Do I still need this *dependency* for my Node.js app?

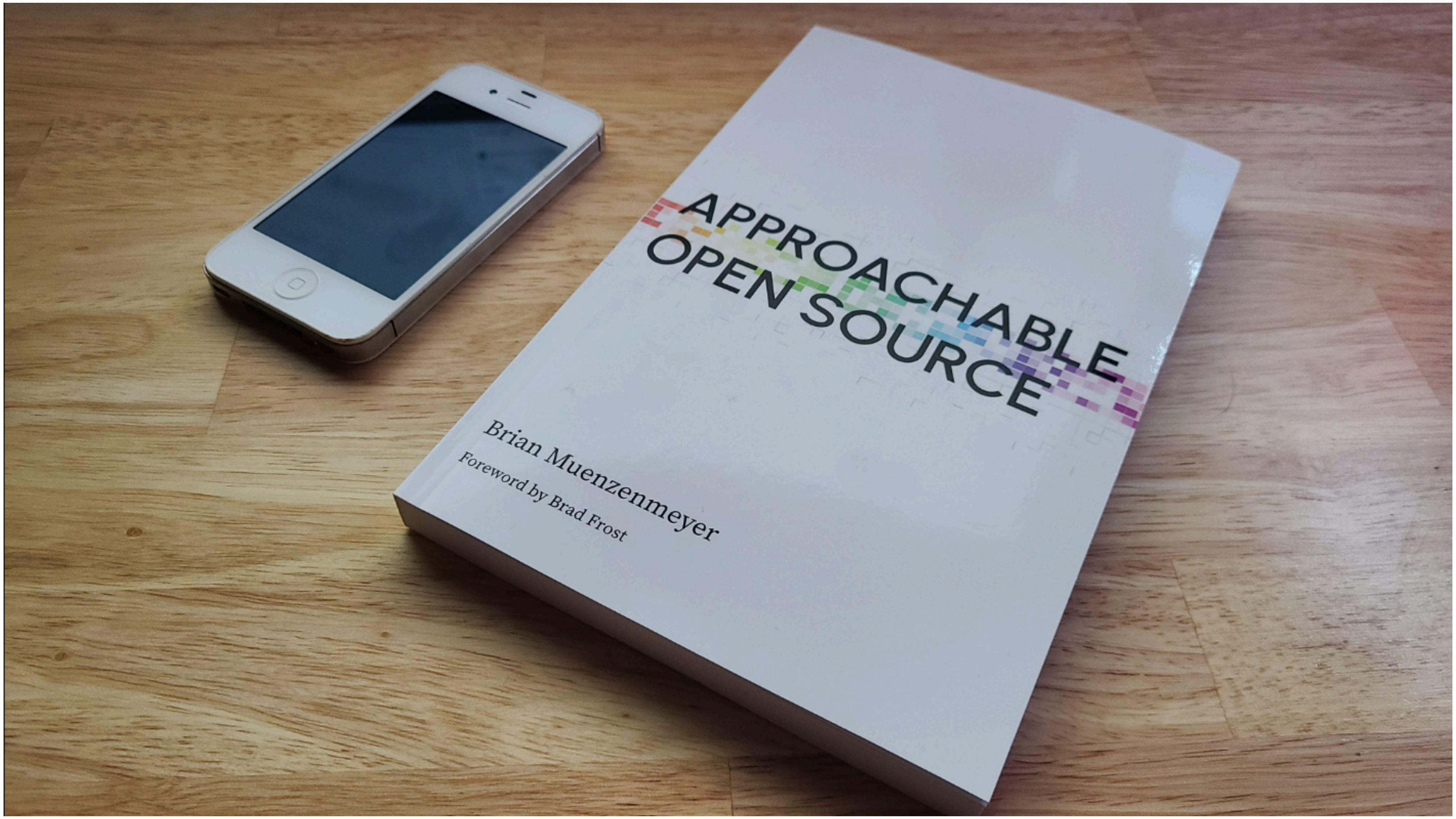
Brian Muenzenmeyer

Minnebar 19

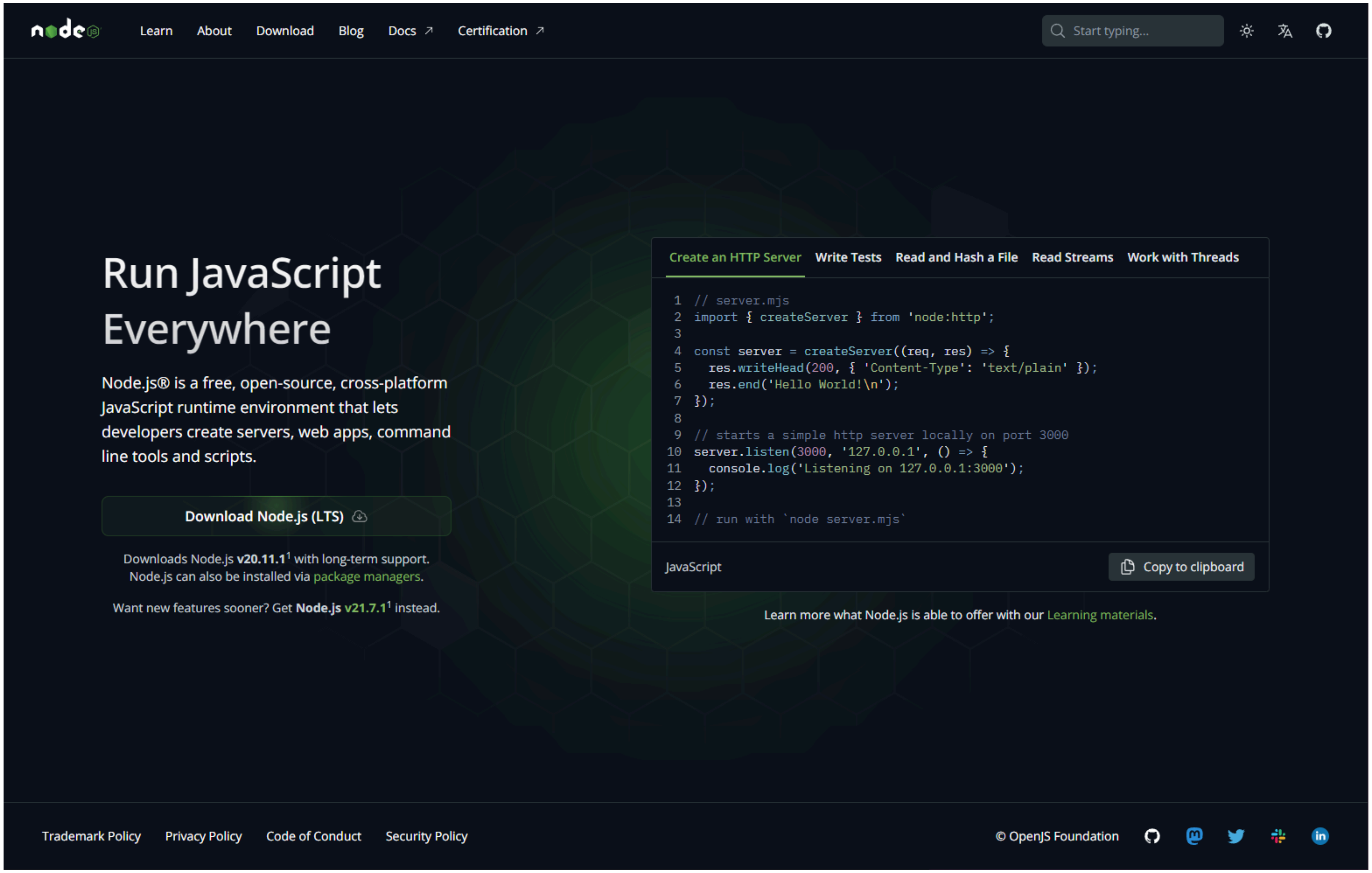
👋 About Me

principal engineer

leader of open source program office



approachableopensource.com



Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Downloads Node.js **v20.11.1**[1] with long-term support.
Node.js can also be installed via package managers.

Want new features sooner? Get **Node.js v21.7.1**[1] instead.

```
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World!\n');
});

// starts a simple http server locally on port 3000
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

Learn more what Node.js is able to offer with our Learning materials.

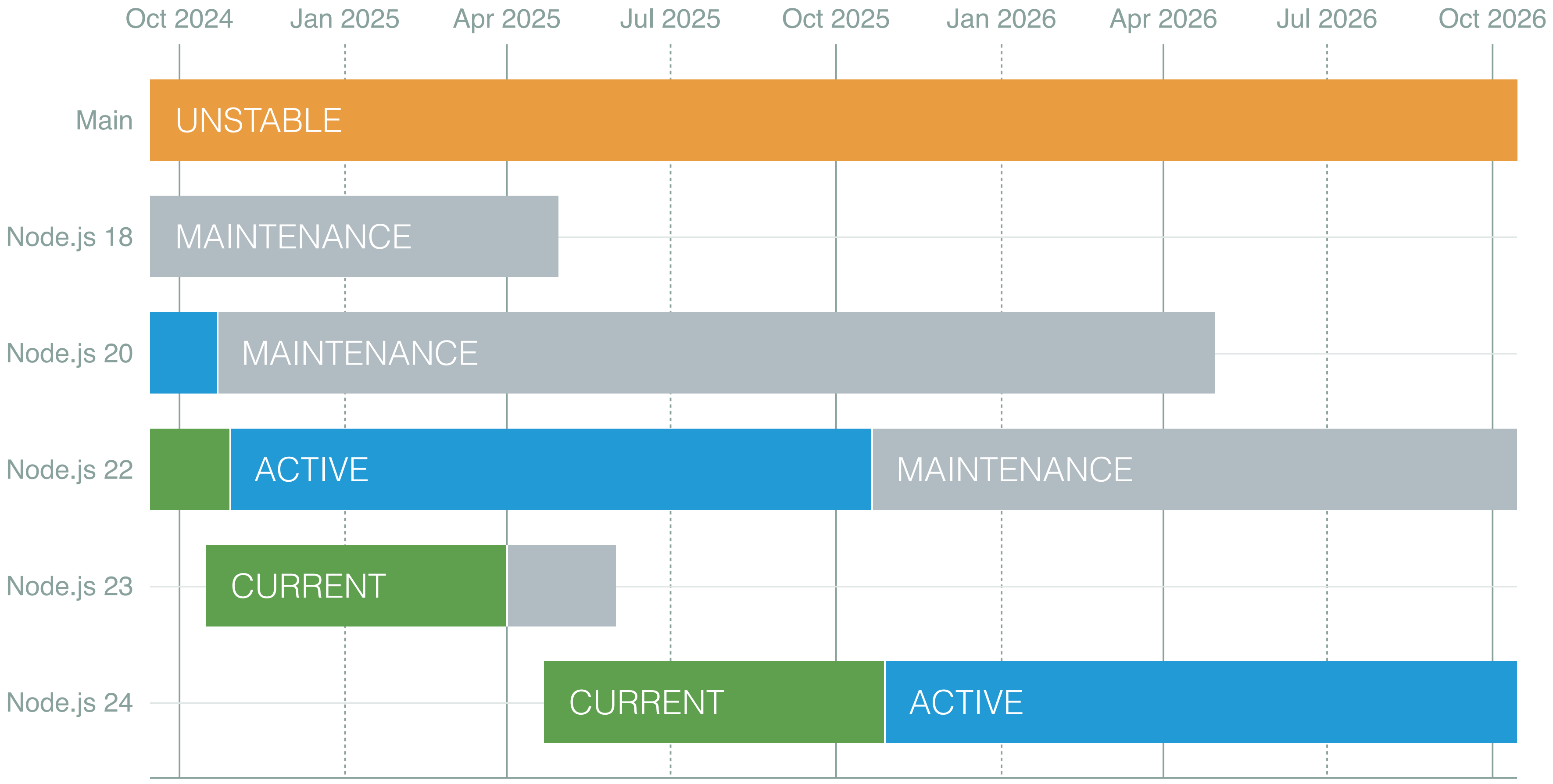
web-infra, moderation, triage

📣

Dependencies are great!

🆕 New(ish) Features

Feature	Introduced	Release Status
testing source code	16.17.0	Stable as of 20.0.0
watching source code	16.19.0	Stable as of 20.13.0
parsing arguments	18.3.0	Stable as of 20.0.0
reading environment	20.6.0	Active Development
styling output	20.12.0	Stable, as of 22.13.0
run scripts	22.0.0	Stable, as of 22.0.0
run typescript	22.6.0	Active Development
transform typescript	22.7.0	Active Development


Oct 2024
Jan 2025
Apr 2025
Jul 2025
Oct 2025
Jan 2026
Apr 2026
Jul 2026
Oct 2026
Main
UNSTABLE
Node.js 18
MAINTENANCE
Node.js 20
MAINTENANCE
Node.js 22
ACTIVE
MAINTENANCE
Node.js 23
CURRENT
Node.js 24
CURRENT
ACTIVE

Interestingly, these releases potentially replace an external dependency in your project.

Feature	Dependency Replaced
testing source code	jest, ava, ts-jest
watching source code	nodemon
parsing arguments	commander, yargs
reading environment	dotenv
styling output	colors, chalk
run typescript	ts-node, tsc

Interestingly, these releases potentially replace an external dependency in your project.

Feature	**Dependency Replaced**
testing source code	jest, ava, ts-jest
watching source code	nodemon
parsing arguments	commander, yargs
reading environment	dotenv
styling output	colors, chalk
run typescript	ts-node, tsc, 🌶️ deno, 🌶️ bun

The Rule of Least Power

TAG Finding 23 February 2006

This version:
http://www.w3.org/2001/tag/doc/leastPower-2006-02-23
Latest version:
http://www.w3.org/2001/tag/doc/leastPower
Previous versions:
