<https://www.geeksforgeeks.org/how-to-deploy-node-js-app-on-heroku-from-github/>

--------------------------------------------------------

**Node.js Introduction**

--------------------------------------------------------

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. ; open-source and cross-platform JavaScript runtime environment which allows backend programming through JavaScript

1. Easy—Node.js is quite easy to start with. It’s a go-to choice for web development beginners. With a lot of tutorials and a large community—getting started is very easy.
2. Scalable—It provides vast scalability for applications. Node.js, being single-threaded, is capable of handling a huge number of simultaneous connections with high throughput.
3. Speed—Non-blocking thread execution makes Node.js even faster and more efficient.
4. Packages—A vast set of open-source Node.js packages is available that can simplify your work. There are more than one million packages in the NPM ecosystem today.
5. Strong backend—Node.js is written in C and C++, which makes it speedy and adds features like networking support.
6. Multi-platform—Cross-platform support allows you to create SaaS websites, desktop apps, and even mobile apps, all using Node.js.
7. Maintainable—Node.js is an easy choice for developers since both the frontend and backend can be managed with JavaScript as a single language.

**Install Node.js on Mac without admin privileges:**

<https://medium.com/javascript-first/how-to-install-npm-nvm-on-mac-without-admin-sudo-or-homebrew-cef757275bbb>

<https://blog.teamtreehouse.com/install-node-js-npm-mac> + <https://superuser.com/questions/619498/can-i-install-homebrew-without-sudo-privileges>

demo.js:

| let name = 'Pesho';  console.log(`My name is ${name}`); |
| --- |

- >

| [11:04:43] donetianpetkov@Donetian-Petkov-NEWs-MacBook-Pro [ ~/WebstormProjects/1-intro-to-nodejs ] $ node demo.js  My name is Pesho |
| --- |

**Initialize a node.js project** - through Node Package Manager - npm:

| npm init -y |
| --- |

**package.json** - the file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies. ; it is used as a manifest, storing information about applications, modules, packages, and more. ; It is a JSON file that lives in the root directory of your project. Your package.json holds important information about the project. It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number and a list of dependencies required by the application. ; which node versions to be used, the scripts which can be executed on the project, the repository of the project and etc:  
  
<https://nodesource.com/blog/the-basics-of-package-json>

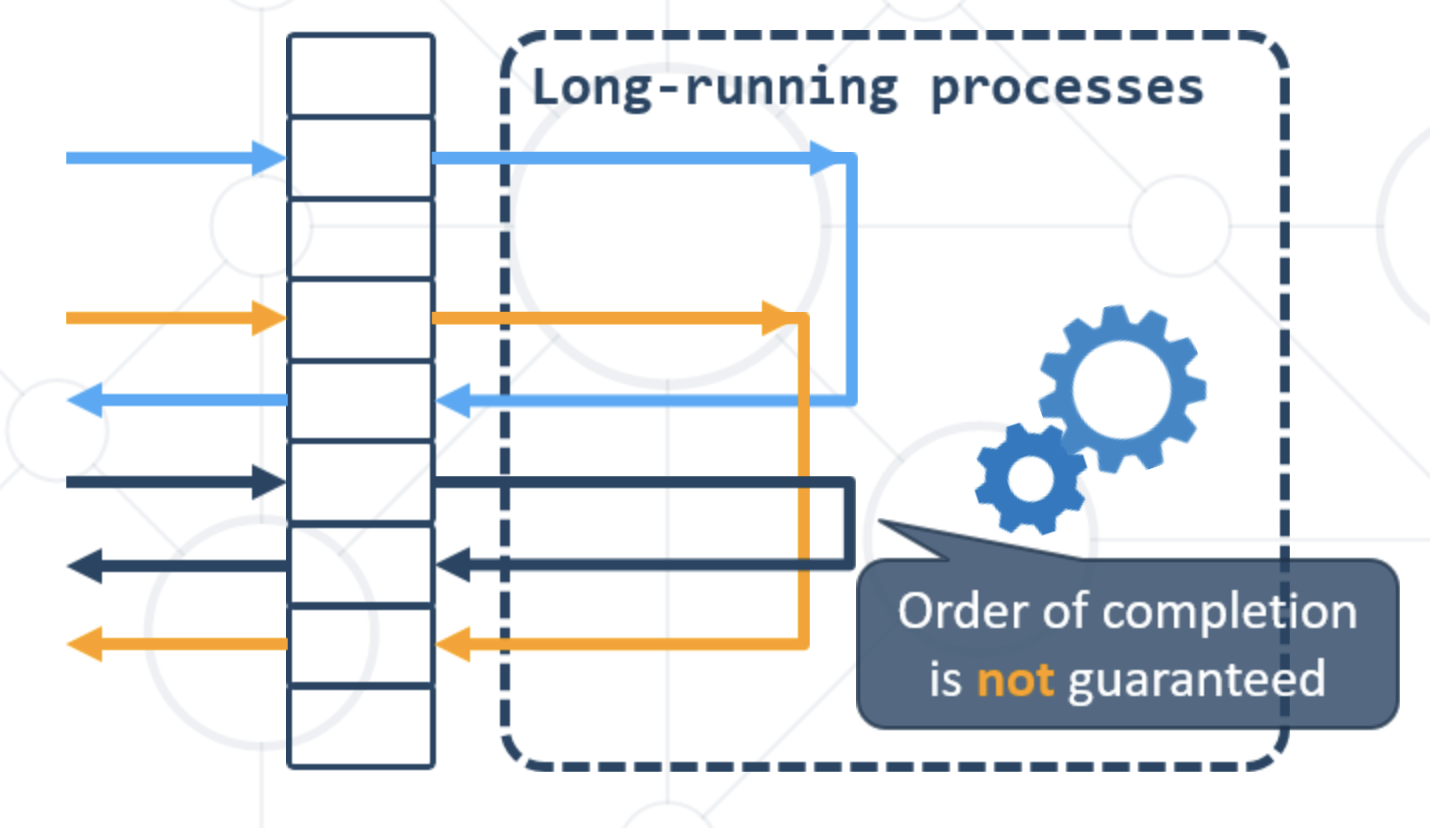
--------------------------------------------------------

