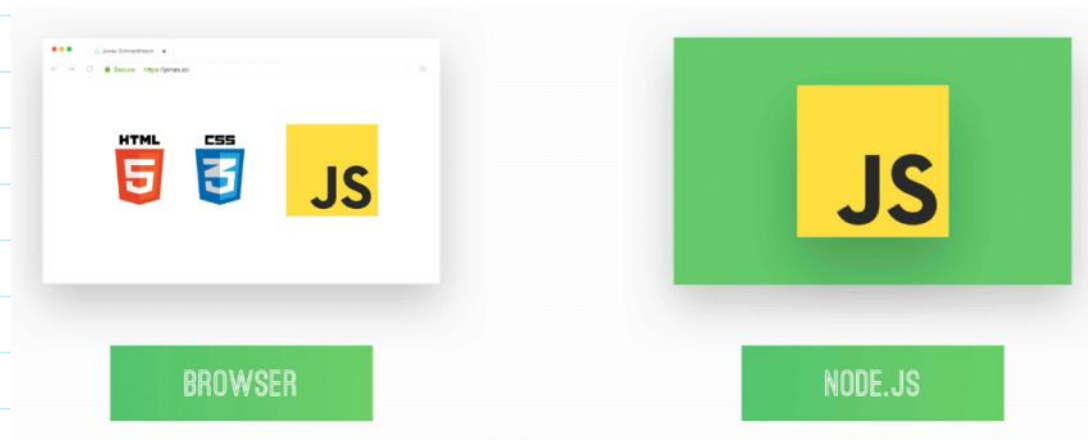


[Node.JS crash course \(bonus from JavaScript Course\)](#)

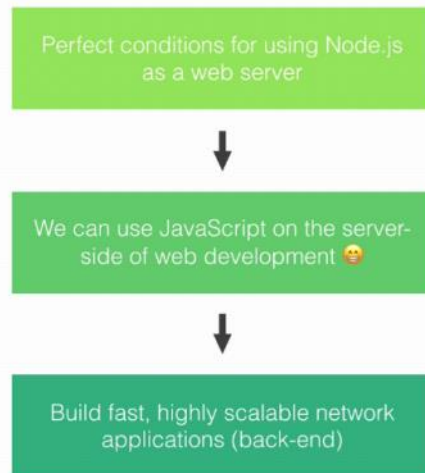
SECTION 12: BONUS: A Node.js Crash Course

164. A Quick Overview of Node.js

- **Node.js:** JavaScript runtime built on google's open source V8 JavaScript Engine
 - JavaScript Runtime: browser is natively the runtime and environment for application created using HTML, CSS, and JavaScript.
 - But, what if we take the JavaScript component out of the browser and execute our code somewhere else without the restrictions that we have in the browser?
 - This is call node.js



- Node.js is just another JavaScript runtime, like a container, or an environment in which JavaScript programs can be executed, but outside of any browser whatsoever
- The V8 Engine is where the JavaScript code is executed
- Now that JavaScript is outside the browser new functionality is introduced
 - Ie accessing file systems
 - Better networking capabilities



165. The Laptop Store Project - Part 1

- Download node and npm
- Run node in terminal
 - node
 - This allows us to write JavaScript lines right on the terminal and it executes right away
 - This also allows us to run js files by performing the following command in the directory where the js file is located
 - Node [filename.js]
- Create js file. We will call it index.js
- Download nodemon that will help in automatically running the file in node every time the js file is saved (-g for global)
 - Npm install nodemon -g
- Run nodemon in our directory where the file is located
 - Nodemon
- Require method allows us to include node packages to be used in our program
 - Fs - filesystem
 - Http
- To load a file, use the readFileSync method from the fs module. Character encoding must be specified
- __dirname is a variable that hold the path of the directory of the project
- For this project we will take our data.json file which is read as a string by the fs module. Now we want to parse it into an object using JSON module so that we can use in in JavaScript

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

const json = fs.readFileSync(`${__dirname}/data/data.json`, 'utf-8');
|
const laptopData = JSON.parse(json);
console.log(laptopData);

const server = http.createServer((req, res) => {
  console.log('Someone accessed the server');
});

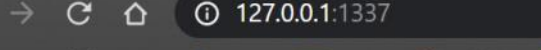
server.listen()
```

- Now lets create web servers which is what Node.js is intended to do
- We will need the http node module
- 1. To create our server we use the createServer method from the http module which accepts a callback function.
 - a. This callback function is that one that is going to be fired each time that someone accesses our webserver (ie, whenever someone opens a page that's on our server.
 - b. This callback function also gets access to the request and response object
- 2. Now we need to listen on a specific port and a specific IP address
 - a. We use the listen method on our newly created server.
 - b. What this does is to basically tell node.js to always keep listening on a certain port on a certain IP address
 - i. Standard port: 1337
 - ii. IP address: localhost = 127.0.0.1
 - c. And then as soon as we start listening, a callback function get fired.
- To check if this is running correctly, open up the browser and set the address to 127.0.0.1:1337. The console should output 'someone has accessed the server'
- To send something back to the browser, we can use the response object
 - o We can send a header which are like small message that we send with a request to let the browser know what kind of data coming in
- Essentially, each time someone accesses our server, our response is to write a header in this case, saying that we have a response so that everything is okay (200 code) and then the content type saying that its going to be text html and then the reponse itself which always happens after the header

