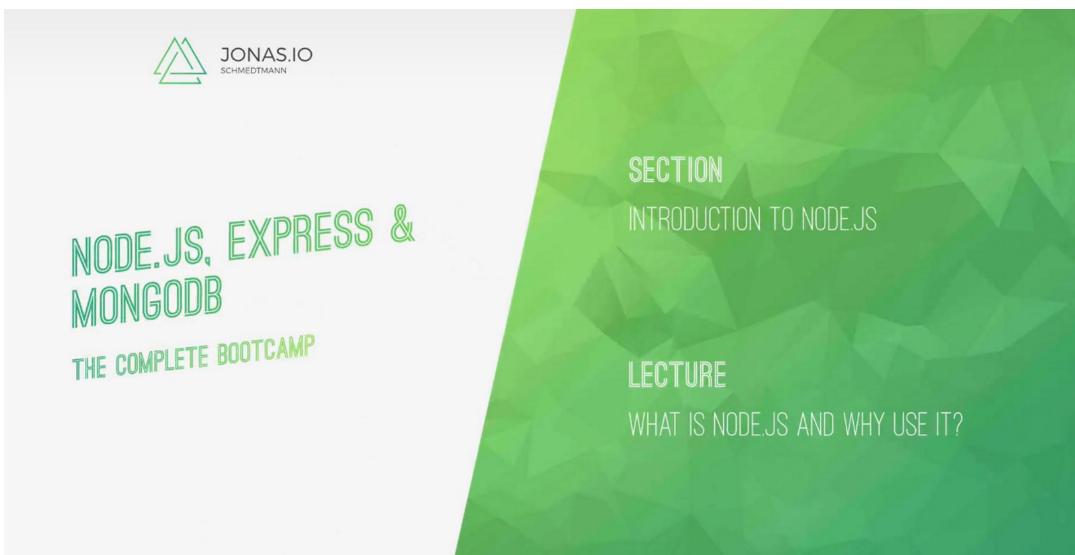


Section 2: Introduction to Node.js and NPM

22 May 2025 19:43



Now before diving into some NodeJS code, let's do a high-level overview of what NodeJS actually is, what we use it for, and why we use Node instead of other technologies.

So, the official definition is that NodeJS is a JavaScript runtime built on Google's open-source V8 JavaScript engine.

Now, what does that actually mean?

Well, let's start by trying to understand what the JavaScript runtime and the V8 engine actually are.

WHAT IS NODE.JS?

NODE.JS

NODE.JS IS A JAVASCRIPT RUNTIME
BUILT ON GOOGLE'S OPEN-SOURCE
V8 JAVASCRIPT ENGINE.

So, you have probably already used JavaScript before, and it was probably always just inside a browser, right?

Because any browser natively understands HTML, CSS, and JavaScript. And no matter if you write vanilla JavaScript or use a framework like React or Angular, that's all just JavaScript that gets executed right in the browser.

So, in this case, the browser is the JavaScript runtime.

But what if we could take JavaScript out of the browser and simply execute our JavaScript code somewhere else, without all the restrictions that we have in the browser?

Well, it turns out that we actually can.

And the solution for this, as you can guess, is called NodeJS.

NodeJS is just another JavaScript runtime. It's like a container—an environment in which a program written in JavaScript can be executed, but outside of any browser whatsoever.

All right, it's actually a bit more complex than this, of course, but for now, this is enough.

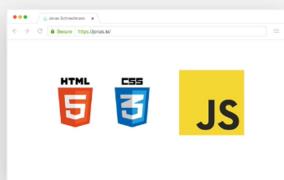
Now, who actually does execute the code if not the browser?

That's where the V8 engine developed by Google comes into play.

Because that is exactly where JavaScript code will be parsed and run in NodeJS, okay?

So, I hope that now the definition of NodeJS being a JavaScript runtime on the V8 JavaScript engine makes a bit more sense.

NODE.JS: JAVASCRIPT OUTSIDE OF THE BROWSER



BROWSER



NODE.JS

We are gonna go into a lot of detail about how Node really works behind the scenes right in the next section.

Anyway, now that we have JavaScript outside of the browser in a kind of standalone environment, which is NodeJS, we can do so many more things with JavaScript that were completely impossible before.

Like accessing the file system or having better networking capabilities—these are now possible with NodeJS.

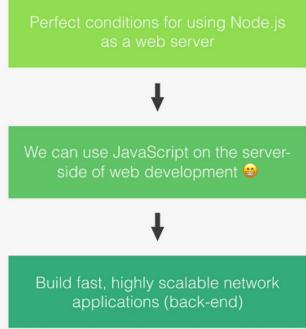
All these factors together give us the perfect conditions for using NodeJS as a web server.

This means we can finally use JavaScript on the server side of web development to build fast, highly scalable network applications that power the back-end for websites or web apps.

And this is absolutely fantastic—and honestly, game-changing—for web development.

So, let's now take a look at some use cases for Node and why it's such a great fit for back-end development.

JAVASCRIPT ON THE SERVER!



And the first thing we need to talk about is the fact that Node applications are so fast and so scalable because NodeJS is single-threaded, based on an event-driven, non-blocking I/O model—which makes NodeJS very lightweight and efficient.

Now, that of course sounds super complicated, I know. But don't worry—we're gonna break down exactly what all of this means a little later in the course.

For now, just keep in mind that Node is perfect for building super fast and scalable data-intensive web applications.

That makes NodeJS a perfect fit for building all kinds of applications—like building an API with a database behind it, preferably a non-relational NoSQL database like MongoDB.

And this is actually *exactly* what we're gonna do later in the course as we dive deeper and deeper into

NodeJS.

But there are, of course, all sorts of other apps we can build—like data streaming apps (think YouTube or Netflix), real-time chat apps, or even server-side web applications where the entire content is generated right on the server.

So as you can see, the possibilities are kinda endless.

But there's actually also a type of app that we *shouldn't* build with Node. And that's when your app needs some super heavy server-side processing—like image manipulation, video conversion, file compression, or anything like that. You'll understand why that's a problem once we dive deeper into how Node actually works.

WHY AND WHEN TO USE NODE.JS?

The infographic is divided into two main sections: 'NODE.JS PROS' (left) and 'USE NODE.JS' (right). Below these is a 'DON'T USE' section.

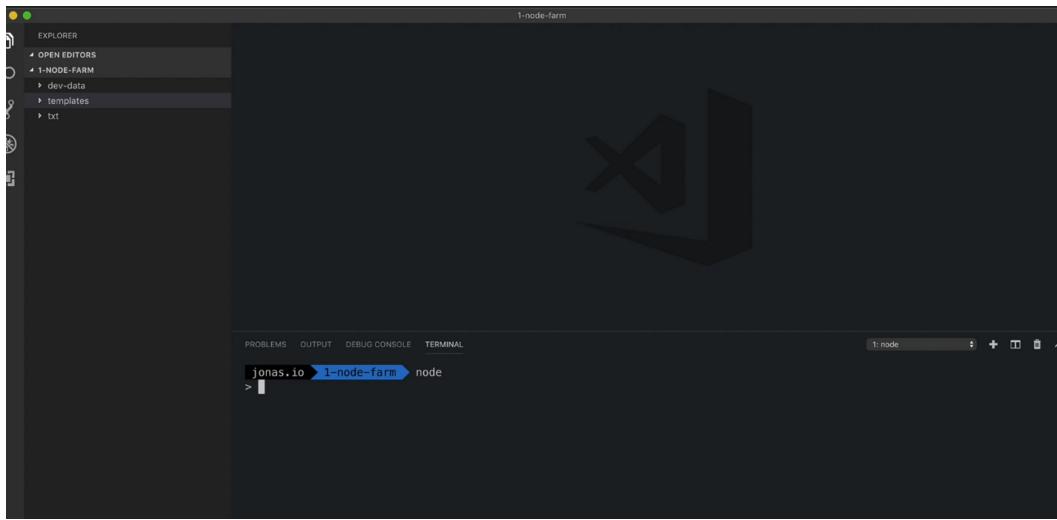
- NODE.JS PROS:**
 - Single-threaded, based on event driven, non-blocking I/O model 😎😊
 - Perfect for building **fast** and **scalable** data-intensive apps;
- USE NODE.JS:**
 - API with database behind it (preferably NoSQL);
 - Data streaming (think YouTube);
 - Real-time chat application;
 - Server-side web application.
- DON'T USE:**
 - Applications with heavy server-side processing (CPU-intensive).

Icons for RAILS, PHP, and PYTHON are shown at the bottom right.

The book cover features a green polygonal background. The title 'NODE.JS, EXPRESS & MONGODB THE COMPLETE BOOTCAMP' is on the left in white. The author's name 'JONAS.IO SCHMIDTMANN' is at the top left. On the right, the word 'SECTION' is above 'INTRODUCTION TO NODE.JS'. Below that is 'LECTURE' and 'RUNNING JAVASCRIPT OUTSIDE THE BROWSER'.

Running JavaScript Outside the Browser

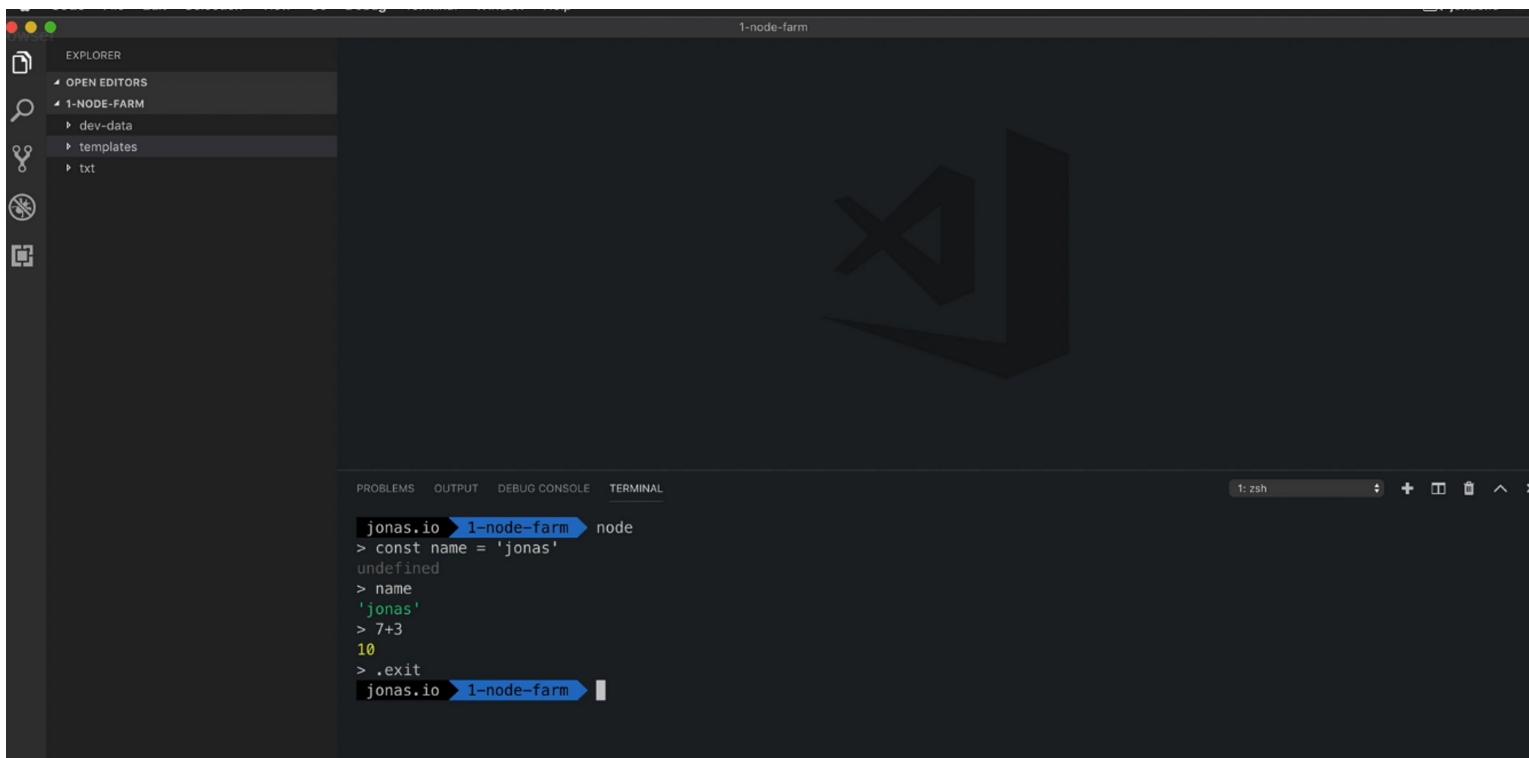
Open up starter project. To start writing some node code here in the console, type "node"



Which will then open up "Node REPL". REPL means 'Read Eval Print Loop'.

Which means we can now write javascript code in the terminal.

Any javascript will work here. Because at the end of the day, NODE JS is a JavaScript runtime.



We use .exit to exit

We can also use CTRL +D

If we hit tab right now, you can see all the global variable that are available in Node.

A screenshot of the Visual Studio Code interface. The title bar shows "jonas.io" and the tab "1-node-farm". The Explorer sidebar on the left has a "1-NODE-FARM" folder expanded, containing "dev-data", "templates", and "txt". The terminal tab is active, showing the Node.js prompt. The output is:

```
jonas.io > 1-node-farm node
> _
jonas.io > 1-node-farm node
>
Array          ArrayBuffer      Atomics        BigInt
BigInt64Array  BigInt64Array   Boolean       Buffer
DTRACE_HTTP_CLIENT_REQUEST DTRACE_HTTP_CLIENT_RESPONSE DTRACE_HTTP_SERVER_REQUEST DTRACE_HTTP_SERVER_RESPONSE
DTRACE_NET_SERVER_CONNECTION DTRACE_NET_STREAM_END DataView       Date
Error          EvalError       Infinity     Float64Array
Function       GLOBAL          Int8Array    Int16Array
Int32Array     Int8Array       Math          NaN
Map            Math            Proxy         Number
Object         Promise         RegExp        RangeError
ReferenceError Reflect        Symbol        Set
SharedArrayBuffer String        URL          SyntaxError
TypeError      URIError       URLSearchParams
```

_ => underscore is basically your previous result.

A screenshot of the Visual Studio Code interface. The title bar shows "1: node". The terminal tab is active, showing the Node.js prompt. The output is:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
unescape      url          util           v8
vm            zlib

__defineGetter__ __defineSetter__ __lookupGetter__ __lookupSetter__
__proto__       hasOwnProperty  isPrototypeOf  valueOf
toLocaleString  toString

constructor

> 3*8
24
> _+6
30
> _-30
0
```

A screenshot of the Visual Studio Code interface. The title bar shows "1: zsh". The terminal tab is active, showing the Node.js prompt. The output is:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
30
> _-30
0
> String.
String.__defineGetter__  String.__defineSetter__  String.__lookupGetter__  String.__lookupSetter__
String.__proto__          String.hasOwnProperty  String.isPrototypeOf  String.propertyIsEnumerable
String.toLocaleString      String.valueOf

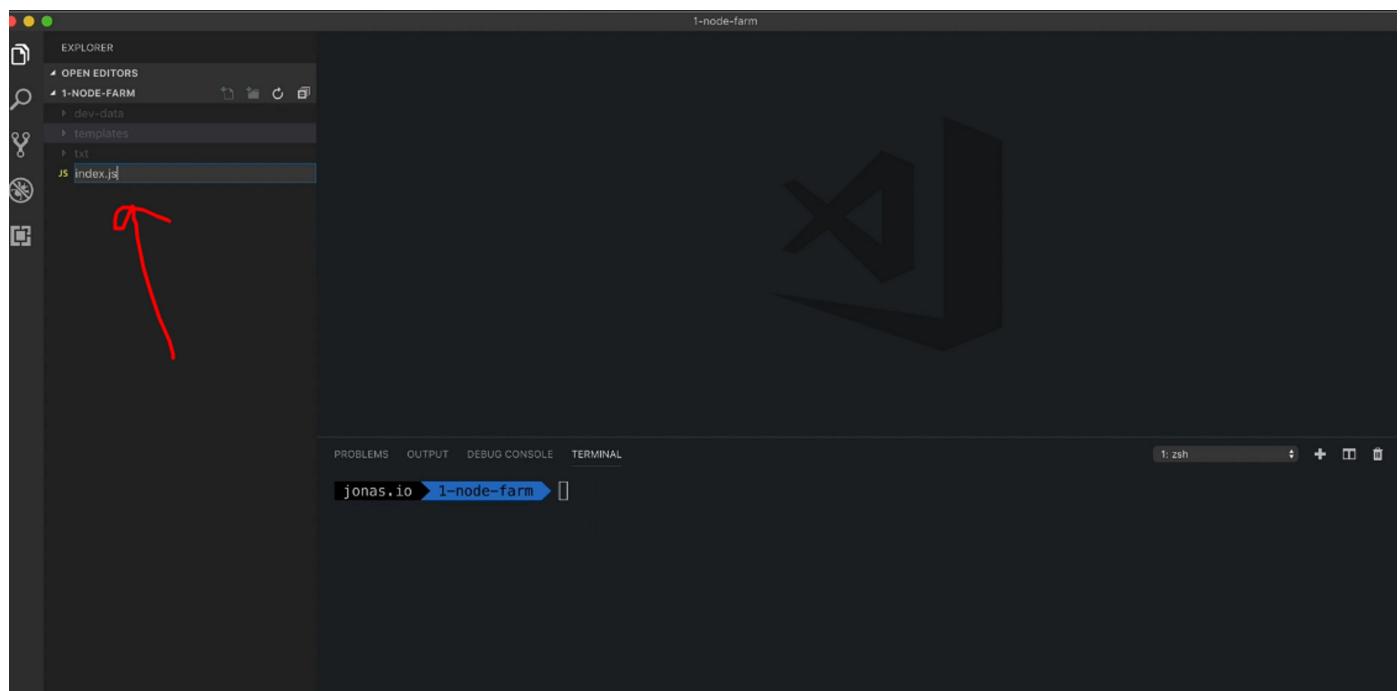
String.apply               String.arguments      String.bind             String.call
String.caller

String.fromCharCode        String.fromCodePoint  String.length          String.name
String.prototype
```

CTRL+K to clear the console.