http://www.w3.org/2001/tag/doc/leastPower-2006-2-13.html http://www.w3.org/2001/tag/doc/leastPower-2006-01-23.html http://www.w3.org/2001/tag/doc/leastPower-2005-12-20.html http://www.w3.org/2001/tag/doc/leastPower-2005-12-19.html
Editors:
Tim Berners-Lee, World Wide Web Consortium <timbl@w3.org>
Noah Mendelsohn, IBM Corporation <noah_mendelsohn@us.ibm.com>

This document is also available in these non-normative formats: XML.

Abstract

When designing computer systems, one is often faced with a choice between using a more or less powerful language for publishing information, for expressing constraints, or for solving some problem. This finding explores tradeoffs relating the choice of language to reusability of information. The "Rule of Least Power" suggests choosing the least powerful language suitable for a given purpose.

Status of this Document

This document has been produced by the W3C Technical Architecture Group (TAG). The TAG approved this finding at its 14 February 2006 teleconference. The text of this finding was adapted from Principle of Least Power in [Axioms]. Earlier drafts of this finding had the working title: The Principle of Least Power. Additional TAG findings, both accepted and in draft state, may also be available. The TAG may incorporate this and other findings into future versions of the [AWWW].

Please send comments on this finding to the publicly archived TAG mailing list www-tag@w3.org (archive).

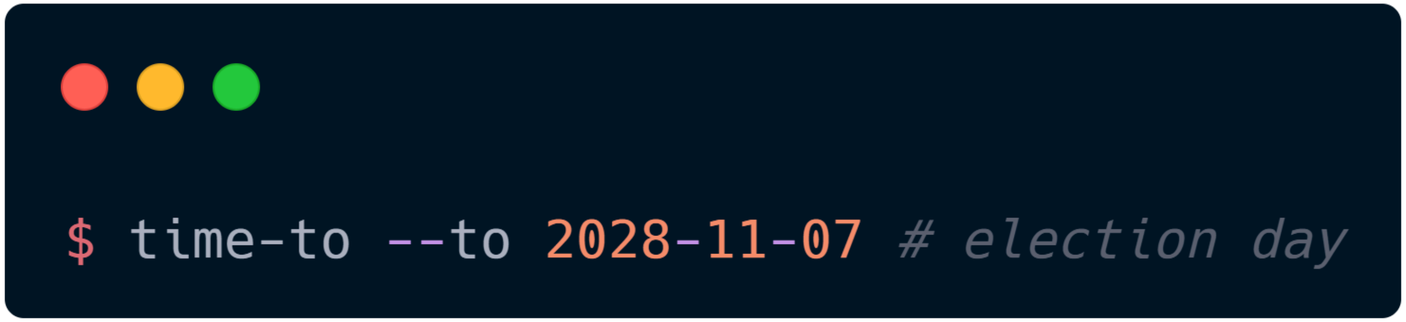
Table of Contents

1 Language Power and Information Reuse
2 Web Technologies and the Rule of Least Power
3 Scalable language families
4 References

1 Language Power and Information Reuse

The World Wide Web is unique in its ability to promote information reuse on a global scale. Information published on the Web can be flexibly combined with other information, read by a broad range of software tools, and browsed by human users of the Web. For such reuse to succeed, the broadest possible range of tools must be capable of understanding the data on the Web, and the relationships among that data. Thus, when publishing information or programs on the Web, the choice of language is important. This finding explores the connection between the choice of computer language and the reusability of information represented in that language.

Sample Project



