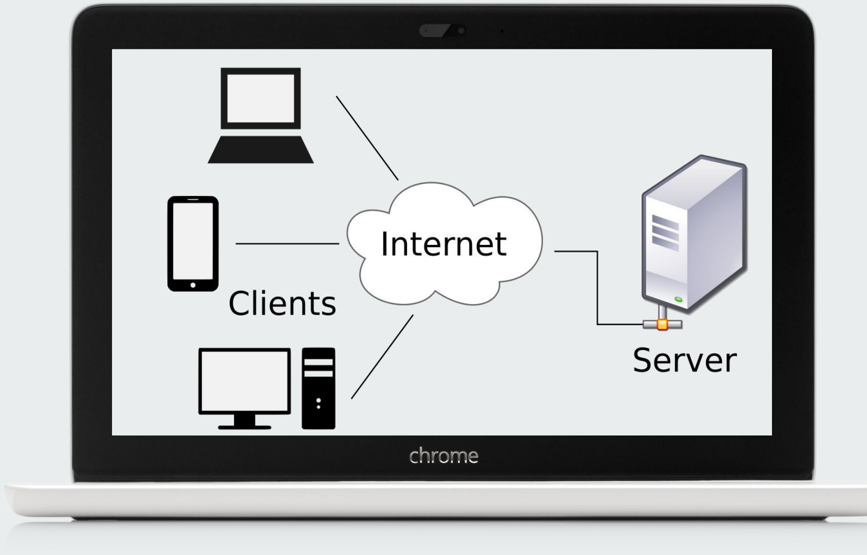
Introduction to Backend Development

With Node.js and Express.js



Prerequisites

Node.js:
https://nodejs.org/en/download/

Postman:
https://www.getpostman.com/downloads/

Outline

What is Backend Development?

What is Node.js and where it fits in?

Some JavaScript fundamentals

Your first server in Node.js

Why do we need Express.js?

Asynchronous nature of Node.js

What next?

What is Backend Development?

Let’s start with what is frontend and what is backend.

Frontend: The part of a website that user interacts with directly.

Backend: It is server side of the website. It is the part of the website that you cannot see and interact with.

Raises more questions:

- What is a server?
- If we cannot see or interact with it, then why is it required?
- Why is backend development required?

We'll answer these questions by understanding how the Internet works.

briefly

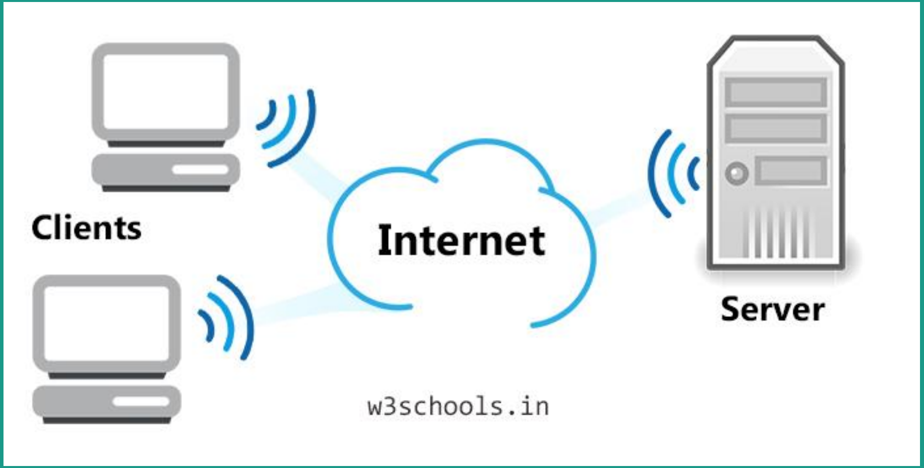
What happens when you visit a website like google.com **or** facebook.com**?**

Your browser makes a request over the internet requesting the google **or** facebook **homepage.**

Who or *what* is listening to these requests and responding with the correct information?

