back-end

Node & NPM

lab-1

today

I. Course (recap)
II. Node
III. Modules
IV. NPM Package

Course

course

description

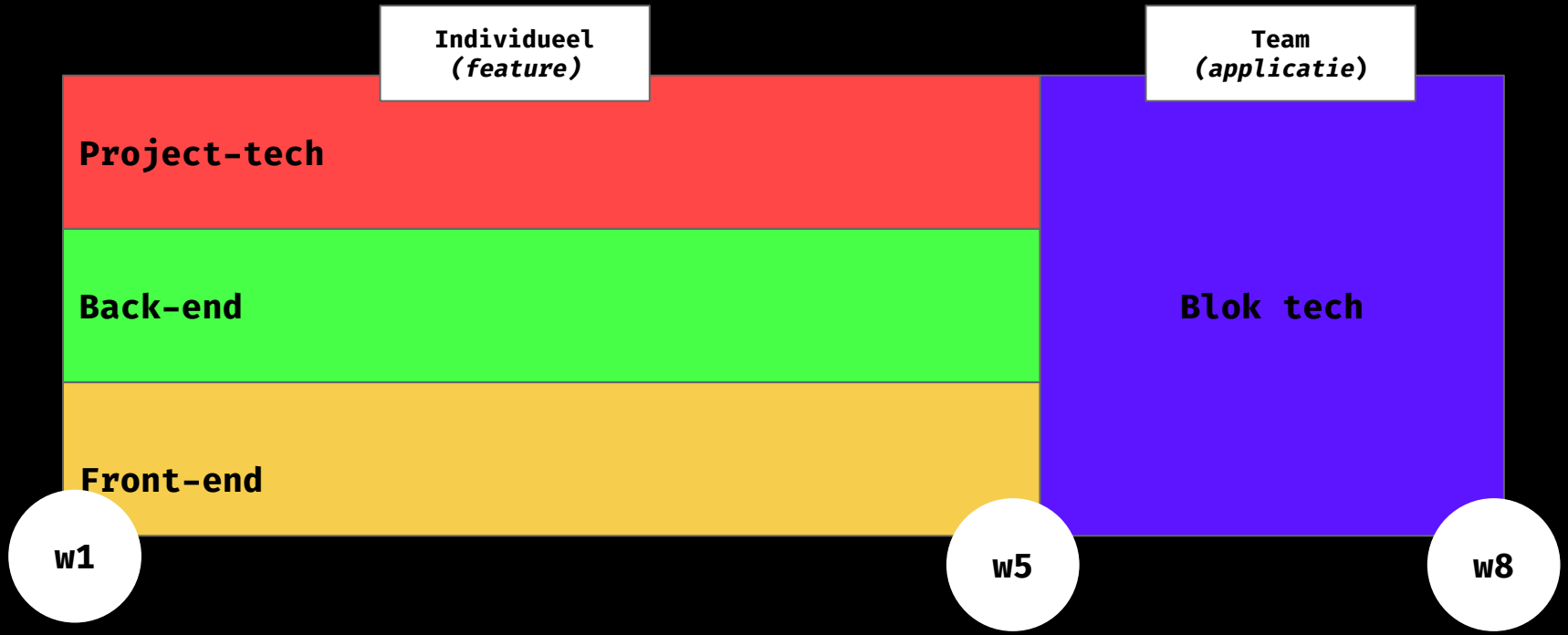
In Back-end we peek behind the curtains and inspects what's behind the web. You build web app with **Node.js**, communicate with **HTTP**, and store data in a database with **MongoDB**. You'll learn to use computers to actually make what you design work, people can actually fill in forms and upload files.

