Day 4: TypeScript - Webpack Setup

Before starting with the following steps, let's have a brief overview of what we are going to do:

- We will be setting up a TypeScript/Node project with Webpack as the module bundler.
- TypeScript is a superset of JavaScript that adds static typing, which helps catch errors and provides better tooling support.
- Webpack is a powerful bundling tool that allows us to bundle our TypeScript code and its dependencies into a single file for deployment.

Step 1: Initializing the project

Start by opening a fresh, empty folder and running the following command in your terminal:

```
JavaScript
npm init -y
```

The `-y` flag accepts the default values for prompts, speeding up the process.

Step 2: Installing Webpack and Webpack CLI

Next, let's install Webpack and the Webpack CLI as dev dependencies:

```
JavaScript
npm install webpack webpack-cli --save-dev
```

This command installs both Webpack and the Webpack CLI. Webpack is a module bundler used to bundle our TypeScript code.

Step 3: Installing TypeScript and ts-loader

Now, let's install TypeScript and the ts-loader as dev dependencies:

```
JavaScript
npm install typescript ts-loader --save-dev
```

The ts-loader is a loader for Webpack that allows it to handle TypeScript.

Step 4: Creating the project structure

Inside your root directory, create a 'dist' and 'src' folder to organize your project:

```
JavaScript
mkdir dist src
```

The 'dist' folder will hold the bundled output for deployment, while the 'src' folder will contain your TypeScript source code.

Step 5: Creating an index.html file

Inside the 'dist' folder, create an index.html file:

```
JavaScript
touch dist/index.html
```

This file will serve as a starting point for your application.

Step 6: Linking the bundle script in the HTML file

Open the index.html file and add the following script tag inside the body:

```
JavaScript
<script src="bundle.js"></script>
```

This script tag will reference the bundle.js file, which will be created during the build process.

Step 7: Initializing TypeScript configuration

In your terminal, run the following command to initialize the TypeScript

configuration file:

```
JavaScript
npx tsc --init
```

If you are using macOS or facing issues with the global TypeScript installation, use `npx` to run `tsc`.

Step 8: Modifying the tsconfig.json file

Open the tsconfig.json file and make the following changes:

```
JavaScript

"moduleResolution": "node",
"module": "ES6",
"sourceMap": true,
"outDir": "./dist",
"allowJs": true
```

These changes configure TypeScript to use the node module resolution strategy, target ES6 modules, generate source maps, output the transpiled code to the 'dist' folder, and allow for JavaScript files in your TypeScript project.

Additionally, here are some best practice changes you can add:

JavaScript

```
"noEmitOnError": true,
"noImplicitAny": true,
"noUnusedLocals": true,
"noUnusedParameters": true,
"noImplicitReturns": true,
"noImplicitOverride": true
```

Step 9: Creating the webpack.config.js file

Create a file called webpack.config.js in the root directory of your project. You can find a sample configuration in the [Webpack TypeScript guide](https://webpack.js.org/guides/typescript/). Copy the contents of the "Webpack.config.js" code block and paste it into your webpack.config.js file.

Step 10: Installing additional development dependencies

Install the npm-run-all package as a dev dependency:

JavaScript

```
npm install npm-run-all --save-dev
```

This package allows us to run multiple npm scripts simultaneously.

Next, install the live-server package:

JavaScript

```
npm install live-server --save-dev
```

Step 11: Setting up package.json for live server and Webpack

Open your package.json file and replace the "scripts" section with the following:

JavaScript

```
"scripts": {
  "start": "npm run bundle && npm-run-all --parallel watch serve",
  "test": "echo \"Error: no test specified\" && exit 1",
  "watch": "webpack --watch",
  "serve": "cd dist && live-server",
  "bundle": "webpack"
}
```

This configuration sets up a start script that runs both the bundle and serve commands simultaneously. The watch script runs Webpack in watch mode, which automatically rebuilds your code on changes. The serve script navigates to the 'dist' folder and starts the live-server to launch your application. The bundle script runs Webpack to bundle your TypeScript code.

That's it! You have now set up your project with Webpack and TypeScript. You can start coding your TypeScript files in the 'src' folder and run the

project using the `npm start` command.