Let's create a javascript file.



A screenshot of the Visual Studio Code interface. The top menu bar includes Code, File, Edit, Selection, View, Go, Debug, Terminal, Window, and Help. The title bar shows "index.js — 1-node-farm". The Explorer sidebar on the left lists "OPEN EDITORS" (index.js), "1-NODE-FARM" (dev-data, templates, txt), and "index.js". The main editor area contains the following code:

```
JS index.js
1 const hello = 'Hello world';
2 console.log(hello);
```

The terminal at the bottom shows the output of running the script: "Hello world".

With Node Js we can do some amazing stuff which is not possible on the browser.

Like reading file from the file system. In order to do that, we need to use a Node module. So Node.js is really built around this concept of modules where all kinds of additional functionality are stored in a module.

And in this case for reading files, that is inside the FS module.

Here FS stands for file system. By using this module (FS) here, we will get access to functions for reading data and writing data to file system.

A screenshot of the Visual Studio Code interface. The top menu bar includes Code, File, Edit, Selection, View, Go, Debug, Terminal, Window, and Help. The title bar shows "index.js — 1-node-farm". The Explorer sidebar on the left lists "OPEN EDITORS" (index.js), "1-NODE-FARM" (dev-data, templates, txt), and "index.js". The main editor area contains the following code:

```
JS index.js
1 const fs = require('fs');
2
3 const hello = 'Hello world';
4 console.log(hello);
```

A red arrow points to the "require('fs');" line in the code editor. The code editor has syntax highlighting with blue for keywords like "const" and "require", and orange for strings like "Hello world".

So calling this function here will then return an object in which there are lots of functions that we can use.

In case you need some other module, you always know look for Node documentation.

All the modules are listed on the left side

The screenshot shows the Node.js v10.15.3 Documentation page. On the left, there is a sidebar with a dark background containing a list of Node.js modules. The 'REPL' module is currently selected, indicated by a green dot next to its name. The main content area has a light background and displays the 'Table of Contents' for the 'REPL' module. The table of contents includes sections like 'Design and Features', 'Commands and Special Keys', and 'Default Evaluation'. Under 'Commands and Special Keys', several commands are listed: '.break', '.clear', '.exit', '.help', '.save', '.load', and '.editor'. Below the table of contents, there is a code editor window showing a sample REPL session. The session starts with '.editor', which enters an editor mode. A function named 'welcome' is defined, which returns 'Hello \$name!'. When the user types 'welcome("Node.js User")', the output is 'Hello Node.js User!'. At the bottom of the code editor, there is a note about keyboard combinations.

This screenshot shows the same Node.js documentation page, but the 'Commands and Special Keys' section is now the active part of the content. The sidebar remains the same. The main content area now displays a list of special commands supported by all REPL instances. These include '.break', '.clear', '.exit', '.help', '.save', '.load', and '.editor'. Below this list is a code editor window showing a sample REPL session. The session demonstrates the '.editor' command, which enters an editor mode where a simple 'Hello World' script is displayed. The user then exits editor mode and runs the script, resulting in the output 'Hello World'. A note at the bottom states that certain key combinations have special effects in the REPL.

NODE.JS, EXPRESS & MONGODB

THE COMPLETE BOOTCAMP

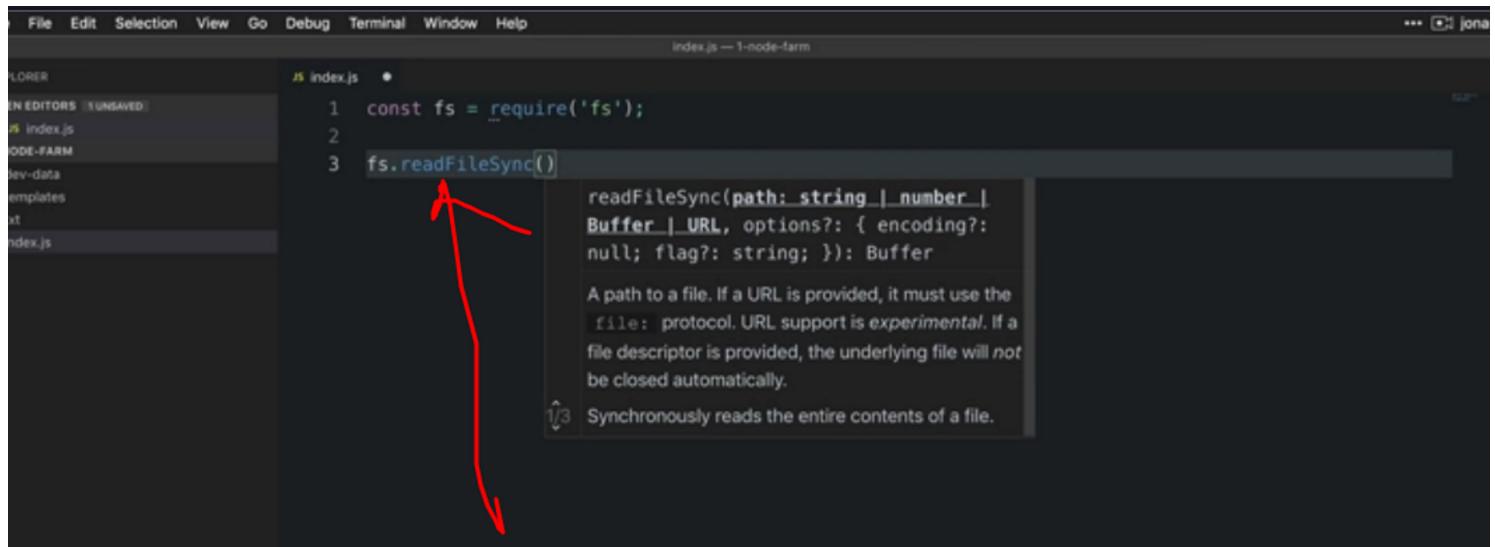
SECTION

INTRODUCTION TO NODEJS

LECTURE

READING AND WRITING FILES

This is really exciting lecture, cause you're gonna learn how to read data from files, and also how to write data into files.



```
File Edit Selection View Go Debug Terminal Window Help
index.js — 1-node-farm
index.js •
1 const fs = require('fs');
2
3 fs.readFileSync()
```

index.js — 1-node-farm

index.js •

```
1 const fs = require('fs');
2
3 fs.readFileSync()
```

1 `readFileSync(path: string | number | Buffer | URL, options?: { encoding?: null; flag?: string; }): Buffer`

A path to a file. If a URL is provided, it must use the `file:` protocol. URL support is experimental. If a file descriptor is provided, the underlying file will not be closed automatically.

1/3 Synchronously reads the entire contents of a file.

This is the synchronous version of file reading. There is also an asynchronous version.

This method takes 2 parameters. First one is the file path and second one is the 'character encoding'.

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows files like index.js, input.txt, and several .txt files in the 1-NODE-FARM folder. The Editor tab has 'index.js' open, displaying code that reads 'input.txt'. The Terminal tab shows the output of running the script, which includes the text from 'input.txt': 'Hello world' and 'The avocado 🥑 is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content 🍩'. A red arrow points from the text 'Here is the content of the text file' to the terminal output.

```
const fs = require('fs');
const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
console.log(textIn);
```

```
jonas.io 1-node-farm> node index.js
Hello world
jonas.io 1-node-farm> node index.js
The avocado 🥑 is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content 🍩
jonas.io 1-node-farm>
```

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows files like index.js, input.txt, and output.txt in the 1-NODE-FARM folder. The Editor tab has 'index.js' open, displaying code that reads 'input.txt', logs its content to the terminal, and then writes it to 'output.txt'. The Terminal tab shows the output of running the script, which includes the text from 'input.txt': 'Hello world' and 'The avocado 🥑 is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content 🍩', followed by 'File written!'. A red arrow points from the text 'File written!' in the terminal output to the 'output.txt' file in the Explorer sidebar.

```
const fs = require('fs');
const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
console.log(textIn);
const textOut = `This is what we know about the avocado: ${textIn}.\\nCreated on ${Date.now()}`;
fs.writeFileSync('./txt/output.txt', textOut);
console.log('File written!');
```

```
jonas.io 1-node-farm> node index.js
Hello world
jonas.io 1-node-farm> node index.js
The avocado 🥑 is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content 🍩
File written!
jonas.io 1-node-farm>
```

NODE.JS, EXPRESS & MONGODB

THE COMPLETE BOOTCAMP

SECTION

INTRODUCTION TO NODE.JS

LECTURE

BLOCKING AND NON-BLOCKING: ASYNCHRONOUS NATURE OF NODE.JS

Let's take a moment to learn about the **asynchronous nature of Node.js**, which includes some absolutely fundamental concepts — like **synchronous**, **asynchronous**, **blocking**, and **non-blocking** code.

Understanding these will be super important for everything that's coming up in this section.

The piece of code we wrote in the last lecture — where we read a file and saved its content into a variable — was done **synchronously**.

That just means each statement is processed one after the other, **line by line**.

First, we require the file system module → then read the file → and finally log the result to the console.
So yeah, **each line waits** for the one before it to finish.

Now, this can be a real issue — especially for **slow operations** — because one line **blocks** the rest of the code from running.

That's why we call synchronous code **blocking code**: stuff only runs after the previous thing is done.
And because of how Node.js is built, this blocking behavior becomes a major problem.
(We'll dig into that more in the next slide.)

SYNCHRONOUS VS. ASYNCHRONOUS CODE (BLOCKING VS. NON-BLOCKING)

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

// Blocking code execution
const input = fs.readFileSync('input.txt', 'utf-8');
console.log(input);
```

SYNCHRONOUS



BLOCKING



The solution to this problem in Node.js is to use **asynchronous, non-blocking code**. With async code, we offload heavy tasks to be handled **in the background**.

And once that task is done, a **callback function** we registered earlier gets called to handle the result.

💡 Meanwhile, the rest of the code keeps running — it **doesn't wait or freeze** — because that heavy task is chillin' in the background.

Basically, we're **deferring** our reaction to the result until it's ready — and that's what makes async code **non-blocking**, which is way more efficient.

Let's look at the example:

We use the `async readFile()` function and pass it a **callback**.

This starts reading the file in the background and then **immediately** moves on to the next line — printing "Reading file..." to the console.

So yeah — no blocking going on here.

Then once the file is fully read, the callback gets triggered, and the file data is finally logged to the console.

Totally different vibe compared to the synchronous version, right?

Now, here's the real question:

Why does it have to be this way in Node.js?

Why is blocking code a big deal?

And why are callbacks used so often?

Let's break that down next... ☀️

SYNCHRONOUS VS. ASYNCHRONOUS CODE (BLOCKING VS. NON-BLOCKING)

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

// Blocking code execution
const input = fs.readFileSync('input.txt', 'utf-8');
console.log(input);
```

