NodeJS

INTRODUCTION: WHAT IS NODEJS

NodeJS (or just 'Node') has been steadily gaining popularity since its creation in 2009. The internet is flooded with courses and articles about it, installing it is a prerequisite for pretty much any front-end development work, and of course the amount of jobs that require knowledge of it are also on the rise.

Learning Outcomes

By the end of this lesson, you should be able to do the following:

- Describe the purpose of a server.
- Describe the differences between static and dynamic sites.
- Explain why you might need a back-end for your project.
- Explain when you wouldn't need a back-end for a project.
- Explain the event loop.
- Understand the origin of the Node.js runtime.
- Write a simple "hello world" application and run it in the console of your machine.
- Understand what Node.js really is.

What *is* Node?

The Node.js website declares:

> "As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications."

This is a definition that requires a little unpacking.

The important bit to understand right up front is that Node is a "JavaScript runtime". When JavaScript was first created, it was designed to run *in the browser*. This means that it was impossible to use JavaScript to write any kind of program that was not a website. Node brings JavaScript *out* of browser-land. This allows developers to use JavaScript to accomplish pretty much anything that other popular server-side languages such as Ruby, PHP, C# and Python can do. So, at its most basic level, Node simply allows you to run JavaScript code on a machine such as your local computer or a server without having to go through a web browser.

To facilitate this, Node has some added functionality that is not found in browser-based JavaScript, such as t. ability to read and write local files, create http connections and listen to network requests

ability to read and write local files, create http connections and listen to network requests.

Event Driven

Back to the definition from Node's website: Node is an **asynchronous event driven** JavaScript runtime. In this context **asynchronous** means that when you write your code you do not try to predict the exact sequence in which every line will run. Instead you write your code as a collection of smaller functions that get called in response to specific events such as a network request (**event driven**).

For example, let's say you are writing a program and you need it to do the following. It should read some text from a file, print that text to the console, query a database for a list of users and filter the users based on their age.

Instead of telling your code to do those steps sequentially like so:

1. Read File
2. Print File Contents
3. Query Database
4. Filter Database Query results

You can break up the task like so:

1. Read File *AND THEN* Print File Contents
2. Query Database *AND THEN* Filter Database Query Results.

When you run this program Node will start at the top and begin reading the file but since that is an action that takes some time it will immediately begin running the second step (querying the database) while it's waiting on the file to finish reading.

While both of these processes are running, Node sits and waits on an *event*. In this case, it is waiting on the completion of both processes, the reading of a file and the database query. When either of these tasks are finished, Node will fire off an event that will run the next function we've defined. So if the read-file process finishes first, it will print the file contents. If the database query finishes first, it will start the filtering process. As the programmer, we don't know or care which order the two processes are going to be completed. If this code was processed synchronously (rather than asynchronously) we would have to wait for each step in the program before moving on to the next one, which could cause things to slow down considerably. If the file that we needed to read was really long then we might have to wait a few seconds before the database query could begin.

This process is almost exactly like the way that you would use `addEventListener` in front-end JavaScript to wait for a user action such as a mouse-click or keyboard press. The main difference is that the events are going to be things such as network requests and database queries. This functionality is facilitated through the use of callbacks. Callbacks are incredibly important to Node, so take a minute to read through this article to make sure you're up to speed.

Let's look at a quick real-world example:

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

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

This snippet is from the very first lesson in a tutorial that you'll be following very soon. Basically this code is creating a server and saying, "any time we get a network request, run this callback function". This function happens to respond with the text 'Hello World!'. So if you go to a browser and navigate to the correct address and port, you would see that text on your screen.

A word of advice

While you may have learned React (or any other frontend framework) before, either of your own volition or earlier in the path, it is not recommended to use it for this course right away. There are many topics that you must learn before you can combine these frameworks effectively. As you move forward through the Node course, you will learn more about how to integrate Node APIs with frontend frameworks. You should follow the course as it is written; deviating from the directions can make it more difficult than it needs to be. Your time spent learning those frameworks will not be wasted, don't worry!

Assignment

1. This short module on "The Server Side" from MDN is a great source for the background knowledge you need. Read through at least the first two articles posted under the 'Guides' section: Introduction to the server side and Client-Server Overview. The other two are interesting and worth reviewing, but less relevant to our immediate concerns.
2. To gain a little more insight into the nature of Node, and to unpack the rest of the above definition, read this article. There's a long, but *really* fantastic video linked in that article, don't skip it!
3. Take a minute or two to browse through the main Node.js website. The getting-started article is a little *too* basic for our needs, and at this point the API documentation is probably a little too complicated for us to make use of just yet. But it will be very useful to become familiar with this resource as we progress.
4. This short video is a great introduction as well!

Additional Resources

This section contains helpful links to other content. It isn't required, so consider it supplemental for if you need to dive deeper into something.

- Read this article on 7 awesome things you can build with Node.js.

Knowledge Check

This section contains questions for you to check your understanding of this lesson. If you're having trouble answering the questions below on your own, clicking the small arrow to the left of the question will reveal th answers.

▶ What is Node?

View Course

Login to track progress

Next Lesson

Improve this lesson on GitHub

Have a question?

Chat with our friendly Odin community in our Discord chatrooms!

Open Discord

THE ODIN PROJECT