Minimal TypeScript React Project Template

A **Matt | Varghese Mathew** production
https://www.linkedin.com/in/matt-varghese-b609979/

Download current **PDF** of this tutorial at:

- https://github.com/mattvarghese/minimal-typescript-react-template/blob/main/documentation.pdf

Blog reference:

- https://mattvarghese-cs.blogspot.com/2022/01/minimal-typescript-react-project.html

I will appreciate (but not mandate) acknowledgement in derivative productions.

Submitting feedback

If you find any errors in this tutorial, or opportunities for improvement / enhancement, or additional things it can cover, or even if you found this tutorial useful, I will appreciate if you will kindly leave your feedback by writing up an issue at https://github.com/mattvarghese/minimal-typescript-react-template/issues

Especially, if you have recommendations on a better way of doing this, please share steps or pointers that I can consider incorporating into this tutorial!

Why?

I've seen developers commit entire node_module folders into source control!! And I have also realized that the create-react-app template is somewhat mysterious and a bit bloated?

What I am trying to do

In this exercise, I explain the process of creating a TypeScript React app, plugged into Visual Studio Code, and supporting active debugging in Visual Studio Code, from scratch, and which works both in Linux and Windows.

I think the mystery behind “create-react-app” baffles many developers. So I figured, I’ll learn how to do this from scratch, and having done that, I am documenting my observations in this tutorial.

I am an advocate of keeping things small and simple – the lean and minimalist approach to everything. Simple means fewer failure points. Likewise, the ability to understand how something works is important – it is not enough that I write React code and it is rendered magically in a website.

This tutorial walks you through creating a minimal React Web Application project from scratch without using the create-react-app template. It is split into 7 parts:

- Part 0 – Explaining how to use create-react-app for those who are absolutely new
- Part 1 – Creating a minimal JavaScript React App from scratch
- Part 2 – Switching from JavaScript to TypeScript
- Part 3 – Switching TypeScript build to skip JavaScript intermediate step
- Part 4 – Integrating into Visual Studio Code debugging capabilities
- Part 5 – Updating the build process for consistency
- Part 6 – Unit testing using Jest & Visual Studio Code

There is a companion Git Repository where the final outcome, and some of the intermediate steps are visible. Sorry, I only started the git repository partway through the effort, but the document should have everything, and the repository is only supplemental.

Whether you use create-react-app, or do things from scratch, knowing how to do it from scratch I hope will remove some of the mystery, and make you a better developer?!

License

Git Repository

If all you care is to get a minimal TypeScript React Template setup for use with Visual Studio Code in both Linux and Windows, you can get the result of this exercise from

https://github.com/mattvarghese/minimal-typescript-react-template

Downloading code

Have Git installed. Then

$ git clone https://github.com/mattvarghese/minimal-typescript-react-template.git

Open the resulting `minimal-typescript-react-template` folder in Visual Studio Code

To run development server

In Visual Studio Code (VSCode) > Terminal > Run Task > *npm: start*

This will pull node_modules and do necessary compilation and start development server. It will also open your default browser and navigate to the App

Be sure once you're done to use CTRL+C in the VSCode terminal to exit out of the development server.

To debug with VSCode

You have to be running development server to debug using VSCode. Follow steps above. Then,

Open App.tsx, and put a breakpoint inside the App function, or wherever you desire.

In VSCode, on the left margin, Click the tab/button for "Run and Debug (CTRL+SHIFT+D)"

On the top, just below the menu bar, select from the drop down for "Launch Chrome against localhost" and click the green arrow button.

Chrome will launch to the App URL, and your breakpoint will hit. When you click continue for your breakpoint, the App loads inside Chrome.

As called out, be sure to close the development server using CTRL+C in the VSCode terminal

Other browsers for debugging

Chromium browser has been added for support for raspberry pi where neither chrome nor edge are available. You must have chromium-browser installed to use this

```
sudo apt install chromium-browser
```

Microsoft Edge has also been added, because why not?

Unit Test debugging

In VSCode, set whatever breakpoints in your .test.tsx files as desired. Then, on the left margin, click the tab/button for "Run and Debug (CTRL+SHIFT+D)". On the top, just below the menu bar, select from the drop down for "Run tests using Jest" and click the green arrow button.

To Build and Run production

Do VSCode > Terminal menu > Run Build Task, **or** VSCode > Terminal Menu > Run Task > *npm: build*

This builds the code into ~/dist folder.

Then, to run the production server, do VSCode > Terminal Menu > Run Task > *npm: serve*

On newer windows versions, this might give you an error related to execution policy.