SYNCHRONOUS



BLOCKING



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

// Non-blocking code execution
fs.readFile('input.txt', 'utf-8', (err, data) => {
  console.log(data);
});
console.log('Reading file...');
```

ASYNCHRONOUS



NON-BLOCKING



To really understand **why** blocking code is a problem in Node.js, we first gotta understand one core fact:

👉 **Node.js is single-threaded.**

That means when your Node.js app runs, it only uses **one thread** — just *one* set of instructions being executed on the CPU.

So basically, all your app's code runs on that **one single thread**. That's how Node was built — by design.

Now here's the kicker:

Everyone using your app — whether it's 5 users or **5 million** — they're **all sharing that same thread**



Yep. Every user interaction, every request, every line of code — it's all running **on the same thread**. So if just **one user's** code is running something **synchronous and slow** (like reading a huge file).

🔴 Boom — the **entire thread** is locked.

That means **all other users** have to *wait* until that task finishes before *anything else* can happen.

That might be okay if you only have a few users...

But if you've got thousands — or millions — of users?

Yeah... now you've got a problem 💀

Imagine a user hits a route that does a synchronous file read that takes 1 second.

For that 1 second, **nobody else can log in**, request data, or even ping the server.

Total traffic jam 😬

That's why we avoid blocking code in Node.js.

That's why we go **asynchronous**.

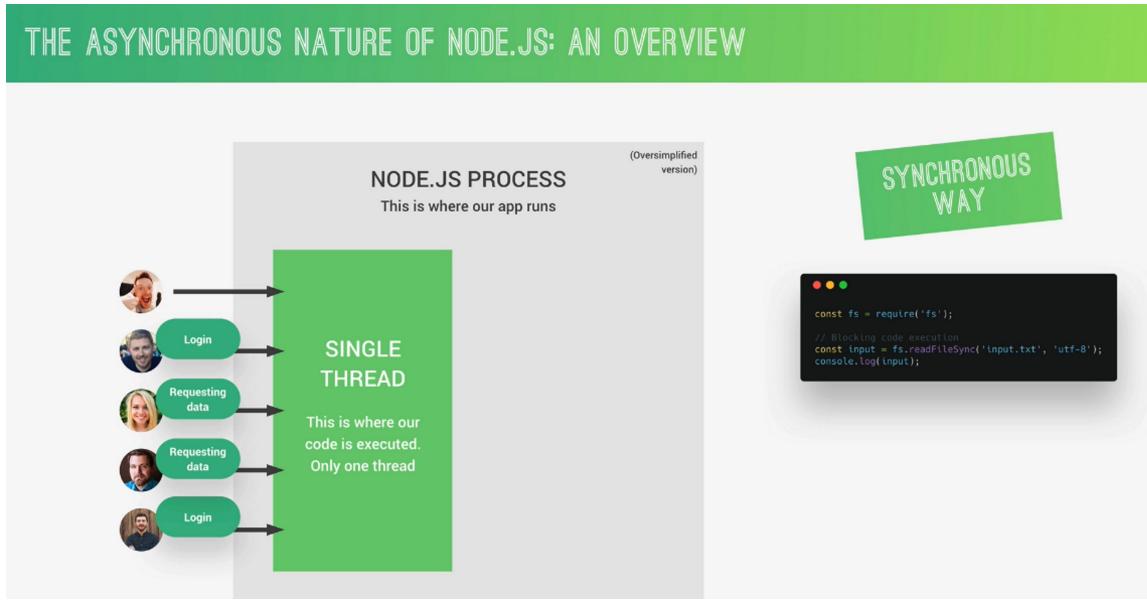
That's why callbacks (and later Promises, async/await, etc.) are a thing.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW



They will have to wait until the file is finished reading. Only when that happens they will finally be able to perform the simpler tasks, one after another.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW

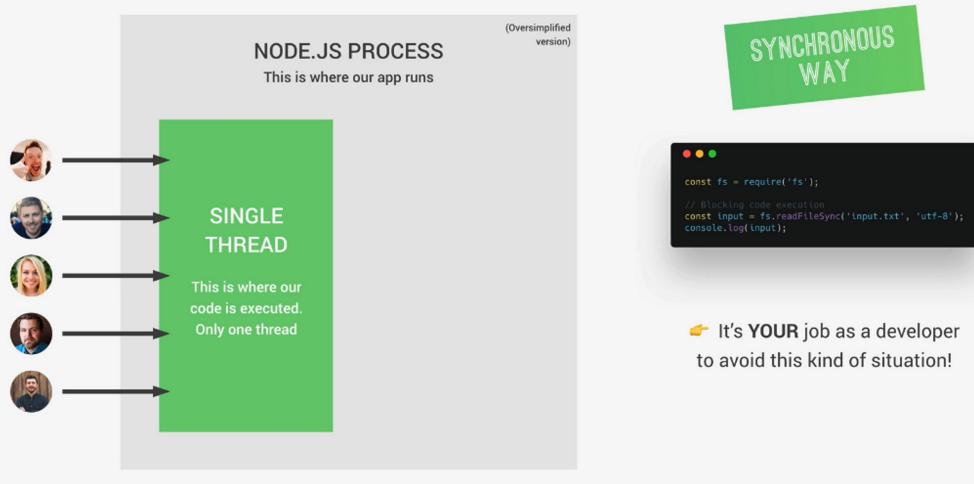


This should be enough to understand the concept. It's better to go step by step and keep things simple from the start, okay?

Now, here's how the situation would play out with synchronous, blocking code—which is obviously a terrible user experience.

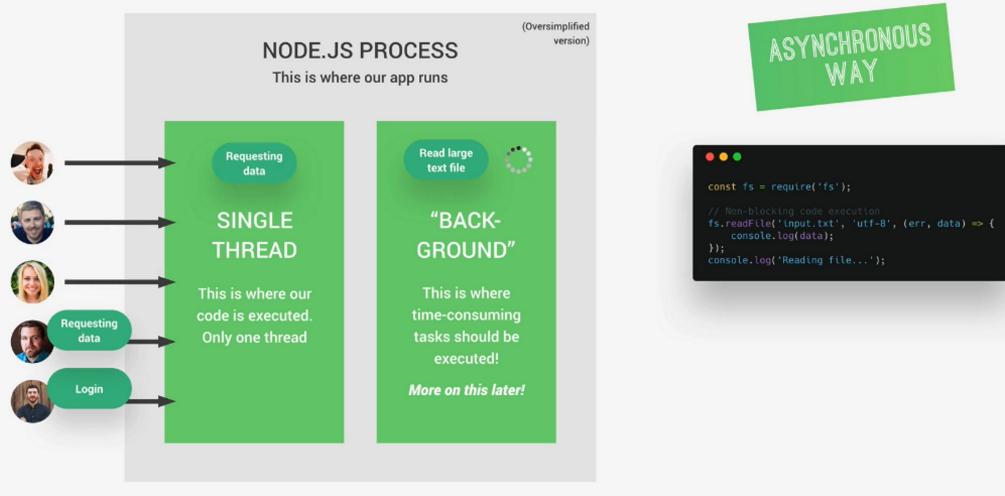
As a developer, it's your responsibility to avoid these issues by using asynchronous code. So, in this case, we should use the asynchronous file read function, which doesn't block the single thread.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW



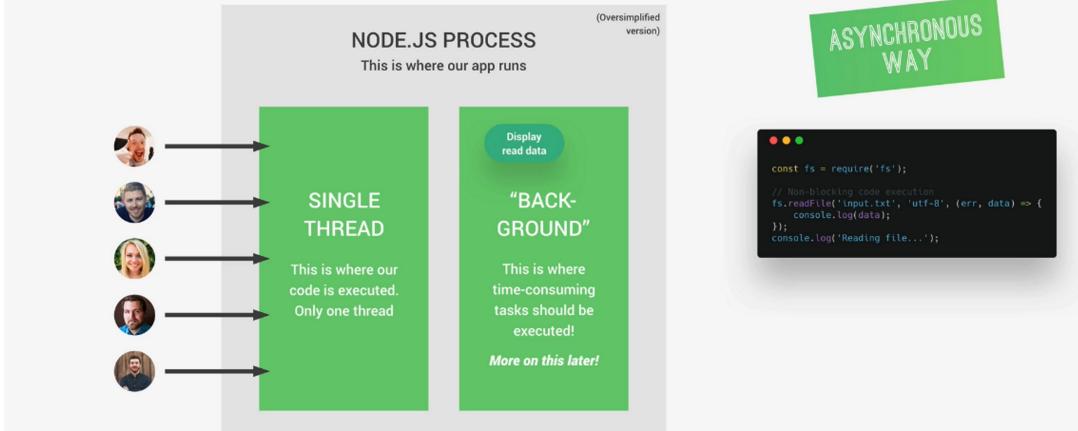
So, in this case, we should use the asynchronous file read function. Instead of blocking the single thread, it handles the heavy work in the background and waits until the file data is fully read. We also register a callback function to be triggered once the data is available. Meanwhile, other users can continue their tasks on the single thread, one after another, while the file is still being read in the background.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW



Now once the data is read, our callback function will get executed.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW



Let's take a minute to understand the **asynchronous nature of Node.js**, which includes essential concepts like synchronous vs. asynchronous and blocking vs. non-blocking code. This is super important for everything coming up next.

In the last lecture, we wrote code to read a file and store its content in a variable. That was done synchronously—meaning the code runs line by line, and each line waits for the previous one to finish. First, we require the file system module, then read the file, then log the result. Simple, but this causes problems with slow operations because each line **blocks** the next.

That's why synchronous code is also called **blocking code**—nothing else can run until the current task is done. In Node.js, that's a huge issue because the entire app runs on **a single thread**.

Yup, **Node.js is single-threaded**, meaning all users share the same thread. If one user triggers a slow, blocking operation like reading a large file, every other user has to wait. That's manageable with five users but a disaster with 5,000.

So, the solution? **Asynchronous, non-blocking code**.

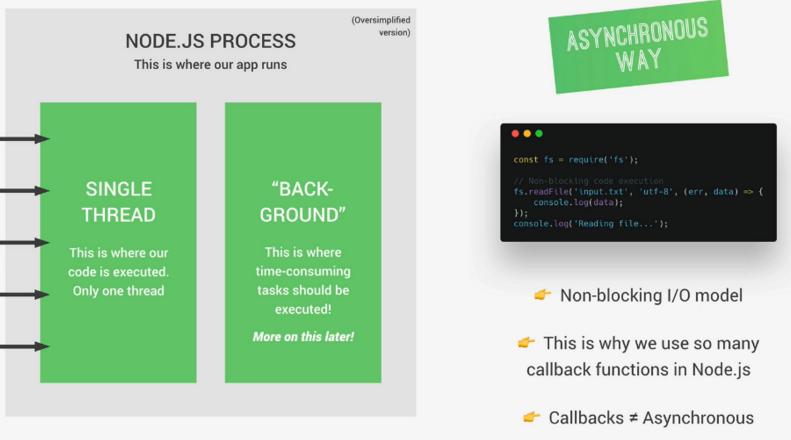
With async code, heavy work is offloaded to the background. While that work runs, Node.js moves on. Once the task finishes, a registered **callback function** gets triggered to handle the result. This keeps the thread free for other users and tasks.

For example, `readFile()` is an async function that starts reading a file in the background and immediately moves on to the next statement—like logging something to the console. When the file is done loading, the callback runs and prints the data. No blocking. Clean. Efficient.

Now, why does Node work like this? Because **there's only one thread**, and blocking it even for a second affects every user. That's why Node leans heavily on callbacks and async behavior—to keep things fast and scalable.

Other languages like PHP spawn a new thread for each user, which is a totally different model. But Node's creator designed it this way to build **high-performance, scalable web apps**, and it just works brilliantly for that.

THE ASYNCHRONOUS NATURE OF NODE.JS: AN OVERVIEW



Now, just as a final note here, it's important to know that, when we use callbacks in our code, that doesn't automatically make it asynchronous, all right? So, passing functions around into other functions is quite common in JavaScript, but of course, again, that doesn't make them asynchronous automatically, okay? It only works this way for some functions in the Node API, such as the readFile function and many, many more, as people explore in the future.

THE PROBLEM: CALLBACK HELL...

CALLBACK HELL

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

fs.readFile('start.txt', 'utf-8', (err, data1) => {
  fs.readFile(`${data1}.txt`, 'utf-8', (err, data2) => {
    fs.readFile('append.txt', 'utf-8', (err, data3) => {
      fs.writeFile('final.txt', `${data2} ${data3}`, 'utf-8', (err) => {
        if (err) throw err;
        console.log('Your file has been saved :D');
      });
    });
  });
});
```

So, this callback model that we just discussed, where one function is called once the one before has finished its work,

can quickly lead to some hard-to-read and unmanageable code. Just take this example where the second file read depends on the first one,

then, the third file read depends on the second one,
and finally, we want to use the final data
to write a file as a result.

That looks quite confusing, right?
I mean, it's gonna work just fine,
but it's just hard to reason about—
and that is just with four levels deep.
Imagine you had like 10 or 20 levels,

which is actually not that uncommon.
Anyway, this is what we call the **callback hell**.
It's such a common problem,
that it already got its own name.
And do you notice this triangular shape here?
That's a very clear sign that you're in callback hell.
Now, how do we actually escape callback hell?

Well, we can use more advanced tools for handling
asynchronous code, like ES6 **promises**
or even better, ES8 **async/await**.

THE PROBLEM: CALLBACK HELL...

CALLBACK HELL

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

fs.readFile('start.txt', 'utf-8', (err, data1) => {
  fs.readFile(`${data1}.txt`, 'utf-8', (err, data2) => {
    fs.readFile('append.txt', 'utf-8', (err, data3) => {
      fs.writeFile('final.txt', `${data2} ${data3}`, 'utf-8', (err) => {
        if (err) throw err;
        console.log('Your file has been saved :D');
      });
    });
  });
});
```

👉 SOLUTION: Using Promises or Async/Await [Optional Section]



In this video, we are gonna read files and write two files just like before. But instead in an asynchronous way.

The screenshot shows a macOS desktop environment. In the top right corner, there's a window titled "index.js — 1-node-farm". The main area displays the following Node.js code:

```
const fs = require('fs');
// Blocking, synchronous way
const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
console.log(textIn);
const textOut = `This is what we know about the avocado: ${textIn}. Created on ${Date.now()}`;
fs.writeFileSync('./txt/output.txt', textOut);
console.log('File written!');

// Non-blocking, asynchronous way
fs.readFile('./txt/start.txt', 'utf-8', (err, data) => {
  console.log(data);
});
console.log('Will read file!');
```

Below the code editor is a terminal window with the title "1: zsh". It shows the command "node index.js" being run, followed by the output "Will read file!" and "read-this".

```
index.js — 1-node-farm
```

```
2
3 // Blocking, synchronous way
4 // const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
5 // console.log(textIn);
6 // const textOut = `This is what we know about the avocado: ${textIn}.\\nCreated on ${Date.now()}`;
7 // fs.writeFileSync('./txt/output.txt', textOut);
8 // console.log('File written!');
9
10 // Non-blocking, asynchronous way
11 fs.readFile('./txt/start.txt', 'utf-8', (err, data1) => {
12   fs.readFile(`./txt/${data1}.txt`, 'utf-8', (err, data2) => {
13     | console.log(data2);
14   });
15 });
16 console.log('Will read file!');
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
```

```
jonas.io > 1-node-farm > node index.js
Will read file!
read-this
jonas.io > 1-node-farm > node index.js
Will read file!
The avocado 🥑 is also used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices.
jonas.io > 1-node-farm >
```

```
index.js — 1-node-farm
```

```
2
3 // Blocking, synchronous way
4 // const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
5 // console.log(textIn);
6 // const textOut = `This is what we know about the avocado: ${textIn}.\\nCreated on ${Date.now()}`;
7 // fs.writeFileSync('./txt/output.txt', textOut);
8 // console.log('File written!');
9
10 // Non-blocking, asynchronous way
11 fs.readFile('./txt/start.txt', 'utf-8', (err, data1) => {
12   fs.readFile(`./txt/${data1}.txt`, 'utf-8', (err, data2) => {
13     | console.log(data2);
14     | fs.readFile('./txt/append.txt', 'utf-8', (err, data3) => {
15       | | console.log(data3);
16     });
17   });
18 });
19 console.log('Will read file!');
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
```

```
jonas.io > 1-node-farm > node index.js
Will read file!
The avocado 🥑 is also used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices.
APPENDIX: Generally, avocados 🥑 are served raw, but some cultivars can be cooked for a short time without becoming bitter.
jonas.io > 1-node-farm >
```

But this code is callback hell!

A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with a folder named '1-NODE-FARM' containing files like 'index.js', 'append.txt', 'start.txt', 'read-this.txt', 'dev-data', 'templates', and 'txt'. The 'index.js' file is open in the editor, displaying code that reads and writes to several text files. The terminal at the bottom shows the execution of 'node index.js' and its output, which includes a list of facts about avocados and a smiley face emoji.

```
6 // const textOut = `This is what we know about the avocado: ${textIn}. Created on ${Date.now()}`;
7 // fs.writeFileSync('./txt/output.txt', textOut);
8 // console.log('File written!');
9
10 // Non-blocking, asynchronous way
11 fs.readFile('./txt/start.txt', 'utf-8', (err, data1) => {
12   fs.readFile('./txt/${data1}.txt', 'utf-8', (err, data2) => {
13     console.log(data2);
14     fs.readFile('./txt/append.txt', 'utf-8', (err, data3) => {
15       console.log(data3);
16
17       fs.writeFile('./txt/final.txt', `${data2}\n${data3}`, 'utf-8', err => {
18         | console.log('Your file has been written 😊');
19       });
20     });
21   });
22 });
23 console.log('Will read file!');
```

jonas.io ➜ 1-node-farm ➔ node index.js
Will read file!
The avocado 🥑 is also used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices.
APPENDIX: Generally, avocados 🥑 are served raw, but some cultivars can be cooked for a short time without becoming bitter.
Your file has been written 😊
jonas.io ➜ 1-node-farm ➔

A screenshot of the Visual Studio Code interface, similar to the one above but with a different file tree. The left sidebar shows a folder named 'final.txt — 1-node-farm' containing files 'index.js', 'final.txt', 'append.txt', 'start.txt', and 'read-this.txt'. The terminal at the bottom shows the execution of 'node index.js' and its output, which includes a list of facts about avocados and a smiley face emoji.

```
1 | The avocado 🥑 is also used as the base for the Mexican dip known as guacamole, as well as a
| spread on corn tortillas or toast, served with spices.
2 | APPENDIX: Generally, avocados 🥑 are served raw, but some cultivars can be cooked for a short
| time without becoming bitter.
```

jonas.io ➜ 1-node-farm ➔ node index.js
Will read file!
The avocado 🥑 is also used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices.
APPENDIX: Generally, avocados 🥑 are served raw, but some cultivars can be cooked for a short time without becoming bitter.
Your file has been written 😊
jonas.io ➜ 1-node-farm ➔

```
index.js — 1-node-farm
Code File Edit Selection View Go Debug Terminal Window Help
OPEN EDITORS
JS index.js
1 NODE-FARM
dev-data
templates
txt
append.txt
final.txt
input.txt
output.txt
read-this.txt
start.txt
JS index.js
9
10 // Non-blocking, asynchronous way
11 fs.readFile('./txt/starttttt.txt', 'utf-8', (err, data1) => {
12   if (err) return console.log('ERROR! 🚨');
13
14   fs.readFile(`./txt/${data1}.txt`, 'utf-8', (err, data2) => {
15     console.log(data2);
16     fs.readFile('./txt/append.txt', 'utf-8', (err, data3) => {
17       console.log(data3);
18
19       fs.writeFile('./txt/final.txt', `${data2}\n${data3}`, 'utf-8', err => {
20         | console.log('Your file has been written 😊');
21       })
22     });
23   });
24 });
25 console.log('Will read file!');

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io ➤ 1-node-farm ➤ node index.js
Will read file!
ERROR! 🚨
jonas.io ➤ 1-node-farm ➤
```



This video is yet another really exciting one, because now we're gonna create our very first real web server capable of accepting requests and sending back responses. And the first step is to include yet another package or another module, and this one is called **http**.

