Creating my first Node.js web server

Welcome to this lab activity

Let's build a Web Server with Node.js! In this lab, you will create a virtual server that serves a simple web page.

Task 1: Create a Node.js/NPM project

Let’s create a new project in Visual Studio Code. Like the previous lab, this lab will be a JavaScript project, but this time you will set up the project to use NPM, the Node package manager. This will allow you to bring in lots of other useful functionality that will support your development of web applications.

1. Create a new folder called `02_simple_server` on your computer.
2. Open the folder in Visual Studio Code.
3. Open a Terminal pane in Visual Studio Code.
4. Run the following command in the Terminal pane:

```
npm init
```

This command will initialise your project as an NPM project. It will ask you lots of questions when you run it. Just hit enter to accept the default answers as shown below:

```
C:\code\02_simple_server>npm init
This utility will walk you through creating a package.json
file.
It only covers the most common items, and tries to guess
sensible defaults.

See `npm help init` for definitive documentation on these
fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (02_simple_server)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
```

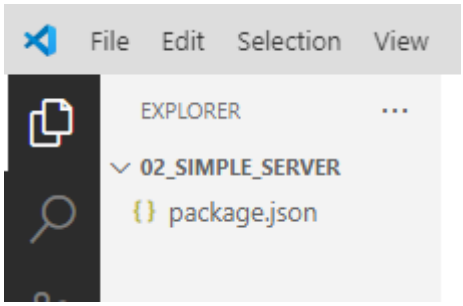
```
keywords:
author:
license: (ISC)
About to write to C:\code\02_simple_server\package.json:

{
  "name": "02_simple_server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)

C:\code\02_simple_server>
```

When this is done you will see a new file, `package.json`, appear in the explorer:



The `package.json` file contains information about your current node project. If you click on it, you will see its contents:

```
{
  "name": "02_simple_server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Task 2: Create the code to serve a web page

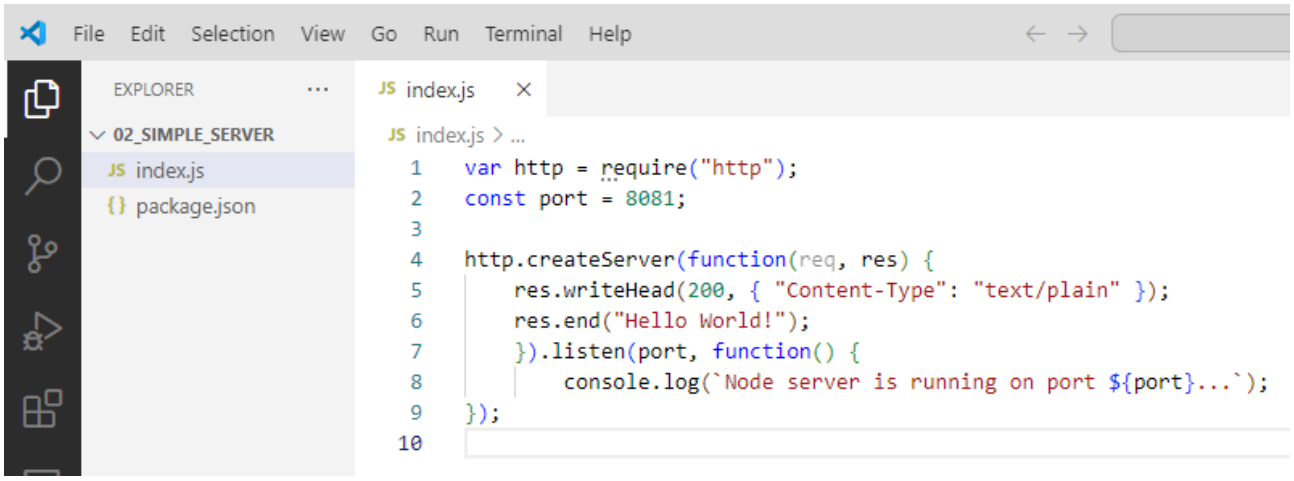
Now you will write the code to serve the web page.

5. Create a file called `index.js` in the project.
6. Add the following code to the `index.js` file and remember to save it:

```
var http = require("http");
const port = 8081;

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

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



Task 3: Run the code

7. Run the `index.js` file with the following Terminal command:

```
node index.js
```

The above command will start a web server running on port 8081. The output in the Terminal pane should confirm this:

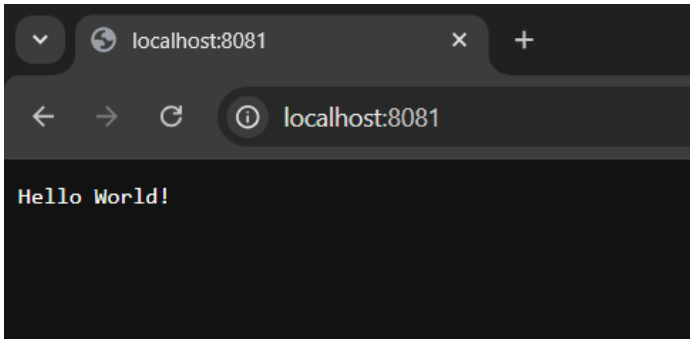
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\code\code\02_simple_server> node .\index.js
Node server is running on port 8081...
```

Task 4: Access your server via HTTP

Now that your code is running and serving your application on port 8081, you can access your web page from a browser!

8. Start up a browser and navigate to http://localhost:8081. If you have correctly followed all the steps, your web browser should show the 'Hello World!' message.



Task 5: Understand the code

The code may be small, but there is a lot going on! Let's go through it.

First of all you **require** the `http` module:

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

You will explore modules in the next session, but they are essentially snippets of code that you can export and import across files. In the above line, you import the Node.js `http` module.

Next you set the port number on which your server will listen for HTTP requests:

```
const port = 8081;
```

Next, you create a server using the built-in method of the `http` module called

`createServer`:

```
http.createServer(function(req, res) {
```

```
   res.writeHead(200, { "Content-Type": "text/plain" });
   res.end("Hello World!");
}).
```

This method creates a web server and provides a function to run when the web server receives an HTTP request from a client. Every time they make such a request to the web server, it will respond with a 200 status code (to say the request has succeeded), setting the response content type to be plain text. Finally, it will send the 'Hello World' text as the response back to the client.

Note the parameters to the function: `req` and `res`. `req` is the request object. It holds information sent in the request from the client. `res` is the response object, which it uses to build the response back to the client.

Now that you've defined the function to run when HTTP requests are received, you can start the web server so that it actually listens for those HTTP requests. You do this using the method `listen`:

```
.listen(port, function() {
```

```
      console.log(`Node server is running on port ${port}...`);
```

Here you are requesting that the web server starts up and listens for HTTP requests on port number 8081. It will only accept requests made to that port number. The function passed as a parameter to the listen method runs when the web server successfully starts up.

Task 6: Shut down your web server

Remember that your web server is still running on port 8081.

9. To shut down your server, press the following combination on your keyboard keys while inside the Terminal:

```
Control + c
```

This will stop the `index.js` script from running, shutting down the web server.

Task 7: Exercise

When tackling these lab activities, it's always good to stretch yourself by doing a bit of research and attempting some changes on your own.

- Do some research on the meaning of ‘require’ in your code.
- What about req and res? What are they? Try adding the following line before the res.writeHead line:

```
console.log(req);
```

Examine the output in the terminal. What useful information can you find there?

- Can you change ‘Hello World!’ to your name?
- Add comments to your code explaining the different pieces of code.

End of section

Congratulations on completing this lab activity!