```
const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-type': 'text/html' });
  res.end('This is the response');
  console.log('Someone accessed the server');
});

server.listen(1337, '127.0.0.1', () => {
  console.log('Listening for requests now');
});
```

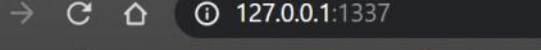
```
[nodemon] starting 'node index.js'
Listening for requests now
Someone accessed the server
Someone accessed the server
```



← → ↻ 🏠 ⓘ 127.0.0.1:1337

Apps 27 📁 📍 📺 📺 📧 📺 📺 📺

This is the response



← → ↻ 🏠 ⓘ 127.0.0.1:1337

Apps 27 📁 📍 📺 📺 📧 📺 📺 📺

This is the response

Has blocked cookies

Name	Status	Type	Initiator	Size	Time	Waterfall
127.0.0.1	200	document	Other	164 B	41 ms	
favicon.ico	200	text/html	Other	164 B	21 ms	

- We will now implement routing
- Routing is basically responding in different ways for different URLs
 - o Different pages have slightly different URLs
 - o Import node package url in our program
 - o We will use parse method from the url module and use it with the request object because that is where the url is actually stored.
 - o The parse method allows us to convert the url string into an object that we can use later on in the code

- With the pathname known, we can now do some proper routing using if else statement with pathname as the condition
- Query is the more specific part of the pathname. It includes the id number

```
const pathName = url.parse(req.url, true).pathname;
const id = url.parse(req.url, true).query.id;
console.log(`pathname: ${pathName}; id: ${id}; datalength: ${laptopData.length}`);

if (pathName === '/products' || pathName === '/') {
  res.writeHead(200, { 'Content-type': 'text/html' });
  res.end('This is the products page');
}

else if (pathName === '/laptop' && id < laptopData.length) {
  res.writeHead(200, { 'Content-type': 'text/html' });
  res.end(`This is the laptop page for laptop ${id}`);
}

else {
  res.writeHead(404, { 'Content-type': 'text/html' });
  res.end('URL was not found in the server');
}
```

166. The Laptop Project - Part 2

- We will be taking care of the application data and the user interface as well by using a technique called templating
- So lets say our HTML template will have a section for the image, description , price etc. The data from our laptopData will be automatically filled into the correct section of the template
- **What we will do is take the HTML and convert it to a template**
 - Create a folder called templates to hold laptop and overview html templates
 - Insert string placeholders

Template-laptop.html

```

<figure class="laptop">
  <p class="laptop_price">${%PRICE%}</p>
  <a href="overview.html" class="laptop__back"><span
class="emoji-left">👉</span>Back</a>
  <div class="laptop_hero">
    
  </div>
  <h2 class="laptop_name">{%PRODUCTNAME%}</h2>
  <div class="laptop_details">
    <p><span class="emoji-left">📺</span> {%SCREEN%}</p>
    <p><span class="emoji-left">💻</span> {%CPU%}</p>
    <p><span class="emoji-left">💾</span> {%STORAGE%}</p>
    <p><span class="emoji-left">📦</span> {%RAM%}</p>
  </div>
  <p class="laptop_description">{%DESCRIPTION%}</p>
  <p class="laptop_source">Source: <a href="https://www.techradar.com/
news/mobile-computing/laptops/best-laptops-1304361"
target="_blank">https://www.techradar.com/news/
mobile-computing/laptops/best-laptops-1304361</a></p>
  <a href="#" class="laptop_link">Buy now for $3199 <span
class="emoji-right">👉👉</span></a>
</figure>

```

Index.js

```

else if (pathName === '/laptop' && id < laptopData.length) {
  res.writeHead(200,{ 'Content-type': 'text/html' });

  fs.readFile(`${__dirname}/templates/template-laptop.html`, 'utf-8',
(err, data) => {
    const laptop = laptopData[id];
    let output = data.replace(/{%PRODUCTNAME%}/g, laptop.productName);
    output = output.replace(/{%IMAGE%}/g, laptop.image);
    output = output.replace(/{%PRICE%}/g, laptop.price);
    output = output.replace(/{%SCREEN%}/g, laptop.screen);
    output = output.replace(/{%CPU%}/g, laptop.cpu);
    output = output.replace(/{%STORAGE%}/g, laptop.storage);
    output = output.replace(/{%DESCRIPTION%}/g, laptop.description);
    res.end(output);
  });
}

```

- Note that we will use something called regex here because some of the strings are repeated in the html more than once. And so regex will help us find that string pattern and replace it with the laptopData object
 - Note that node uses a single thread
 - Sync methods are blocking
 - Async are not

- Next we will implement the overview page.
- Here we will loop through each laptop in our data and create a card for each on in our overview page
- In the HTML template-overview, we will have to split it in two to implement our loop. We need one template for everything that is outside of the card html block so that we have a container where we can then put the cards into
- Note that the image will not show up immediately and needs to be fixed. This is because the URL that we pass in is treated as a route. Almost everything in node.js is about routing.
- When we load the products page for example, it is not just one request but also for style.css, and for the five images for the laptop cards. All of the resources on the page are also requests and we need to respond to them
- So it looks for the URL in our server and does not find it. Therefore we need to add the URL as a route in our server