It is the servers that listen to the *requests* and *respond* to the clients ie your browser.

This is called the client-server architecture.



Let’s look at a formal definition on MDN web docs.

Link

Revisiting

- What is a server?
- If we cannot see or interact with it, then why is it required?
- Why is backend development required?

How to Backend Development?

There are many options. MANY.

Flask

RAILS

Java™

django

spring
BOOT

php

Perl

Dart

Laravel 6

GO

node JS™

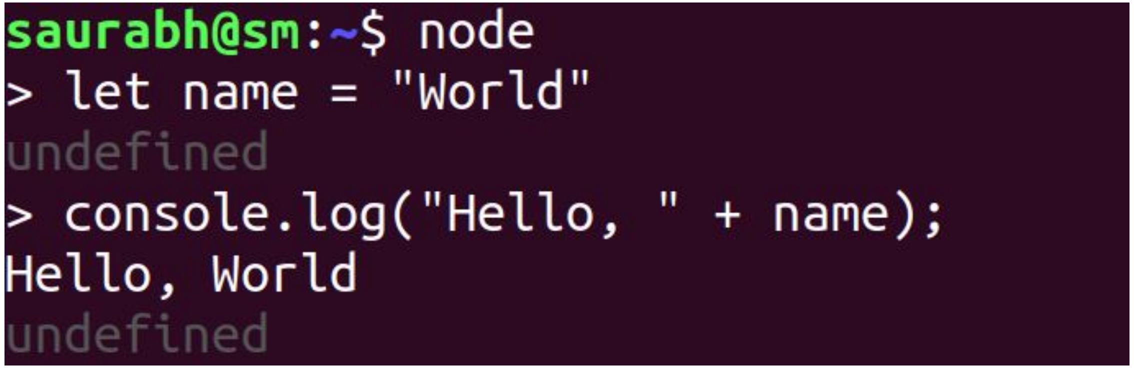
What is Node.js?

Is it a library? No

Is it a framework? No

Node.js is a JavaScript runtime.

Node can run any valid JavaScript which is usually executed on a browser.

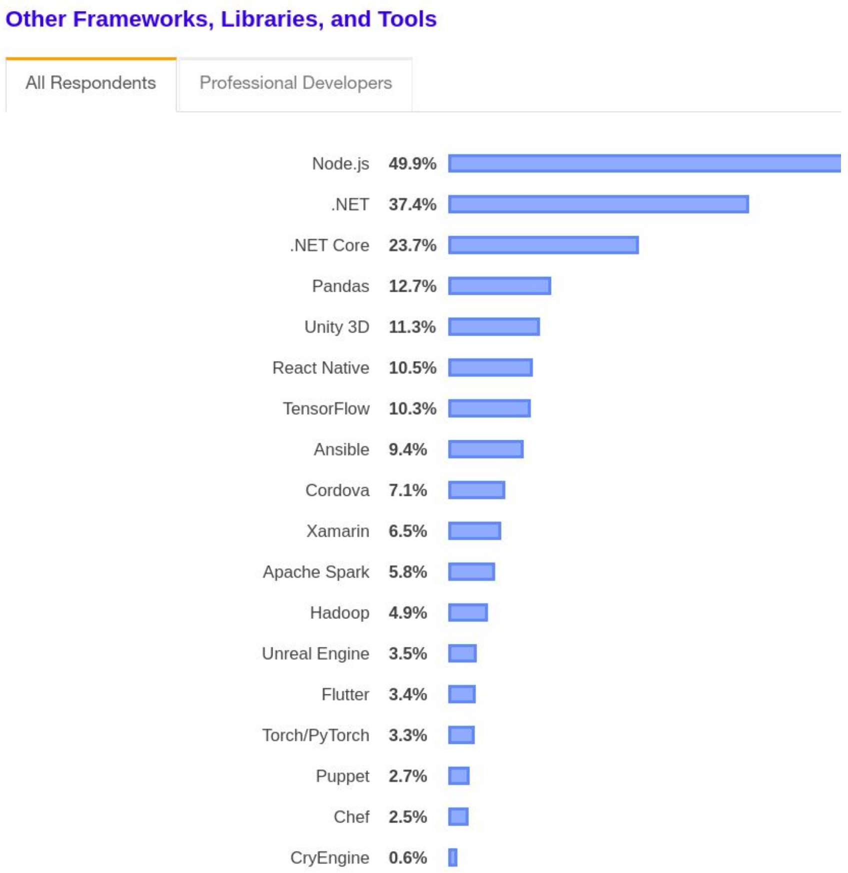


```
saurabh@sm:~$ node
> let name = "World"
undefined
> console.log("Hello, " + name);
Hello, World
undefined
```

