Using TypeScript in a Project

Add TypeScript type checking and transpilation on a project level.

We can use TypeScript's CLI, `tsc`, to transpile and type check a TypeScript file. While installing `tsc` globally is convenient, it also means that we can only use one version of `tsc` whenever we'd like to transpile or type check a file. This presents a problem: as newer versions of `tsc` are released, we'll eventually have projects that require different versions of `tsc`.

To solve this problem, we can install `tsc` as a dependency inside each project. With this approach, all of our TypeScript projects will contain the version of `tsc` they need and they'll continue to work even when other projects require different versions of `tsc`.

Create a TypeScript Project

To set up TypeScript inside a project, we'll need to create a TypeScript project. If you set up the files from the Using TypeScript on the Command Line article, then you can skip down to the "Installing TypeScript Locally Inside a Project" section.

Run the following commands in the terminal to create a folder called `tsc-clock` and a file `clock.js`:

```
mkdir tsc-clock
touch tsc-clock/clock.ts
```

Then open up the file with VSCode or your editor of choice. Once open, we'll write a program that outputs the current time to the console. Copy and paste the following code in `clock.ts`:

```
function logTime(date: Date): void {
  console.log(`The time is ${date.toLocaleTimeString()}`);
}

logTime(new Date());
```

Installing TypeScript Locally Inside a Project

To install packages locally, let's initialize this project with `npm`:

```
npm init --yes
```

Then, install TypeScript as a dev dependency:

```
npm install --save-dev typescript
```

In `package.json` we'll see `typescript` installed under `devDependencies`.

To run the instance of `typescript` installed as a dependency of our project, we'll need to add a script to `package.json`. Under the `scripts` key in `package.json`, add the following `"tsc"` command:

```
{
  // ...
  "scripts": {
    "tsc": "tsc",
    // ...
  },
  // ...
}
```

By adding this `"tsc"` script, the `npm` command will find and run the instance of `tsc` installed in `node_modules`.

Configuring TypeScript

When we use `tsc` in the command line, we can pass CLI flags to customize how TypeScript will transpile and type check our code. Often, projects require many options. Running a command with many flags is cumbersome, so TypeScript allows us to encode all of its flags inside a configuration file named `tsconfig.json`. We can generate the default `tsconfig.json` with:

```
npm run tsc -- --init
```

Here's what this command accomplishes:

- `npm run tsc`: This runs the `"tsc"` script in `package.json`, which runs the instance of `tsc` installed as a dependency of our project.
- `--`: This allows us to pass flags to the `"tsc"` script in `package.json`.
- `--init`: This is TypeScript CLI's initialization flag, which will produce a `tsconfig.json` file.

After this command, we'll see a new file named `tsconfig.json`, which has a few options set by default. It also contains many options commented out with explanations about what they do. Learn more about the `tsconfig.json` file in our The `tsconfig.json` File article.

Running TypeScript

We're finally ready to transpile and type check our project. Run the following command in the terminal:

```
npm run tsc
```

After running this comand, we will see a transpiled `clock.js` file in our project.

Now, let's test out TypeScript's ability to type check our project. In `clock.ts`, change the `void` type to be a `number` type. The `number` type is intentionally incorrect so that we can see a type checking error.

```
function logTime(date: Date): number {
// ...
```

Then, run `npm run tsc` again. We will see the following output:

```
> tsc-clock@1.0.0 tsc
> tsc

clock.ts:1:31 - error TS2355: A function whose declared type is neither 'void' nor 'any' must
return a value.

1 function logTime(date: Date): number {
```

This output shows us an incorrect type in our project, as expected. We can change `number` back to `void` to fix the error.

Running `tsc` automatically

Now that we can check for type errors, we will want to check for errors frequently so that we can catch errors as we code. `tsc` has a mode that continuously checks for file changes and re-runs the `tsc` command when it detects a change, called a "watch mode". To start `tsc` in watch mode, we can run the following command in our terminal:

```
npm run tsc -- --watch
```

Once TypeScript is running in watch mode, the type-checking output will automatically update as we save changes to TypeScript files.

Wrap Up

Setting up TypeScript inside a project allows us to use a specific version of TypeScript in each project. While installing TypeScript locally inside a project takes more time, it will save us time in the future as newer versions of TypeScript are released. Now, we're ready to configure TypeScript inside for all of our future projects.