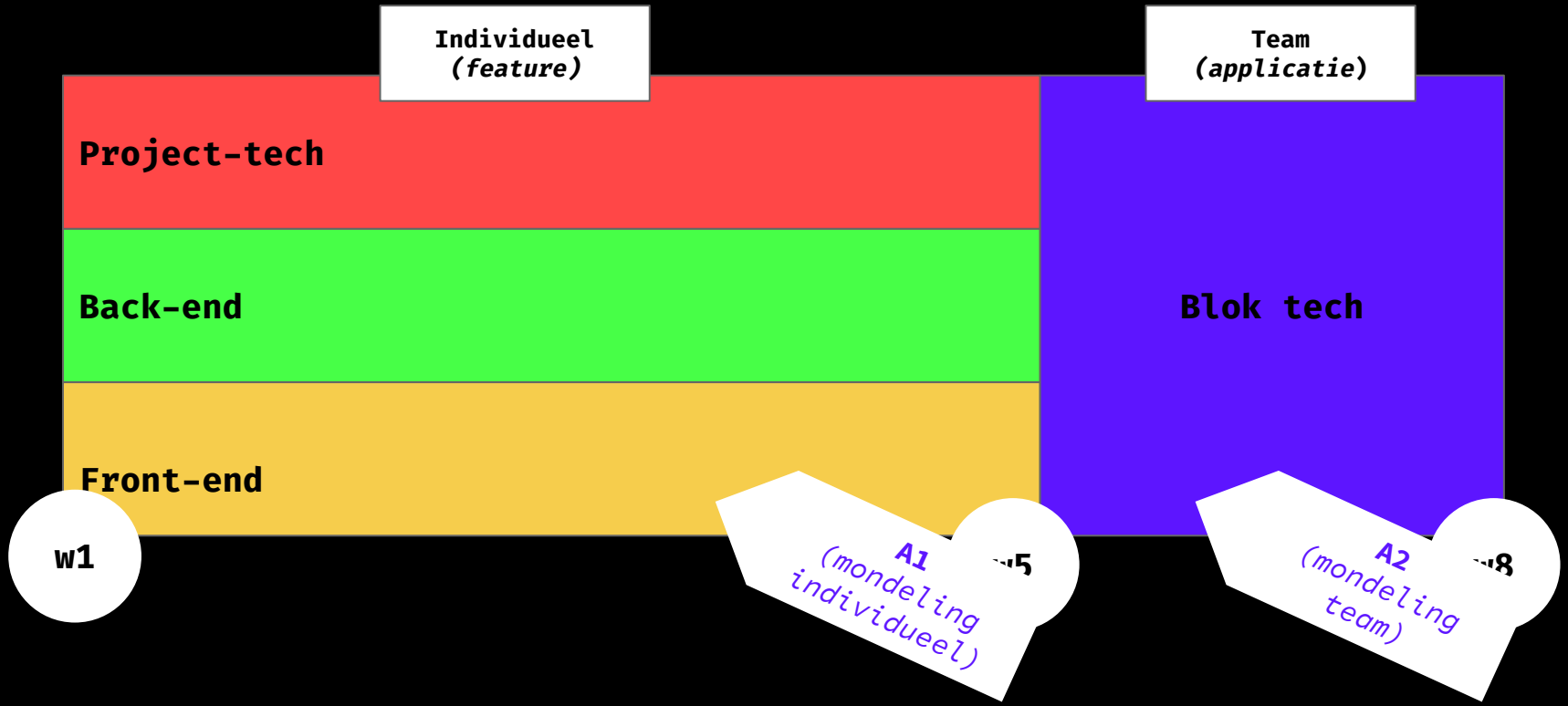
/readme.md

course

goals

- You can build web apps with Node and NPM
- You understand client/server flow (http)
- You can render data server-side with a template engine
- You can store data in a database and update that data
- You can write docs and explain your code cohesion


Individueel
(feature)
Team
(applicatie)
Project-tech
Back-end
Front-end
Blok tech
w1
w5
w8


Individueel
(feature)
Team
(applicatie)
Project-tech
Back-end
Front-end
Blok tech
w1
A1
(mondeling
individueel)
A2
(mondeling
team)

course

deliverables

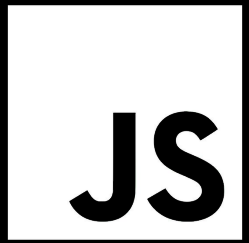
- **Individual Prototype:** working **interactive feature** for matching application
- **Team Prototype:** working **interactive matching application**
- **Process book (wiki):** that provides insight into the weekly iterative process

Node.js

node

javascript?

JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.



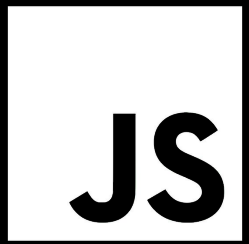
developer.mozilla.org

node

environment

JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.

...and so much more!