**Event Loop**

--------------------------------------------------------

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible. ; an event-listener which functions inside the NodeJS environment and is always ready to listen, process, and output for an event. ;

1. Event loop is an endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks.
2. The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
3. The event loop allows us to use callbacks and promises.
4. The event loop executes the tasks starting from the oldest first.



<https://blog.risingstack.com/node-js-at-scale-understanding-node-js-event-loop/> - explanation with the call stack

!!!!

| function func() {  console.log('start');    setTimeout(() => {  console.log('executed');  }, 0)    console.log('end'); }  func(); |
| --- |

- >  
  
start

end

executed

due to:  
  
**The event loop executes tasks from the event queue only when the call stack is empty**

1) console.log(‘start’);  
  
2) console.log('executed'); is registered in the Event Callback  
  
3) console.log(‘end’)  
  
4) the call stack is empty, so the console.log('executed') is executed !!!!

--------------------------------------------------------

**Modules**

--------------------------------------------------------

In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality. Modules can be a single file or a collection of multiples files/folders. The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks. ;

* Allow larger apps to be split and organized
* Each module has its own context
* It cannot pollute the global scope

**Node.js includes three types of modules**

**Core Modules**: Node.js has many built-in modules that are part of the platform and comes with Node.js installation. These modules can be loaded into the program by using the require function.

**Local Modules**: Unlike built-in and external modules, local modules are created locally in your Node.js application

Two ways of importing modules:

* **common.js**: parses automatically JSON to JavaScript objects

**1) default exporting (in the other module it can be calc or anything else we want):**

calc.js:

| function calc(a,b) {  return a + b; }  module.exports = calc; |
| --- |

calc-application.js:

| const calculator = require('./calc')  function init() {  let result = calculator(2,3);  console.log(result); } |
| --- |

**2) named export (it can only be calc):**

| function calc(a,b) {  return a + b; }  exports.calc = calc; |
| --- |

| const { calc } = require('./calc')  function init() {  let result = calc(2,3);  console.log(result); } |
| --- |

<https://nodejs.org/api/modules.html>

* **es6 modules:** does not parse automatically JSON to JavaScript objects

| function calc(a,b) {  return a + b; }  export { calc } |
| --- |

| import { calc} from "./calc";  function init() {  let result = calc(2,3);  console.log(result); } |
| --- |

<https://nodejs.org/api/esm.html>

**Third-Party Modules**: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react. <https://www.npmjs.com/>

**Example Common Module:**

<https://nodejs.org/docs/latest-v16.x/api/url.html>

**url:**

**const url = require('url');**

The url module provides utilities for URL resolution and parsing

| const url = require('url');  let softUniURL = url.parse('https://softuni.bg/trainings/3473/angular-july-2021');  console.log(softUniURL); |
| --- |

| Url {  protocol: 'https:',  slashes: true,  auth: null,  host: 'softuni.bg',  port: null,  hostname: 'softuni.bg',  hash: null,  search: null,  query: null,  pathname: '/trainings/3473/angular-july-2021',  path: '/trainings/3473/angular-july-2021',  href: 'https://softuni.bg/trainings/3473/angular-july-2021' } |
| --- |

