Buildless JavaScript

February 17, 2023 · 4 minutes to read

I stumbled upon Julia Evans' recent blog post "Writing Javascript without a build system" and I can highly empathize with it. I too wanted to write JavaScript without a build system and bit rot is never fun. I would like to offer a few alternatives to Julia's approach, which I think already covers the basics.

I remember the first time I opened an old JavaScript project (my application project for my first React-related role) and it didn't build at all. Back then we didn't have widespread usage of lockfiles yet, and an incompatible dependency was the reason it didn't build. The situation is much better now than how it was before, but old projects still can and will fail to build for various reasons.

Bit rot is an especially important consideration for me, but I also like to have simpler projects with fewer moving parts. I remember hearing that we wouldn't need bundling with HTTP/2, but that future never came. If we could do less work during the build, we would waste less time and resources during CI.

Lastly, let's not forget that JavaScript is a build-less programming language by design, and we implemented many tools on top of it. Even though I like ensuring the quality of my code and making sure it's minimal and backward-compatible, I also yearn for a time when we could actually read the source code of a random website. [1]

If you must have a build step and want maximum reproducibility

Coincidentally, my first React job also introduced me to Nix, a reproducible build system and friends. [2] I know it has a steep learning curve, but the ecosystem has been growing and there are projects to make it easier. My current favorite way of building JavaScript projects is dream2nix.

Traditionally, Nix requires you to define all your dependencies using the Nix programming language. But dream2nix makes the process easier by reading your `package.json` and your lockfile to derive a Nix expression on-the-fly, so you don't have to write any Nix code yourself. There's a very nice quickstart you can follow, with a one-liner to initialize it in an existing project.

Simple ways to achieve a little bit more reproducibility

Let's say you have a simple Node.js application. You may have locked down your dependencies with a lockfile and avoided using any build steps. Your project still may fail to start, because Node.js and package managers have breaking changes from time to time, even when they try their best to keep backward-compatibility.

To not have any surprises in the future, it's a good practice to add an engines field to your `package.json` file to indicate which Node.js version is compatible with your project. Node.js will warn you if you use an unsupported version. You can also add a packageManager field to declare your preferred package manager (npm, yarn, pnpm, etc.) and its supported version. It looks something like this:

```json
{
  "engines": {
    "node": ">=14.0.0"
  },
  "packageManager": {
    "npm": ">=7.0.0"
  }
}
```

Note: Be warned that `packageManager` field is marked as experimental, but in my experience it seems to be working well. If you'd rather not use it, you can put your package manager and its version in the `engines` field instead.

If you want a modern frontend with no build step

If you want to build a modern frontend, but rather not spend minutes waiting for CI, I can highly recommend Deno (which is very anti-build-step) and one of its frameworks. My favorite is Fresh, which implements “islands architecture”, is very lightweight, and of course runs without a build step. You also get type safety for free, since Deno runs TypeScript code natively.

You can see Fresh as an alternative to metaframeworks like Next.js and Remix, with support for file-system based routing and server-side rendering. It does all that just-in-time, without a build step. By default, it doesn’t ship any JavaScript to the client, which is an interesting choice if you care about your payload size. It will definitely be a breath of fresh air if you’d like to try it in your next project, no pun(s) intended.

This topic is dear to my heart, as I have already written on this topic before. In “Do we really need node_modules or npm?” I talked about using UNPKG to manage dependencies and some ideas for the future. In “What I look forward to about web development in 2021” I have a whole section about bundling and CDNs capable of serving ES Modules with a hat tip to Deno.

I’m happy to see that some of these solutions became more stable and approachable. I can safely say that it’s possible to build websites without a build step, and if you need, there are robust solutions to keep them working for years into the future.

1. While sourcemaps ease the pain a little bit, it’s an optional solution and not all websites enable them.↩
2. It would take a series of blog posts to explain what Nix is, so I won’t get into that. Someday...↩

Share on Twitter · Edit on GitHub

Written by **Fatih Altinok**, who cares a lot about user experience, teamwork and functional programming.

← The Suspense is Killing Me: Part 2

Newsletter

Please subscribe to get new posts and monthly updates right to your inbox.

you@yoursite.com Subscribe

I promise I won't send you spam or sell your email.