Fullstack Development

Module

- Mechanisms for splitting JavaScript programs up into separate "pieces".
- When JavaScript modules first came into use, different developers created their own solutions.
  - Asynchronous Module Definition (AMD)
  - Universal Module Definition (UMD)
  - **CommonJS**
  - **ES Modules**

CommonJS vs ES Modules

CommonJS

- **Older** type of writing modules
- Popularized by `NodeJS`
  - `CommonJS` modules was the only supported style of modules in `NodeJS` up until version 12.

Using CommonJS

- You can mark your file as a `CommonJS` module by either
  - Naming it with the `.cjs` extension
  - Using `type: "commonjs"` in `package.json` (default)

CommonJS syntax

onefile.cjs

```
module.exports.add = function (a, b) {
  return a + b;
};
```

anotherfile.cjs

```
const { add } = require("./util");

console.log(add(5, 5)); // 10
```

EcmaScript (ES) Modules

- EcmaScript's standard way of writing modules.
  - Newer system
- Natively supported module style in browsers and all modern runtimes

Using ESM

- You can mark your file as a `ES` module by either
  - Naming it with the `.mjs` extension
  - Using `type: "module"` in `package.json`

ESM syntax

util.mjs

```
export function add(a, b) {
  return a + b;
}
```

app.mjs

```
import { add } from "./util.mjs"; // Note the extension is required.

console.log(add(5, 5)); // 10
```

Comparison

- Synchronous vs. asynchronous loading
  - CommonJS modules ➡️ synchronous loading ➡️ blocks execution
  - ESM modules ➡️ asynchronous loading ➡️ does not block execution.
- Static vs. dynamic imports
  - CommonJS ➡️ dynamic imports ➡️ less optimization
  - ESM ➡️ static imports ➡️ more optimization

Comparison

- Compatibility
  - CommonJS ➡️ server-side
  - ESM ➡️ server-side + web browsers
- More details

What about TypeScript?

- From TypeScript Handbook

> In TypeScript, just as in ECMAScript 2015, any file containing a top-level import or export is considered a module.

- Additional info
- *This means we are actually using ESM.*

TypeScript Output

- Let's take a look at https://github.com/fullstack-67/pf-backend
  - If you `npm run build` this project and inspect JavaScript files, you will see `CommonJS` module.
- The reason is in the `tsconfig.json`

```
{
  "compilerOptions": {
    "module": "commonjs"
  }
}
```

Problem

Start encountering packages that start to drop CommonJS support.

- `nanoid`
  - https://github.com/ai/nanoid/blob/main/CHANGELOG.md#40
- `@auth/express`
  - https://authjs.dev/reference/express#notes-on-esm

Let's do it the ~~right~~ (modern) way.

Steps

- `npm init -y`
- `pnpm install -D typescript @types/node nodemon`
- `pnpm install -D @tsconfig/node-lts` (Option)
- `pnpm install tsx`
  - Much better than `ts-node`, trust me.

Steps

- `package.json`

```
{
  "type": "module" <---- Note this change
}
```

- `tsconfig.json`
- `nodemon.json`
    - Notice that I used `tsx`, not `ts-nocde`
- Files in `src` folder

Notes

- Inspect linked `@tsconfig/node-lts/tsconfig.json`:

```
{
  "module": "node16"
}
```

- I have to import with `js` extension. *(What!)*

```
import { msg } from "@src/lib.js";
```

- Run `npm run build` and inspect the output to see ESM module.

Further

- `CommonJS` version
  - `git clone -b cjs https://github.com/fullstack-67/typescript-esm.git cjs`
- `nanoid` with ES module (*working*)
  - `git clone -b test-nanoid https://github.com/fullstack-67/typescript-esm.git test-nanoid`
- `nanoid` with CommonJS module (*not working*)
  - `git clone -b test-nanoid-cjs https://github.com/fullstack-67/typescript-esm.git test-nanoid-cjs`