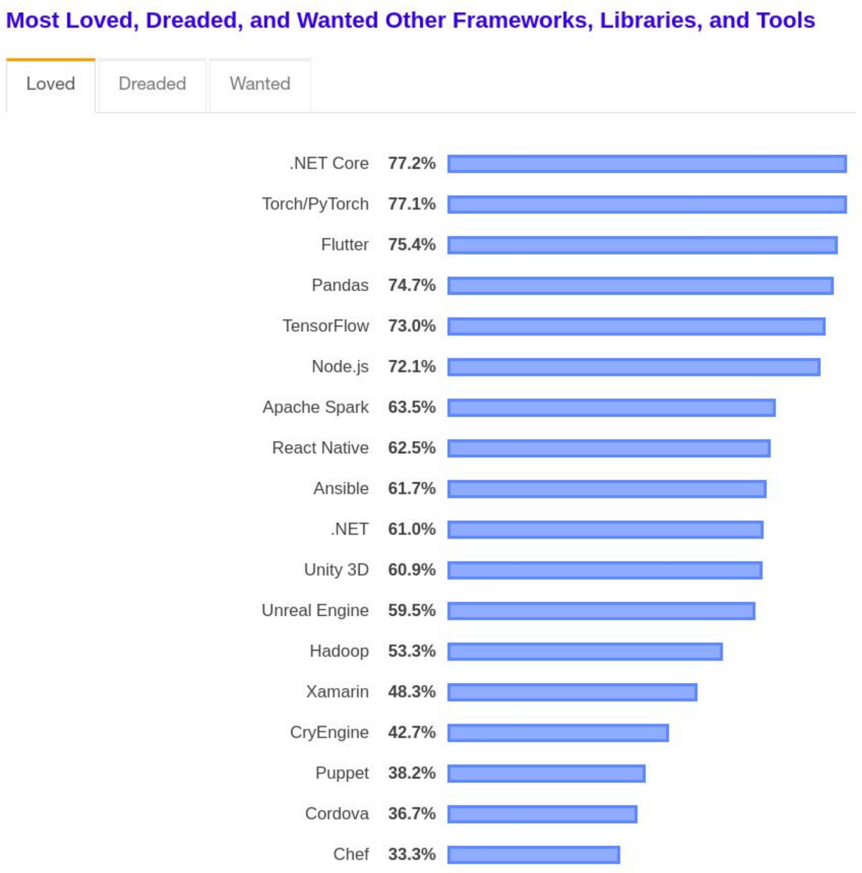
Node.js is built on Chrome’s V8 engine

V8 is the JavaScript execution engine which was initially built for Google Chrome. It was then open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code during runtime.

