NodeJS

CCAPDEV

For this lecture:

1. NodeJS Features
2. Creating a Server
3. The Request object
4. The Response object
5. Status Codes
6. URL routing

Let’s do a quick review…

on the **last few things** we have discussed

What is NodeJS?

- NodeJS is an open-source JavaScript runtime environment
  - JavaScript was originally meant to run only on browsers
  - NodeJS allowed JavaScript to run outside of browsers
- No need to learn another programming language since it uses JavaScript everywhere

Back to our current lecture...

Let’s talk about NodeJS’ **features**

Features of NodeJS

- So what can NodeJS even do?
- NodeJS can…
  - generate dynamic page content
  - create, open, read, write, delete, and close server files
  - collect form data
  - add, delete, modify data in the database

Features of NodeJS

- So what can NodeJS even do?
- NodeJS can…
  - generate dynamic page content
  - create, open, read, write, delete, and close server files
  - collect form data
  - add, delete, modify data in the database

For today's session, we'll be **trying out** the **first two** highlighted points

NodeJS Modules

- Node has built-in modules to aid development
- The full list of modules can be found in the documentation
  - https://nodejs.org/docs/latest/api/
- Commonly used modules include:
  - http – interact with the web as a server
  - fs – access the file system

Let’s create our first **server**

Let’s see NodeJS in **action**

Installing NodeJS

- Download and install NodeJS:
  - Download the latest version of NodeJS here
    - https://nodejs.org/en/
  - If you have already downloaded a different version, it should be fine.
  - Install it with the default settings.
- Confirm your NodeJS installation by running the following command in Command Prompt:
  - `C:\Users\Me>node -v`
  - It should show your NodeJS version

Creating a **Server**

- To create a server, you need to include the http module
- A common way to include modules is as follows:

```
const varName = require('modulename');
```

➥ `varName`

constant variable to hold your module

➥ `modulename`

the name of the module to include

Creating a **Server**

- To create a server, you need to include the http module
- A common way to include modules is as follows:

```
const http = require(‘http');
```

➥ `varName`

constant variable to hold your module

➥ `modulename`

the name of the module to include

Creating a **Server**

- After including the http module, let’s use the http variable to create a server

```
http.createServer([requestListener]);
```

➥ `.createServer()`

method that return the http.Server object

➥ `[requestListener]`

listens to a port and executes a function each time a request is made to the server

1

2

3

4

5

6

The **Request Listener**

- The requestListener handles requests and responses from the client and back to the client respectively

```
function (req, res) { }
```

➥ `req`

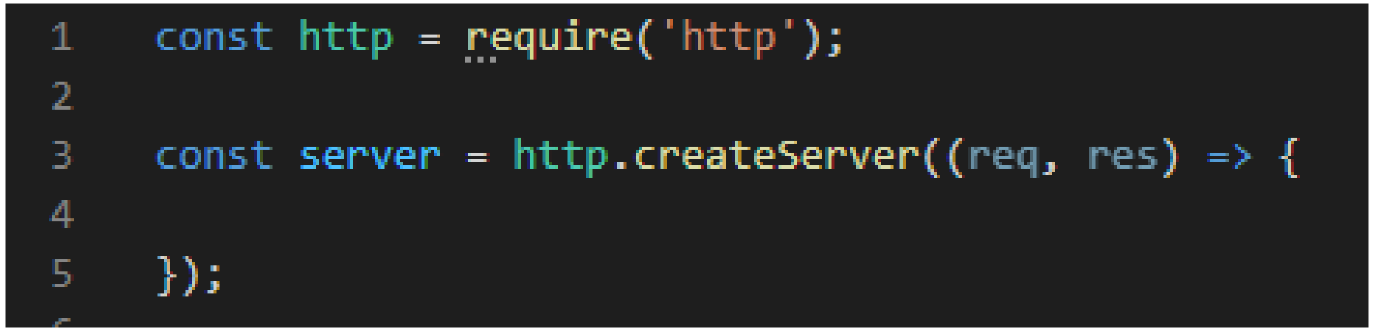
represents an **`IncomingMessage`** object containing details about the client request

➥ `res`

represents a **`ServerResponse`** object, responsible for the response stream back to the client

Creating a **Server**

- Our createServer now looks like this:



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

const server = http.createServer((req, res) => {

});
```

- Note that at this point, the server is not ‘activated’ yet.

Creating a **Server**

- To activate the server, we have to make it **listen** to a port.
- This can be done using `server.listen`