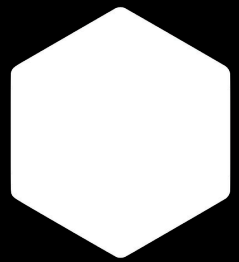


developer.mozilla.org

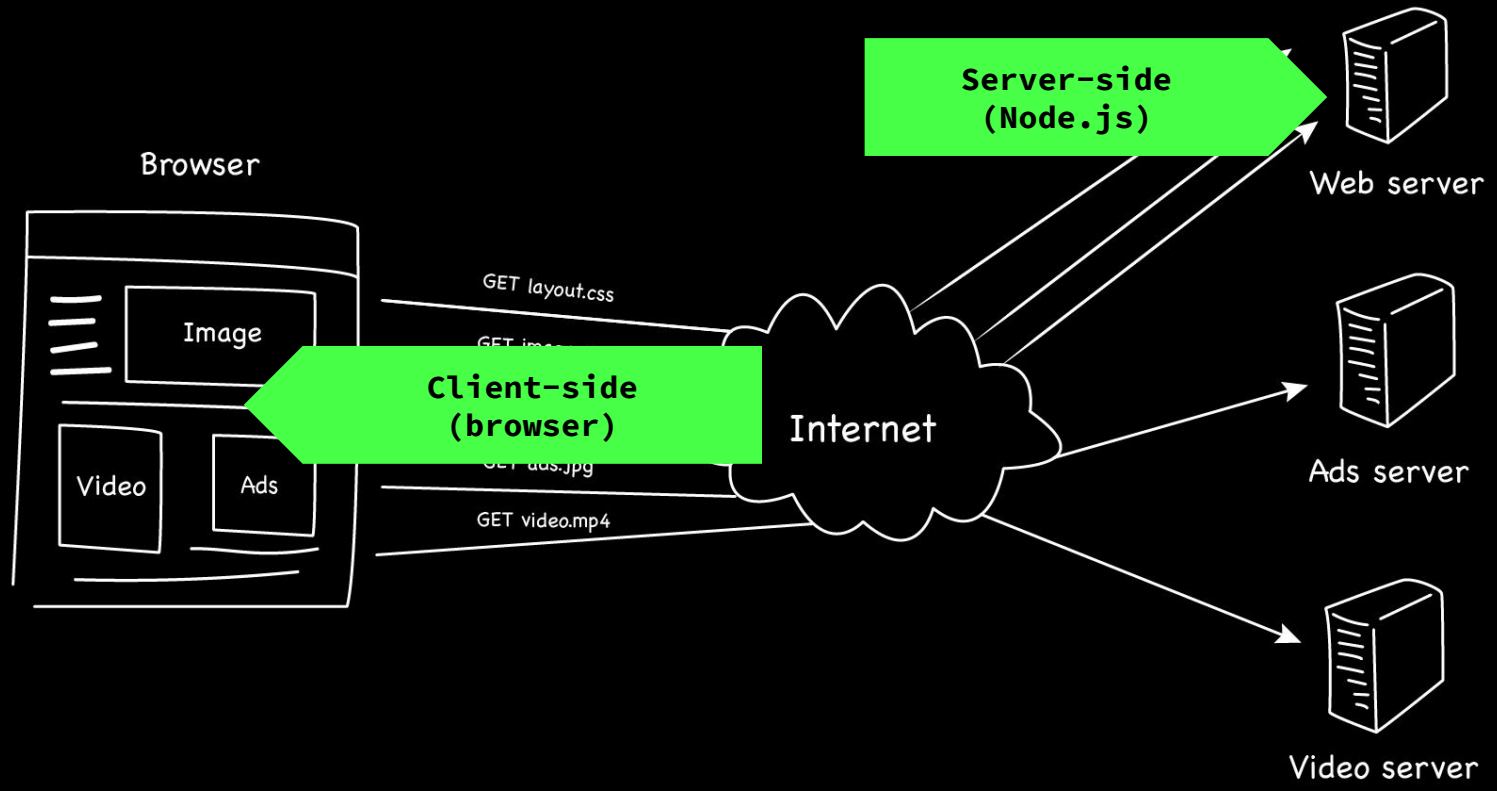
node

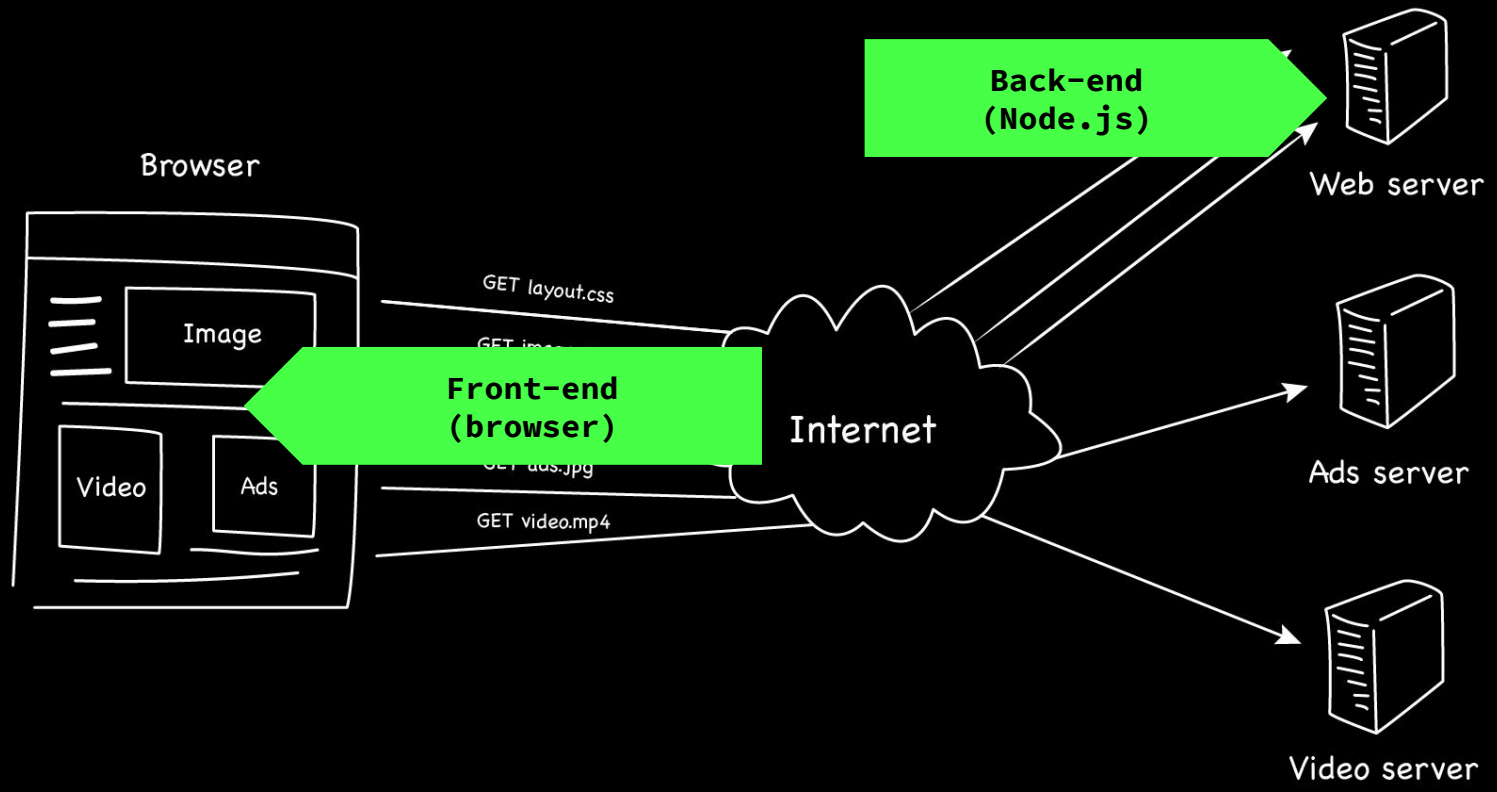
node?

