Bristech

KNOWLEDGE SHARED

Next month:
Isomorphic React / Networking for Devs

// Ben Gourley - Enjoy modern JavaScript

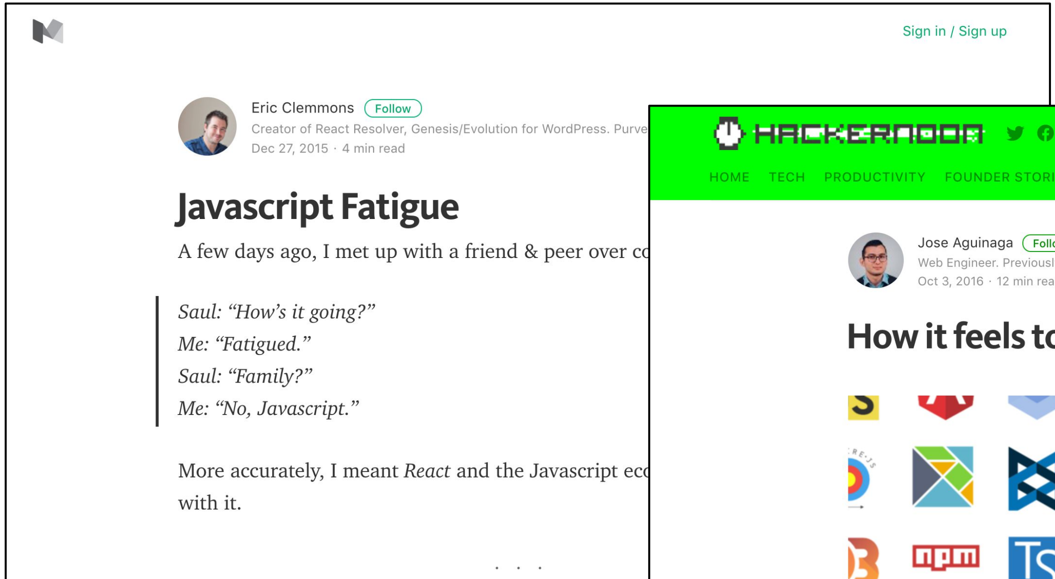
// Fionnuala Costello - Anti-counterfeiting

softwire POLECAT SCOTT LOGIC ALTOGETHER SMARTER CACI

Enjoying JS

@bengourley

- Avoiding “JavaScript fatigue”
- Help you make good decisions about dependencies
- Use the language and ecosystem to your advantage


Sign in / Sign up

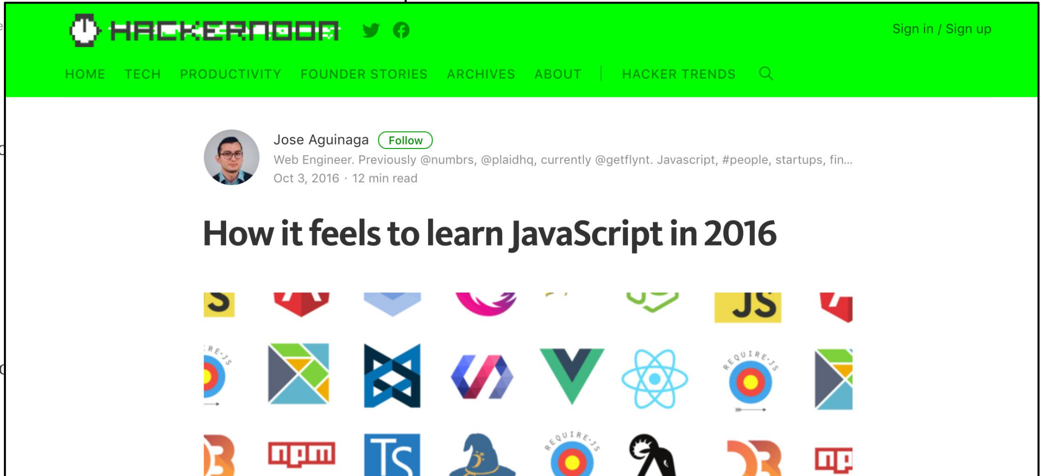
Eric Clemmons Follow
Creator of React Resolver, Genesis/Evolution for WordPress. Purve
Dec 27, 2015 · 4 min read

Javascript Fatigue

A few days ago, I met up with a friend & peer over co

Saul: “How’s it going?”
Me: “Fatigued.”
Saul: “Family?”
Me: “No, Javascript.”