!!!! There is also **URL**, which is included in the basic nodeJS library by default (without import):

| let softUniURL2 = new URL('https://softuni.bg/trainings/3473/angular-july-2021');  console.log(softUniURL2); |
| --- |

->

| URL {  href: 'https://softuni.bg/trainings/3473/angular-july-2021',  origin: 'https://softuni.bg',  protocol: 'https:',  username: '',  password: '',  host: 'softuni.bg',  hostname: 'softuni.bg',  port: '',  pathname: '/trainings/3473/angular-july-2021',  search: '',  searchParams: URLSearchParams {},  hash: '' } |
| --- |

!!!!

--------------------------------------------------------

**NodeJS Web Server**

--------------------------------------------------------

Web servers are software products that use the operating system to handle web requests and serve Web content ; the requests are redirected to other software products

(ASP.NET, PHP, etc.), depending on the webserver settings

**Creating simple NodeJS web server with the HTTP module:**   
  
**index.js:**

| const http = require('http'); const port = 5000;  const server = http.createServer((req, res) => {  res.write('Hello from NodeJS!');  res.end(); });  server.listen(port, () => {  console.log(`Server is listening on port ${port}...`); }); |
| --- |

!!! By default you must restart the web server for any changes to take effect

However, we may install and use **nodemon** - the third-party module will monitor for any changes in our source code and automatically restart our server.:

<https://nodemon.io/>

1) npm i nodemon

2) in package.json ->

| "scripts": {  "start": "nodemon index.js", |
| --- |

3) npm start !!!!  
  
!!! We must end the res with res.end, otherwise no content will be served to the visitor !!!

**Basic way to serve content based on the URL in the request:**

| if (req.url === '/cats') {  res.write('Some Cats');  } else if (req.url === '/dogs') {  res.write('Some Dogs');  } else {  res.write('Hello from NodeJS!');  } |
| --- |

**The request wrapper (req) has many properties, which we may use, such as:**

| console.log('Method: ', req.method); console.log('URL: ', req.url); console.log('Headers: ', req.headers); |
| --- |

the console log on the node.js server (not on the browser):

| Method: GET URL: /dogs Headers: {  host: 'localhost:5000',  connection: 'keep-alive',  'cache-control': 'max-age=0',  'sec-ch-ua': '" Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"',  'sec-ch-ua-mobile': '?0',  'sec-ch-ua-platform': '"macOS"',  'upgrade-insecure-requests': '1',  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36',  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  'sec-fetch-site': 'none',  'sec-fetch-mode': 'navigate',  'sec-fetch-user': '?1',  'sec-fetch-dest': 'document',  'accept-encoding': 'gzip, deflate, br',  'accept-language': 'en-US,en;q=0.9,bg;q=0.8,ja;q=0.7,ar;q=0.6,sl;q=0.5',  cookie: 'Webstorm-f2b156=8a94a01a-6b01-41b9-b2ff-dd2f71fab48d' } |
| --- |

**With the response wrapper we may send content to the client and change the response headers, including for example the content type of the response**:

| res.writeHead(200, {  'Content-type': 'text/plain'  }); |
| --- |

If we set the content type to be html, we can set HTML in our response:

| res.writeHead(200, {  'Content-type': 'text/html'  });  res.write('<h1>Hello from NodeJS!</h1>'); |
| --- |

we can serve CSS from JavaScript string by setting the content-type of the response to text/css:

index.js:

| const http = require('http'); const homePage = require('./home'); const siteCSS = require('./resources/content/styles/site.js');  const server = http.createServer((req, res) => {  if (req.url == '/resources/content/styles/site.css') {  res.writeHead(200, {  'Content-type': 'text/css'  });  res.write(siteCSS);  } else {  res.writeHead(200, {  'Content-type': 'text/html'  });    }  res.end(); }).listen(5000, () => {  console.log('Server is listening on port 5000...') }) |
| --- |

site.js:

| module.exports = ` @import url('https://fonts.googleapis.com/css?family=Lato:400,700&display=swap');  \* {  margin: 0;  padding: 0; }  html, body {  font-family: 'Lato', sans-serif;  background-color: #F2F2F2;  color: #333; }  ........... `; |
| --- |