```
server.listen([port], [host], [callback]);
```

➥ `port`

the port to bind the server into

➥ `host`

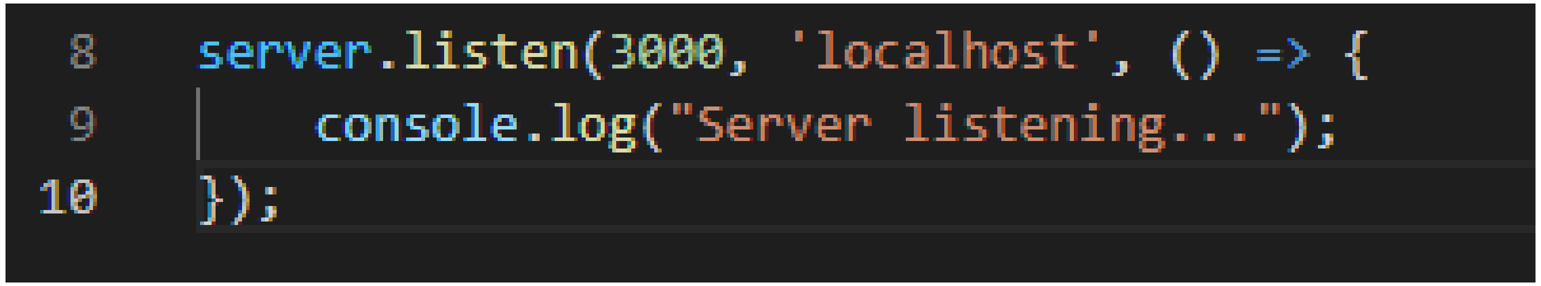
the IP address to bind the server

➥ `callback`

the function executed after the server has been bounded

Creating a **Server**

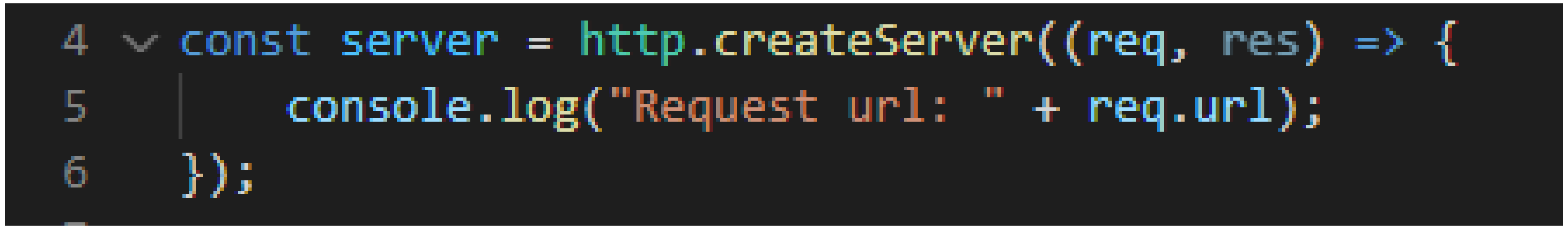
- To activate the server, we have to make it **listen** to a port.
- This can be done using `server.listen`



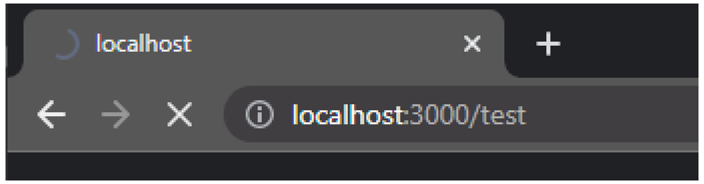
```
server.listen(3000, 'localhost', () => {
    console.log("Server listening...");
});
```

The Request Object

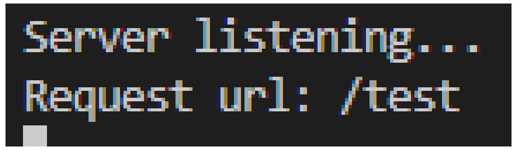
- Using the Request Object (req), do the following inside the createServer callback:
  - Log the request object's URL via the url property



```
const server = http.createServer((req, res) => {
    console.log("Request url: " + req.url);
});
```



in browser