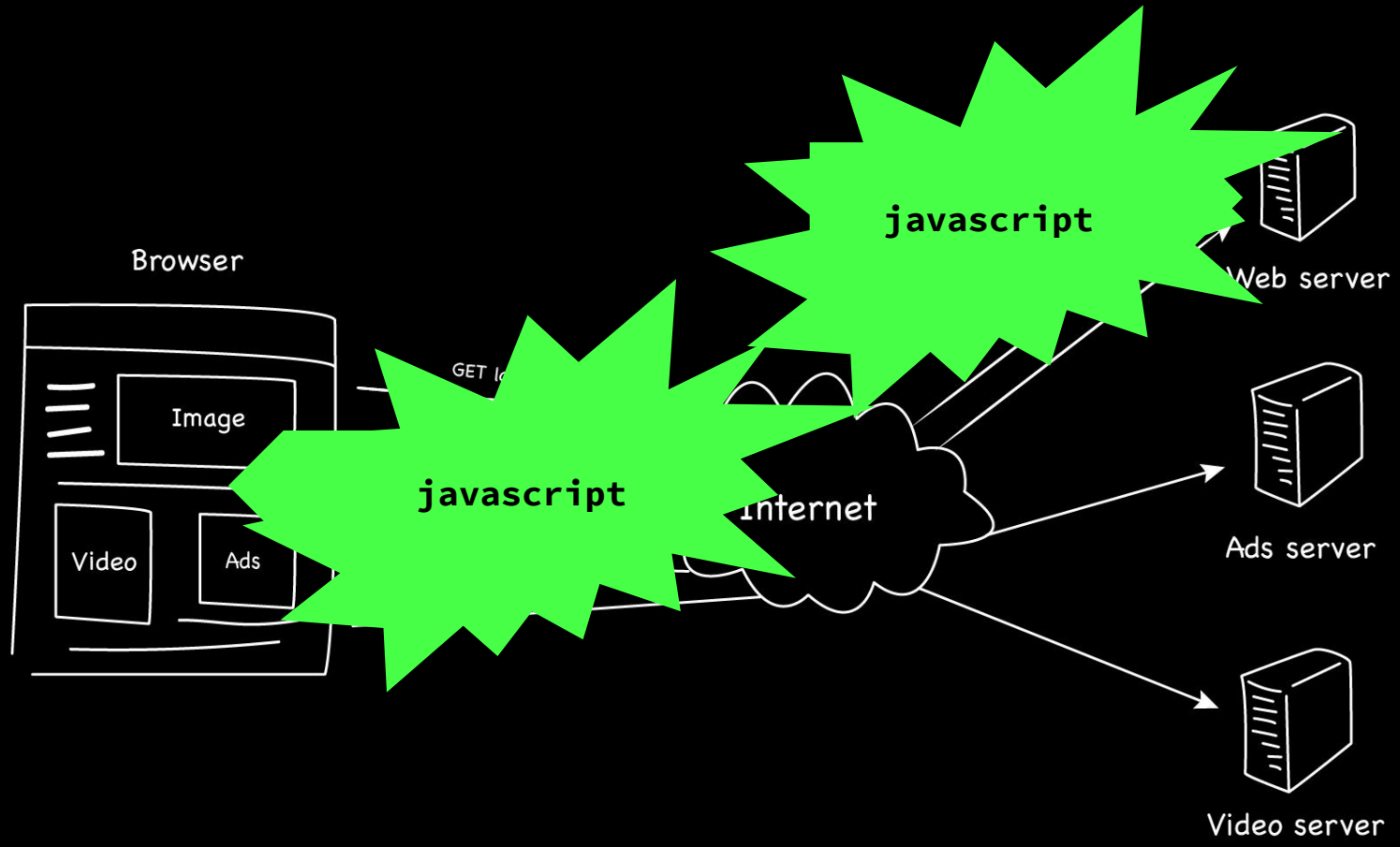
Node.js is an open-source, cross-platform […] run-time environment for **executing JavaScript code server-side.** […] Node enables JavaScript to be used for server-side scripting, and runs scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.



wikipedia.org


Server-side
(Node.js)
Web server
Browser
Image
Video
Ads
GET layout.css
Client-side
(browser)
Internet
GET video.mp4
Ads server
Video server


Back-end
(Node.js)
Web server
Browser
Image
Video
Ads
GET layout.css
Front-end
(browser)
Internet
Ads server
GET video.mp4
Video server


javascript
Browser
Image
Video
Ads
javascript
Internet
Web server
Ads server
Video server

node

libs

In the **Browser** you get JS and…

- Console (console.log, …)
- Timers (setTimeout, …)
- window
- document (DOM)
- XMLHttpRequest (or fetch)
- <script>
- Canvas / WebGL
- …

In **Node.js** you get JS and…

- Console (console.log, …)
- Timers (setTimeout, …)
- global
- File System (fs)
- http
- require / module
- Buffer
- …

node

~/index.js

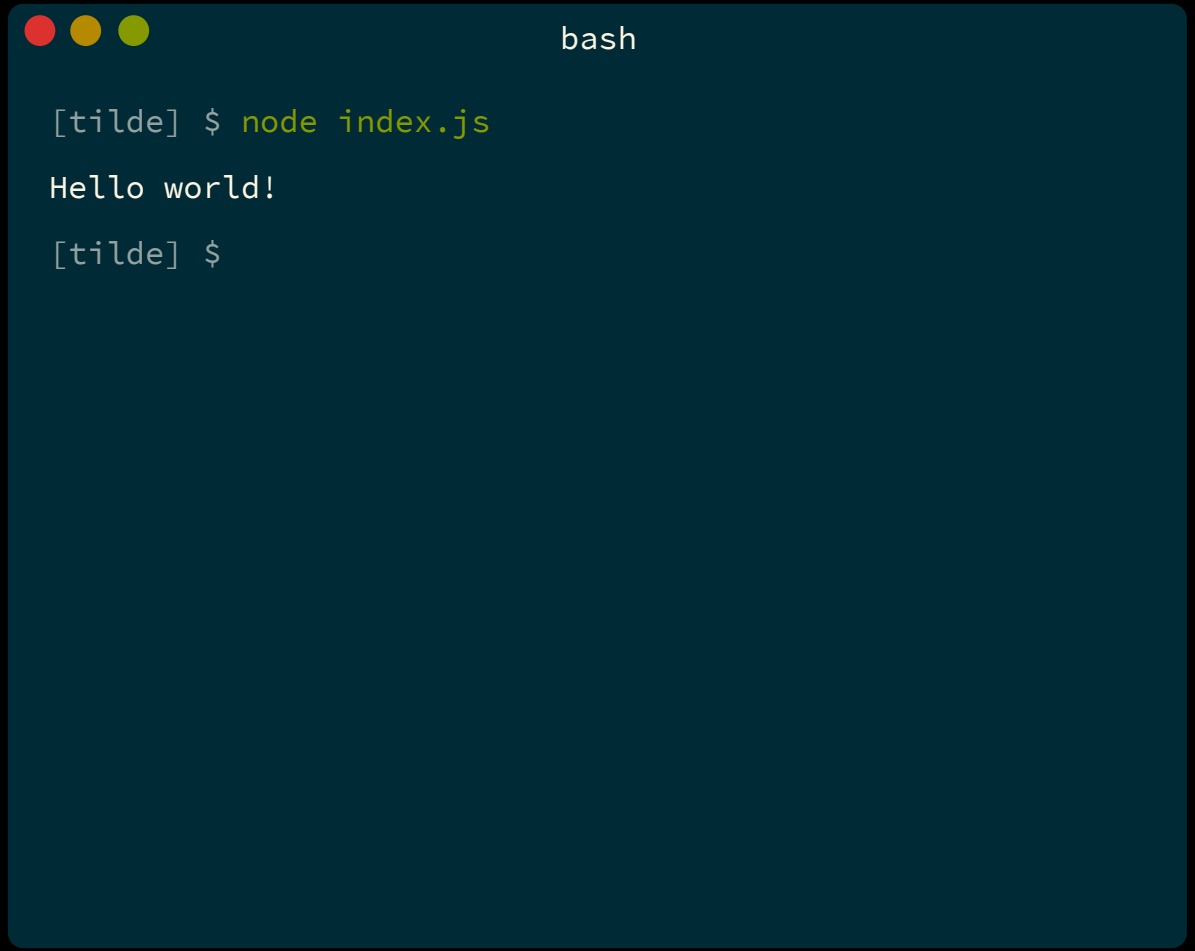
```
console.log('Hello world!')
```

browser

~/index.html

```
<script src=index.js></script>
```

node



repl

node

why learn node?