```
index.js — 1-node-farm
JS index.js ●
1 const fs = require('fs');
2 const http = require('http'); -----^
3
4 // Blocking, synchronous way
5 // const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
6 // console.log(textIn);
7 // const textOut = `This is what we know about the avocado: ${textIn}.\\nCreated
8 // `;
9 // fs.writeFileSync('./txt/output.txt', textOut);
10 // console.log('File written!');
11
12 // Non-blocking, asynchronous way
13 fs.readFile('./txt/start.txt', 'utf-8', (err, data1) => {
14   if (err) return console.log('ERROR! ★');
15
16   fs.readFile(`./txt/${data1}.txt`, 'utf-8', (err, data2) => {
17     console.log(data2);
18     fs.readFile(`./txt/append.txt`, 'utf-8', (err, data3) => {
19       fs.writeFile(`./txt/final.txt`, `${data2}\\n${data3}`, 'utf-8', err => {
20         if (err) console.log(`Your file has been written 😊`); -----^
21       });
22     });
23   });
24 });
25 });
26 });
27 });
28 });
29 // console.log('Will read file!');
30
31 /////////////////////////////////
32 // SERVER
33 const server = http.createServer((req, res) => {
34   res.end('Hello from the server!');
35 });
36
37 server.listen(8000, '127.0.0.1', () => {
38   console.log('Listening to requests on port 8000');
39 })
```

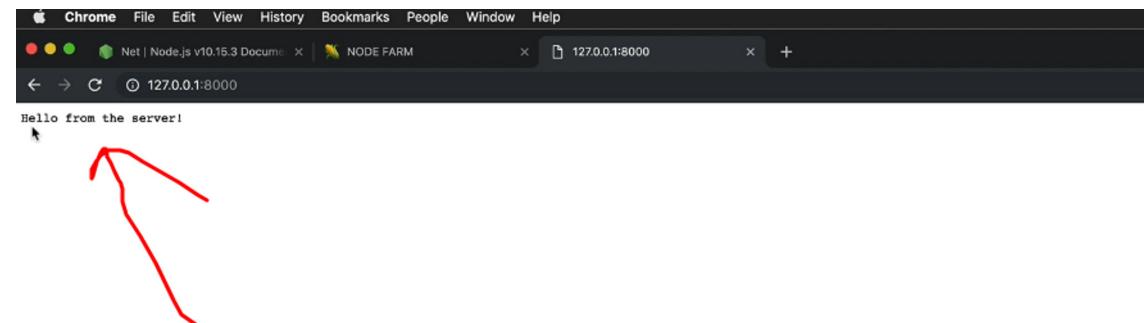
In order to build our server, we have to do two things. First we have to create a server then we start the server.

So `http.createServer` will accept a callback function, which will be fired off each time a new request hits our server. And this callback function gets access to two very important and fundamental variables. It is the `request` variable and `response` variable.

```
index.js — 1-node-farm
JS index.js ●
23 //           fs.writeFile(`./txt/final.txt`, `${data2}\\n${data3}`, 'utf-8', err => {
24 //             console.log(`Your file has been written 😊`); -----^
25 //           });
26 //         });
27 //       });
28 //     });
29 //   });
30
31 /////////////////////////////////
32 // SERVER
33 const server = http.createServer((req, res) => {
34   res.end('Hello from the server!');
35 });
36
37 server.listen(8000, '127.0.0.1', () => {
38   console.log('Listening to requests on port 8000');
39 })
```

Default IP address. This is the host which is localhost.

As you can see the app keeps running, because that's because of something called the event loop.



Chrome

File Edit View History Bookmarks People Window Help

Net | Node.js v10.15.3 Document 127.0.0.1:8000

Hello from the server!

VS Code

Code File Edit Selection View Go Debug Terminal Window Help

index.js

```

23 //     fs.writeFile('./txt/final.txt', `${data2}\n${data3}`, 'utf-8', err => {
24 //         console.log('Your file has been written 😊');
25 //     })
26 // });
27 // });
28 // });
29 // console.log("Will read file!");
30
31 /////////////////////////////////
32 // SERVER
33 const server = http.createServer((req, res) => {
34     console.log(req);
35     res.end('Hello from the server!');
36 });
37
38 server.listen(8000, '127.0.0.1', () => {
39     console.log('Listening to requests on port 8000');
40 });

```

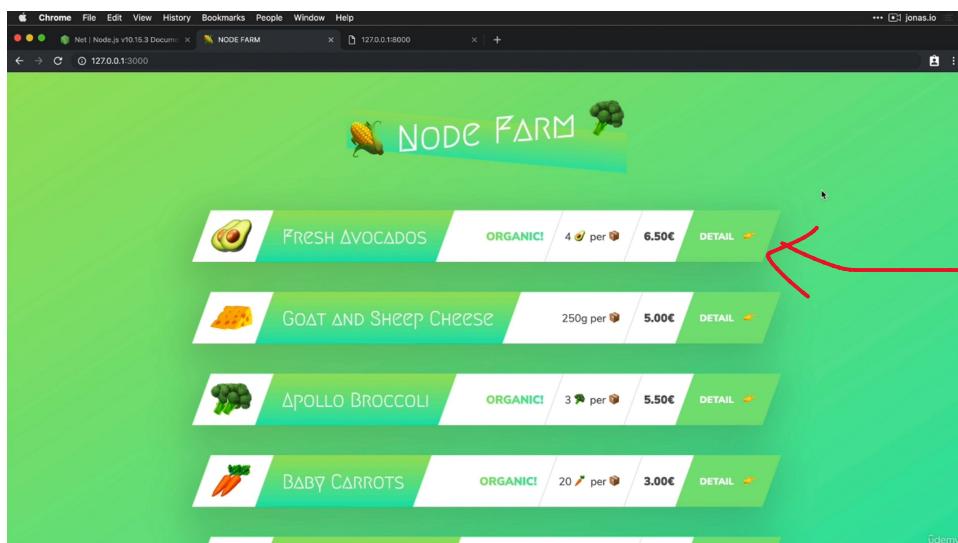
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

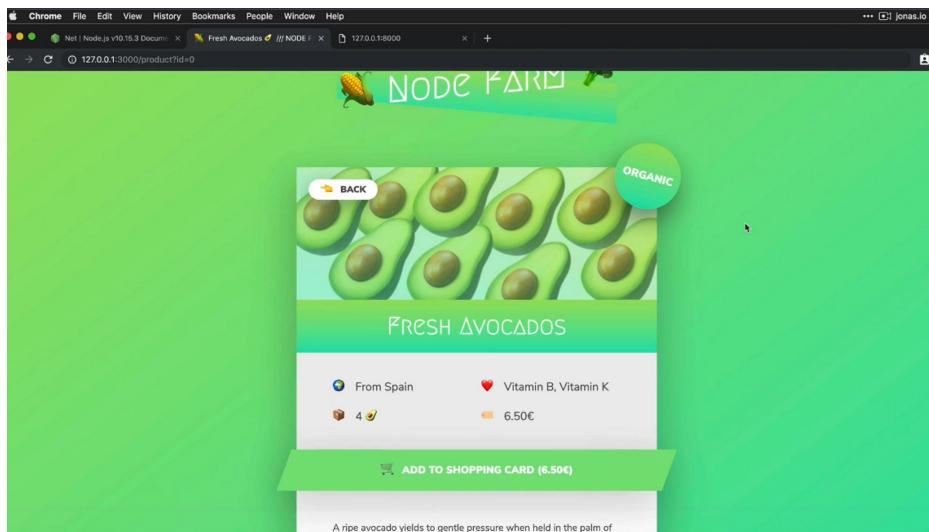
```

jonas.io > l-node-farm > node index.js
listening to requests on port 8000
jonas.io > l-node-farm > node index.js
Listening to requests on port 8000

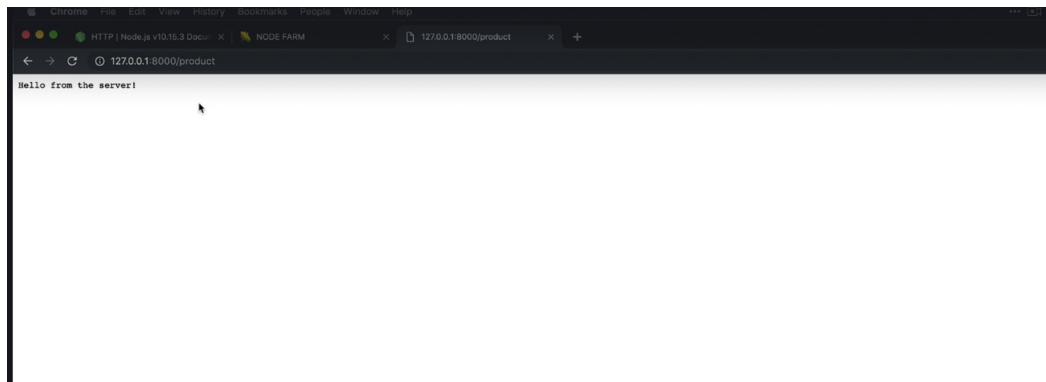
```

We are going to build this project in the next lecture.

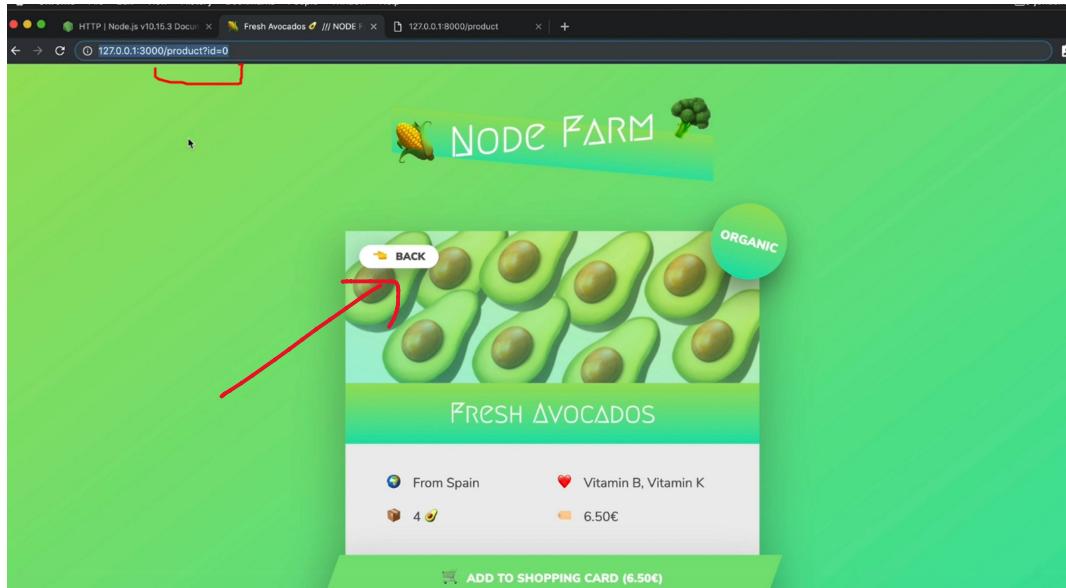
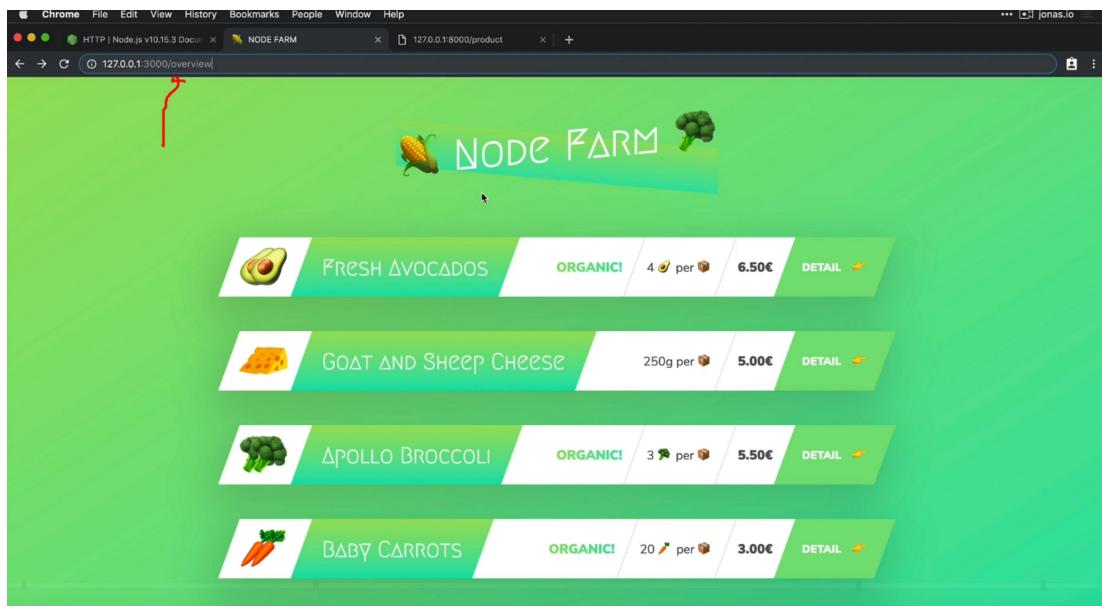




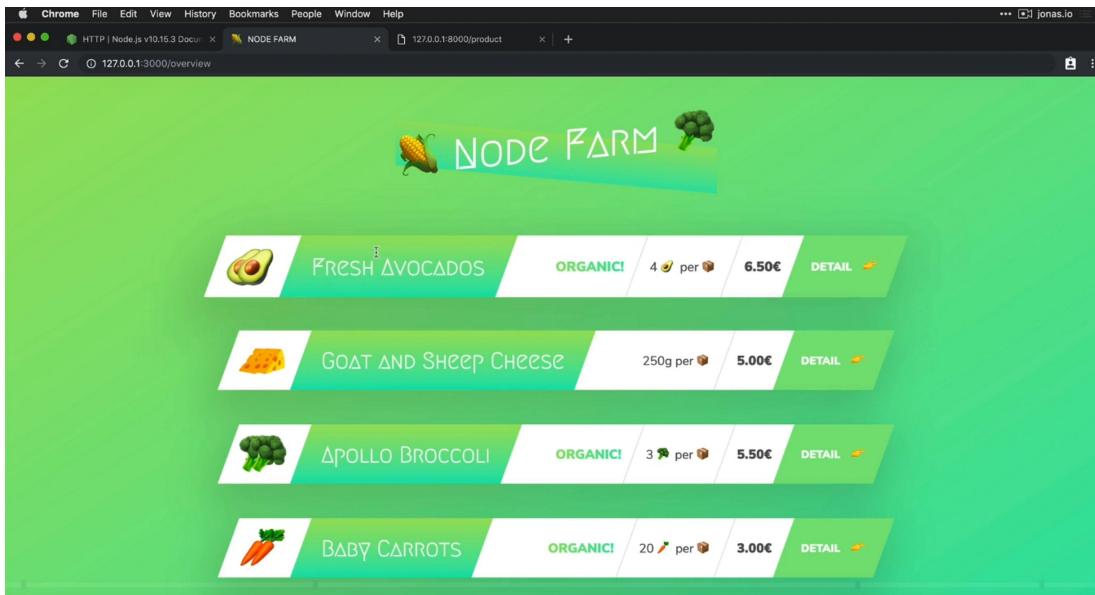
Not work! We get the exact same result.



But what we want to build bro? Lets' look at our final project.



Click on 'Back' button and get back to the 'overview'



So Routing basically means implementing different actions for different URL.

Routing can actually become very complicated in a big real world application. In that case we use a tool like **Express**.



A screenshot of the Visual Studio Code interface. The title bar shows "index.js — 1-node-farm". The Explorer sidebar on the left lists files and folders: "OPEN EDITORS 1 UNSAVED" (index.js), "1-NODE-FARM" (dev-data, templates, txt), and several text files (append.txt, final.txt, input.txt, output.txt, read-this.txt). The main editor area shows the following code:

```
1 const fs = require('fs');
2 const http = require('http');
3 const url = require('url'); // Line 3 has a red underline
4
5 /////////////////////
6 // FILES
7
8 // Blocking, synchronous way
9 // const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
```

The screenshot shows the Visual Studio Code interface with the following details:

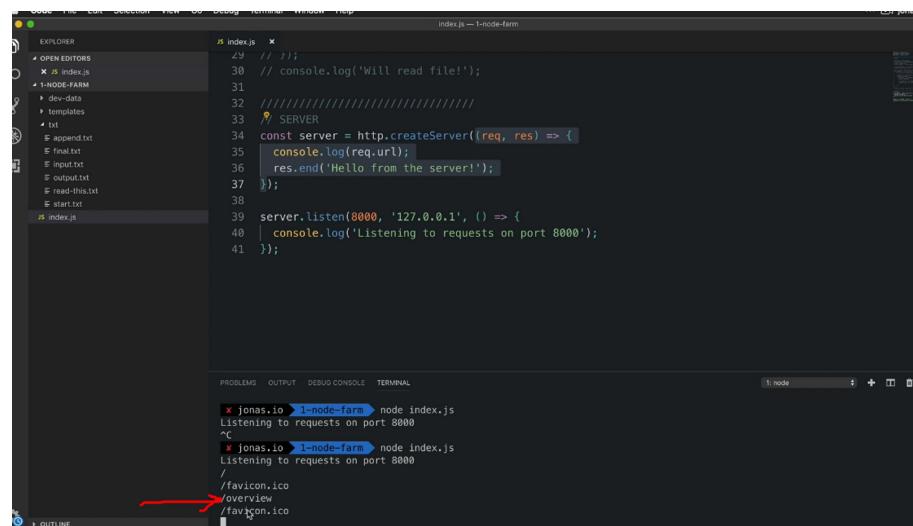
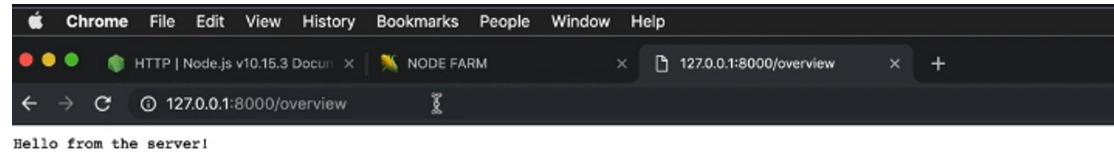
- File Structure (EXPLORER):** Shows files like `index.js`, `1-NODE-FARM`, `dev-data`, `templates`, and several `.txt` files.
- Code Editor (index.js):** The code defines a simple HTTP server. A red arrow points to the line `console.log(req.url);`.

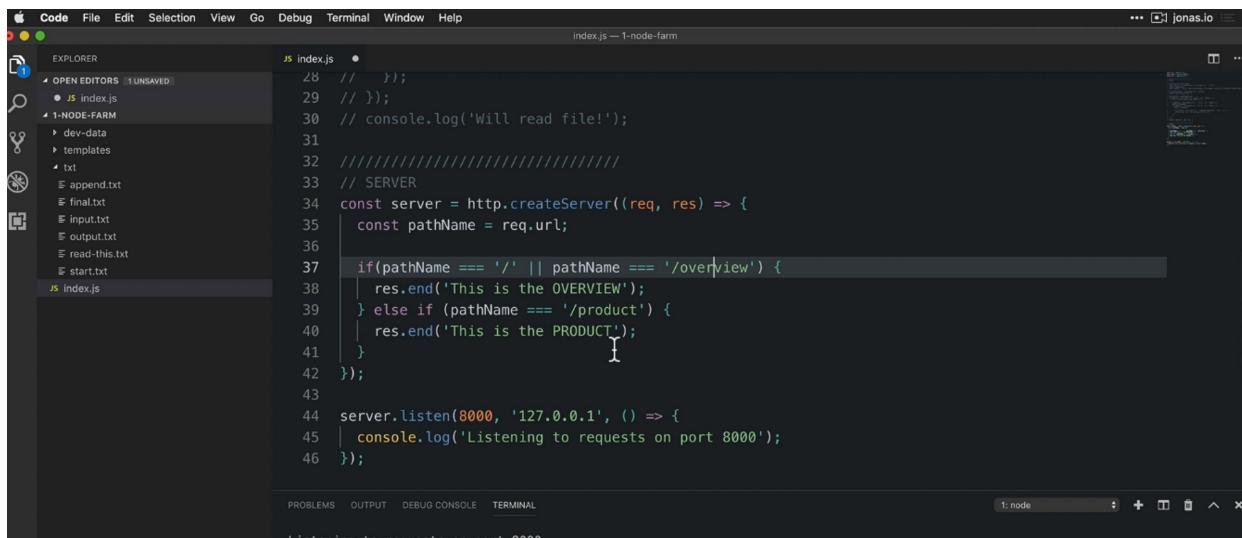
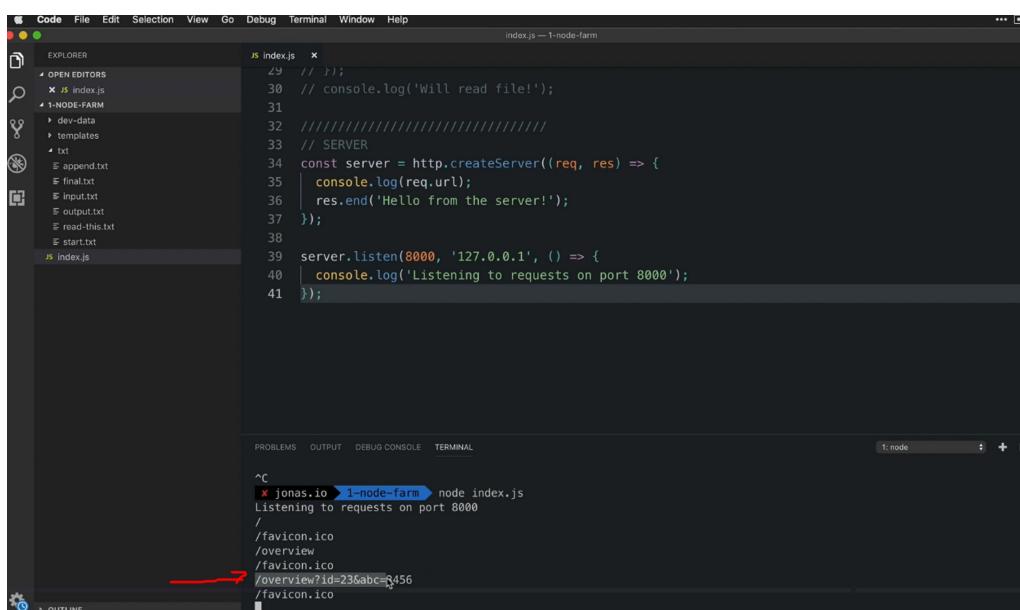
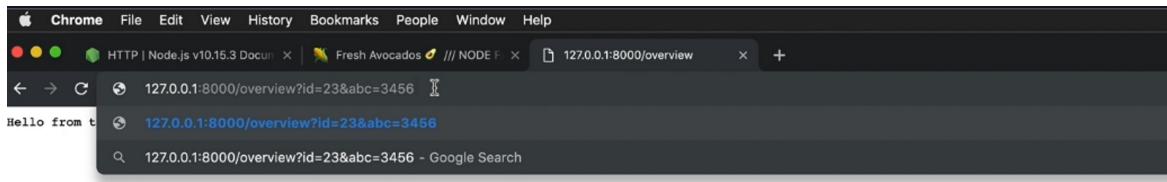
```
JS index.js x
29 // i);
30 // console.log('Will read file!');
31
32 ///////////////////////////////
33 // SERVER
34 const server = http.createServer((req, res) => {
35   console.log(req.url); ←
36   res.end('Hello from the server!');
37 });
38
39 server.listen(8000, '127.0.0.1', () => {
40   console.log('Listening to requests on port 8000');
41 });
```
- Terminal:** Shows the output of running the script with `node index.js`. It prints "Listening to requests on port 8000" and then displays the file tree structure under the root directory, with a red bracket highlighting the path `/favicon.ico`.

Open up the browser and hit the URL, but wait a minute. We got two response. Because

When we are using the browser, browser automatically performs a request for the website's favicon. Ignore it now.

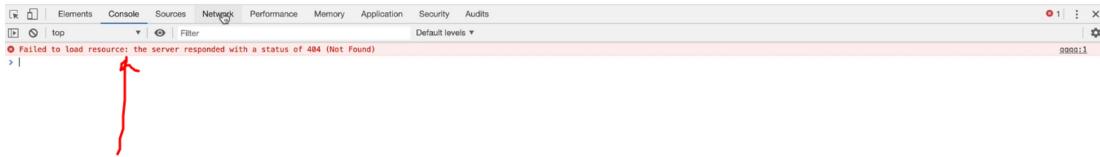
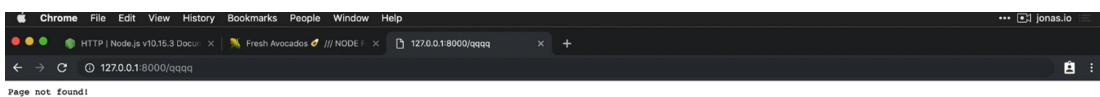
Hit /overview





```
js index.js *  
28 //     });  
29 // }));  
30 // console.log('Will read file!');  
31  
32 ////////////////  
33 // SERVER  
34 const server = http.createServer((req, res) => {  
    const pathName = req.url;  
    if(pathName === '/' || pathName === '/overview') {  
        res.end('This is the OVERVIEW');  
    } else if (pathName === '/product') {  
        res.end('This is the PRODUCT');  
    } else {  
        res.writeHead(404);  
        res.end('Page not found!');  
    }  
});  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47 server.listen(8000, '127.0.0.1', () => {  
    console.log('Listening to requests on port 8000')  
});
```

The screenshot shows the VS Code interface with the file 'index.js' open. A red arrow points from the word '404' in the code to the corresponding line in the browser's developer tools network tab.



HTTP header is basically a piece of information about the response we are sending back.

```
js index.js *  
30 // console.log('Will read file!');  
31  
32 ////////////////  
33 // SERVER  
34 const server = http.createServer((req, res) => {  
    const pathName = req.url;  
    if(pathName === '/' || pathName === '/overview') {  
        res.end('This is the OVERVIEW');  
    } else if (pathName === '/product') {  
        res.end('This is the PRODUCT');  
    } else {  
        res.writeHead(404, {  
            'Content-type': 'text/html',  
            'my-own-header': 'hello-world'  
        });  
        res.end('<h1>Page not found!</h1>');  
    }  
});  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50
```

The screenshot shows the VS Code interface with the file 'index.js' open. A red arrow points from the 'my-own-header' line in the code to the corresponding line in the browser's developer tools network tab.



Now, to start, what actually is an API?

Well, the long answer, you're gonna learn in one of the next sections, but for now, the short answer, at least in this context of web APIs, basically, an API is a service from which we can request some data, so in this case, the data that the user wants to request is data about the products that we are offering in this node farm, so in this project.

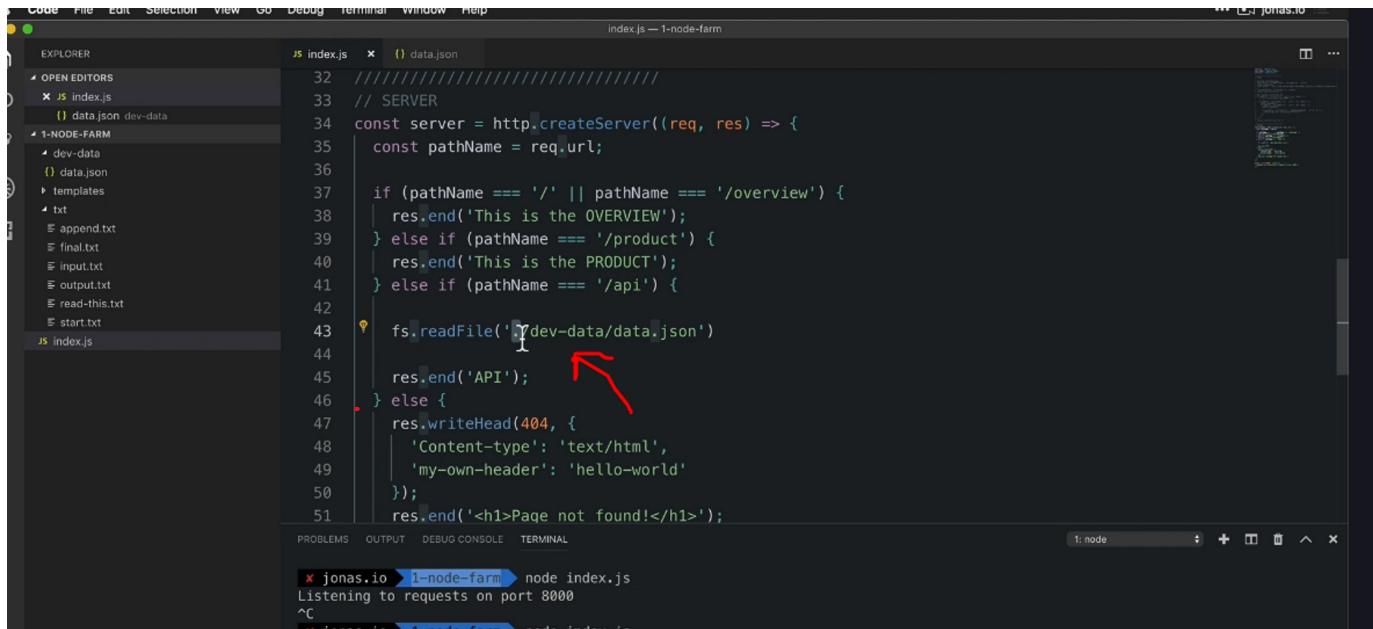
So I have here this dev-data folder, and in there, I have a JSON file, and JSON, in case you don't know, is a very simple text format that looks a lot like JavaScript object.

```
index.js          data.json
1-node-farm > starter > dev-data > data.json > ...
You, 13 hours ago | 1 author (You)
[     You, 13 hours ago • initial
1 [      {
2       "id": 0,
3       "productName": "Fresh Avocados",
4       "image": "avocado.png",
5       "from": "Spain",
6       "nutrients": "Vitamin B, Vitamin K",
7       "quantity": "4 🥑",
8       "price": "6.50",
9       "organic": true,
10      "description": "A ripe avocado yields to gentle pressure when held in the palm of the hand and squeezed. The fruit is not sweet, but distinctly and subtly flavored, with smooth texture. The avocado is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content. Generally, avocado is served raw, though some cultivars, including the common 'Hass', can be cooked for a short time without becoming bitter. It is used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices."
11    },
12  ],
13  {
14    "id": 1,
15    "productName": "Goat and Sheep Cheese",
16    "image": "cheese.png",
17    "from": "Portugal",
```

this dot here actually refers to the directory from which we run the node command in the terminal. So, again, this dot here right now represents this current folder here because that is where we actually run the node command,

so in this 1-node-farm folder, which is our current working directory anyway in this case. But we could have run the node command somewhere else, and then the dot would mean something else. So, for example, we could perfectly fine go to the desktop and run node there, and then specify the whole path to index.js.

But we could do that, but then in this case, the dot would mean the desktop. So if we started the script from the desktop, then this here, this dot, would mean the desktop. And so that is not always ideal, and therefore, there is a better way.

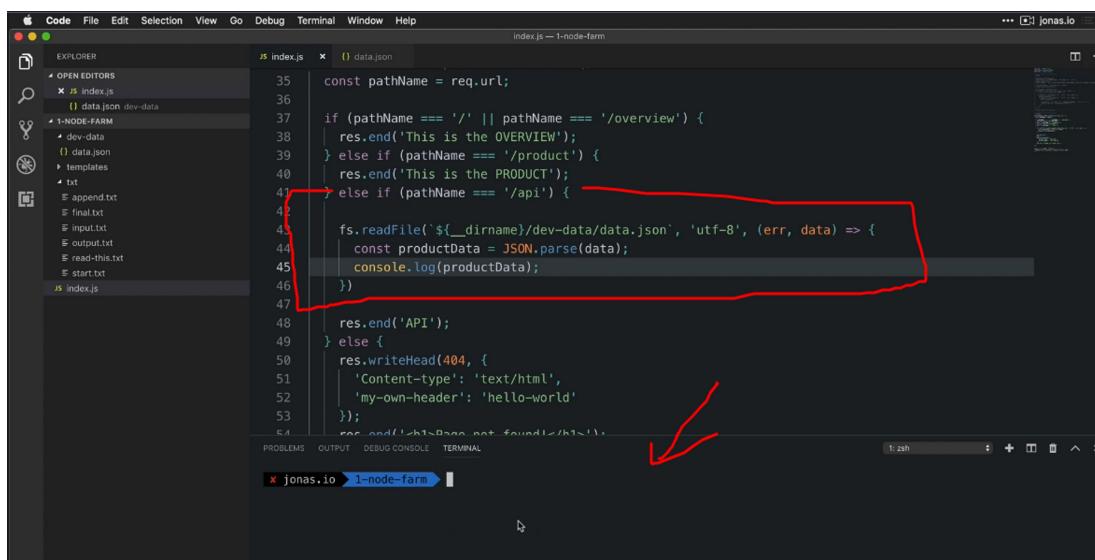


A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with 'index.js' and 'data.json' under 'OPEN EDITORS'. Below them is a folder named '1-NODE-FARM' containing several text files: 'dev-data', 'templates', 'txt', and 'start.txt'. The 'index.js' editor tab is open, showing the following code:

```
32 //////////////////////////////////////////////////////////////////
33 // SERVER
34 const server = http.createServer((req, res) => {
35   const pathName = req.url;
36
37   if (pathName === '/' || pathName === '/overview') {
38     res.end('This is the OVERVIEW');
39   } else if (pathName === '/product') {
40     res.end('This is the PRODUCT');
41   } else if (pathName === '/api') {
42
43     fs.readFile(__dirname + 'dev-data/data.json')
44       .then(data => {
45         res.end('API');
46       })
47     .catch(error => {
48       res.writeHead(404, {
49         'Content-type': 'text/html',
50         'my-own-header': 'hello-world'
51       });
52       res.end('<h1>Page not found!</h1>');
53     });
54   }
55 });
56
57 module.exports = server;
```

A red arrow points to the line `fs.readFile(__dirname + 'dev-data/data.json')` in the code. The terminal at the bottom shows the command `node index.js` being run and the output "Listening to requests on port 8000".

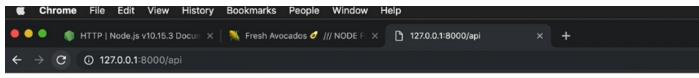
`__dirname` => directory name, where the current file is located.



A screenshot of the Visual Studio Code interface, similar to the previous one but with some changes. The file tree and 'index.js' editor tab are the same. The code has been modified:

```
35 const pathName = req.url;
36
37 if (pathName === '/' || pathName === '/overview') {
38   res.end('This is the OVERVIEW');
39 } else if (pathName === '/product') {
40   res.end('This is the PRODUCT');
41 } else if (pathName === '/api') {
42
43   fs.readFile(`${__dirname}/dev-data/data.json`, 'utf-8', (err, data) => {
44     const productData = JSON.parse(data);
45     console.log(productData);
46   })
47
48   res.end('API');
49 } else {
50   res.writeHead(404, {
51     'Content-type': 'text/html',
52     'my-own-header': 'hello-world'
53   });
54   res.end('<h1>Page not found!</h1>');
55 }
56
57 module.exports = server;
```

A red box highlights the entire block of code starting with `fs.readFile`. A red arrow points to the line `console.log(productData);` in the same block.



The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a tree view of files and folders. Opened files include `index.js` and `data.json`. Folders like `dev-data` and `1-NODE-FARM` are also listed.
- Code Editor:** The `index.js` file contains Node.js code. A red box highlights the JSON parsing logic where `JSON.parse(data)` is called. The code handles different path names and returns JSON data if the path is `/api`.
- Terminal:** A tooltip in the bottom right corner displays the output of the command `node ./bin/www`, which includes a detailed description of an avocado's properties and a JSON object definition.
- Status Bar:** Shows the current file is `index.js — 1-node-farm`, line count as `Ln 45, Col 32`, and other settings like `Spaces: 2`, `UTF-8`, `LF`, `JavaScript`, and `Prettier`.

The screenshot shows a code editor with a file named `index.js` open. The code handles requests based on the path name. It reads a JSON file named `data.json` and returns its contents as JSON if the path is `/product`. If the path is `/api`, it returns the same JSON data. For any other path, it sends a 404 response with a custom header `my-own-header`.

