure has already been partially constructed for you and organised into different topics. For the purpose of this lab, you will be making changes inside the *topic2* folder structure. Let's get started!

Please do not delete or move any existing folders/files inside the lab environment.

Use the Visual Studio Code Terminal and run the following commands:

- **cd topic2/mySecondNode**: type this command and press *Enter*. This command will change your working directory to be the *mySecondNode* folder.
- **npm init**: type this command and press *Enter*. This command will set up a new or existing npm package. You can skip all the npm initialisation questions by pressing *Enter*. Once the npm package file has been created, you can view it inside the *mySecondNode* directory (*package.json*). The *package.json* file contains a set of information regarding your current node project.

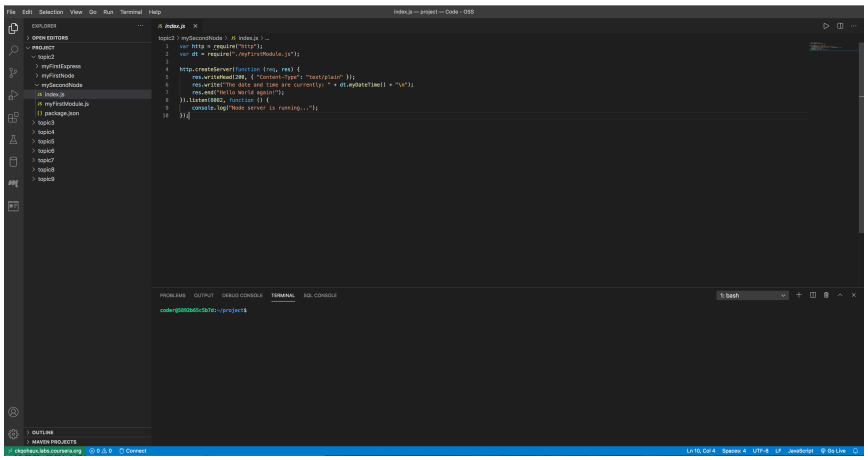
The very next step is to add some code to the *index.js* file located inside the *topic2/mySecondNode* folder in order to create your web server.

Add the following code to the *index.js* file and remember to save it:

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

http.createServer(function(req, res) {
res.writeHead(200, { "Content-Type": "text/plain" });
res.end("Hello World again!");
}).listen(8082, function() {
console.log("Node server is running...");
});
```

If you have correctly followed all the above steps, your environment should be similar to the one below:



Running the index.js file

Run the *index.js* file with the following Terminal command:

- **node index.js**: type this command and press Enter. The node command, followed by the file name, tells Node.js to execute the content of the file.

The above command will start a web server running on port 8082.

Task 2: Access your server via HTTP

Now that your code is running, serving your application on port 8082, you can access your web page from a browser!

Use the “Browser Preview” plugin to visualise your web application.

If you do not remember how to use the “Browser Preview” plugin, please refer to the “Creating my first Node.js web server” lab instructions.

Type *localhost:8082* on the “Browser Preview” tab and press *Enter* on your keyboard to visualise your web application:

If you have correctly followed all the steps, your web application should show the “Hello World again!” message.

Task 3: Add a new module to your code

It is now the time to create your own module and import it inside the *topic2/mySecondNode/ index.js* file.

You can create your own modules, and easily include them in your application.
As an example let's create a module that returns a date and time object so that you can import it and use it in the *index.js* file. The goal is to show a message with the current date and time just before the "Hello Word again!" message in the web page served by your server.

Create a new file called *myFirstModule.js* inside *topic2/mySecondNode/*; you can either use the Terminal or the Visual Studio GUI to perform this operation.

Add the following code to the *myFirstModule.js* file and remember to save it:

```
exports.myDateTime = function () {
 return Date();
};
```

The "exports" keyword makes properties and methods available outside of the module (in this case outside of the myFirstModule.js script). In particular, we export and make the Date() object ready to be included in other scripts.

At this point you have exported your module and you only need to require it inside the *topic2/ mySecondNode/ index.js* file to be able to use it.

Add this line of code just below the "http require" line in the *index.js* file:

```
var dt = require("./myFirstModule.js");
```

In addition, add this line of code to your *index.js* file:

```
res.write("The date and time are currently: "+ dt.myDateTime() + "\n");
```

In order to visualise your latest changes, you will have to both restart your web server (index.js) and also refresh the "Browser Preview" tab.

Can you guess where you should add this piece of code? Remember we would like to show this in the browser before the "Hello World again!" message:

End of session

Well done for completing this session. If you have correctly followed all the steps you should be able to see your web page with both the data-time and the welcome message like in the above picture.