Yes, you should know Node (but it depends)

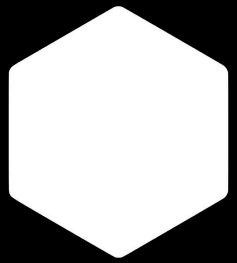
Frontend developers **should**…

- Know what backend developers do
- Be comfortable with Node-based tooling
- Have a basic understanding of web servers and databases
- Be able to build a basic prototype

Modules

folder/index.js

```
console.log(sum(1, 2, 3))

function sum() {
  var args = arguments
  var total = 0
  var index = -1

  while (++index < args.length) {
    total += args[index]
  }

  return total
}
```



modules

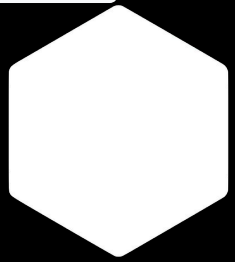
scripts

folder/sum.js

```
function sum() {
  var args = arguments
  var total = 0
  var index = -1
  while (++index < args.length) {
    total += args[index]
  }
  return total
}
```

folder/index.js

```
console.log(sum(1, 2, 3))
```

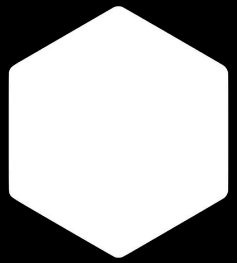


modules

scripts

folder/index.html

```
<script src=sum.js></script>
<script src=index.js></script>
```



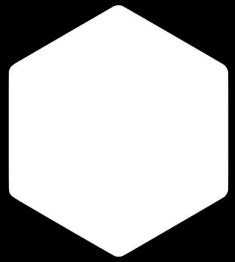
modules

errors

folder/index.html

```
<script src=index.js></script>
<script src=sum.js></script>
```

ReferenceError: Can't find variable: sum



modules

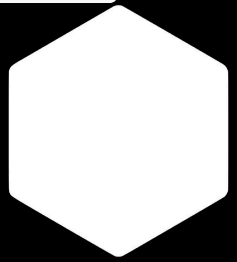
require

folder/sum.js

```
module.exports = sum

function sum() {
  var args = arguments
  var total = 0
  var index = -1
  while (++index < args.length) {
    total += args[index]
  }
  return total
}
```

folder/index.js

```
var sum = require('./sum.js')

console.log(sum(1, 2, 3))
```

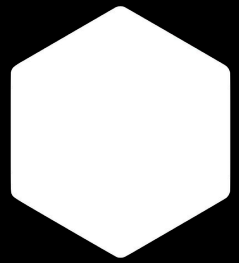


modules

folders

```
// Files
folder/
├─ index.js
└─ modules/
   └─ sum.js

// Command line
[folder] $ node index.js
6
```

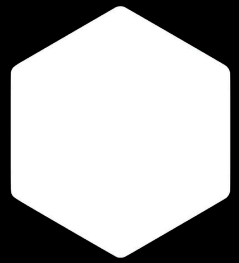
```
// index.js
var sum = require('./modules/sum')

console.log(sum(1, 2, 3))

// sum.js
module.exports = sum

function sum() {
  var args = arguments
  var total = 0
  var index = -1

  while (++index < args.length) {
    total += args[index]
  }

  return total
}
```



modules

?

