

COMP 1842

Node.js server

Matt Prichard

Introduction

NoSQL recap

Npm recap from week 6

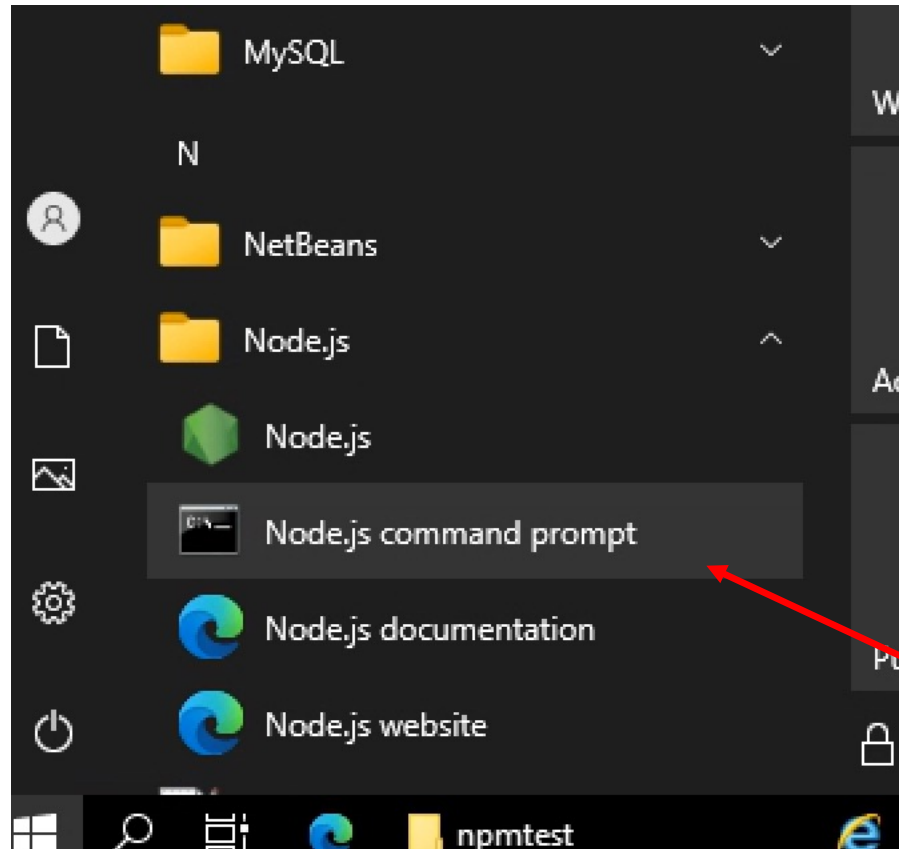
Node js server

Recap -NoSQL

<https://www.sitepoint.com/sql-vs-nosql-differences/>

<https://www.sitepoint.com/sql-vs-nosql-choose/>

Recap -Running Node locally



- For these examples I am running the files from my G Drive
- You can install Node and npm on your own machines, but I can't directly support that.
- In the labs locate and open the Node.js command prompt