```
if (pathName === '/' || pathName === '/overview') {
  res.end('This is the OVERVIEW');
} else if (pathName === '/product') {
  res.end('This is the PRODUCT');
} else if (pathName === '/api') {
}

fs.readFile(`${__dirname}/dev-data/data.json`, 'utf-8', (err, data) => {
  const productData = JSON.parse(data);
  res.writeHead(200, { 'Content-type': 'application/json'});
  res.end(data);
});

} else {
  res.writeHead(404, {
    'Content-type': 'text/html',
    'my-own-header': 'hello-world'
  });
  res.end('<h1>Page not found!</h1>');
}
```

Below the code editor, a terminal window shows the application running on port 8000. The command `node index.js` was run, and the message "Listening to requests on port 8000" is displayed.

But our code is not efficient. We have to read the file every time once the api hits.

```

  {
    "id": 0,
    "productName": "Fresh Avocados",
    "image": "🥑",
    "from": "Spain",
    "nutrients": "Vitamin B, Vitamin K",
    "quantity": "4 🥑",
    "price": "6.00",
    "organic": true,
    "description": "A ripe avocado yields to gentle pressure when held in the palm of the hand and squeezed. The fruit is not sweet, but distinctly and subtly flavored, with smooth texture. The avocado is popular in vegetarian cuisine as a substitute for meats in sandwiches and salads because of its high fat content. Generally, avocado is served raw, though some cultivars, including the common 'Hass', can be cooked for a short time without becoming bitter. It is used as the base for the Mexican dip known as guacamole, as well as a spread on corn tortillas or toast, served with spices."
  },
  {
    "id": 1,
    "productName": "Goat and Sheep Cheese",
    "image": "🧀",
    "from": "Portugal",
    "nutrients": "Vitamin A, Calcium",
    "quantity": "250g",
    "price": "5.00",
    "organic": false,
    "description": "Creamy and distinct in flavor, goat cheese is a dairy product enjoyed around the world. Goat cheese comes in a wide variety of flavors and textures, from soft and spreadable fresh cheese to salty, crumbly aged cheese. Although it's made using the same coagulation and separation process as cheese made from cow's milk, goat cheese differs in nutrient content."
  },
  {
    "id": 2,
    "productName": "Apollo Broccoli",
    "image": "🥦",
    "from": "Portugal",
    "nutrients": "Vitamin C, Vitamin K",
    "quantity": "3 🥦",
    "price": "5.50",
    "organic": true,
    "description": "Broccoli is known to be a hearty and tasty vegetable which is rich in dozens of nutrients. It is said to pack the most nutritional punch of any vegetable. When we think about green vegetables to include in our diet, broccoli is one of the foremost veggies to come to our mind. Broccoli is a cruciferous vegetable and part of the cabbage family, which includes vegetables such as Brussels sprouts and kale. Although the tastes are different, broccoli and these other vegetables are from the same family."
  },
  {
    "id": 3,
    "productName": "Baby Carrots",
    "image": "🥕",
    "from": "France",
    "nutrients": "Vitamin A, Vitamin K",
    "quantity": "200g",
    "price": "4.00",
    "organic": true
  }
}

```

```

  ...
  34 const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
  35 const dataObj = JSON.parse(data);
  ...
  45 res.writeHead(200, { 'Content-type': 'application/json'});
  46 res.end(data);
  ...
}

  ...

```

The screenshot shows the VS Code interface with the 'index.js' file open. The code is annotated with red boxes highlighting specific sections: lines 34-35 where the JSON data is read and parsed, and lines 45-46 where the response is written with a content type of 'application/json'. The terminal at the bottom shows the command 'node index.js' being run and the message 'Listening to requests on port 8000'.

It works as before. But more efficient.

```

  ...
  34 const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
  35 const dataObj = JSON.parse(data);
  ...
  45 res.writeHead(200, { 'Content-type': 'application/json'});
  46 res.end(data);
  ...
}

  ...

```

The screenshot shows a browser window displaying the JSON data returned by the API endpoint. The data is identical to the one shown in the previous screenshot, containing details about three products: Fresh Avocados, Goat and Sheep Cheese, and Apollo Broccoli.



Let's start with 'product.html' file

```

Code File Edit Selection View Go Debug Terminal Help
product.html — 1-node-farm
OPEN EDITORS 1 UNSAVED
JS index.js 284
● product.html templates 285
  ○ data.json dev-data 286
  ○ overview.html templates 287
  ○ product.html 288
  ○ product.html 289
  ○ product.html 290
  ○ product.html 291
  ○ product.html 292
  ○ product.html 293
  ○ product.html 294
  ○ product.html 295
  ○ product.html 296
  ○ product.html 297
  ○ product.html 298
  ○ product.html 299
  ○ product.html 300
  ○ product.html 301
  ○ product.html 302
  ○ product.html 303
  ○ product.html 304
  ○ product.html 305
  ○ product.html 306
  ○ product.html 307
  ○ product.html 308
  ○ product.html 309
JS index.js
product.html
<span class="product_emoji product_emoji--7">🔗</span>
<span class="product_emoji product_emoji--8">🕒</span>
<span class="product_emoji product_emoji--9">👉</span>
</div>
<h2 class="product_name">{%PRODUCTNAME%}</h2>
<div class="product_details">
  <p><span class="emoji-left">📍</span>From {%FROM%}</p>
  <p><span class="emoji-left">❤️</span>{%NUTRIENTS%}</p>
  <p><span class="emoji-left">⌚</span>{%QUANTITY%}</p>
  <p><span class="emoji-left">👉</span>{%PRICE%}€</p>
</div>
<a href="#" class="product_link">
  <span class="emoji-left">🛒</span>
  Add to shopping card ({%PRICE%}€)</span>
</a>
<p class="product_description">
  A ripe avocado yields to gentle pressure when held in the palm of the
  hand and squeezed. The fruit is not sweet, but distinctly and subtly
</p>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
x jonas.io 1-node-farm node index.js
Listening to requests on port 8000

```

```

Code File Edit Selection View Go Debug Terminal Help
product.html — 1-node-farm
OPEN EDITORS 1 UNSAVED
JS index.js 296
● product.html templates 297
  ○ data.json dev-data 298
  ○ overview.html templates 299
  ○ product.html 300
  ○ product.html 301
  ○ product.html 302
  ○ product.html 303
  ○ product.html 304
  ○ product.html 305
  ○ product.html 306
  ○ product.html 307
  ○ product.html 308
  ○ product.html 309
JS index.js
product.html
<a href="#" class="product_link">
  <span class="emoji-left">🛒</span>
  Add to shopping card ({%PRICE%}€)</span>
</a>
<p class="product_description">
  {>DESCRIPTION</p>
</p>
</figure>
</div>
</body>
</html>

```

```
product.html — 1-node-farm
index.js  product.html  data.json  overview.html

271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290

<figure class="product">
  <div class="product__organic"><h5>Organic</h5></div>
  <a href="#" class="product__back">
    <span class="emoji-left">></span>Back
  </a>
  <div class="product__hero">
    <span class="product__emoji product__emoji--1">%IMAGE%</span>
    <span class="product__emoji product__emoji--2">%IMAGE%</span>
    <span class="product__emoji product__emoji--3">%IMAGE%</span>
    <span class="product__emoji product__emoji--4">%IMAGE%</span>
    <span class="product__emoji product__emoji--5">%IMAGE%</span>
    <span class="product__emoji product__emoji--6">%IMAGE%</span>
    <span class="product__emoji product__emoji--7">%IMAGE%</span>
    <span class="product__emoji product__emoji--8">%IMAGE%</span>
    <span class="product__emoji product__emoji--9">%IMAGE%</span>
  </div>
  <h2 class="product__name">{PRODUCTNAME}</h2>
  <div class="product__details">
    <p><span class="emoji-left">></span>From {FROM}</p>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

jonas.io 1-node-farm node index.js
Listening to requests on port 8000

```
product.html — 1-node-farm
index.js  product.html  data.json  overview.html

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

<link
  rel="icon"
  href="https://emojipedia-us.s3.dualstack.us-west-1.amazonaws.com/thumbs/240/apple/155/
  ear-of-maize_1f33d.png"
/>
<title>{PRODUCTNAME} /// NODE FARM</title>
<style>
  *
  *::before,
  *::after {
    margin: 0;
    padding: 0;
    box-sizing: inherit;
  }
```

EXPLORER Code File Edit Selection View Go Debug Terminal Window Help

jonas.io

```
product.html — 1-node-farm
index.js  product.html  data.json  overview.html

271
272
273
274
275
276
277
278

<figure class="product">
  <div class="product__organic {NOT_ORGANIC}"><h5>Organic</h5></div>
  <a href="#" class="product__back">
    <span class="emoji-left">></span>Back
  </a>
  <div class="product__hero">
    <span class="product__emoji product__emoji--1">%IMAGE%</span>
```

```
template-product.html — 1-node-farm
index.js  template-product.html  data.json  overview.html

1
<!DOCTYPE html>
2
<html lang="en">
3
  <head>
4
    <meta charset="UTF-8" />
5
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
7
    <link
      href="https://fonts.googleapis.com/css?family=Megrin|Nunito+Sans:400,900"
      rel="stylesheet"
    />
8
    <link
      rel="icon"
      href="https://emojipedia-us.s3.dualstack.us-west-1.amazonaws.com/thumbs/240/apple/155/
      ear-of-maize_1f33d.png"
    />
9
10
<title>{PRODUCTNAME} {IMAGE} /// NODE FARM</title>
11
12
13
14
15
16
17
18
19

<style>
  *
```

EXPLORER Code File Edit Selection View Go Debug Terminal Window Help

jonas.io 1-node-farm node index.js
Listening to requests on port 8000

VS Code interface showing the Explorer, Editor, and Terminal panes. The Explorer pane shows a project structure with folders like 'OPEN EDITORS', '1-NODE-FARM', and 'txt'. The Editor pane has tabs for 'index.js', 'data.json', 'template-overview.html', and 'template-card.html'. The Terminal pane is empty.

VS Code interface showing the Explorer, Editor, and Terminal panes. The Explorer pane shows a project structure. The Editor pane displays the code for 'template-card.html' (lines 1-20). A red box highlights the tab for 'template-card.html' in the Editor tab bar.

```
<figure class="card">
  <div class="card_emoji">🐐</div>
  <div class="card_title-box">
    <h2 class="card_title">Goat and Sheep Cheese</h2>
  </div>

  <div class="card_details">
    <div class="card_detail-box">
      <h6 class="card_detail">250g per 🐑</h6>
    </div>

    <div class="card_detail-box">
      <h6 class="card_detail card_detail--price">5.00€</h6>
    </div>
  </div>

  <a class="card_link" href="#">
    <span>Detail <i class="emoji-right">➡</i></span>
  </a>
</figure>
```

VS Code interface showing the Explorer, Editor, and Terminal panes. The Explorer pane shows a project structure. The Editor pane displays the code for 'template-overview.html' (lines 186-200). A red box highlights the tab for 'template-overview.html' in the Editor tab bar.

```
<style>
</head>

<body>
  <div class="container">
    <h1>Node Farm 🐑</h1>
    <div class="cards-container">
      {%PRODUCT_CARDS%}
    </div>
  </div>
</body>
</html>
```

VS Code interface showing the Explorer, Editor, and Terminal panes. The Explorer pane shows a project structure. The Editor pane displays the code for 'template-card.html' (lines 1-6). A red box highlights the tab for 'template-card.html' in the Editor tab bar.

```
<figure class="card">
  <div class="card_emoji">%IMAGE%-%IMAGE%</div>
  <div class="card_title-box">
    <h2 class="card_title">%PRODUCTNAME%</h2>
  </div>
```

```
index.js
template-card.html templates
1-NODE-FARM
dev-data
data.json
templates
template-card.html
template-overview.html
template-product.html
txt
append.txt
final.txt
input.txt
output.txt
read-this.txt
start.txt
index.js

<div class="card__emoji">%{IMAGE}%</div>
<div class="card__title-box">
| <h2 class="card__title">%(PRODUCTNAME)%</h2>
</div>

<div class="card__details">
<div class="card__detail-box %{NOT_ORGANIC%}">
| <h6 class="card__detail">%(QUANTITY)% per 🍎</h6>
</div>

<div class="card__detail-box">
| <h6 class="card__detail card__detail--price">%(PRICE)%</h6>
</div>
</div>

<a class="card__link" href="/product?id=%{ID%}">
| <span>Detail <i class="emoji-right">👉</i></span>
| </a>
</figure>
```



Let's replace the placeholder with the actual content.

Let's add some comment

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like `index.js`, `data.json`, and `templates`.
- Code Editor:** The file `index.js` contains the following code:

```
const server = http.createServer((req, res) => {
  const pathName = req.url;

  // Overview page
  if (pathName === '/' || pathName === '/overview') {
    res.end('This is the OVERVIEW');

    // Product page
  } else if (pathName === '/product') {
    res.end('This is the PRODUCT');

    // API
  } else if (pathName === '/api') {
    res.writeHead(200, { 'Content-type': 'application/json'});
    res.end(data);

    // Not found
  } else {
    res.writeHead(404, {
      'Content-type': 'text/html',
      'my-own-header': 'hello-world'
    });
    res.end('<h1>Page not found!</h1>');
  }
});
```
- Annotations:** Red arrows point to specific parts of the code:
 - A red arrow points to the first `if` block under the heading `// Overview page`.
 - A red arrow points to the second `if` block under the heading `// Product page`.
 - A red arrow points to the third `if` block under the heading `// API`.
 - A red arrow points to the `else` block under the heading `// Not found`.
- Terminal:** Shows the command `node index.js` being run.
- Status Bar:** Shows the path `jonas.io → 1-node-farm` and the status `node index.js`.