In **Stack Overflow Developer Survey 2019**, Node.js was the most popular technology and the 6th most loved technology



Node.js	49.9%
.NET	37.4%
.NET Core	23.7%
Pandas	12.7%
Unity 3D	11.3%
React Native	10.5%
TensorFlow	10.3%
Ansible	9.4%
Cordova	7.1%
Xamarin	6.5%
Apache Spark	5.8%
Hadoop	4.9%
Unreal Engine	3.5%
Flutter	3.4%
Torch/PyTorch	3.3%
Puppet	2.7%
Chef	2.5%
CryEngine	0.6%



.NET Core	77.2%
Torch/PyTorch	77.1%
Flutter	75.4%
Pandas	74.7%
TensorFlow	73.0%
Node.js	72.1%
Apache Spark	63.5%
React Native	62.5%
Ansible	61.7%
.NET	61.0%
Unity 3D	60.9%
Unreal Engine	59.5%
Hadoop	53.3%
Xamarin	48.3%
CryEngine	42.7%
Puppet	38.2%
Cordova	36.7%
Chef	33.3%

Why Node.js?

Because it is fast

Node uses Chrome’s V8 engine. It is used to compile functions written in JavaScript into machine code, and it does the job at an impressive speed.

Because it is scalable

Since Node is a lightweight technology, using Node.js for microservices architecture is a great choice.

This architectural style is best described as “an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms”.

Because it has an active community

Node.js has a large and active community of developers who keep on continuously contributing towards its further development and improvement.

Because it has a Rich Ecosystem

One word – **npm**, a default Node.js package manager, it also serves as a marketplace for open source JavaScript tools. There are about 836,000 libraries available in the npm registry as of now, and over 10,000 new ones being published every week.

Because it’s JavaScript

Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language.

Also it’s easy to learn

Before getting into Node.js, let’s cover some JavaScript concepts.

- *var* vs *let* vs *const*
- Destructuring
- Callbacks
- Promises
- Async Await

var vs **let** vs **const**

These keywords are used to declare variables but there are differences between them.

var	**let**
function scoped	block scoped
undefined when accessing a variable before it's declared	ReferenceError when accessing a variable before it's declared

Variables declared using **const** can't be reassigned.

Destructuring

It’s a common way to cleanly extract properties from objects.

```
const name = {
  first: 'Harry',
  second: 'Potter'
}
const { first, second } = name;
```

We can also extract properties under a different name.

```
const name = {
  first: 'Harry',
  second: 'Potter'
}
const { first: fname, second: lname } = name;
```

Callback Functions

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```javascript
function greeting(name) {
  alert('Hello ' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting);
```

```
const verifyUser = function(username, password, callback){
   dataBase.verifyUser(username, password, (error, userInfo) => {
       if (error) { callback(error) }
       else{
           dataBase.getRoles(username, (error, roles) => {
               if (error){ callback(error) }
               else {
                   dataBase.logAccess(username, (error) => {
                       if (error){ callback(error) }
                       else{
                           callback(null, userInfo, roles);
                       }
                   })
               }
           })
       }
   })
};
```

This is callback hell

Promises

The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

```
var prms = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});

proms.then(function(value) {
  console.log(value);
});
```

```
const verifyUser = function(username, password) {
   database.verifyUser(username, password)
       .then(userInfo => dataBase.getRoles(userInfo))
       .then(rolesInfo => dataBase.logAccess(rolesInfo))
       .then(finalResult => {
           //do whatever the 'callback' would do
       })
       .catch((err) => {
           //do whatever the error handler needs
       });
};
```

Promises help in avoiding the callback hell

Async Await

Async Await is just “syntactic sugar” on top of promises. This makes asynchronous code easier to write and to read afterwards. They make async code look more like old-school synchronous code

```
var prms = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});


async function init() {
    const result = await prms;
    console.log(result)
}
```