If so, run this command in the terminal before running serving

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

Clean Everything

To delete the node_modules folder, dist folder etc. and thereby do a cleanup of build, VSCode > Terminal menu > Run Task > *npm: clean*

You do not need to clean everything before every build / start / change etc. This option is there for when you need to package up files / clear dependencies and build etc.

Using without Visual Studio Code

Even though this project template is developed with the perspective of using with Visual Studio Code, you can still use it without VSCode. Do the following steps

- Download code. Then do the below commands from the root folder.
- Do `npm run start`, **or** `npm start`, to run the development server
- Do `npm run build` to build production
- Do `npm run` to serve production locally
- Do `npm test`, **or** `npm run test` to run unit tests
- Do `npm run clean` to clean up everything

To disconnect from GitHub repository

Delete the `.git` folder and the `.gitignore` file in the root directory.

Part 0: The “black-box” way to do this

To create a starter React Typescript project, use the command

```
$ npx create-react-app my-app --template typescript
```

Or

```
$ yarn create react-app my-app --template typescript
```

Reference: https://create-react-app.dev/docs/adding-typescript/

To make a minimal typescript app

```
$ npx create-react-app my-app --template minimal-typescript
```

Or

```
$ yarn create react-app my-app --template minimal-typescript
```

Reference: https://www.npmjs.com/package/cra-template-minimal-typescript
The difference is not actually in the node_modules used, but just the code organization, and skipping CSS etc

Issues

Permission Denied

This happens if NPM and NodeJS are installed globally.
In the user (not root), home folder, there is a .npm folder. However, this ends up with root ownership.
So the fix can be

- `$ rm -rf ~/.npm`
- Or `$ sudo chown -R $USER:$USER '/home/REPLACE_WITH_YOUR_USERNAME/.npm/'`

Reference: https://www.google.com/search?client=firefox-b-1-d&q=sh%3A+1%3A+create-react-app%3A+Permission+denied

NodeJS version is old in Linux

To update nodejs in linux, `apt update nodejs` doesn't fix this. Instead do

- `$ npm cache clean -f`
- `$ sudo npm install -g n`
- `$ sudo n stable` or `$ sudo n latest`

Reference: https://exerror.com/create-react-app-requires-node-14-or-higher-please-update-your-version-of-node/

Sample Output

Only important lines retained

```
ubuntu@mate-2110x64-play:~/Desktop/projects/react-typescript$ npx
create-react-app my-app --typescript
Installing react, react-dom, and react-scripts with cra-template...
Inside that directory, you can run several commands:
  npm start
    Starts the development server.
  npm run build
    Bundles the app into static files for production.
  npm test
    Starts the test runner.
  npm run eject
    Removes this tool and copies build dependencies, configuration
    files and scripts into the app directory. If you do this, you
    can't go back!
We suggest that you begin by typing:
  cd my-app
  npm start
```

```
npm notice Run npm install -g npm@8.4.0 to update!
```

If you do `$ npm run build`, you see this about serving locally

- `$ sudo npm install -g serve`
- `$ serve -s build`

Prerequisites

NodeJS

In Linux, you can

```
$ sudo apt install npm nodejs
```

In Windows, download from https://nodejs.org/en/download/

Visual Studio Code

Download from: https://code.visualstudio.com/download

In Linux, install .deb using `$ dpkg –configure filename.deb`

Part 1: Creating a JavaScript React app

In this section, I am shamelessly copying from https://dev.to/riddhiagrawal001/create-react-app-without-create-react-app-2lgd Refer to that page for more explanations etc.

$ npm init -y

- This creates package.json
- holds important information which is required before publishing to NPM, and also defines attributes of a project that is used by npm to install dependencies, run scripts, and identify the entry point of the project.

$ npm install react react-dom

- React-DOM is a glue between React and browser DOM
- Creates node_modules and package.lock.json

$ npm install --save-dev @babel/core @babel/preset-env @babel/preset-react babel-loader

- Babel is a JS compiler / JSX compiler
- Babel-loader transpiles JS files using Babel and webpack

Create .babelrc file

```
{
 "presets": [
        "@babel/preset-react",
        "@babel/preset-env"
            ]
}
```

$ npm install --save-dev webpack webpack-cli webpack-dev-server

- Webpack compiles / bundles JavaScript modules
- Webpack-CLI provides interface of options webpack uses