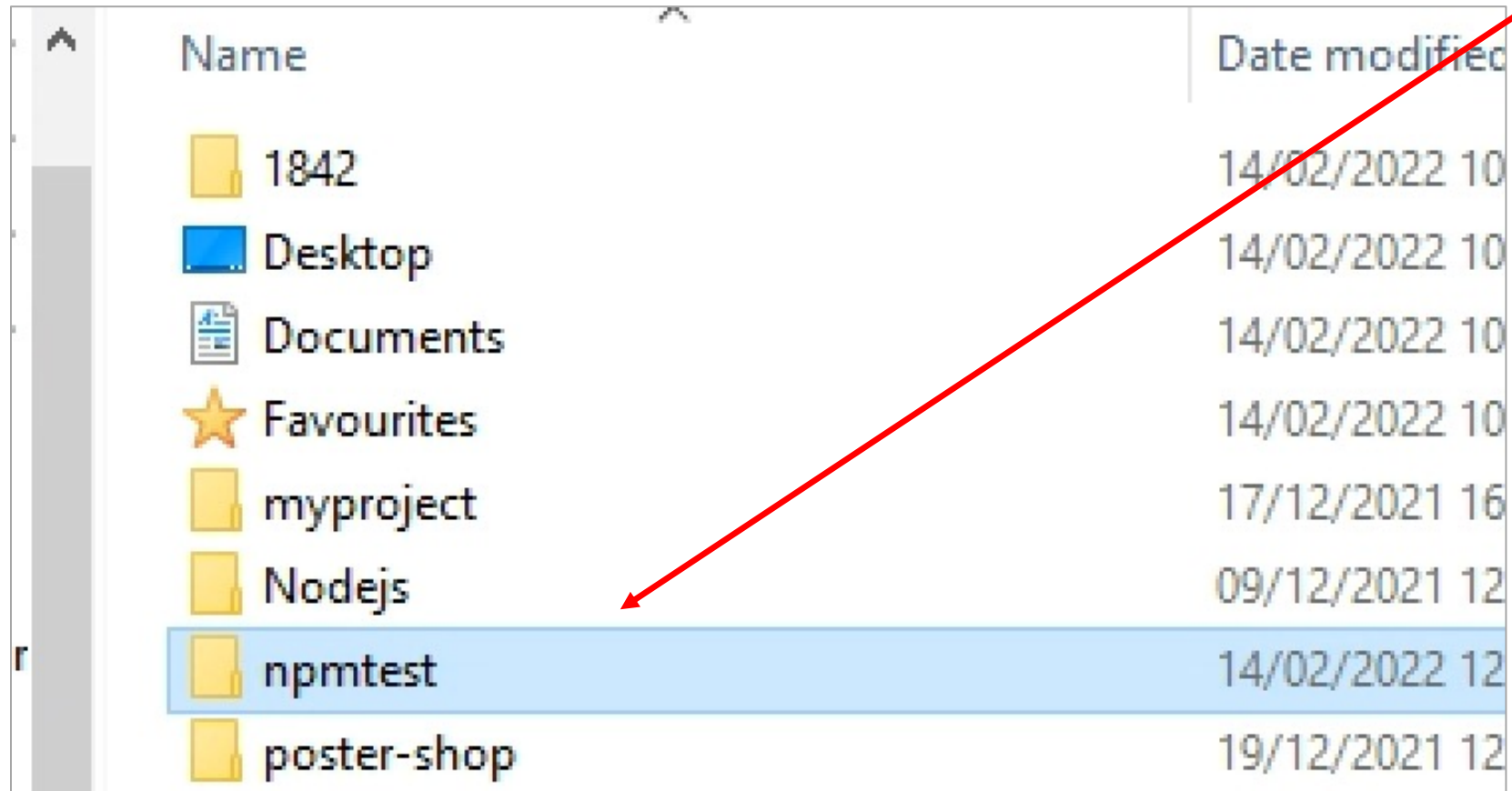
Install check

```
Node.js command prompt - "C:\Program
Your environment has been set u
G:\>node -v
v16.6.2
G:\>npm -v
7.20.3
G:\>_
```

Type `node -v` and press enter, this will give the current version of node that is installed, do the same with