```
$ time-to --to 2028-11-07 # election day
```

I've prepared a contrived CLI and server as a code sample of incremental migration.

🙉

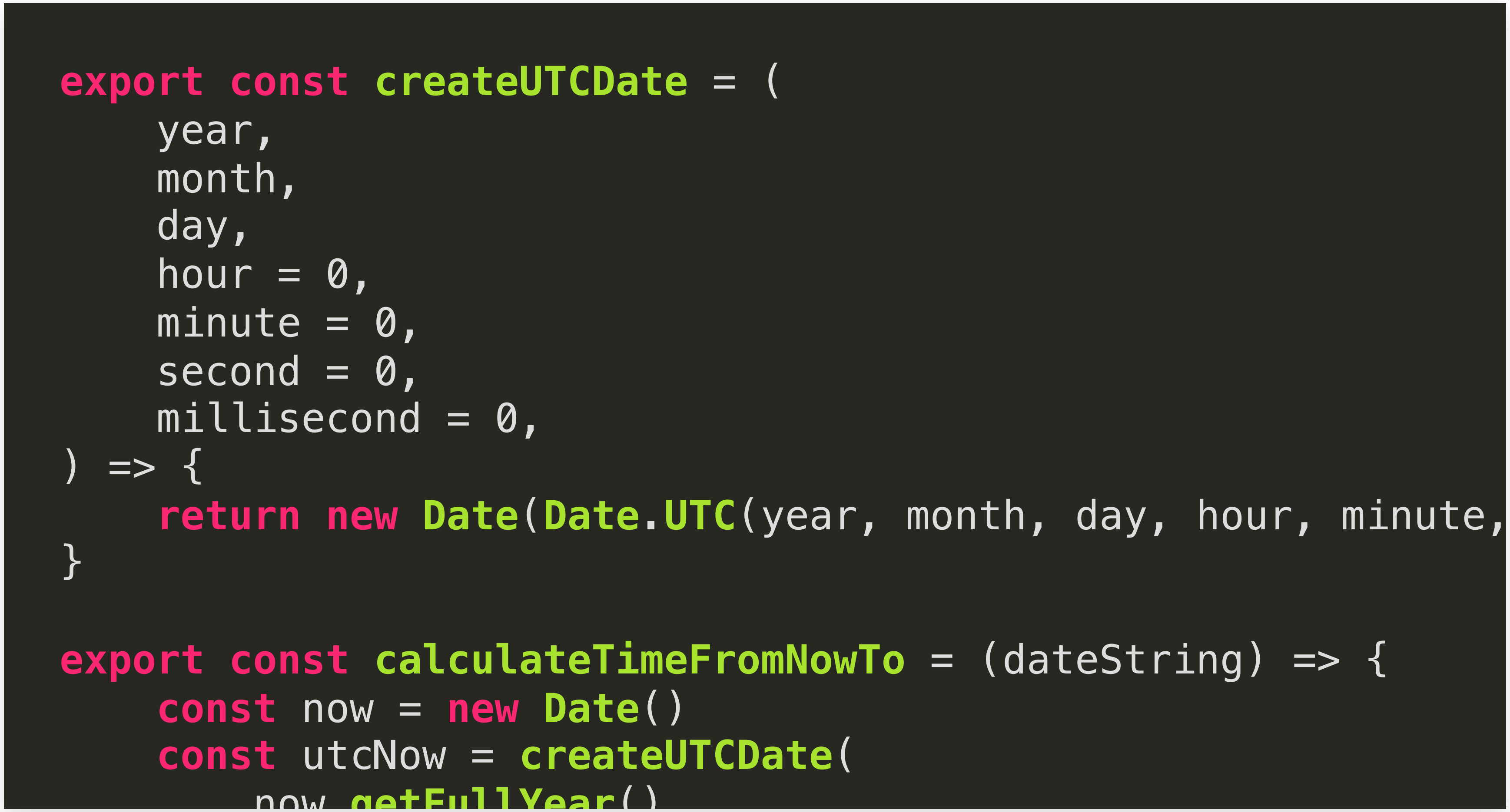
Timezones... say what?? The business logic isn't the star here.

🙉

Timezones... say what?? The business logic isn't the star here.

Let's get started!

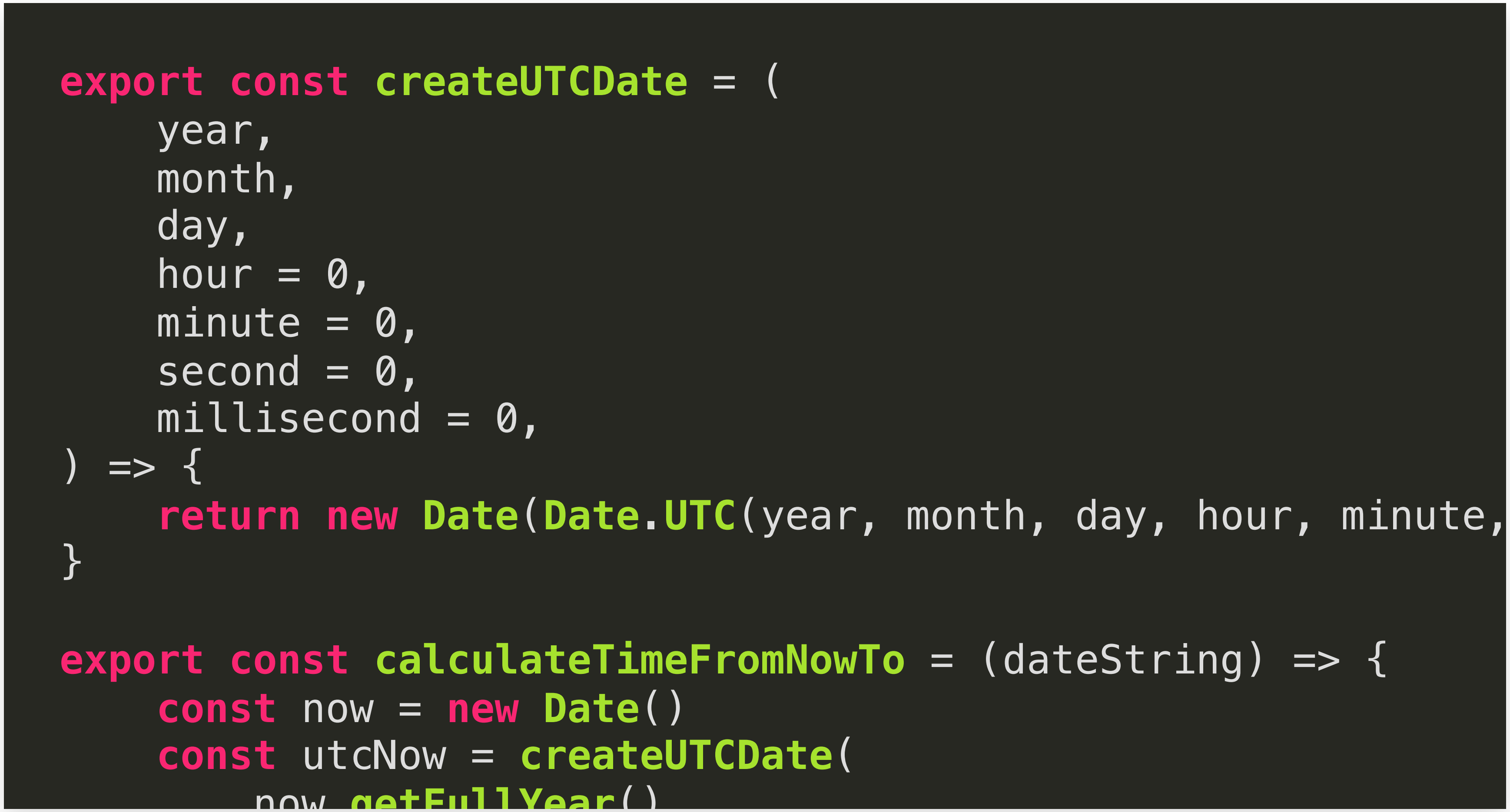
Core business logic ripped straight from an LLM cause they are good at generating garbage:



```javascript
export const createUTCDate = (
    year,
    month,
    day,
    hour = 0,
    minute = 0,
    second = 0,
    millisecond = 0,
) => {
    return new Date(Date.UTC(year, month, day, hour, minute,
}

export const calculateTimeFromNowTo = (dateString) => {
    const now = new Date()
    const utcNow = createUTCDate(
        now.getFullYear()
```

Core business logic ripped straight from an LLM cause they are good at generating garbage:



```
export const createUTCDate = (
    year,
    month,
    day,
    hour = 0,
    minute = 0,
    second = 0,
    millisecond = 0,
) => {
    return new Date(Date.UTC(year, month, day, hour, minute,
}

export const calculateTimeFromNowTo = (dateString) => {
    const now = new Date()
    const utcNow = createUTCDate(
        now.getFullYear()
```

these days we call that vibin'

You saw the CLI earlier. It also has a server.js file copy-pastaed from the Node.js homepage:

```
import { createServer } from 'node:http'

import { calculateTimeFromNowTo } from './lib/calculate.js'

const server = createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' })
    res.end(calculateTimeFromNowTo('2028-11-07')))
})

server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
})
```

Now, with `node src/server.js` one can visit http://localhost:3000 and see the same output.

🧪 Testing Source Code

Introduced 16.17.0

For many projects, I'd turn to jest to test my code.

🧪 Testing Source Code

Introduced 16.17.0

For many projects, I'd turn to jest to test my code.

It's been the default for so long, is part of the OpenJS Foundation, and enjoys a large ecosystem of tools and attention, making it hard to argue against.

We can test our `createUTCDate` function with this test:

```
import { createUTCDate } from '../calculate.js'

describe('createUTCDate', () => {
    it('should create a date in UTC time', () => {
        const date = createUTCDate(2026, 3, 20) // zero-inde
        expect(date.toISOString()).toEqual('2026-04-20T00:00
    })
})
```

And then run it with:

```
"test": "node --experimental-vm-modules node_modules/jest/bi
"test": "pnpm test:jest --watch",
```

And then run it with:

```
"test": "node --experimental-vm-modules node_modules/jest/bi
"test": "pnpm test:jest --watch",
```

Already there's trouble brewing...

Node.js now includes a built-in test runner,

`node --test`

improving with each successive release. We can replace the jest scripts with:

```
-"test": "node --experimental-vm-modules node_modules/jest/b
-"test": "pnpm test:jest --watch",
+"test": "node --test",
+"test:watch": "node --test --watch",
```

Here's a test diff:

```
+import { describe, it } from 'node:test' // no globals
import { createUTCDate } from '../calculate.js'

describe('createUTCDate', () => {
- it('should create a date in UTC time', () => {
+ it('should create a date in UTC time', (test) => {
        const date = createUTCDate(2026, 3, 20)
-       expect(date.toISOString()).toEqual('2026-04-20T00:00
+       test.assert.strictEqual(date.toISOString(),'2026-04-
    })
})
```

I'm not interested in the code golf here, but it is worth emphasizing two things:

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.

1. Jest's support for ESM is still evolving (pinned issue since 2020), not yet with a polished developer experience.
2. Jest is slower, even with one test. Benchmarking via `time pnpm test` against both showed the Node.js test runner to be 2.2 times faster.

👀 Watching Source Code

Introduced 16.19.0

It's a common convenience to have tasks re-run when as your source code changes during development.

👀 Watching Source Code

Introduced 16.19.0

It's a common convenience to have tasks re-run when as your source code changes during development.

Stopping and restarting the server each time is a pain.

Many frameworks and tools have this built in.

If not, a common choice is nodemon. Our sample project contains this script in our `package.json`:

```
"dev": "nodemon --watch src src/server.js"
```

Many frameworks and tools have this built in.

If not, a common choice is nodemon. Our sample project contains this script in our `package.json`:

```
"dev": "nodemon --watch src src/server.js"
```

Works great, and has for a long time.

But now, Node.js has built-in arguments `--watch` and `--watch-path`.

We can replace this with:

```diff
-"dev": "nodemon --watch src src/server.js",
+"dev": "node --watch-path src src/server.js",
```

Not much different at the surface. It works for this use case.

Not much different at the surface. It works for this use case.

Critically, it survives parsing errors in the JavaScript.

💬 Parsing Arguments

Introduced 18.3.0

Our server and CLI should accept the date to calculate the "time until".

```
node src/index.js --to 2027-03-21 # my 40th birthday 💀
```

Node.js supports `process.argv` since `0.1.27`
so what's the fuss?

Node.js supports `process.argv` since `0.1.27`
so what's the fuss?

Tools like yargs and commander have been the go-to for a long time.

Our arguments start by being parsed with yargs like this:

```
import yargs from 'yargs'
import { hideBin } from 'yargs/helpers'

const values = yargs(hideBin(process.argv))
.option('to', {
    type: 'string',
    description: 'Date string to measure time until'
  })
.parseSync()
```

But we can simplify.

```diff
+import { parseArgs } from 'node:util'

-import yargs from 'yargs'
-import { hideBin } from 'yargs/helpers'

-const values = yargs(hideBin(process.argv))
-.option('to', {
-    type: 'string',
-    description: 'Date string to measure time until'
-  })
-.parseSync()

+const { values } = parseArgs({
+ strict: true,
+ options: {
+   to: {
```

Lops off the two first arguments by default, synchronous by default, and enough for my needs.

Lops off the two first arguments by default, synchronous by default, and enough for my needs.

Node.js also throws an error for missing or extra params - which is nice - and again, perhaps enough.

Lops off the two first arguments by default, synchronous by default, and enough for my needs.

Node.js also throws an error for missing or extra params - which is nice - and again, perhaps enough.

Need something stronger? You already know where to look, or could try a newcomer like bubbletea.

🌲 Reading Environment

Introduced 20.6.0 | Active Development

Environment variables provide flexibility and portability to code.

The obvious choice for years and years has been dotenv.

One would likely reach for this wherever they want to read env:

```
import 'dotenv/config'

const { PORT } = process.env
...
```

But Node.js can do this too!.

Delete that dotenv import.

We add this to our `package.json` dev script:

```diff
-"dev": "node --watch-path src src/server.js",
+"dev": "node --env-file=.env --watch-path src src/server.js
```

We get multiple file support with overrides and a familiar enough syntax to dotenv files.

We get multiple file support with overrides and a familiar enough syntax to dotenv files.

Node.js can also error or gracefully handle missing env files, the choice is yours.

🖌️ Styling Output

Introduced 20.12.0 | Stable as of 22.13.0

Say we got that `server.js` start message. Maybe we wanna highlight the full URL that some terminals can click on.

The ubiquitous node_module chalk suffices:

```
const { PORT } = process.env
import chalk from 'chalk'
...
server.listen(PORT, '127.0.0.1', () => {
    console.log(
        `Listening on ${chalk.blue(chalk.underline(`http://1
    )
})
```

But look at this native Node.js code, quite new if I do say so myself:

```
const { PORT } = process.env
-import chalk from 'chalk'
+import { styleText } from 'node:util'

server.listen(PORT, '127.0.0.1', () => {
    console.log(
-    `Listening on ${chalk.blue(chalk.underline(`http://127.0
+    `Listening on ${styleText(['underline', 'blue'], `http:/
    )
})
```

📜 Run Scripts

Introduced 22.0.0 | Stable as of 22.0.0

Agnostic, faster script invocation.

Make your package.json a bit more portable with:

```diff
- "lint:fix": "pnpm lint --fix"
+ "lint:fix": "node --run lint -- --fix"
```

Executables must be in `/node_modules/.bin`

Executables must be in `/node_modules/.bin`

In benchmarking, its 200ms faster than `npm run`

Executables must be in `/node_modules/.bin`

In benchmarking, its 200ms faster than `npm run`

Does not run pre-and post- scripts

🎮 Checkpoint

Nice, we got through quite a bit.

But it's always smart to save before the big boss fight.

What's left?

What's left?

TypeScript.

🏗️ TypeScript

Introduced 22.6.0 | Active Development

🏗️ TypeScript

Introduced 22.6.0 | Active Development

Native TypeScript support has been a consistent community ask, and an obvious wedge issue.

We have a TypeScript error in that server code snippet.

We have a TypeScript error in that server code snippet.

Did you see it?

We have a TypeScript error in that server code snippet.

Did you see it?

How to start?

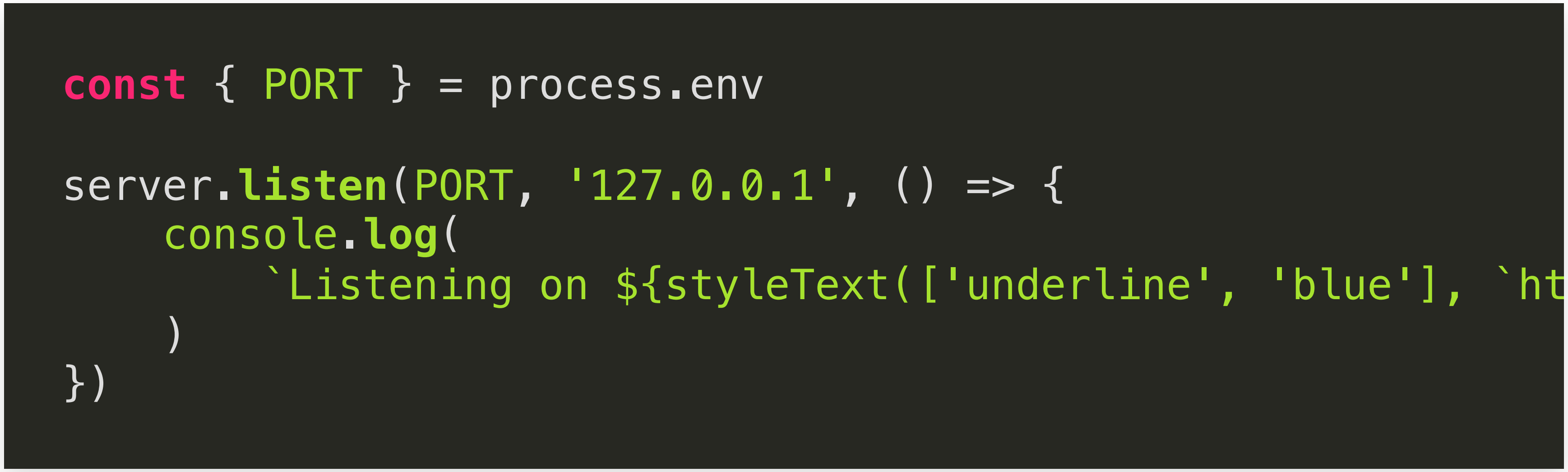
We have a TypeScript error in that server code snippet.

Did you see it?

How to start?

With a bold rename to server.ts

This is the code in question:



```
const { PORT } = process.env

server.listen(PORT, '127.0.0.1', () => {
    console.log(
        `Listening on ${styleText(['underline', 'blue'], `ht
    )
})
```

Hint…

```
const { PORT } = process.env // string | undefined, whoops

server.listen(PORT, '127.0.0.1', () => {
    console.log(
        `Listening on ${styleText(['underline', 'blue'], `ht
    )
})
```

Hint...

```
const { PORT } = process.env // string | undefined, whoops

server.listen(PORT, '127.0.0.1', () => {
    console.log(
        `Listening on ${styleText(['underline', 'blue'], `ht
    )
})
```

Missing overloads with our inferred typings.

Fixing this is easy enough:

```diff
-const { PORT } = process.env
+const PORT = Number(process.env.PORT)
```

To run it? We can replace the dev script with:

```diff
-"dev:node": "node --env-file=.env --watch-path src src/serv
+"dev:node": "node --experimental-strip-types --env-file=.en
```

And as of v23.6.0 in January, this runs without the flag at all!