$ npm install --save-dev html-webpack-plugin style-loader css-loader file-loader

- More stuff for Webpack

Create webpack.config.js file

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "bundle.[hash].js",
    path: path.resolve(__dirname, "dist"),
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
    }),
  ],
  resolve: {
    modules: [__dirname, "src", "node_modules"],
    extensions: ["*", ".js", ".jsx", ".tsx", ".ts"],
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: require.resolve("babel-loader"),
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"],
      },
      {
        test: /\.png|svg|jpg|gif$/,
        use: ["file-loader"],
      },
    ],
  },
};
```

Create "src" folder, and create "src/App.js"

```
import React from "react";
```

```
const App = () => (
    <div>
        <h1>Hello React</h1>
    </div>
);

export default App;
```

Create "src/index.js"

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App/>,document.querySelector("#root"));
```

Create "src/index.html"

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>React</title>
</head>

<body>
    <div id="root"></div>
</body>

</html>
```

In package.json, replace scripts with this

```
"scripts": {
    "start": "webpack serve --mode development --hot --open",
    "build": "webpack --config webpack.config.js --mode
production"
  }
```

Run the App

Either npm start, or npm run start work - npm start seems to be a shortcut

```
$ npm start
```

Build the App

- `$ npm run build`
- `$ sudo npm install -g serve`
- `$ serve -s dist`

On newer windows versions, this might give you an error related to execution policy. If so, run this command in the terminal before running serve -s dist

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

Part 2: Adding TypeScript

Install relevant types

```
$ npm install --save typescript @types/node @types/react
@types/react-dom
```

Setup tsconfig.json

```
$ npx tsc --init --rootDir src --outDir build --esModuleInterop --
resolveJsonModule --module commonjs --allowJs true --noImplicitAny
true --sourceMap true --jsx react -t ES2016 --lib "ES2016","DOM"
```

Next, the tsconfig.json file created by the above command contains only the top level "compilerOptions" property. We need to also add another top level property "include" to tell typescript to only try to compile stuff inside the "src" folder. So at the end of compilerOptions, but not outside the main object, add:

```
,
"include": [
  "src"
]
```

Note: the comma on first line is to start next property after "compilerOptions"

Rename files

- src/index.js to src/index.tsx
- Src/App.js to src/App.tsx

Add TypeScript build task

In VSCode, Terminal > Configure Tasks

- Select "tsc: build - tsconfig.json"

This will compile .tsx files in "./src" to .js files in "./build"
The tasks.json file is generated inside folder .vscode

Update webpack.config.js to look in build

Our webpack.config.js right now looks in "src" for the .js files.
We need to update it to look in build folder, since that's where the TS compiler puts stuff.
There are three "src" references in webpack.config.js that need to be converted.
These are highlighted in the above code listing

Add NPM Build task

In VSCode, Terminal > Configure Tasks

- Select "npm: build / webpack --config webpack.config.js --mode production"
- In the new task that gets created in tasks.json, add a dependency on our Typescript build task
  "dependsOn": ["tsc: build - tsconfig.json"]

At this point, if you do VSCode > Terminal > Run Task..., webpack will complain that there is no index.html in build folder. This is because, while the typescript compiler created the .js files in build, index.html is still in src.

Add an additional task to copy files

In tasks.json, add another task to copy index.html from src, to build.
Note that because "build folder is created by the TypeScript build task, we need a dependency.

```
    {
        "label": "Copy Additional Files",
        "type": "shell",
        "command": "cp ./src/index.html ./build/",
        "windows": {
            "command": "copy .\\src\\index.html .\\build\\"
        },
        "dependsOn":["tsc: build - tsconfig.json"]
    }
```

Add a dependency to this task for npm: build.
You may also want to create a similar npm: start task from VSCode > Terminal > Configure Tasks
Your tasks.json will look like this

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "typescript",
            "tsconfig": "tsconfig.json",
            "problemMatcher": [
                "$tsc"
            ],
            "group": "build",
            "label": "tsc: build - tsconfig.json"
        },
        {
            "label": "Copy Additional Files",
            "type": "shell",
            "command": "cp ./src/index.html ./build/",
            "windows": {
                "command": "copy .\\src\\index.html .\\build\\"
            },
            "dependsOn": [
                "tsc: build - tsconfig.json"
            ]
        },
        {
            "type": "npm",
            "script": "build",
            "group": {
```

```
                "kind": "build",
                "isDefault": true
            },
            "problemMatcher": [],
            "label": "npm: build",
            "detail": "webpack --config webpack.config.js --mode
            production",
            "dependsOn": [
                "tsc: build - tsconfig.json",
                "Copy Additional Files"
            ]
        },
        {
            "type": "npm",
            "script": "start",
            "problemMatcher": [],
            "label": "npm: start",
            "detail": "webpack serve --mode development --hot --open",
            "dependsOn": [
                "tsc: build - tsconfig.json",
                "Copy Additional Files"
            ]
        }
    ]
}
```