```
Server listening...
Request url: /test
```

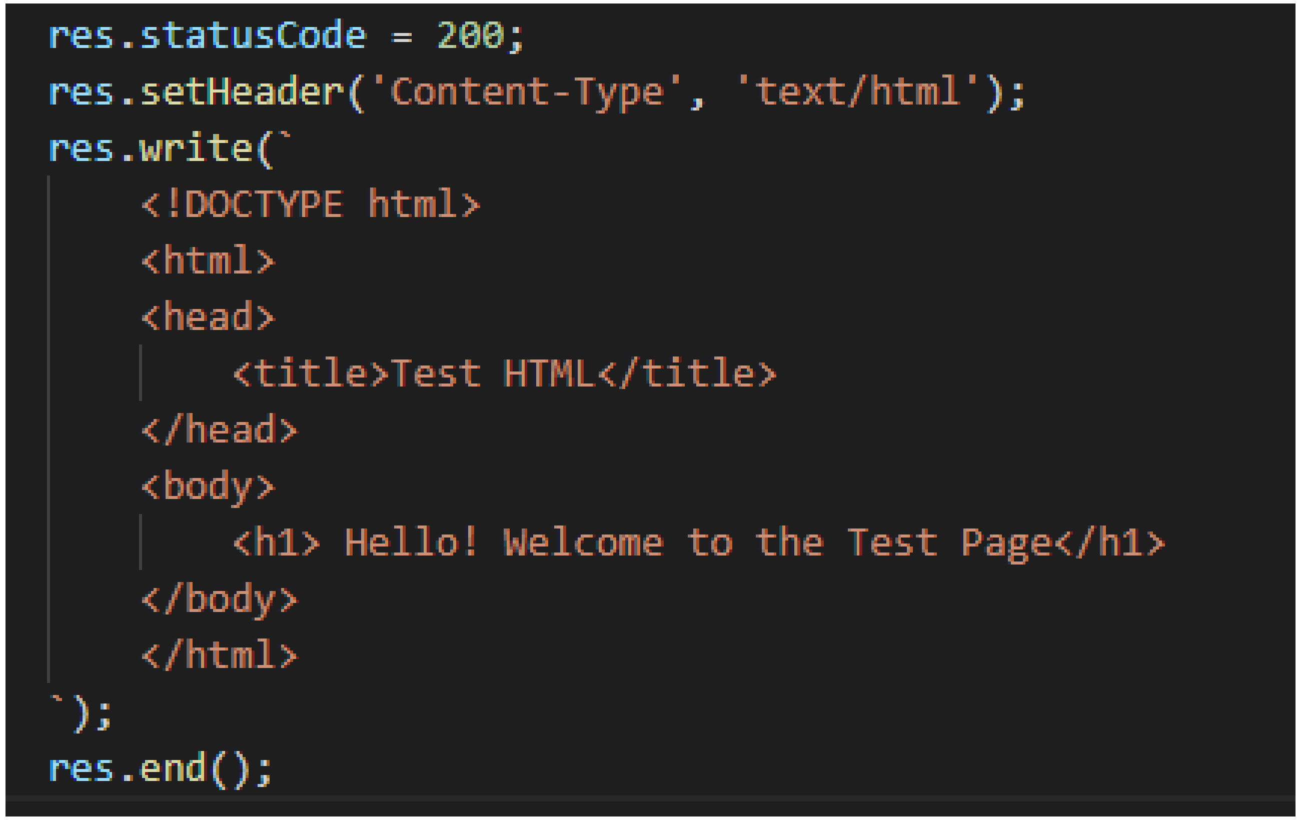
in server

1
2
3
4
5
6

The **Response Object**

- Using the Response Object (res), we can do the following:
  - Create the header
    - The header contains meta-information regarding the data that is sent by the server to the client
    - in V16.15, this is done through `setHeaders()` method
    - status code is set through the `.statusCode` property
  - Create the content
    - refers to the main data that will be sent to the client.
    - done through the `write()` method
  - End the response
    - signals that the response is complete
    - done through the `end()` method

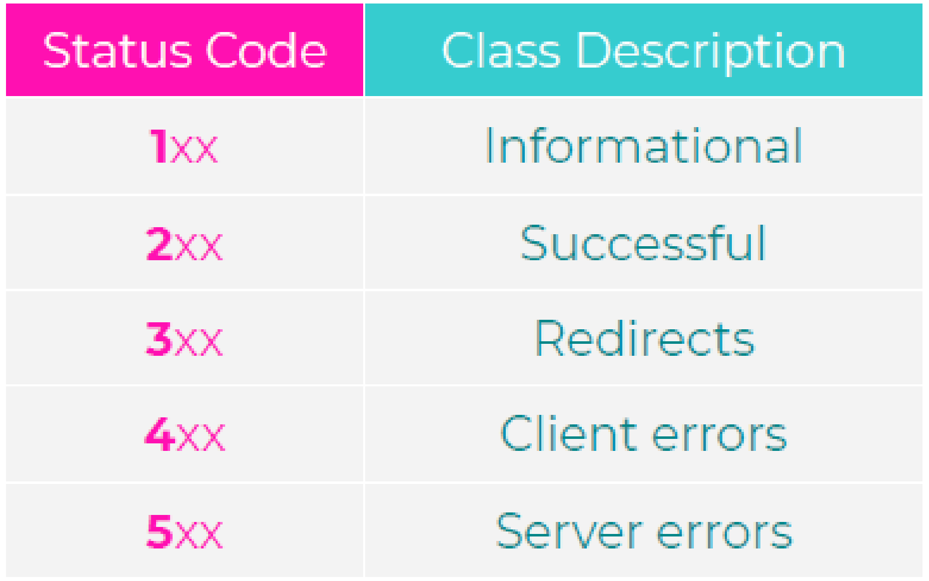
The **Response Object**