```
const verifyUser = async function(username, password) {
    try {
        const userInfo = await database.verifyUser(username, password)
        const rolesInfo = await dataBase.getRoles(userInfo)
        const finalResult = await dataBase.logAccess(rolesInfo))
        //do whatever the 'callback' would do
    } catch(error) {
       //do whatever the error handler needs
    }
};
```

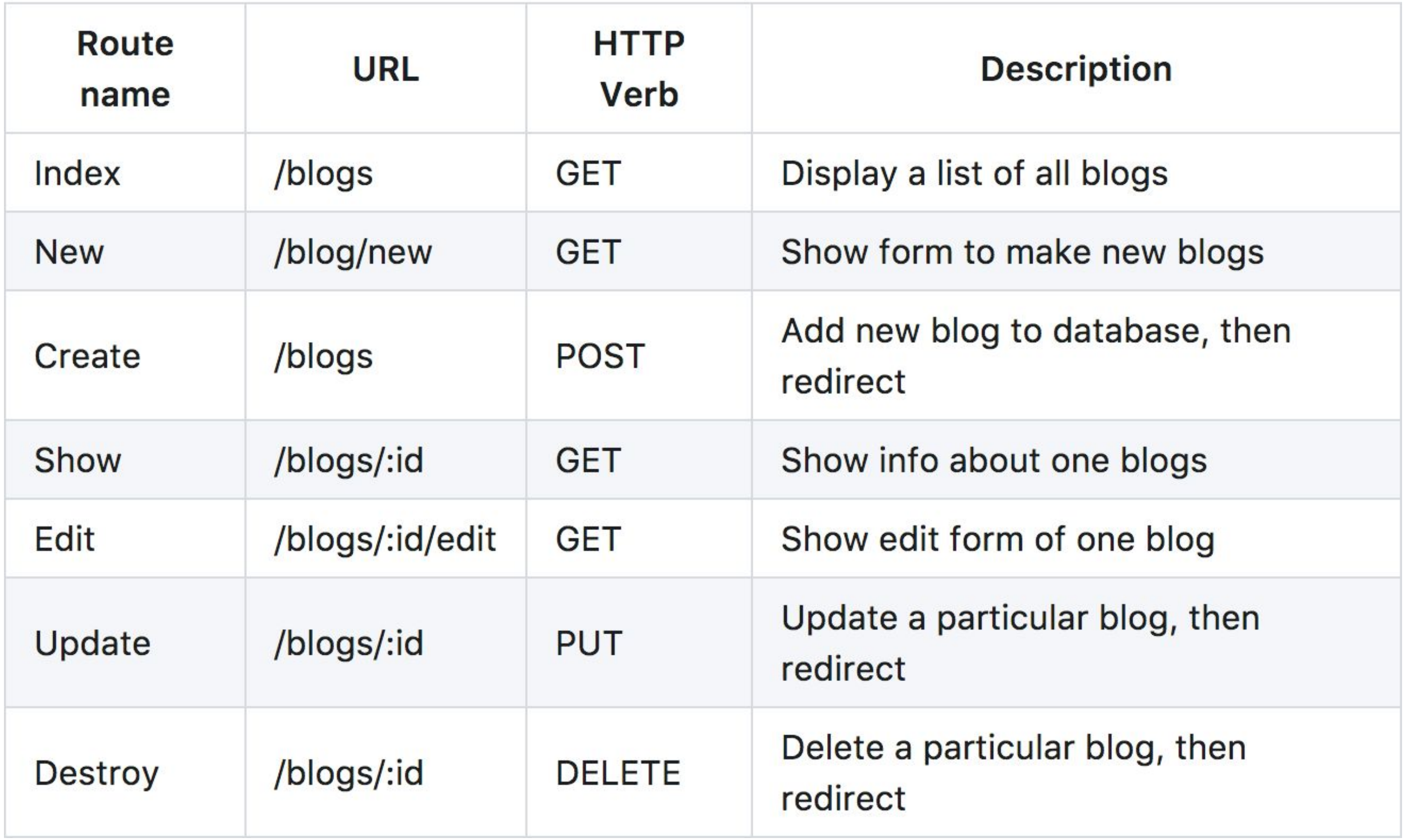
Async Await makes asynchronous code cleaner and look more like synchronous code.

Your First Server in Node.js

HTTP Request Methods

- **GET**: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- **POST**: The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- **PUT**: The PUT method replaces all current representations of the target resource with the request payload.
- **DELETE**: The DELETE method deletes the specified resource.

RESTful Routing



Route name	URL	HTTP Verb	Description
Index	/blogs	GET	Display a list of all blogs
New	/blog/new	GET	Show form to make new blogs
Create	/blogs	POST	Add new blog to database, then redirect
Show	/blogs/:id	GET	Show info about one blogs
Edit	/blogs/:id/edit	GET	Show edit form of one blog
Update	/blogs/:id	PUT	Update a particular blog, then redirect
Destroy	/blogs/:id	DELETE	Delete a particular blog, then redirect

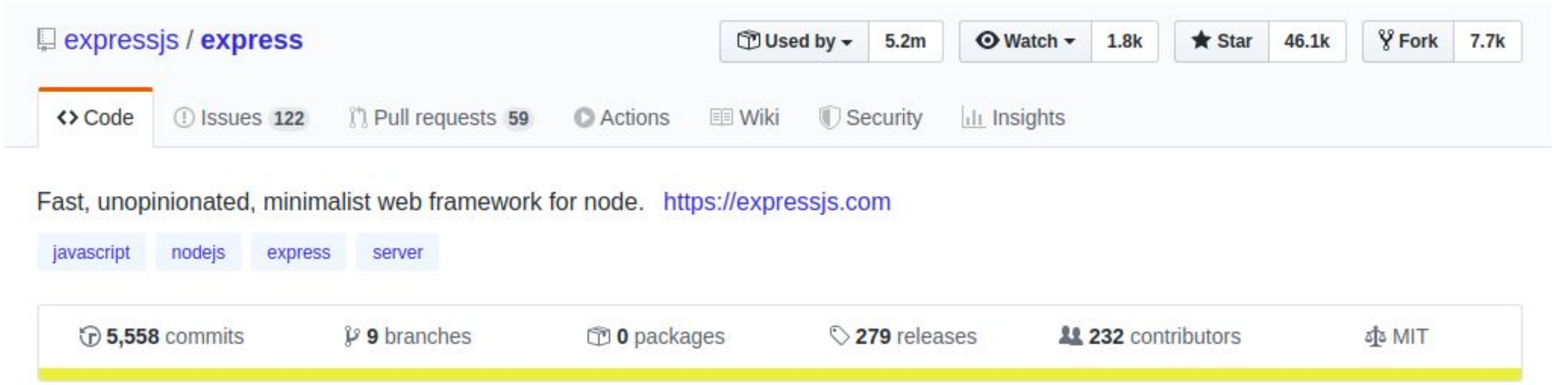
Demo

Simple HTTP Server

What is Express.js?

Express.js