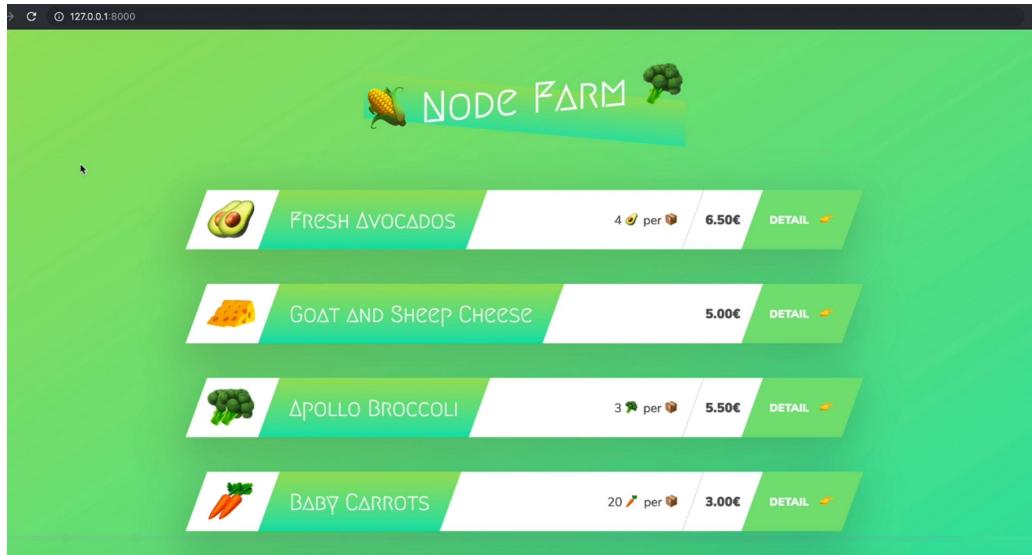
index.js — 1-node-farm

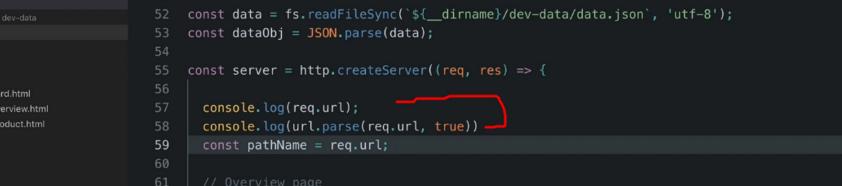
```

31
32 ///////////////////////////////////////////////////////////////////
33 // SERVER
34 const tempOverview = fs.readFileSync(`${__dirname}/templates/template-overview.html`, 'utf-8');
35 const tempCard = fs.readFileSync(`${__dirname}/templates/template-card.html`, 'utf-8');
36 const tempProduct = fs.readFileSync(`${__dirname}/templates/template-product.html`, 'utf-8');
37
38 const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
39 const dataObj = JSON.parse(data);
40
41 const server = http.createServer((req, res) => {
  const pathName = req.url;
42
43 // Overview page
44 if (pathName === '/' || pathName === '/overview') {
45
46   res.end('This is the OVERVIEW');
47
48 // Product page
49 } else if (pathName === '/product') {
50   res.end('This is the PRODUCT');
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
868
869
869
870
871
872
873
874
875
876
877
877
878
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
938
938
939
939
940
941
942
943
944
945
946
946
947
947
948
948
949
949
950
951
952
953
954
955
956
957
958
958
959
959
960
961
962
963
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
976
976
977
977
978
978
979
979
980
981
982
983
984
985
986
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1824
1825
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1834
1835
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1844
1845
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1854
1855
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1864
1865
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1874
1875
1875
1876
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1884
1885
1885
1886
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1895
1896
1896

```

```
50 | const pathname = req.url;
51 |
52 | // Overview page
53 | if (pathname === '/' || pathname === '/overview') {
54 |   res.writeHead(200, { 'Content-type': 'text/html'});
55 |
56 |   const cardsHtml = dataObj.map(el => replaceTemplate(tempCard, el)).join('');
57 |   const output = tempOverview.replace('{%PRODUCT_CARDS%}', cardsHtml);
58 |
59 |   res.end(output);
60 |
61 |   // Product page
62 | } else if (pathname === '/product') {
63 |   res.end('This is the PRODUCT');
64 |
65 |
66 | }
```





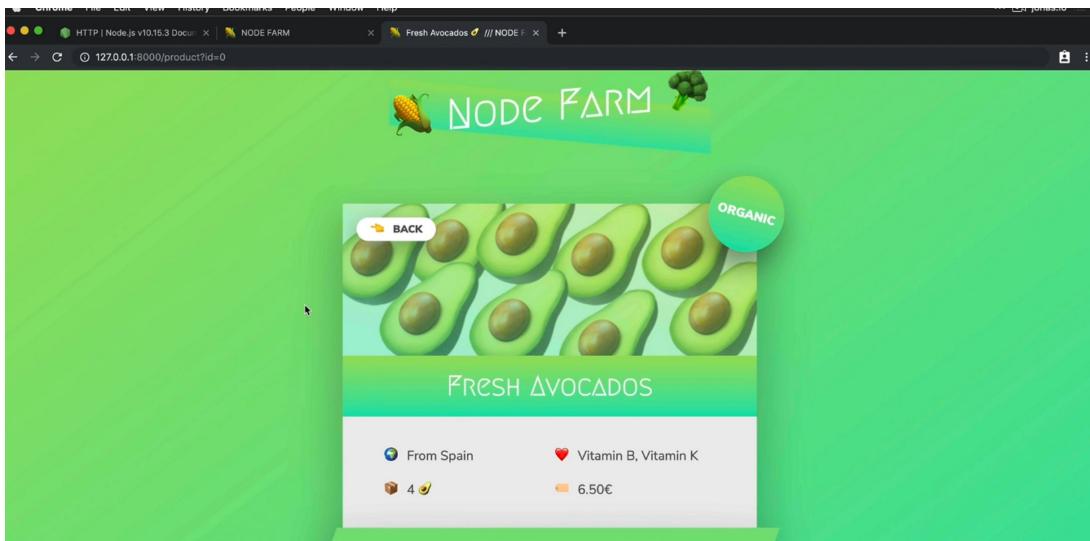
```
index.js — 1-node-farm
index.js x [] data.json

51
52 const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
53 const dataObj = JSON.parse(data);
54
55 const server = http.createServer((req, res) => {
56
57   console.log(req.url);
58   console.log(url.parse(req.url, true));
59   const pathName = req.url;
60
61   // Overview page
62   if (pathName === '/' || pathName === '/overview') {
63     res.writeHead(200, { 'Content-type': 'text/html'});
64
65     const cardsHTML = dataObj.map(el => replaceTemplate(tempCard, el)).join('');
66     const output = tempOverview.replace('{%PRODUCT_CARDS%}', cardsHTML);
67     res.end(output);
68
69   // Product page
70   } else if (pathName === '/product') {
71     res.end('This is the PRODUCT');
```



```
50 const cardsTemplate = fs.readFileSync(`${__dirname}/temp-views/cards.html`, 'utf-8');
51
52 const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
53 const dataObj = JSON.parse(data);
54
55 const server = http.createServer((req, res) => {
56
57   const { query, pathname } = url.parse(req.url, true); ←
58
59   // Overview page
60   if (pathname === '/' || pathname === '/overview') {
61     res.writeHead(200, { 'Content-type': 'text/html'});
62
63     const cardsHtml = dataObj.map(el => replaceTemplate(tempCard, el)).join('');
64     const output = tempOverview.replace('{%PRODUCT_CARDS%}', cardsHtml);
65     res.end(output);
66
67   // Product page
68   } else if (pathname === '/product') {
69     res.end('This is the PRODUCT');
```

```
66 // Product page
67 } else if (pathname === '/product') {
68     res.writeHead(200, { 'Content-type': 'text/html'});
69     const product = dataObj[query.id];
70     const output = replaceTemplate(tempProduct, product);
71     res.end(output);
72
73 // API
```



Okay, and so that is what we're gonna do in this lecture.

So let's say that we actually had
a bunch of different JavaScript files
in which we used this **replaceTemplate** function.

So right now we're just using it here in index.js.
We use it twice and that is why we have a function,
but imagine if we wanted to use this function in multiple files.
Okay, so what we can do is create a new module
and export that function from it
and then import it back here.

So, the first thing that you need to know
is that in Node.js, actually every single file is treated as a module.
And so this index.js here actually is also a module,
which in this case imports other modules—
and particularly these three.

The screenshot shows the VS Code interface with the following details:

- Left Sidebar (EXPLORER):** Shows files in the project: `index.js`, `dev-data`, `data.json`, `templates`, `template-card.html`, `template-overview.html`, `template-product.html`, `txt`, `append.txt`, `final.txt`, `input.txt`, `output.txt`, `read-this.txt`, and `start.txt`.
- Current File:** `index.js` is open in the editor.
- Annotations:** There are two red arrows pointing to the code block where `cardsHtml` is assigned. One arrow points to the assignment operator (`=`) and another points to the opening brace of the `map` loop. A red T-shaped mark is placed over the `const product = dataObj[query.id];` line.
- Bottom Status Bar:** Shows the terminal command `* jonas.io > 1-node-farm node index.js` and the message "Listening to requests on port 8000".

Cut this function from 'index.js' file

The screenshot shows a code editor interface with several tabs open. The left sidebar lists files and folders: **EXPLORER**, **OPEN EDITORS** (containing `index.js` and `replaceTemplate.js`), **1-NODE-FARM** (containing `dev-data`, `data.json`, `modules.js`, `replaceTemplate.js`, `templates` (with sub-items `template-card.html`, `template-overview.html`, `template-product.html`), and `txt` (with sub-items `append.txt`, `final.txt`, `input.txt`, `output.txt`, `read-this.txt`, `start.txt`), and `index.js`. The main editor area displays `index.js` and `replaceTemplate.js`. The `replaceTemplate.js` file contains a function to replace placeholders in a template string with values from a product object. The `index.js` file imports this function and uses it to process data from a JSON file, reading templates from sub-directories, and writing the results to files. The bottom status bar shows the terminal output: "jonas.io > 1-node-farm > node index.js Listening to requests on port 8000".

And paste it here.

CREATE A FOLDER

'module.js' -> replaceTemplate.js

Every node js module file has '**module**'

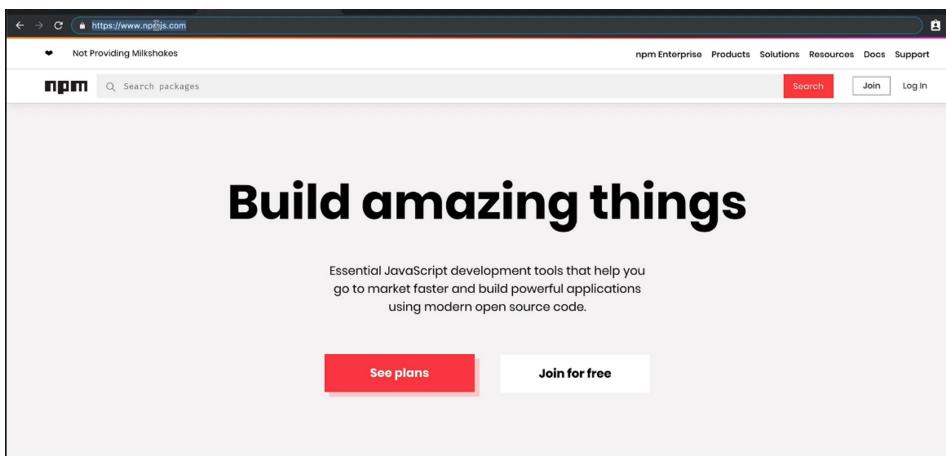
```

OPEN EDITORS
JS index.js
JS replaceTemplate.js ×
1 module.exports = (temp, product) => [
2   let output = temp.replace(/%PRODUCTNAME%/g, product.productName);
3   output = output.replace(/%IMAGE%/g, product.image);
4   output = output.replace(/%PRICE%/g, product.price);
5   output = output.replace(/%FROM%/g, product.from);
6   output = output.replace(/%NUTRIENTS%/g, product.nutrients);
7   output = output.replace(/%QUANTITY%/g, product.quantity);
8   output = output.replace(/%DESCRIPTION%/g, product.description);
9   output = output.replace(/%ID%/g, product.id);
10
11   if(!product.organic) output = output.replace(/%NOT_ORGANIC%/g, 'not-organic');
12   return output;
13 ]

```



Now that you know the basics of Node.js, let's talk about **npm** — the Node Package Manager. npm is a command-line tool that comes with Node.js. We use it to install and manage open-source packages. These packages usually come from the **npm repository**, which you can explore at [npmjs.com](https://www.npmjs.com). npm is both the tool we use in the terminal **and** the online registry that stores the packages. It's the **largest software registry** in the world. At the time of this recording, there are around **800,000+ packages**, and it's growing fast — soon hitting over a million. For example, let's search for **Express**, a Node.js framework we'll be using next.



The screenshot shows the npm search results for the package 'express'. At the top, it says '16964 packages found'. Below this, the 'express' package is listed with its version 4.16.4, status as 'Public', and a note that it was published 6 months ago by dougwilson. The package is described as a 'Fast, unopinionated, minimalist web framework'. It has 30 dependencies and 32,298 dependents. There are tabs for 'Readme', '30 Dependencies', '32,298 Dependents', and '261 Versions'. A red arrow points to the 'Search' button at the top right.

The screenshot shows the detailed page for the 'express' package. It features a large image of the word 'express'. Below it, there's a snippet of code showing how to use the module. To the right, there's information about weekly downloads (7,815,512), version (4.16.4), license (MIT), and links to open issues and pull requests. At the bottom, there are links to the homepage (expressjs.com) and repository (github).

Whenever we start a new project, to start with , we use npm as the command then init.

`npm init`

This will basically create a package.json file. Which is kind of configuration file of our project where all kinds of data about the project is stored.

The screenshot shows a terminal window within a code editor interface. The user has run the command `npm init`. The terminal output shows the following message:

```
jonas.io 1-node-farm$ 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 json' for definitive documentation on these fields
and exactly what they do.

Press ^C at any time to quit.
package name? (1-node-farm)
```

It will ask me some question.

First name is package name.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

save it as a dependency in the package.json file.

Press ^C at any time to quit.

```
package name: (1-node-farm) node-farm ↗
version: (1.0.0)
description: Learning node.js
entry point: (index.js)
test command:
git repository:
keywords:
author: Jonas Schmedtmann
license: (ISC)
About to write to /Users/jonas.io/Desktop/1-node-farm/package.json:
```

{
 "name": "node-farm",
 "version": "1.0.0",
 "description": "Learning node.js",
 "main": "index.js",
 "scripts": {
 "test": "echo \\\"Error: no test specified\\\" && exit 1"
 },
 "author": "Jonas Schmedtmann",
 "license": "ISC"
}



In this video, we'll look at **two types of npm packages** and **two ways to install them**. The two types are:

Dependencies – Packages that your app needs to run (like Express).

Dev Dependencies – Packages only needed during development (like testing tools). Let's install our first dependency: **Slugify**.

It's a small tool that turns names into URL-friendly slugs — perfect for things like product names in our Node Farm project.

```

1 {
2   "name": "node-farm",
3   "version": "1.0.0",
4   "description": "Learning node.js",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "author": "Jonas Schmedtmann",
10  "license": "ISC"
11 }
12

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io ➜ 1-node-farm ➔ npm install slugify

```

2 {
3   "name": "node-farm",
4   "version": "1.0.0",
5   "description": "Learning node.js",
6   "main": "index.js",
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1"
9   },
10  "author": "Jonas Schmedtmann",
11  "license": "ISC",
12  "dependencies": {
13    "slugify": "^1.3.4"
14  }
15

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io ➜ 1-node-farm ➔ npm install slugify
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN node-farm@1.0.0 No repository field.

Besides these regular dependencies, we also have development dependencies.

And these are usually tool for development. For example, a code bundler like WebPack, or a debugger tool or a testing library. So these are development dependencies. They are needed for production, so our codes does not really depend on them, we simply use them to develop our application.

--save-dev => defines it is a development dependencies.

Nodemon is a very nice tool that helps us develop NodeJS application whenever we change some files in our working directory.

```

13  }
14 }


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io ➜ 1-node-farm ➔ npm install nodemon --save-dev
[redacted] : extract:color-convert: sill extract color-convert@^1.9.0 extracted to /Users/jonas.io/Desktop/1-node-fa

You might wonder why we see so many packages when we only installed **Slugify** and **nodemon**. That's because these packages have their own dependencies.

For example, if **Slugify** needs another package, npm installs it too. That's how we end up with a long list of dependencies in our project.

```

{
  "name": "1-node-farm",
  "version": "0.0.1",
  "description": "Learning node.js",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "Jonas Schmedtmann",
  "license": "ISC",
  "dependencies": {
    "nodemon": "^1.18.11"
  }
}

```

node-pre-gyp WARN Using needle for node-pre-gyp https download
[fsevents] Success: "/Users/jonas.io/Desktop/1-node-farm/node_modules/fsevents/lib/binding/Release/node-v64-darwin-x64/fse.node" is installed via remote
> nodemon@1.18.11 postinstall /Users/jonas.io/Desktop/1-node-farm/node_modules/nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> <https://opencollective.com/nodemon/donate>

We installed the packages locally, so they only work in this project.
But with **npm**, we can also do **global installs**, which work anywhere on your machine.
Global installs are for tools that you run from the command line — like **nodemon**,
which I use in all my Node projects.

