Writing Javascript without a build system

Hello! I've been writing some Javascript this week, and as always when I start a new frontend project, I was faced with the question: should I use a build system?

I want to talk about what's appealing to me about build systems, why I (usually) still don't use them, and why I find it frustrating that some frontend Javascript libraries require that you use a build system.

I'm writing this because most of the writing I see about JS assumes that you're using a build system, and it can be hard to navigate for folks like me who write very simple small Javascript projects that don't require a build system.

what's a build system?

The idea is that you have a bunch of Javascript or Typescript code, and you want to translate it into different Javascript code before you put it on your website.

Build systems can do lots of useful things, like:

- combining 100s of JS files into one big bundle (for efficiency reasons)
- translating Typescript into Javascript
- typechecking Typescript
- minification
- adding polyfills to support older browsers
- compiling JSX
- treeshaking (remove unused JS code to reduce file sizes)
- building CSS (like tailwind does)
- and probably lots of other important things

Because of this, if you're building a complex frontend project today, probably you're using a build system like webpack, rollup, esbuild, parcel, or vite.

Lots of those features are appealing to me, and I've used build systems in the past for some of these reasons: Mess With DNS uses `esbuild` to translate Typescript and combine lots of files into one big file, for example.

the goal: easily make changes to old tiny websites

I make a lot of small simple websites, I have approximately 0 maintenance energy for any of them, and I change them very infrequently.

My goal is that if I have a site that I made 3 or 5 years ago, I'd like to be able to, in 20 minutes:

- get the source from github on a new computer
- make some changes
- put it on the internet

But my experience with build systems (not just Javascript build systems!), is that if you have a 5-year-old site, often it's a huge pain to get the site built again.

And because most of my websites are pretty small, the *advantage* of using a build system is pretty small – I don't really need Typescript or JSX. I can just have one 400-line `script.js` file and call it a day.

example: trying to build the SQL playground

One of my sites (the sql playground) uses a build system (it's using Vue). I last edited that project 2 years ago, on a different machine.

Let's see if I can still easily build it today on my machine. To start out, we have to run `npm install`. Here's the output I get.

```
$ npm install
[lots of output redacted]
npm ERR! code 1
npm ERR! path /Users/bork/work/sql-playground.wizardzines.com/node_modules/grpc
npm ERR! command failed
npm ERR! command sh /var/folders/3z/g3qrs9s96mg6r4dmzryjn3mm0000gn/T/install-b52c96ad.sh
npm ERR! CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/surface/init.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/avl/avl.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/backoff/backoff.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/channel/channel_args.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/channel/channel_stack.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/channel/channel_stack_builder.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/channel/channel_trace.o
npm ERR!   CXX(target) Release/obj.target/grpc/deps/grpc/src/core/lib/channel/channelz.o
```

There's some kind of error building `grpc`. No problem. I don't really need that dependency anyway, so I can just take 5 minutes to tear it out and rebuild. Now I can `npm install` and everything works.

Now let's try to build the project:

```
$ npm run build
  ?  Building for production...Error: error:0308010C:digital envelope routines::unsupported
    at new Hash (node:internal/crypto/hash:71:19)
    at Object.createHash (node:crypto:130:10)
    at module.exports (/Users/bork/work/sql-playground.wizardzines.com/node_modules/webpack/lib/util/createHash.js:13
    at NormalModule._initBuildHash (/Users/bork/work/sql-playground.wizardzines.com/node_modules/webpack/lib/NormalMo
    at handleParseError (/Users/bork/work/sql-playground.wizardzines.com/node_modules/webpack/lib/NormalModule.js:467
    at /Users/bork/work/sql-playground.wizardzines.com/node_modules/webpack/lib/NormalModule.js:499:5
```

```
    at /Users/bork/work/sql-playground.wizardzines.com/node_modules/webpack/lib/NormalModule.js:356:12
    at /Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRunner.js:373:3
    at iterateNormalLoaders (/Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRun
    at iterateNormalLoaders (/Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRun
    at /Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRunner.js:236:3
    at runSyncOrAsync (/Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRunner.js
    at iterateNormalLoaders (/Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRun
    at Array.<anonymous> (/Users/bork/work/sql-playground.wizardzines.com/node_modules/loader-runner/lib/LoaderRunner
    at Storage.finished (/Users/bork/work/sql-playground.wizardzines.com/node_modules/enhanced-resolve/lib/CachedInpu
    at /Users/bork/work/sql-playground.wizardzines.com/node_modules/enhanced-resolve/lib/CachedInputFileSystem.js:79:
```