```
res.statusCode = 200;
res.setHeader('Content-Type', 'text/html');
res.write(`
    <!DOCTYPE html>
    <html>
    <head>
        <title>Test HTML</title>
    </head>
    <body>
        <h1> Hello! Welcome to the Test Page</h1>
    </body>
    </html>
`);
res.end();
```

Status Codes

- HTTP Response status codes are standard codes which indicate whether the request was successfully completed or not.
- Status codes are grouped in 5 classes:



Status Code	Class Description
1xx	Informational
2xx	Successful
3xx	Redirects
4xx	Client errors
5xx	Server errors

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Status Codes

- **2xx** – Successful
  - **200 OK**
    - Informs the client that the request has been successfully served
  - **204 No content**
    - There’s no content to send for the request but there might be useful headers

Status Codes

- **3xx** – Redirects
  - **301 Moved permanently**
    - The URL has changed, and the new URL is provided in the response body
  - **304 No content**
    - Used in caching. It means the response has not changed since it was last requested
  - **308 Permanent Redirect**
    - Tells the browser to make another request with the URL specified in the ‘Location’ HTTP response header.

Status Codes

- **4xx** – Client Errors
  - **400 Bad Request**
    - Server could not understand the request due to invalid syntax
  - **401 Unauthorized**
    - Clients must authenticate (login or provide valid credentials) to access the resource
  - **403 Forbidden**
    - Client has no permission to access (even if logged in)
  - **404 Not Found**
    - Endpoint is valid but resource does not exist

Status Codes

- **5xx** – Server Errors
  - **500 Internal Server Error**
    - Server has encountered a situation it doesn’t know how to handle
  - **503 Service Unavailable**
    - Server is not ready to handle the request. Usually, it means the server is down or it is overloaded

URL **Routing**

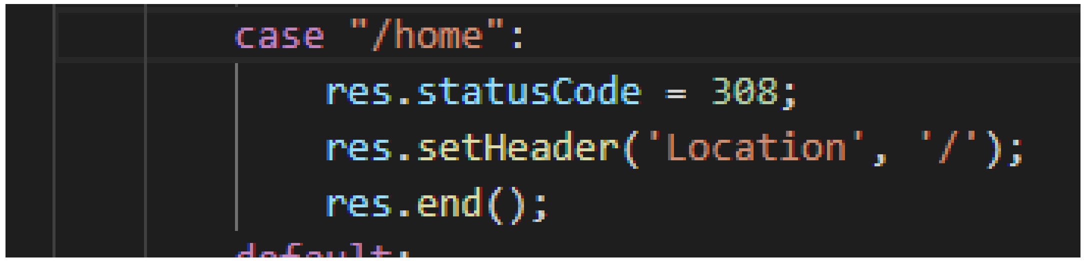
- We don’t want to just keep returning one page over and over again, regardless of the URL
- We need a way to handle various URLs in our web application
- This problem is known as URL routing, and for now, we can achieve this through the use of conditional statements.

URL **Routing**

- Using the Request Object (req), do the following inside the createServer callback:
  - Do a switch case with the URL
  - If URL is "/", read home.html, set res status code to 200
  - If URL is "/profile", read profile.html, set res status code to 200
  - If URL is "/myprofile", redirect to profile, set res status code to 301
  - If URL is neither case, read 404.html, set res status code to 404

URL **Routing**

- When redirecting, you may set the Response Object’s header location to the URL you wish to redirect to.
- An example of URL redirecting is shown below:



```
        case "/home":
            res.statusCode = 308;
            res.setHeader('Location', '/');
            res.end();
```

URL Routing

- So do we need to keep adding switch cases whenever we add new pages to our web app?
  - Essentially yes
- Thankfully, this has been simplified by 3rd party frameworks
  - One example of which is **Express.js**

Further Reading

- There are a couple of useful modules for nodejs backend development:
  - url module: https://nodejs.org/docs/latest-v16.x/api/url.html
    - Provides utility functions for parsing URLs sent by clients
  - fs module: https://nodejs.org/docs/latest-v16.x/api/fs.html
    - Provides utility functions for accessing the server's file system

Assignment

- Learn how to install 3rd party packages using npm
  - You may watch Crash Course 5 (16:48 mins)
    - https://www.youtube.com/watch?v=bdHE2wHT-gQ
  - If you're in a rush, Playback Speed 1.5 is your friend :)
- Install Express.js
  - You may also refer to Crash Course 6 for help (0:00 to 2:27 only)
    - https://www.youtube.com/watch?v=Lr9WUkeYSA8

- Try to have these done before our next meeting

NodeJS

CCAPDEV