Update code to use TypeScript features

src/App.tsx

```
import React from "react";

interface IProps {
    heading: string;
    body: string;
}

const App: React.FunctionComponent<IProps> = (props: IProps):
JSX.Element | null => {
    const { heading, body } = props;
    return (
        <div>
            <h1>{heading}</h1>
            <p>{body}</p>
        </div>
    );
};

export default App;
```

src/index.tsx

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(
    <App heading="Hello React" body="Hope you have a ton of
fun!"/>,
    document.querySelector("#root")
);
```

Part 3: Fixing TypeScript build

Right now, we're using as separate TypeScript compile step, and then chaining that through webpack as JavaScript. This has a problem that it doesn't work well with debugging.

Remove TypeScript Build Tasks

In tasks.json, remove the tasks

- tsc: build - tsconfig.json
- Copy Additional Files

We have also added a "Clean" task to this, and a Restore Node Modules task.
The two npm tasks now depend on "Restore Node Modules" and not the ones we removed.
Your tasks.json should look like this:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "npm",
            "script": "build",
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "problemMatcher": [],
            "label": "npm: build",
            "detail": "webpack --config webpack.config.js --mode
            production",
            "dependsOn":["Restore Node Modules"]
        },
        {
            "type": "npm",
            "script": "start",
            "problemMatcher": [],
            "label": "npm: start",
```

```
            "detail": "webpack serve --mode development --hot --
            open",
            "dependsOn":["Restore Node Modules"]
        },
        {
            "label": "Clean All",
            "type": "shell",
            "command": "rm -rf dist/ node_modules/",
            "windows": {
                "command": "Remove-Item -Path dist -Force -
                Recurse; Remove-Item -Path node_modules -Force -
                Recurse"
            },
            "problemMatcher": []
        },
        {
            "label": "Restore Node Modules",
            "type": "shell",
            "command": "./restore-node-modules.sh",
            "windows": {
                "command": ".\\restore-node-modules.bat"
            },
            "problemMatcher": []
        }
    ]
}
```

The Restore Scripts are

restore-node-modules.sh

```
if [ ! -d "./node_modules" ]; then
    npm install react react-dom
    npm install --save-dev @babel/core @babel/preset-env
@babel/preset-react babel-loader
    npm install --save-dev webpack webpack-cli webpack-dev-server
    npm install --save-dev html-webpack-plugin style-loader css-
loader file-loader
    npm install --save typescript @types/node @types/react
@types/react-dom
    npm install --save-dev ts-loader
fi
```

resotre-node-modules.bat

```
if not exist "node_modules" (
    npm install react react-dom
    npm install --save-dev @babel/core @babel/preset-env
@babel/preset-react babel-loader
    npm install --save-dev webpack webpack-cli webpack-dev-server
```

```
    npm install --save-dev html-webpack-plugin style-loader css-
loader file-loader
    npm install --save typescript @types/node @types/react
@types/react-dom
    npm install --save-dev ts-loader
)
```

Install ts-loader

```
$ npm install --save-dev ts-loader
```

Update webpack.config.js to support TypeScript

In webpack.config.js,

- Change all build references back to src
- Update entry property to ./src/index.tsx
- In module/rules, add an entry for tsx:

```
      {
        test: /\.tsx?$/,
        exclude: /node_modules/,
        use: "ts-loader",
      },
```

Your webpack.config.js should now look like this.

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const path = require("path");

module.exports = {
  entry: "./src/index.tsx",
  output: {
    filename: "bundle.[hash].js",
    path: path.resolve(__dirname, "dist"),
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
    }),
  ],
  resolve: {
    modules: [__dirname, "src", "node_modules"],
    extensions: ["*", ".js", ".jsx", ".tsx", ".ts"],
  },
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        exclude: /node_modules/,
        use: "ts-loader",
      },
      {
```