More accurately, I meant *React* and the Javascript ec
with it.



Jose Aguinaga Follow
Web Engineer. Previously @numbrs, @plaidhq, currently @getflynt. Javascript, #people, startups, fin...
Oct 3, 2016 · 12 min read

How it feels to learn JavaScript in 2016

JS is THE BEST

JS is ~~THE BEST~~

You’re doing it WRONG!

~~You’re doing it WRONG!~~

So what then?

1.

npm + node + browserify

npm

- The javascript package manager
- server/client parity aka. “isomorphism”
- 437,000 modules
- ~~Solved~~ reduced dependency hell

```
# install a module
npm install --save express

# puts it on the file system
node_modules
└ express@4.14.0

# puts it in package.json
{
  dependencies: {
    “express”: “^4.14.0”
  }
}
```

Dependency Hell

parsnip depends on **vegetable < v1.5**

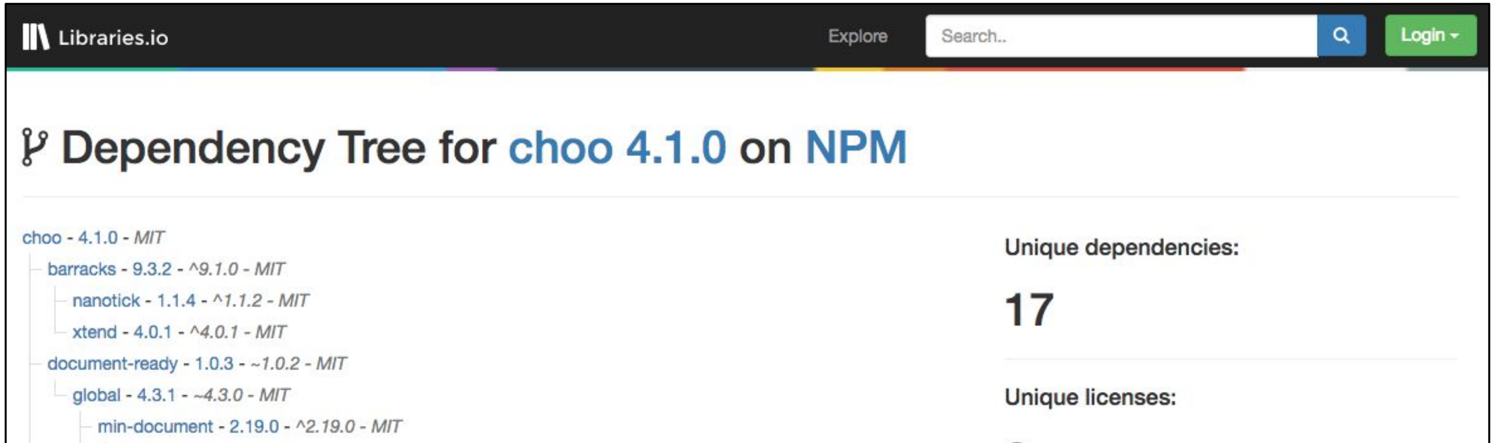
cabbage depends on **vegetable > v2.1**

Your application depends on **parsnip** <u>AND</u> **cabbage**

Which version of **vegetable** should get installed?

Choosing a good module

- Do you really need a module for this?
- Shallow dependency tree
- Trusted module author
- Good test suite
- Don’t write off low levels of activity



Dependency Tree for choo 4.1.0 on NPM

```
choo - 4.1.0 - MIT
  barracks - 9.3.2 - ^9.1.0 - MIT
    nanotick - 1.1.4 - ^1.1.2 - MIT
    xtend - 4.0.1 - ^4.0.1 - MIT
  document-ready - 1.0.3 - ~1.0.2 - MIT
    global - 4.3.1 - ~4.3.0 - MIT
      min-document - 2.19.0 - ^2.19.0 - MIT
```

Unique dependencies:

17

Unique licenses:

2.

Minimise tooling

tooling should help you and get out of the way, not weigh you down

Running tasks

- Make?
- Rake?
- Jake?
- Gulp?
- Grunt?
- Broccoli?
- Grulp?!
- pliers?

```
# install the task runner
npm install -g gulp

# install the task adapter module
npm install --save gulp-less

# which also installs less
node_modules
├─ gulp-less@x.x.x
└─ less@x.x.x

# then configure the “task”
gulp.task('less', function () {
  ...
})

# run the less task
$ gulp less
```

npm scripts

- Install the tools you need
- Alias their usage in npm scripts
- More than adequate for any small/medium size project

```
# install less cli directly
npm install --save less

# package.json
{
  scripts: {
    “buildcss”:
      “lessc styles.less styles.css”
  }
}

# run it
npm run buildcss
```

3.