Q: So, do I need to write all my modules?
A: **No!**



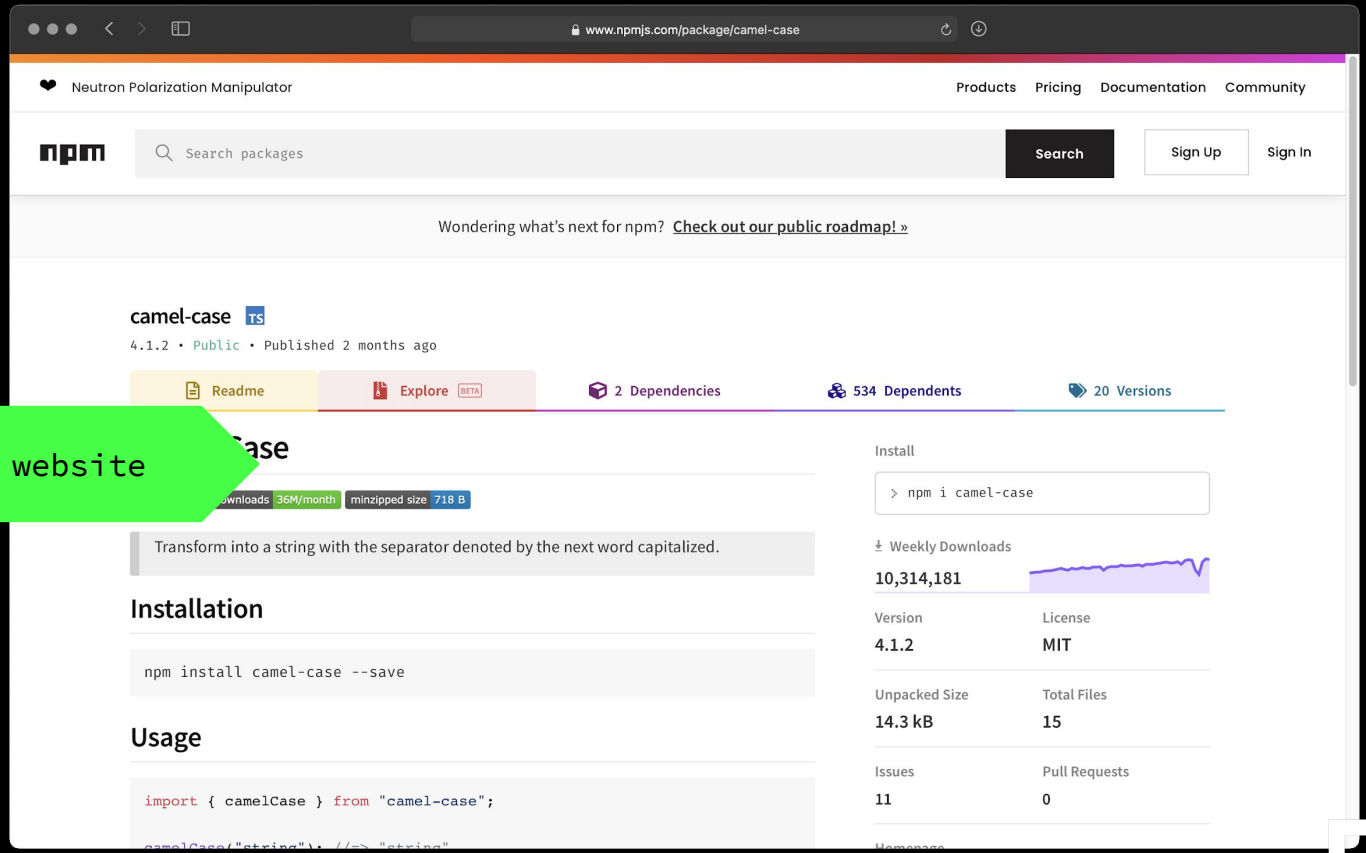
NPM

NPM

?

npm is a package manager for the JavaScript programming language. It is the default package manager for […] Node.js. It consists of a command line client, also called npm, and an online database of […] packages, called the npm registry.

wikipedia.org

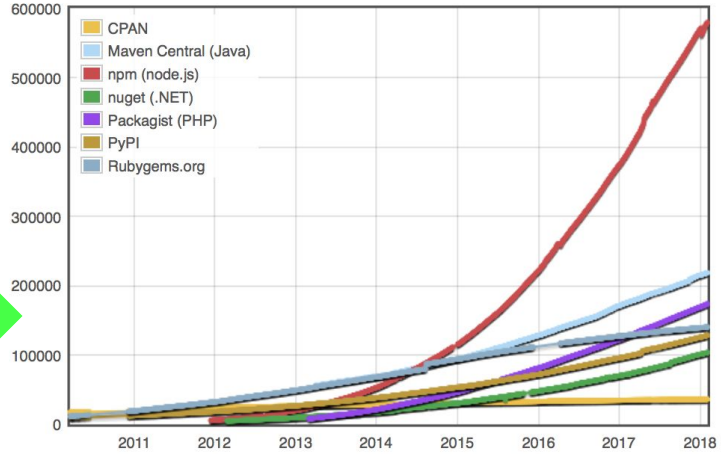


The **npm** website

npm is huge

www.modulecounts.com

Module Counts



Include

- [ ] Clojars (Clojure)
- [x] CPAN
- [ ] CPAN (search)
- [ ] CRAN (R)
- [ ] Crates.io (Rust)
- [ ] Drupal (php)
- [ ] DUB (dlang)
- [ ] Gopm (go)
- [ ] Hackage (Haskell)
- [ ] Hex.pm (Elixir/Erlang)
- [ ] Julia
- [ ] LuaRocks (Lua)
- [x] Maven Central (Java)
- [ ] MELPA (Emacs)
- [ ] Nimble (Nim)
- [x] npm (node.js)
- [x] nuget (.NET)
- [x] Packagist (PHP)
- [ ] Pear (PHP)
- [ ] Perl 6 Ecosystem (perl 6)
- [x] PyPI
- [x] Rubygems.org
- [ ] Vim Scripts

time period ◉ all time ○ last year ○ last 90 days ○ last 30 days ○ last 7 days

	Jan 26	Jan 27	Jan 28	Jan 29	Jan 30	Jan 31	Feb 1	Avg Growth
Clojars (Clojure)	20942	20947	20952	20957	20969	20976	20984	7/day

arstechnica.com/information-tech

ars TECHNICA SUBSCRIPTIONS SIGN IN

BIZ & IT —

Rage-quit: Coder unpublished 17 lines of JavaScript and “broke the Internet”