```
node --env-file=.env --watch-path src src/server.ts
```

🤓

Reply guy: Well actually, you didn't build the project at all, it only removed the typings, flow-style. Sad!

🤓

Reply guy: Well actually, you didn't build the project at all, it only removed the typings, flow-style. Sad!

But wait, I say, my editor gave me immediate feedback of the error, without needing a build process at all.

Some notes:

- no type-stripping under node_modules

Some notes:

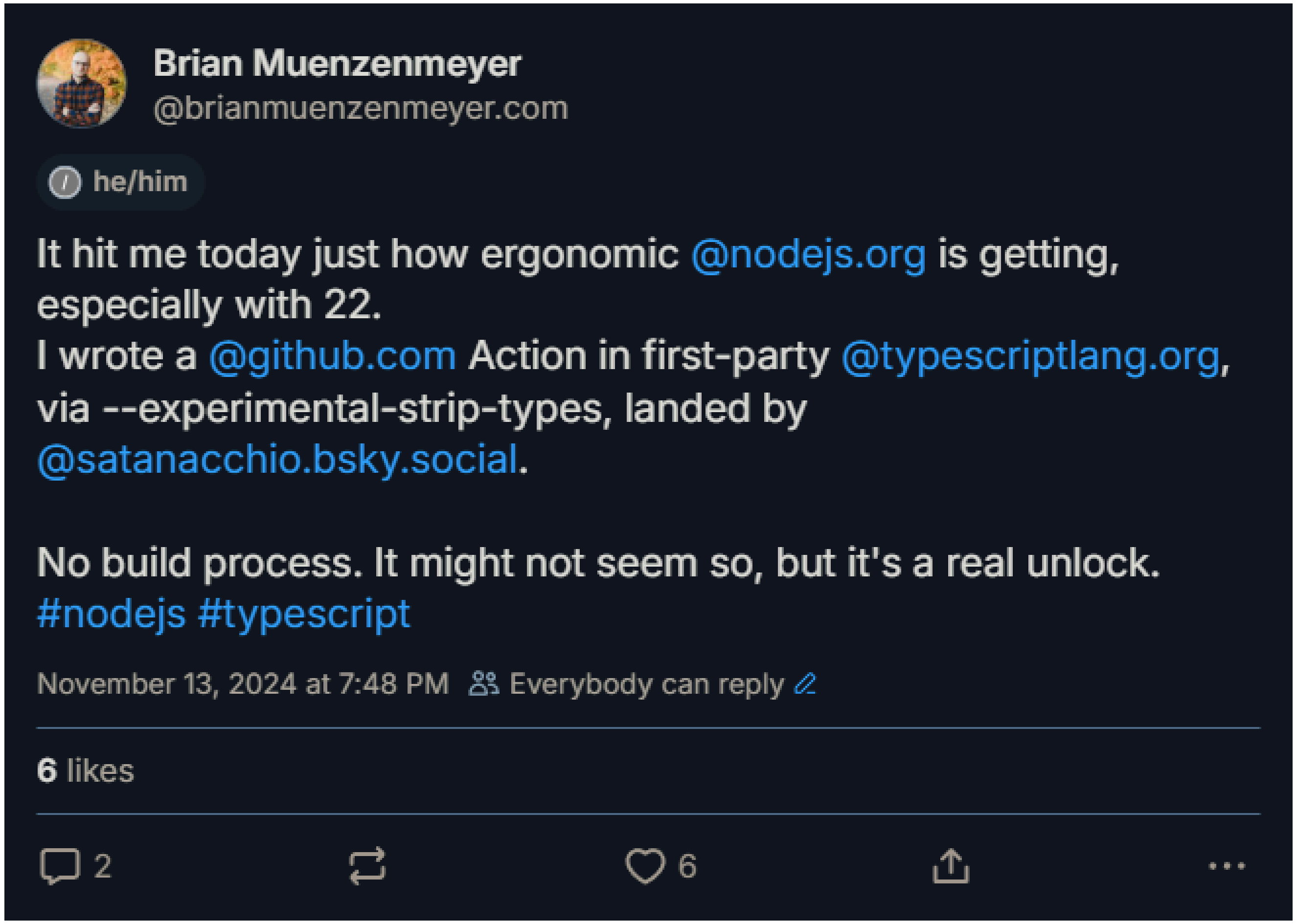
- no type-stripping under node_modules (yet)

Some notes:

- no type-stripping under node_modules (yet)
- `--experimental-transform-types`

Some notes:

- no type-stripping under node_modules (yet)
- `--experimental-transform-types`
- `--erasableSyntaxOnly` for TS@5.8



Brian Muenzenmeyer
@brianmuenzenmeyer.com

he/him

It hit me today just how ergonomic @nodejs.org is getting, especially with 22.
I wrote a @github.com Action in first-party @typescriptlang.org, via --experimental-strip-types, landed by @satanacchio.bsky.social.

No build process. It might not seem so, but it's a real unlock.
#nodejs #typescript

November 13, 2024 at 7:48 PM Everybody can reply

6 likes

2 6

🛑 STOP

Let's not go deeper today.

I'll spare you the ESM + TypeScript + Jest headache.

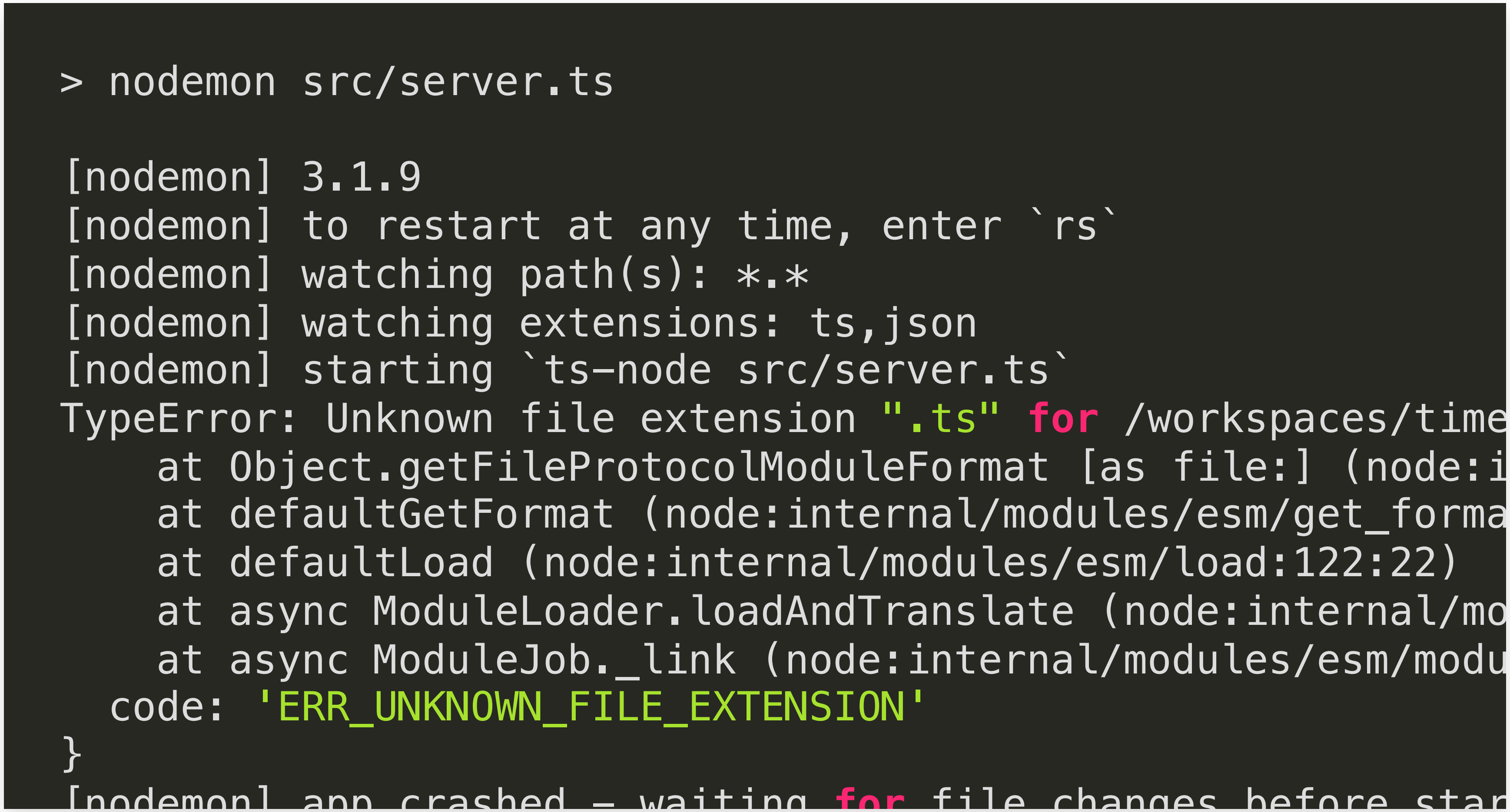
👉

Okay, well, can we do the same incremental running of our server.ts file with a dependency? Of course we can.

In fact, the ts-node and nodemon docs allude to the fact that this should just work:

```
"dev:nodemon": "nodemon --watch src src/server.js",
```

This was after temporarily dropping the watch glob, as the default is… `*.*` 🤝. What we uncover, however, is a problem lurking around the whole post, ESM.