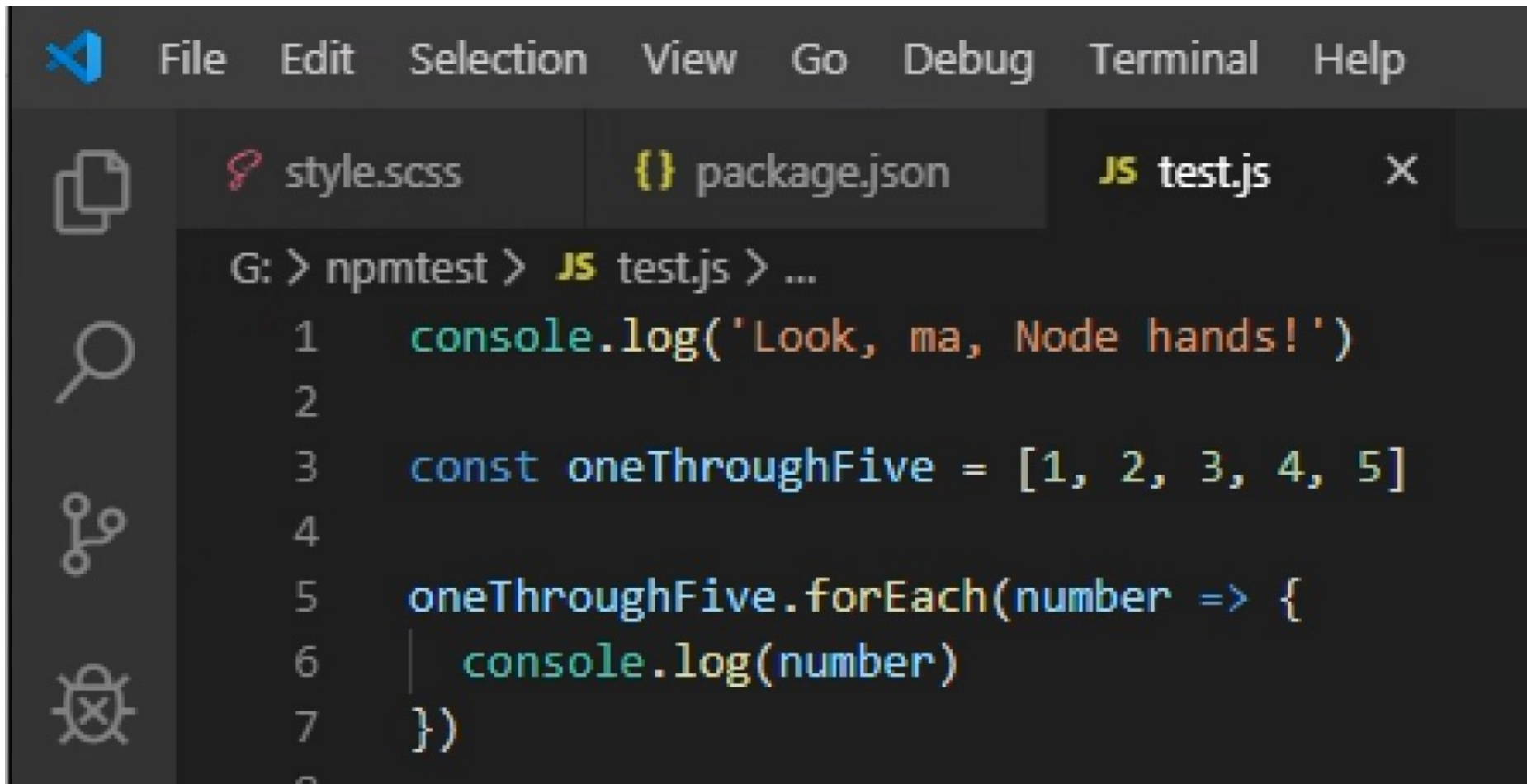
`npm -v`

G Drive – new folder npmtest



Name	Date modified
1842	14/02/2022 10
Desktop	14/02/2022 10
Documents	14/02/2022 10
Favourites	14/02/2022 10
myproject	17/12/2021 16
Nodejs	09/12/2021 12
npmtest	14/02/2022 12
poster-shop	19/12/2021 12

Create test.js and save to the new folder



The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The Explorer sidebar on the left shows a file named 'test.js' with a JavaScript icon. The main editor area displays the following code:

```
G: > npmtest > JS test.js > ...  
1 console.log('Look, ma, Node hands!')  
2  
3 const oneThroughFive = [1, 2, 3, 4, 5]  
4  
5 oneThroughFive.forEach(number => {  
6   | console.log(number)  
7   | })  
8
```

Run node

Open the Node command prompt, navigate to where the file is (using `cd npmtest` or “change directory to npmtest”), and run `node test.js` to get the following output.

```
[matt@Matts-MacBook-Pro ~ % cd npmtest
[matt@Matts-MacBook-Pro npmtest % node test.js
Look, ma, Node hands!
1
2
3
4
5
matt@Matts-MacBook-Pro npmtest %
```


Navigating folders

cd folder_name/ change directory

cd ../ back up a directory

Can chain together

cd folder1/folder2/folder3/

cd ../../../../

cd/ root directory

Node

Node.js can be used to build different types of applications such as command line applications, web applications, real-time chat application, REST API server etc.

