Bryan Braun

A minimalist development workflow using ES modules and esinstall

Aug 27, 2021
Updated: Oct 12, 2023

When a new-ish web developer wants to start building a frontend JavaScript application these days, they typically reach for a project-starting tool—like Create React App or Parcel—to set up their codebase. This works and gets you running quickly, but it's at the cost of massive tooling complexity abstracted beneath the surface (if you don't believe me, check your `node_modules` directory)!

Fortunately, we can bypass this complexity by skipping the bundler and using ECMAScript modules in our apps directly (browser support for modules is good now). This is a big deal! As DHH recently said, "*An entire class of complexity stands at the precipice.*" The only problem is that lots of 3rd-party JavaScript packages continue to only export to CommonJS, making them incompatible with ES module apps.

In this post, I'll share a how we can use esinstall to pull in *any* 3rd-party package as an ES module, thus unlocking a minimalist, bundle-free development workflow for modern JavaScript apps.

Setup: HTML, CSS, and JavaScript

To get set up, we just need a basic HTML file that pulls in `main.js` as our JavaScript entrypoint:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Project</title>
</head>
<body>
  <h1>Hello World</h1>
  <script type="module" src="main.js"></script>
```

```
</body>
</html>
```

By declaring `type="module"` on our script tag, we're safe to use `import` and `export` statements in `main.js` without needing a bundler. 💪

Pulling in dependencies

To pull in dependencies, we'll create a basic `package.json` and list them under `dependencies` like normal. Here's an example, pulling in a `dom-confetti` package:

```
{
  "name": "esinstall-demo",
  "dependencies": {
    "dom-confetti": "^0.2.2"
  }
}
```

`dom-confetti` only exports to CommonJS, so we're not able to import it into our app yet. This is where `esinstall` comes in.

esinstall

`esinstall` is a development tool that converts any JavaScript package into a single ES6-module-friendly file that you can check into your codebase and `import`. You can run this conversion just once after install, so there's no need to set up file watching or other unnecessary tooling.

To set it up, we'll update our `package.json` like so:

```
{
  "name": "esinstall-demo",
  "scripts": {
    "build": "node install-modules.mjs"
  },
  "dependencies": {
    "dom-confetti": "^0.2.2"
  },
  "devDependencies": {
    "esinstall": "^1.1.7"
  },
  "esinstall": {
    "install": [
      "dom-confetti"
    ]
  }
}
```

This includes a few important changes:

We declared `esinstall` as a dev-dependency, so we can use it for our one-time builds.

We set up a new "esinstall" section, for listing any packages we want to install as ES modules.

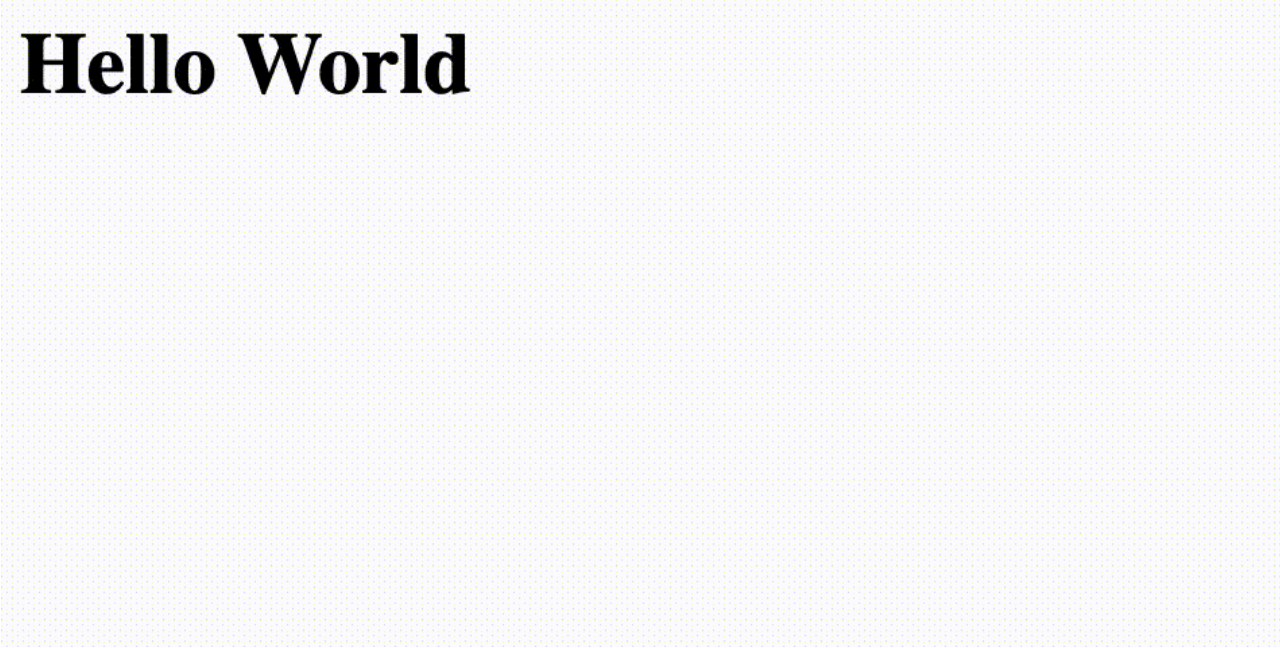
We created a `build` task for running the one-time build.

This `build` task runs a small node script that I've created called `install-modules.mjs`. All this script does is gets the list of modules from our package.json and runs them through `esinstall`:

```
// install-modules.mjs

import { install } from 'esinstall';
import { readFile } from 'fs/promises';

const json = JSON.parse(
  await readFile(
    new URL('./package.json', import.meta.url)
  )
);
const moduleList = json.esinstall.install;
const installOptions = json.esinstall.installOptions;

await install(moduleList, installOptions);
```

Now all we need to do is run `npm install && npm run build` and our `dom-confetti` package gets downloaded, converted to an ES Module, and dropped into a `web_modules` folder, ready for us to import:

```
// main.js

import { confetti } from "./web_modules/dom-confetti.js";

confetti(document.body.firstElementChild, {
  angle: "60",
  duration: "6000",
  stagger: "30"
});
```



Is that all we need?

Pretty much! Now you can organize everything in your codebase as modules and ship them straight to the browser bundle-free. If you have questions about how modules handle things like caching, performance, or browser support, feel free to check out my earlier post, ES modules in production: my experience so far.

Some final thoughts

One interesting thing about `esinstall` is that it was born out of the Snowpack bundler. Snowpack's big idea was to embrace cross-browser support for ES modules to speed up and simplify development. While Snowpack is no longer actively maintained, the Snowpack team decided to take its core functionality for converting code to ES modules, publish it as its own package, and name it `esinstall` .

Hopefully, our industry will get to a place where all 3rd-party JavaScript packages provide an ESM-formatted export, ready to use in in the browser. Until then, tools like `esinstall` can help bridge the gap without introducing unnecessary complexity.

Edit this post

Related posts: A minimalist development workflow using ES6 modules and Snowpack , ES modules in production: my experience so far , How searching for a bundle-free React led me to web components , Alt-React and Nonsense

---

Comments