This stack overflow answer suggests running `export NODE_OPTIONS=--openssl-legacy-provider` to fix this error.

That works, and finally I can `npm run build` to build the project.

This isn't really that bad (I only had to remove a dependency and pass a slightly mysterious node option!), but I would rather not be derailed by those build errors.

for me, a build system isn't worth it for small projects

For me, a complicated Javascript build system just doesn't seem worth it for small 500-line projects – it means giving up being able to easily update the project in the future in exchange for some pretty marginal benefits.

esbuild seems a little more stable

I want to give a quick shoutout to esbuild: I learned about esbuild in 2021 and used for a project, and so far it does seem a more reliable way to build JS projects.

I just tried to build an `esbuild` project that I last touched 8 months ago on a new computer, and it worked. But I can't say for sure if I'll be able to easily build that project in 2 years. Maybe it will, I hope so!

not using a build system is usually pretty easy

Here's what the part of nginx playground code that imports all the libraries looks like:

```
<script src="js/vue.global.prod.js"></script>
<script src="codemirror-5.63.0/lib/codemirror.js"></script>
<script src="codemirror-5.63.0/mode/nginx/nginx.js"></script>
<script src="codemirror-5.63.0/mode/shell/shell.js"></script>
<script src="codemirror-5.63.0/mode/javascript/javascript.js"></script>
<link rel="stylesheet" href="codemirror-5.63.0/lib/codemirror.css">
<script src="script.js "></script>
```

This project is also using Vue, but it just uses a `<script src` to load Vue – there's no build process for the frontend.

a no-build-system template for using Vue

A couple of people asked how to get started writing Javascript without a build system. Of course you can write vanilla JS if you want, but my usual framework is Vue 3.

Here's a tiny template I built for starting a Vue 3 project with no build system. It's just 2 files and ~30 lines of HTML/JS.

some libraries require you to use a build system

This build system stuff is on my mind recently because I'm using CodeMirror 5 for a new project this week, and I saw there was a new version, CodeMirror 6.

So I thought – cool, maybe I should use CodeMirror 6 instead of CodeMirror 5. But – it seems like you can't use CodeMirror 6 without a build system (according to the migration guide). So I'm going to stick with CodeMirror 5.

Similarly, you used to be able to just download Tailwind as a giant CSS file, but Tailwind 3 doesn't seem to be available as a big CSS file at all anymore, you need to run Javascript to build it. So I'm going to keep using Tailwind 2 for now. (I know, I know, you're not supposed to use the big CSS file, but it's only 300KB gzipped and I really don't want a build step)

I'm not totally sure why some libraries don't provide a no-build-system version – maybe distributing a no-build-system version would add a lot of additional complexity to the library, and the maintainer doesn't think it's worth it. Or maybe the library's design means that it's not possible to distribute a no-build-system version for some reason.

I'd love more tips for no-build-system javascript

My main strategies so far are:

- search for "CDN" on a library's website to find a standalone javascript file
- use https://unpkg.com to see if the library has a built version I can use
- host my own version of libraries instead of relying on a CDN that might go down
- write my own simple integrations instead of pulling in another dependency (for example I wrote my own CodeMirror component for Vue the other day)
- if I want a build system, use esbuild

A couple of other things that look interesting but that I haven't looked into:

- this typescript proposal for type syntax in Javascript comments
- ES modules generally