```
> nodemon src/server.ts

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/server.ts`
TypeError: Unknown file extension ".ts" for /workspaces/time
    at Object.getFileProtocolModuleFormat [as file:] (node:i
    at defaultGetFormat (node:internal/modules/esm/get_forma
    at defaultLoad (node:internal/modules/esm/load:122:22)
    at async ModuleLoader.loadAndTranslate (node:internal/mo
    at async ModuleJob._link (node:internal/modules/esm/modu
  code: 'ERR_UNKNOWN_FILE_EXTENSION'
}
[nodemon] app crashed - waiting for file changes before star
```

Ugh. Googling around, this is potentially a "famous" problem with ESM + TypeScript. I won't even discuss Jest right now. I did get it working but we shouldn't mention it.

> 🙊 *I'm staying true to this process, so no, we aren't talking about B~~r~~un~~o~~ and Deno. That's not the point, yet. We're almost there, I promise.*

`tsx` I guess is maybe something? This worked:

```
"dev:nodemon": "nodemon --exec pnpm tsx src/server.ts"
```

⚖️ Comparisons

This ain't your entire app...

⚖️ Comparisons

This ain't your entire app...

...and we can all caveat this with enough asterisks to call in Legal.

⚖️ Comparisons

This ain't your entire app...

...and we can all caveat this with enough asterisks to call in Legal.

But numbers are numbers.

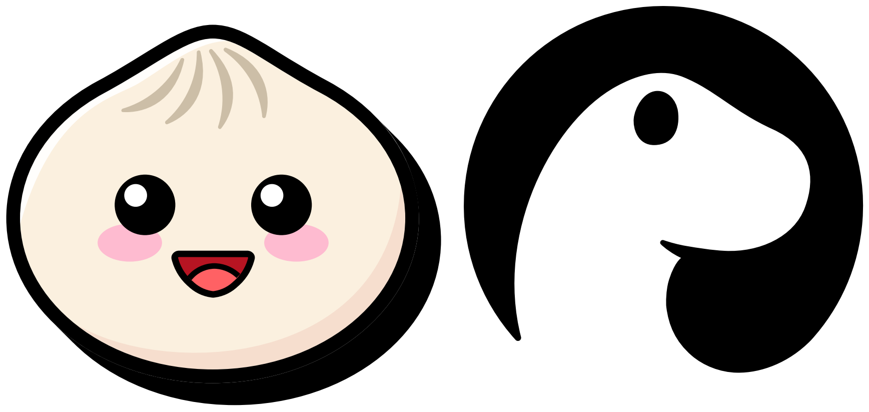
Metric	Before	After	Delta
# node_modules	215	2	0.9%, or 107 times smaller
size node_modules	49 MB	2.6 MB	5.3%, or 18 times smaller

🔬 *npm-built node_modules, omitting biome dev dependencies*

⚡ *All this, with two orders of magnitude less dependencies. Dependabot will be bored.*

node

JS®

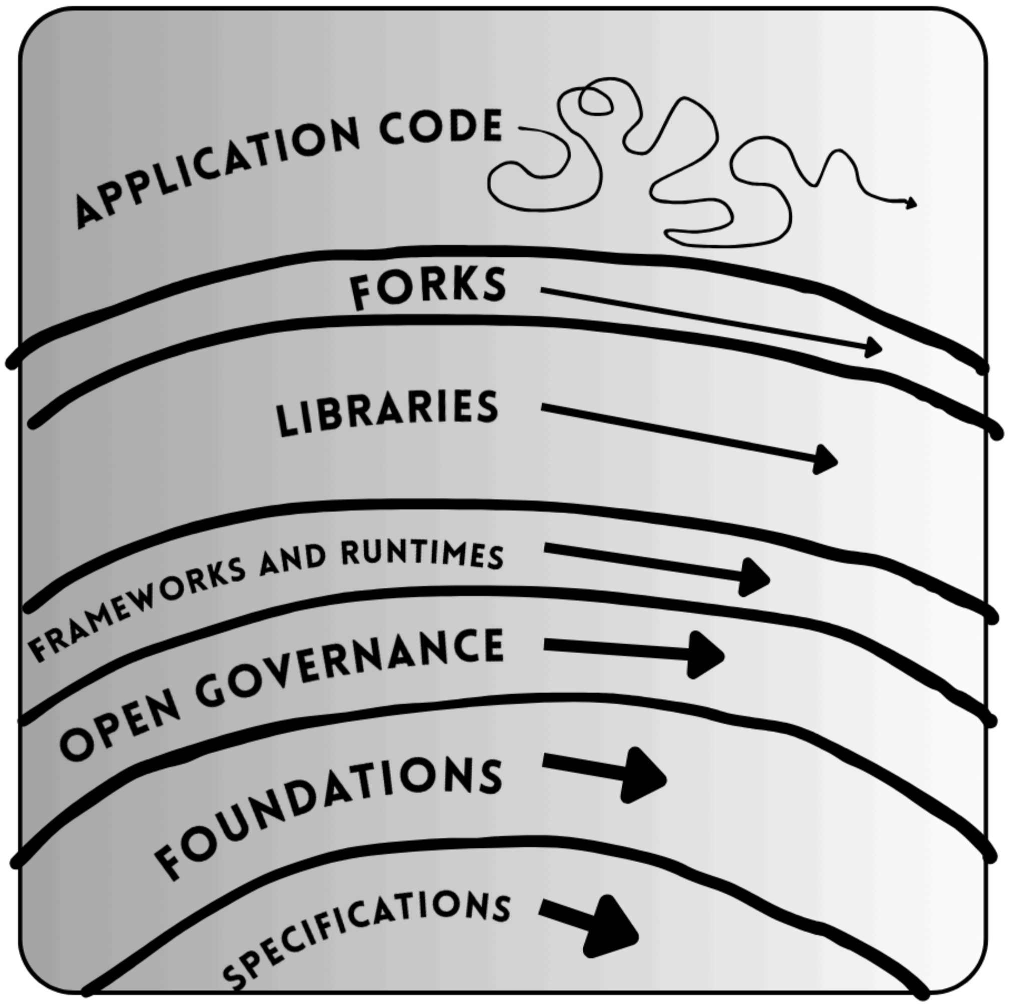


☀️ Pace Layers

symmathesy: together, learn

An entity composed by transcontextual mutual learning through interaction
— Nora Bateson

Open Source Pace Layers, v1.0.0



Source: Brian Muenzenmeyer, 2025, brianmuenzenmeyer.com and approachableopensource.com

🔥 *Churn is pace layers in motion.*

By design, we can celebrate that:

By design, we can celebrate that:

- innovation can be quick / creators have agency to explore

By design, we can celebrate that:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve

By design, we can celebrate that:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve
- maintainers craving momentum and stability have space and time to cultivate

By design, we can celebrate that:

- innovation can be quick / creators have agency to explore
- competition puts pressure on established systems to improve
- maintainers craving momentum and stability have space and time to cultivate
- layers exist for anyone to contribute within their means

“Fast learns; slow remembers.”

— Stewart Brand

Find your layer.

There's room for all of us.

Thanks