```

{
  "devDependencies": {
    "nodemon": "^1.18.11"
  }
}

```

jonas.io ➤ 1-node-farm ➤ npm i nodemon --global

nodemon node index.js

```

{
  "devDependencies": {
    "nodemon": "^1.18.11"
  }
}

```

jonas.io ➤ 1-node-farm ➤ node index.js → Listening to requests on port 8000 → X

^C
x jonas.io ➤ 1-node-farm ➤ nodemon index.js
[nodemon] 1.18.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node index.js`
Listening to requests on port 8000

What if we don't want to install nodemon globally. Just locally ?

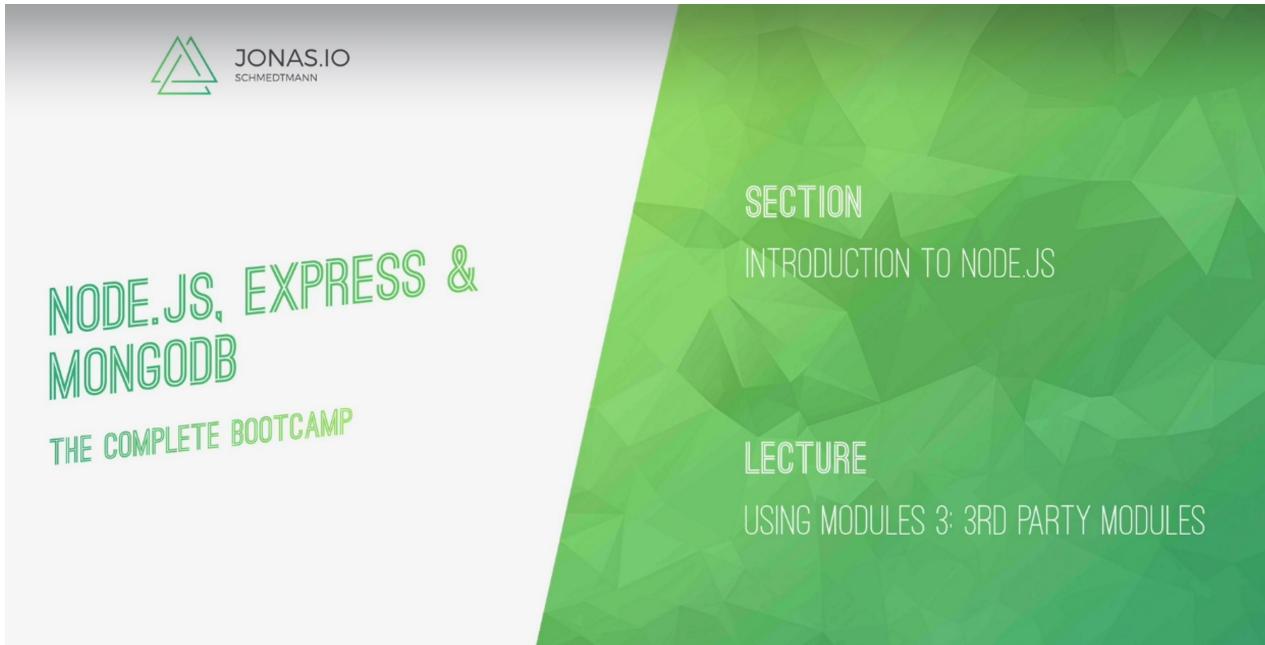
```

1  {
2    "name": "node-farm",
3    "version": "1.0.0",
4    "description": "Learning node.js",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js" ←
8    },
9    "author": "Jonas Schmedtmann",
10   "license": "ISC",
11   "dependencies": {
12     "slugify": "^1.3.4"
13   },
14   "devDependencies": {
15     "nodemon": "^1.18.11"
16   }
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
x jonas.io > 1-node-farm > npm run start ←
> node-farm@1.0.0 start /Users/jonas.io/Desktop/1-node-farm
> nodemon index.js
[nodemon] 1.18.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting 'node index.js'
listening to requests on port 8080.

Or 'npm start'



```

1  {
2    "name": "node-farm",
3    "version": "1.0.0",
4    "description": "Learning node.js",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js"
8    },
9    "author": "Jonas Schmedtmann",
10   "license": "ISC",
11   "dependencies": {
12     "slugify": "^1.3.4" ←
13   },
14   "devDependencies": {
15     "nodemon": "^1.18.11"
16   }
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
x node-farm > 1-node-farm > npm start
> node-farm@1.0.0 start /Users/jonas.io/Desktop/1-node-farm
> nodemon index.js

```

const fs = require('fs');
const http = require('http');
const url = require('url');
const slugify = require('slugify');
const replaceTemplate = require('./modules/replaceTemplate');

// FILES

// Blocking, synchronous way
const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
const textOut = `This is what we know about the avocado: ${textIn}. Created on ${Date()}`;
fs.writeFileSync('./txt/output.txt', textOut);
console.log('File written!');

// Non-blocking, asynchronous way
fs.readFile('./txt/start.txt', 'utf-8', (err, data) => {
  if (err) throw err;
  const template = data;
  const output = replaceTemplate(template, { productName: 'Avocado' });
  fs.writeFile('./txt/end.txt', output);
  console.log('File written!');
});

```

```

// SERVER
const tempOverview = fs.readFileSync(`${__dirname}/templates/template-overview.html`, 'utf-8');
const tempCard = fs.readFileSync(`${__dirname}/templates/template-card.html`, 'utf-8');
const tempProduct = fs.readFileSync(`${__dirname}/templates/template-product.html`, 'utf-8');
const data = fs.readFileSync(`${__dirname}/dev-data/data.json`, 'utf-8');
const dataObj = JSON.parse(data);

const slugs = dataObj.map(el => slugify(el.productName, { lower: true }));
console.log(slugs);

const server = http.createServer((req, res) => {
  const { query, pathname } = url.parse(req.url, true);

  // Overview page
  if (pathname === '/' || pathname === '/overview') {
    res.writeHead(200, { 'Content-type': 'text/html' });

    const cardsHtml = dataObj.map(el => replaceTemplate(tempCard, el)).join('');
    res.end(cardsHtml);
  }
  // Product page
  else if (pathname === `/product/${slugs[0]}`) {
    res.writeHead(200, { 'Content-type': 'text/html' });

    const product = dataObj.find(el => el.productName === slugs[0]);
    const cardHtml = replaceTemplate(tempProduct, product);
    res.end(cardHtml);
  }
});

server.listen(8000, () => {
  console.log(`Listening to requests on port 8000`);
});

```

[nodemon] starting `node index.js`
 fresh-avocados
 Listening to requests on port 8000
 [nodemon] restarting due to changes...
 [nodemon] starting `node index.js`
 ['fresh-avocados',
 'goat-and-sheep-cheese',
 'apollo-broccoli',
 'baby-carrots',
 'sweet-corncobs']
 Listening to requests on port 8000



```

index.js
1 {
  "name": "node-farm",
  "version": "1.0.0",
  "description": "Learning node.js",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "Jonas Schmedtmann",
  "license": "ISC",
  "dependencies": {
    "slugify": "^1.3.4" ←
  },
  "devDependencies": {
    "nodemon": "1.18.11" ←
  }
}

```

package.json — t-node-farm

We use **semantic versioning**:

- **MAJOR.MINOR.PATCH**
- **Patch** (last number): bug fixes only (e.g. 1.18 → 1.18.1 → 1.18.2...)
- **Minor** (middle): new, backward-compatible features (e.g. 1.18 → 1.19)
- **Major** (first): big releases that can break stuff (e.g. 1.x → 2.0)

So if you see Slugify go from 1.4 to 2.0, watch out—your code might need updates. 🚀

```

index.js
1 {
  "name": "node-farm",
  "version": "1.0.0",
  "description": "Learning node.js",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "Jonas Schmedtmann",
  "license": "ISC",
  "dependencies": {
    "slugify": "^1.3.4" ←
  },
  "devDependencies": {
    "nodemon": "1.18.11" ←
  }
}

```

package.json — t-node-farm

$\wedge \Rightarrow$

Because of that, it is also important to talk about updating packages.
In our package.json file, this small symbol (\wedge) here that comes in front of the version number is what specifies which updates we accept for each of the packages. This symbol here, which npm specifies here by default means that we accept patch and minor releases.

How to check if there are any outdated packages?

npm outdated

Right now they are all up to date.

```

{
  "name": "node-farm",
  "version": "1.0.0",
  "description": "Learning node.js",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "Jonas Schmedtmann",
  "license": "ISC",
  "dependencies": {
    "slugify": "^1.3.4"
  },
  "devDependencies": {
    "nodemon": "^1.18.11"
  }
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
x jonas.io > 1-node-farm > npm outdated
jonas.io > 1-node-farm > npm install slugify@1.0.0
((...)) : rollbackFailedOptional: verb npm-session aa79588d58b1af1f

Let's now see how to install a certain package with a certain version number?

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
x jonas.io > 1-node-farm > npm outdated
jonas.io > 1-node-farm > npm install slugify@1.0.0
((...)) : rollbackFailedOptional: verb npm-session aa79588d58b1af1f

If we now run

`npm outdated`

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io > 1-node-farm > npm outdated
Package Current Wanted Latest Location
slugify 1.0.0 1.3.4 1.3.4 node-farm
x jonas.io > 1-node-farm

\wedge => we accept all the (patch+minor) releases

\sim => we accept (patch) releases

Change in the 'package.json' file and run 'npm outdated' again.

The screenshot shows the VS Code interface. In the top left, there's a file tree with files like index.js, package.json, and node-farm.html. The main editor area shows package.json with the following content:

```

1  {
2   "name": "node-farm",
3   "version": "1.0.0",
4   "description": "Learning node.js",
5   "main": "index.js",
6   "scripts": {
7     "start": "nodemon index.js"
8   },
9   "author": "Jonas Schmedtmann",
10  "license": "ISC",
11  "dependencies": {
12    "slugify": "~1.0.0"
13  },
14  "devDependencies": {
15    "nodemon": "^1.18.11"
16  }
17 }
18

```

A red arrow points to the "slugify" entry in the dependencies section. Below the editor is the terminal window with the following output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jonas.io > 1-node-farm npm outdated
Package Current Wanted Latest Location
slugify 1.0.0 1.3.4 1.3.4 node-farm
x jonas.io > 1-node-farm npm outdated
Package Current Wanted Latest Location
slugify 1.0.0 1.0.2 1.3.4 node-farm
x jonas.io > 1-node-farm

```

A red arrow points to the "slugify" entry in the devDependencies section of the terminal output.

See the 'wanted' column. It's 1.0.2 version that is accepted.

Run 'npm update slugify' command.

The terminal shows the command being run and its output:

```

slugify 1.0.0 1.3.4 1.3.4 node-farm
x jonas.io > 1-node-farm npm outdated
Package Current Wanted Latest Location
slugify 1.0.0 1.0.2 1.3.4 node-farm
x jonas.io > 1-node-farm npm update slugify
npm WARN node-farm@1.0.0 No repository field.

+ slugify@1.0.2
updated 1 package and audited 2237 packages in 2.698s
found 0 vulnerabilities

```

A red arrow points to the "WARN" message about the repository field.

The screenshot shows the VS Code interface with the same package.json content as before. A red arrow points to the "slugify": "1.0.2" entry. A tooltip appears over the "1.0.2" value, providing information about the package:

Slugifies a String
Latest version: 1.3.4
[https://github.com/simov\(slugify](https://github.com/simov(slugify)

If there is "*" and we update package that means it will be updated to the latest version.

```
4 "description": "Learning node.js",
5 "main": "index.js",
6 "scripts": {
7   "start": "nodemon index.js"
8 },
9 "author": "Jo", Slugifies a String
10 "license": "I", Latest version: 1.3.4
11 "dependencies": "https://github.com/simov/slugify"
12   "slugify": "1.3.4"
13 },
14 "devDependencies": {
15   "nodemon": "1.18.11"
16 }
17 }
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
jonas.io ➜ 1-node-farm ➔ npm update slugify
npm WARN node-farm@1.0.0 No repository field.
```

Safest option is to use ~

We can also delete packages.

Lets now install 'express' so we can delete

```
jonas.io ➜ 1-node-farm ➔ npm i express
npm WARN node-farm@1.0.0 No repository field.

+ express@4.16.4
updated 1 package and audited 2358 packages in 1.899s
found 0 vulnerabilities

jonas.io ➜ 1-node-farm ➔ npm uninstall express
```

NEVER SHARE 'NODE_MODULES' FOLDER TO GITHUB, OR ANY WHERE.

You can get it by pressing 'npm i'

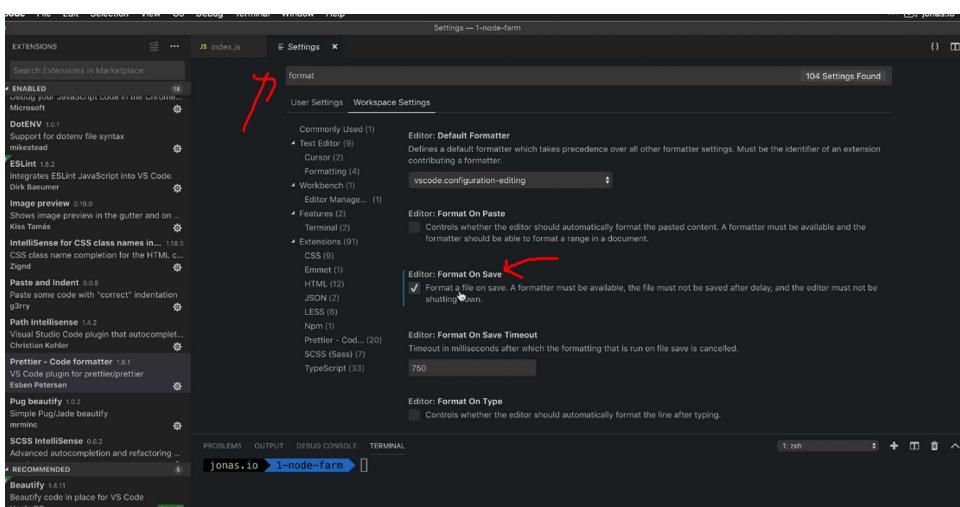
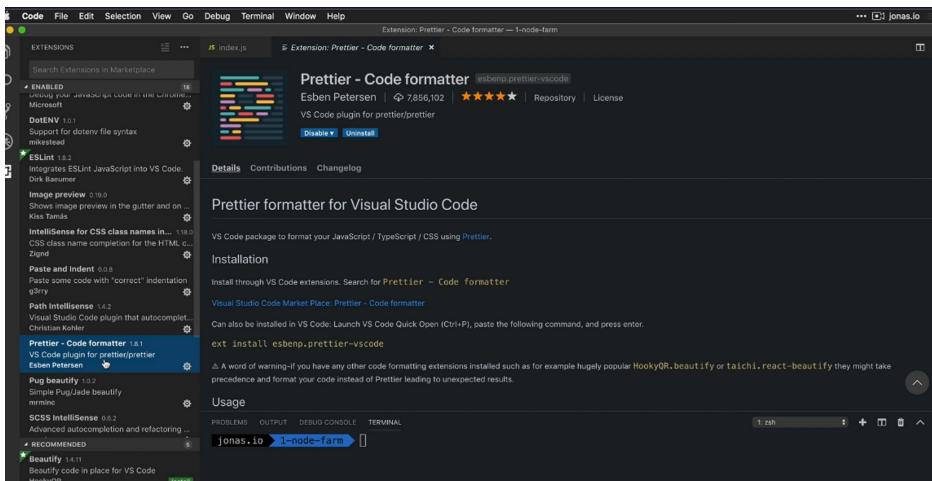
Basically it reads our 'package.json' file and install corresponding dependencies.

Now one important piece of this puzzle
is the package-lock.json file.

If we open that up, we get a list of
all the versions of all the packages
that we're actually using.

That includes the dependencies of our dependencies.
Let's go to Slugify, for example.
Slugify, and so here we see that
we're using version 1.3.4.

That is very important because
if you share your code, you want the other developer
or even yourself to be using
the exact same package versions.



Now we can configure prettier now. Let's define a configuration file.

We also can do in the vs code. But we prefer to create a separate file.

(Helpful for working in a team)

A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with various files and folders, including a .prettierrc file under the '1-NODE-FArm' folder. The main editor area displays a .prettierrc configuration file with the following content:

```
1 {
2   "singleQuote": true,
3   "printWidth": 120
4 }
```

The terminal at the bottom shows the command `jonas.io ➜ 1-node-farm`.