Learn the language

(not frameworks)

It’ll work! But…

- Non-trivial things will be difficult
- Your code might look like spaghetti
- “Hey Ben, I should have just used a framework”

Or you might just…

- Realise that you didn’t need a framework
- Rejoice at the lack of bloat

4.

If it ain’t broke…

(aka. The “blinkers”)

But…

- It is beneficial to keep up
- Do it at your own leisure
- Observe from a high level e.g.

React: (state) => ui

Redux: (state, action) => new state

If something sticks long after the hype then you know there must be something to it – just be sure it does something for you.

5.

Standards are improving

You might not need…

jQuery

DOM APIs have massively improved. Now redundant?

```
// jQuery
$(‘.my-thing’)

// DOM methods
document.querySelectorAll(‘.my-thing’)
```

underscore/lodash

ES5/6 has widespread support and adoption.

```
// underscore.js
_.map(array, fn) _.reduce(array, fn) …

// Array.prototype methods
array.map(fn) array.reduce(fn) …
```

You might want to start using…

Promises

```
// promise constructor
new Promise((resolve, reject) => {})

// async control flow
Promise.all([ ...promises ])
Promise.race([ ...promises ])

// ensure the thing you have is a promise
Promise.resolve(valOrPromise)
```

You might want to start using…

Destructuring

```
// object destructuring
const { x, y } = getCoords()

// array destructuring
Promise.all([ fetchCatPics(), fetchDogPics() ])
  .then(function ([ cats, dogs ]) => {
    // do something with cats, dogs
  })
```

You might want to start using…

Arrow functions

```
function Widget() {
  this.clicked = 0
  document.addEventListener(‘click’, () => {
    // “this” does what you expect
    console.log(this.clicked++)
  })
}

new Widget()
```

6.

Avoiding bloat

The caveat of npm.

- Just because you *can* install something doesn’t mean you should
- Finding a good module is hard
- Sub-ecosystems each with their own NIH
- Installing is easy, pain comes later

Logical conclusion: fewer dependencies is better.

my full stack

for api-backed web apps

browserify

require('modules') for the browser

- Uses **exactly** the same module resolution algorithm as node. Compatibility FTW!
- watchify for fast recompilation in development
- Sourcemaps
- "Do one thing well" approach
  - Inject functionality with transforms/plugins
  - Pipe output to other tools, e.g. uglify js

choo

🚂🚋🚋🚋🚋🚋

frontend framework

- Views: (state) => ui
- Models: (state, action) => new state
- Batteries included
  - routing
  - asynchronous effects
  - subscriptions
  - server rendering
- No virtual dom, uses an efficient dom-diffing algorithm
- Designed for use with browserify
- 5kb

express

server framework

- expressive routing
- simple middleware layer
- require('helmet') !! security !!
- fast
- static asset serving

postcss

css preprocessor

- Generic css preprocessor with arbitrary plugin system
- Functionality is defined by what plugins you use
- cssnext: future css standards for use today

tooling

linting, testing etc.

- nodemon
- standardjs
- tape
- test utilities:
  - proxyquire
  - nsp
  - istanbul
- cssnano
- uglifyjs

```
"scripts": {
  "start": "node app",
  "clean": "rm -fr static && mkdir -p static/js static/css",
  "test": "npm run lint && npm run unittest && npm run coverage && npm run security",
  "unittest": "istanbul cover tape -- **/*.test.js'",
  "security": "nsp check",
  "coverage": "istanbul check-coverage",
  "lint": "standard",
  "build": "npm run clean && npm run build:js && npm run build:css",
  "watch": "nodemon app & npm run watch:js & npm run watch:css",
  "build:js": "browserify src/client.js | uglifyjs > static/js/bundle.js",
  "watch:js": "watchify -d src/client.js -o static/js/bundle.js",
  "build:css": "postcss src/styles/global.css | cssnano > static/css/bundle.css",
  "watch:css": "postcss -w -m static/css/bundle.css src/styles/global.css"
}
```

Summary

1. Use npm
2. Minimise tooling
3. Learn ~~frameworks~~ the language
4. Keep the blinkers on
5. Make use of new features
6. Only install necessary things

Thank you!

Any questions?