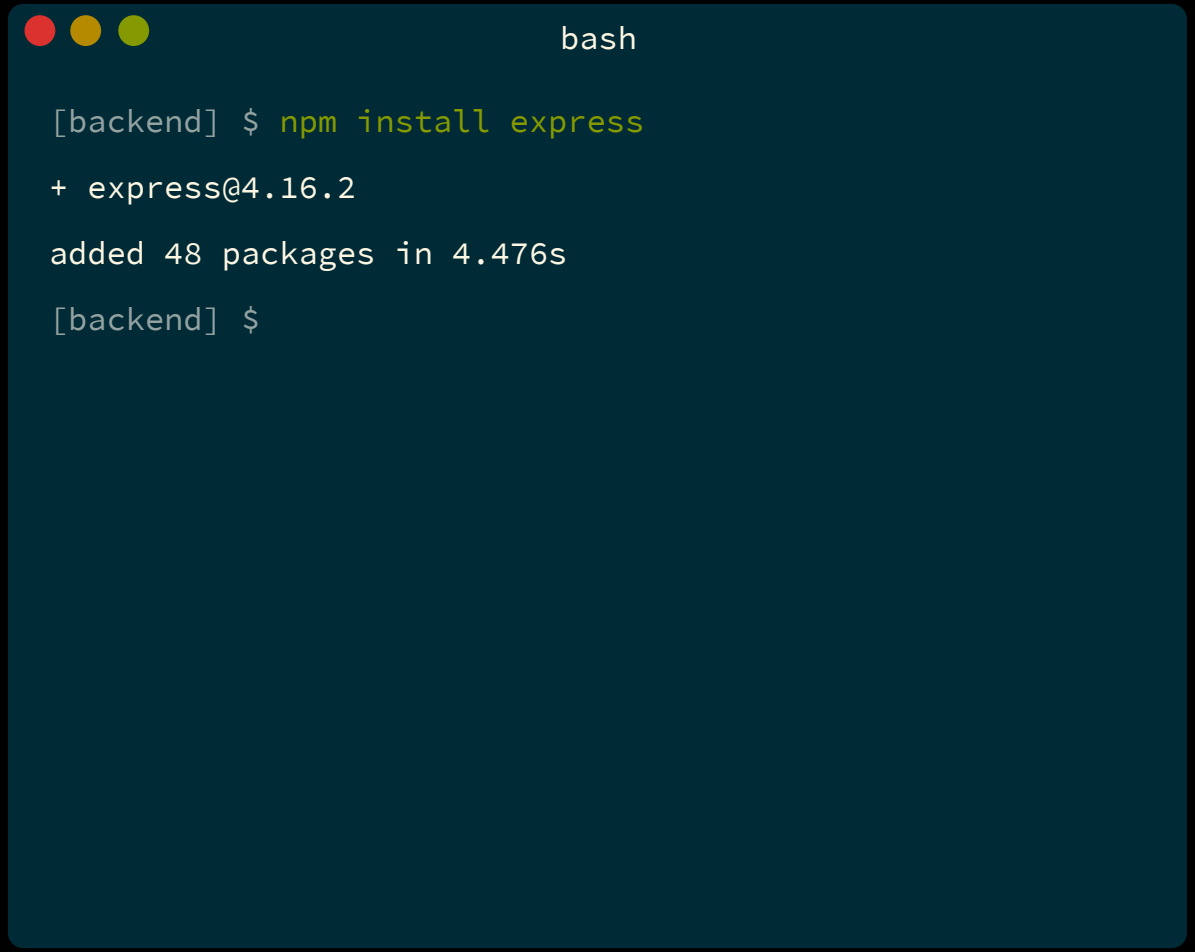
Dispute over module name in npm registry became giant headache for developers.

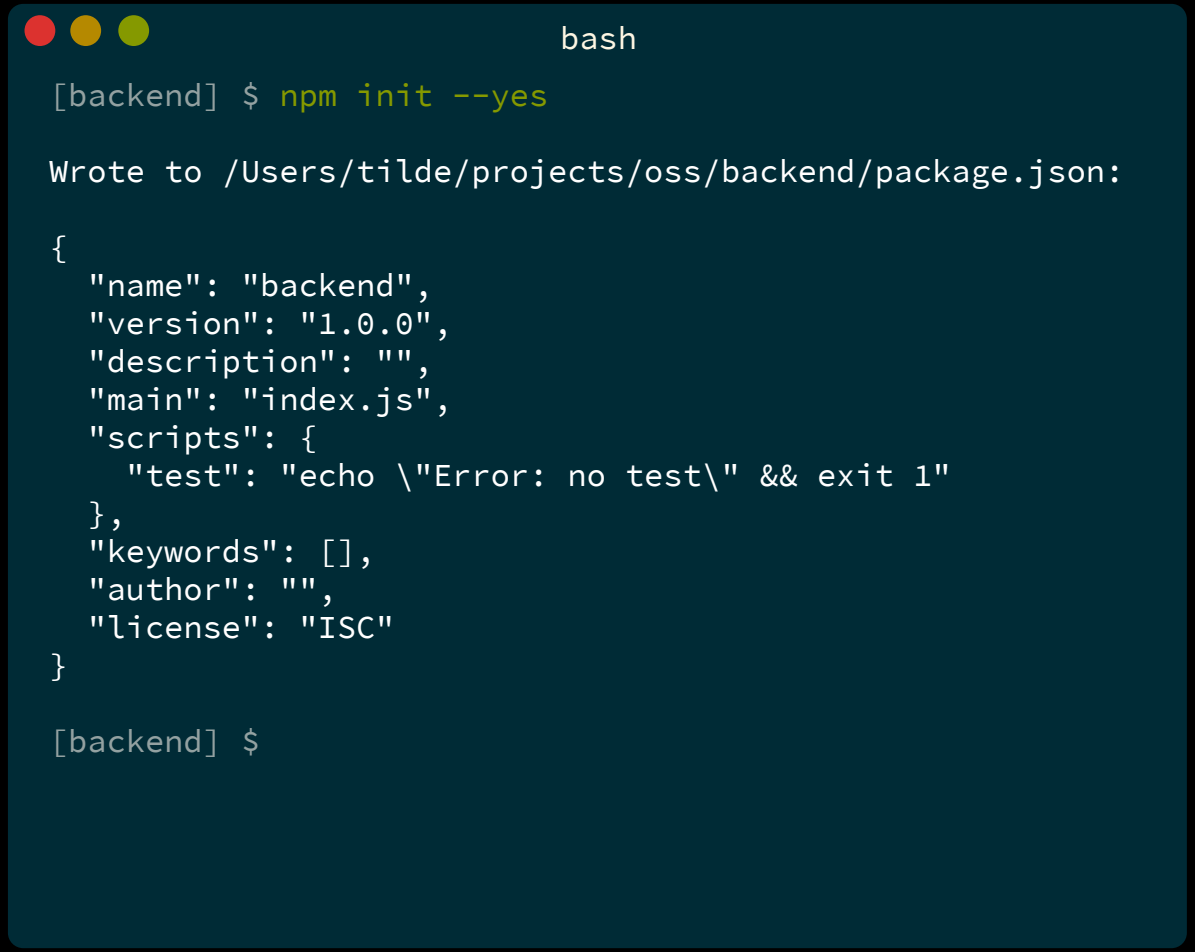
AN GALLAGHER - 3/25/2016, 3:10 AM

Developer got mad

ch Lawson

npm


bash
[backend] $ npm install express
+ express@4.16.2
added 48 packages in 4.476s
[backend] $


bash
[backend] $ npm init --yes
Wrote to /Users/tilde/projects/oss/backend/package.json:
{
"name": "backend",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}
[backend] $

NPM

dependencies

bash

```
$ npm install repeat-string
```

Dependencies are used in the project itself

```
$
```

package.json

```
{
  …
  "dependencies": {
    "repeat-string": "^1.6.1"
  },
  "devDependencies": {
    "standard": "^10.0.3",
    "tape": "^4.8.0",
    …
  },
  …
```