However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.

Cont...

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Cont...

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

<https://medium.com/@monuchaudhary/single-threaded-non-blocking-asynchronous-and-concurrent-nature-of-javascript-a0d5483bcf4c>

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can collect form data
- Node.js can add, delete, modify data in your database
- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

An aside => Arrow functions

Arrow functions are one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions. For example:

```
// function expression
let x = function(x, y) {
    return x * y;
}
```

```
// using arrow functions
let x = (x, y) => x * y;
```

Arrow function syntax

```
let myFunction = (arg1, arg2, ...argN) => {  
  statement(s)  
}
```

- `myFunction` is the name of the function
- `arg1, arg2, ...argN` are the function arguments
- `statement(s)` is the function body

If the body has single statement or expression, you can write arrow function as:

```
let myFunction = (arg1, arg2, ...argN) => expression
```

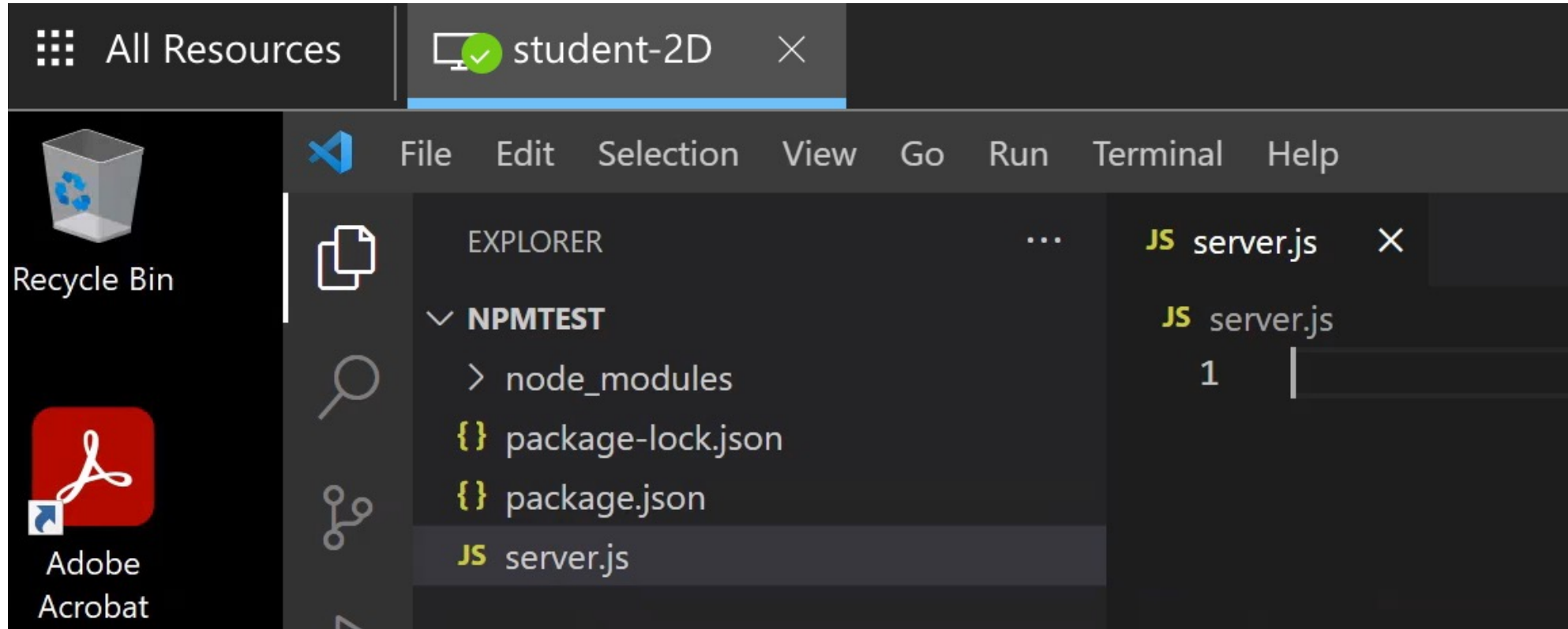


Further
reading

https://www.w3schools.com/Js/js_arrow_function.asp

<https://www.freecodecamp.org/news/arrow-function-javascript-tutorial-how-to-declare-a-js-function-with-the-new-es6-syntax/>

Create new file



In the 'npmtest' folder create a new file > server.js

Node.js web server

- To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application e.g
- IIS is a web server for ASP.NET web applications.
- Apache is a web server for PHP or Java web applications.
- Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web applications but it is recommended to use Node.js web server.

Create Node.js web server

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper-Text Transfer Protocol (HTTP).

To include the HTTP module, use the `require()` method:

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

- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- Use the `createServer()` method to create an HTTP server:

Create Node.js web server 2

The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 8000.

```
JS server.js  X
JS server.js > ...
1  const http = require('http'); //Import Node.js core module
2  const server = http.createServer(function(req, res){ // creating server
3    //handle incoming requestd here
4  })
5  server.listen(8000); //listen for any incoming requests
6  console.log('Node.js web serverr at port 8000 is running..');
7
```

Run the server

> node server.js

Control C to stop the server

```
C:\Users\pm76\fakeg\npmtest>node server.js
Node.js web server at port 8000 is running..
_
```

Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type: Line 3 set response header, Line 4 set response content.

JS server.js X

JS server.js > ...

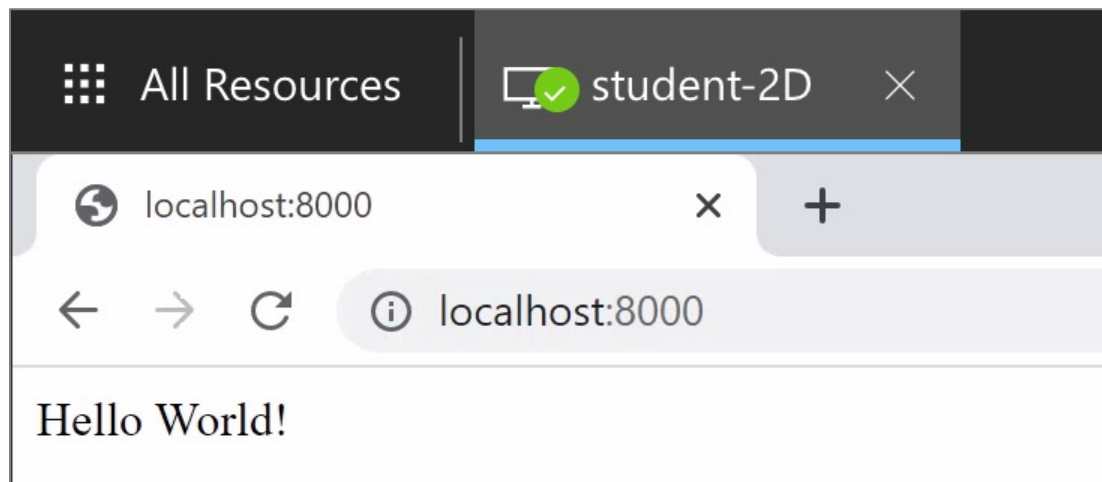
```
1  const http = require('http'); //Import Node.js core module
2  const server = http.createServer(function(req, res){ // creatin
3    res.writeHead(200,{ 'Content-Type': 'text/html' });
4    res.write('Hello World!');
5    res.end();
6  })
7  server.listen(8000); //listen for any incoming requests
8  console.log('Node.js web server at port 8000 is running..');
```