ExpressJS is a prebuilt Node.js framework that can help you in creating server-side web applications faster and smarter.


expressjs / express

Used by 5.2m | Watch 1.8k | Star 46.1k | Fork 7.7k

Code | Issues 122 | Pull requests 59 | Actions | Wiki | Security | Insights

Fast, unopinionated, minimalist web framework for node. https://expressjs.com

javascript nodejs express server

5,558 commits | 9 branches | 0 packages | 279 releases | 232 contributors | MIT

Why do we need Express.js?

- Express.js is simple, minimal, flexible and scalable and since it is made in Node.js itself, it inherited its performance as well.

- In short, Express.js did for Node.js what Bootstrap did for HTML/CSS and responsive web design.

- It made coding in Node.js a piece of cake and gave programmers some additional features to extend their server-side coding.

Demo

Basic Express API

Let’s start by generating some fake data.

Asynchronous and Non-Blocking nature of Node.js

- Asynchronous (or async) execution refers to execution that doesn’t run in the sequence it appears in the code.
- In async programming the program doesn’t wait for the task to complete and can move on to the next task.
- Blocking refers to operations that block further execution until that operation finishes while non-blocking refers to code that doesn’t block execution.

Demo

Let’s see this in the demo by making request to a third party API

What next?

is.gd/K0Lo3S

Thanks

Github

Facebook

Instagram

Website