NPM

dependencies

bash

```
$ npm install tape --save-dev

+ tape@4.8.0
updated 1 package in 1.44s
```

devDependencies are used to build, check, and test the project

package.json

```
{
  …
  "dependencies": {
    "repeat-string": "^1.5.4"
  },
  "devDependencies": {
    "standard": "^10.0.3",
    "tape": "^4.8.0",
    …
  },
  …
```

NPM

semver

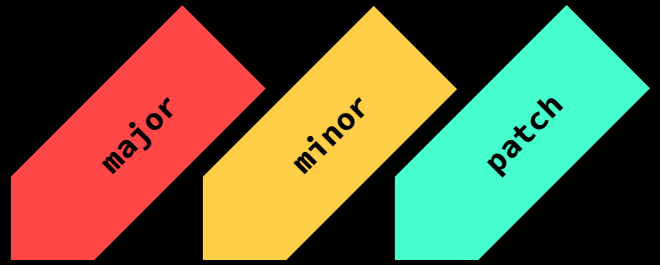
package.json

```
{
  ...
  "dependencies": {
    "repeat-string": "^1.5.4"
  },
  "devDependencies": {
    "standard": "^10.0.3",
    "tape": "^4.8.0",
    ...
  },
  ...
```

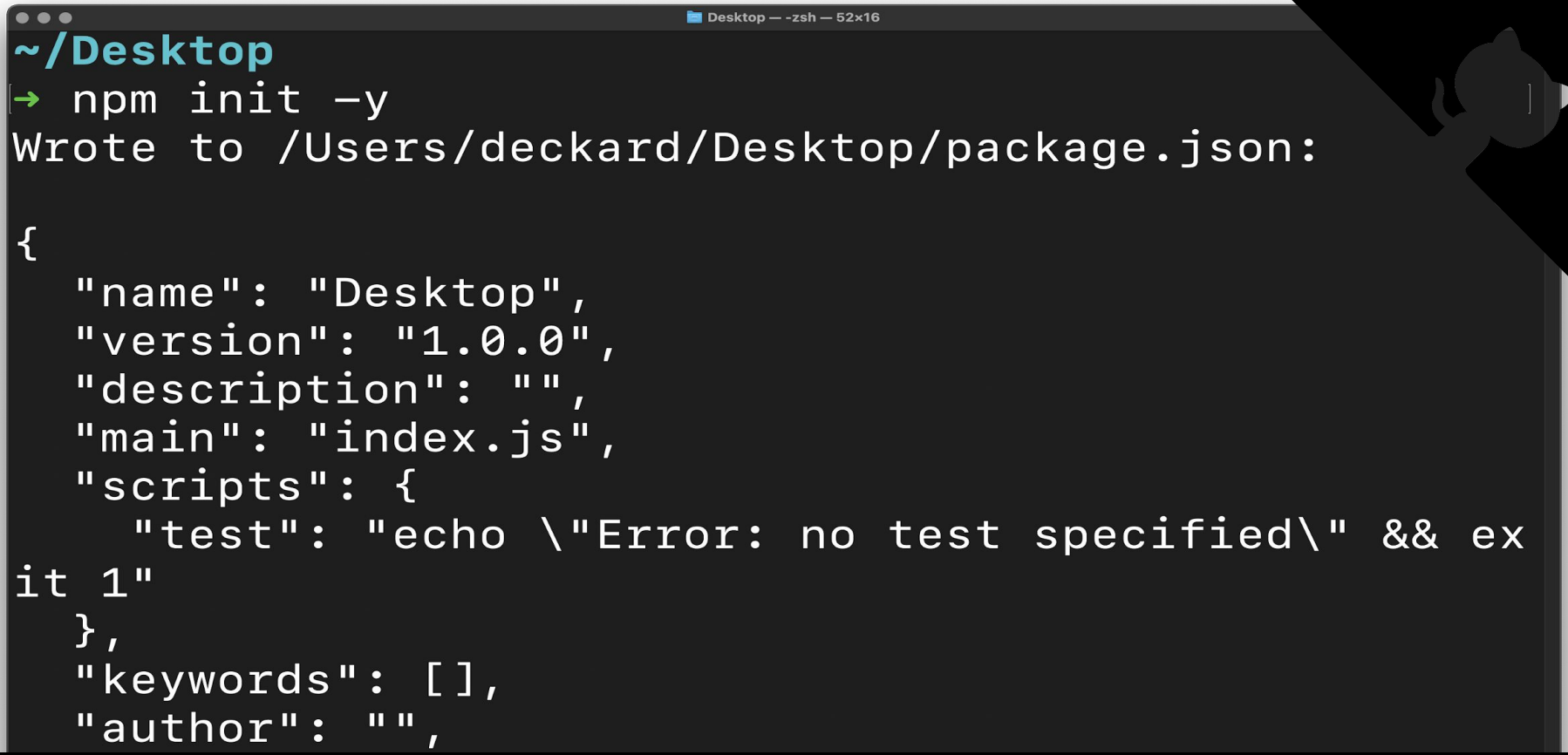
semver
(semantic versioning)

NPM

versions




```
"version": "2.5.1"
```



Live demo **npm en packages**

Package

> Learn the basics of node modules and npm packages and setup a boilerplate for your own feature.

Synopsis

- **Time**: 6:00h
- **Goals**: subgoal 1, subgoal 2
- **Due**: before week 2

1. Create the boilerplate for the matching app you are going to create. Include a `package.json` with a correct name, version, dependencies, and other metadata. See npm's documentation on `package.json`. For examples of `package.json` files, see `repeat-string`, `longest-streak`, or `skin-tone`.

2. Look trough the NPM registry and install a package from npm that would be helpful for your job story and try it out in `index.js`. Not sure what package to pick? You can try playing around with `camelcase` or `lodash` to get comfortable requiring packages and using them.

3. Improve the *developer experience* of your application. Look for so called 'developer dependencies' on NPM. `nodemon` is a good example,

work on package

exit;

see you in `lab-2a`!