Restart the server

Restart the server > node server.js

```
C:\Users\pm76\fakeg\npmtest>node server.js
Node.js web server at port 8000 is running..
^C
C:\Users\pm76\fakeg\npmtest>node server.js
Node.js web server at port 8000 is running..
```

In your browser go to localhost:8000



```

JS server.js > ...
1  const http = require('http'); // Import Node.js core module
2  const server = http.createServer(function (req, res) { //create web server
3      if (req.url == '/') { //check the URL of the current request
4          // set response header
5          res.writeHead(200, { 'Content-Type': 'text/html' });
6          // set response content
7          res.write('<html><body><p>This is home Page.</p></body></html>');
8          res.end();
9      }
10     else if (req.url == "/student") {
11         res.writeHead(200, { 'Content-Type': 'text/html' });
12         res.write('<html><body><p>This is student Page.</p></body></html>');
13         res.end();
14     }
15     else if (req.url == "/admin") {
16         res.writeHead(200, { 'Content-Type': 'text/html' });
17         res.write('<html><body><p>This is admin Page.</p></body></html>');
18         res.end();
19     }
20     else if (req.url == "/data") {
21         res.writeHead(200, { 'Content-Type': 'application/json' });
22         res.write(JSON.stringify({ message: "Hello World JSON" }));
23         res.end();
24     }
25     else
26         res.end('Invalid Request!');
27 });
28 server.listen(8000); //6 - listen for any incoming requests
29 console.log('Node.js web server at port 8000 is running..')
30

```

Fuller example

Handle HTTP Requests

The `http.createServer()` method includes **request** and **response** parameters which are supplied by Node.js.

The request object can be used to get information about the current HTTP request e.g., url, request header, and data.

The response object can be used to send a response for a current HTTP request.

Detail view

Using if and else if. We can use the 'url' property of the request object to check the url of the current request lines 3 and 10

```
JS server.js > ...
1  const http = require('http'); // Import Node.js core module
2  const server = http.createServer(function (req, res) { //create web server
3      if (req.url == '/') { //check the URL of the current request
4          // set response header
5          res.writeHead(200, { 'Content-Type': 'text/html' });
6          // set response content
7          res.write('<html><body><p>This is home Page.</p></body></html>');
8          res.end();
9      }
10     else if (req.url == "/student") {
11         res.writeHead(200, { 'Content-Type': 'text/html' });
12         res.write('<html><body><p>This is student Page.</p></body></html>');
13         res.end();
14     }
15 }
```

Detail view 2

Again, `req.url` is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using `writeHead()` method and then writes a string as a response body using `write()` method. Finally, Node.js web server sends the response using `end()` method.

```
13     res.end();  
14 }  
15 else if (req.url == "/admin") {  
16     res.writeHead(200, { 'Content-Type': 'text/html' });  
17     res.write('<html><body><p>This is admin Page.</p></body></html>');  
18     res.end();  
19 }
```

Sending JSON response

Inserting lines 20-24 shows how to serve a JSON response from the Node.js web server. this way we can create a simple web server that serves different responses.

```
20     else if (req.url == "/data") {
21         res.writeHead(200, { 'Content-Type': 'application/json' });
22         res.write(JSON.stringify({ message: "Hello World JSON" }));
23         res.end();
24     }
25     else
26         res.end('Invalid Request!');
27 });
28 server.listen(8000); //6 - listen for any incoming requests
29 console.log('Node.js web server at port 8000 is running..')
30
```

Lab task for this week (#5 of 6)

Implement the code from slide 17 onwards from this lecture on your G Drive

Submit a screen shot of the server running in the command line on port 8000

Show the different responses in the browser via localhost:8000

Home, student, admin and data (the JSON response)

Write 200-300 words explaining your understanding of the code.



sources

<https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server>

https://www.tutorialspoint.com/nodejs/nodejs_response_object.htm

<https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/>

https://www.w3schools.com/nodejs/nodejs_http.asp