```
            test: /\.jsx?$/,
            exclude: /node_modules/,
            loader: require.resolve("babel-loader"),
          },
          {
            test: /\.css$/,
            use: ["style-loader", "css-loader"],
          },
          {
            test: /\.png|svg|jpg|gif$/,
            use: ["file-loader"],
          },
        ],
      },
    };
```

At this point, the "npm: build" task, and the "npm: start" task should work correctly.

Part 4: Debugging

launch.json

In order to debug, go to the debugging tab on the far left of VSCode
Click "Run and Debug" button. This suggests you to pick what debugging type - choose Chrome.
This introduces a launch.json in your .vscode folder.
Note: in the github version of this project, additional configurations have been added for launching edge, and for launching chromium browser – the latter to support raspberry pi, where neither chrome nor edge are available.
Reference: https://askubuntu.com/questions/1048540/using-vs-code-with-chromium-snap

First try

At this point, you can put a breakpoint in your App.tsx
Now, run the "npm: start" task. This launches dev server and firefox. Keep dev server running.
Then go to debugging tab, hit the green run button which should day "Launch Chrome"
The app runs correctly, but our breakpoint doesn't hit. This is because, our webpack bundled javascript doesn't include source maps.

Add sourcemap to webpack.config.js

Note - we only want to add sourcemap when running development server. So we do this. Also note that we're changing the port on which webpack serves from automatic (8080) to 3000.

```
    const HtmlWebpackPlugin = require("html-webpack-plugin");
    const path = require("path");
```

```
var config = {
  entry: "./src/index.tsx",
  devServer: {
    port: 3000
  },
  output: {
    filename: "bundle.[hash].js",
    path: path.resolve(__dirname, "dist"),
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
    }),
  ],
  resolve: {
    modules: [__dirname, "src", "node_modules"],
    extensions: ["*", ".js", ".jsx", ".tsx", ".ts"],
  },
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        exclude: /node_modules/,
        use: "ts-loader",
      },
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: require.resolve("babel-loader"),
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"],
      },
      {
        test: /\.png|svg|jpg|gif$/,
        use: ["file-loader"],
      },
    ],
  },
};

module.exports = (env, argv) => {
  if (argv.mode === "development") {
    config.devtool = "inline-source-map"
  }

  if (argv.mode === "production") {
    // Nothing special
  }
```

```
  return config;
};
```

This adding of properties based on mode works, because if you look at your tasks.json, you'll see that mode is a parameter in each task. The build task specifies production, the start task specifies development.

Update your launch.json

Because we changed the port, we need to update the port from 8080 to 3000 in your launch.json. The launch.json should look like this.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "pwa-chrome",
            "request": "launch",
            "name": "Launch Chrome against localhost",
            "url": "http://localhost:3000",
            "webRoot": "${workspaceFolder}"
        }
    ]
}
```

Second Try

Now,

- Set a breakpoint in App.tsx
- Terminal > Run Task > npm: start
    - Leave the development server running. Up to you if you close firefox or not
- Go to the debug tab on the left, hit the green "Launch Chrome" button

This time, your breakpoint should hit!

Part 5: Using a consistent build model

After all of the above, I realized that I'm using a mix of different things for the VSCode tasks vs. the npm scripts. I want this project to be usable with and without VSCode. So I decided to make the following updates to the tasks.json and package.json, so everything is managed by npm/node, and VSCode just calls into these.

Install serve locally

We will install serve locally, so that we are not dependent on serve being globally installed, and can more easily package the application for deployment on docker etc.

```
$ npm install -D serve
```

npm install

We don't really need .sh files and .bat files for restoring node modules. So delete these two files. Rather, running `npm install` from the command line will do this for us, since all necessary information is already in the package.json.

Add a clean-all.js

There doesn't seem to be a quick command for cleaning everything. So we do still need some custom handling. And we need OS specific handling for linux vs. windows. For this, we add a file named clean-all.js at the root folder, that looks like this:

```
var console = require('console');
var exec = require('child_process').exec;
var os = require('os');

function puts(error, stdout, stderr) { console.log(stdout) }

// Run command depending on the OS
if (os.type() === 'Linux')
   exec("rm -rf dist node_modules", puts);
else {
    exec("rmdir /s /q node_modules", puts);
    exec("rmdir /s /q dist", puts);
}
```

Updated package.json

See this updated package.json. We have scripts for start, build, serve, and clean

```
{
  "name": "my-app2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "npm install && webpack serve --mode development --
hot --open",
    "build": "npm install && webpack --config webpack.config.js -
-mode production",
```

```
    "serve": "npm run build && serve -s dist",
    "clean": "node clean-all.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@types/node": "^17.0.13",
    "@types/react": "^17.0.38",
    "@types/react-dom": "^17.0.11",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "typescript": "^4.5.5"
  },
  "devDependencies": {
    "@babel/core": "^7.16.12",
    "@babel/preset-env": "^7.16.11",
    "@babel/preset-react": "^7.16.7",
    "babel-loader": "^8.2.3",
    "css-loader": "^6.5.1",
    "file-loader": "^6.2.0",
    "html-webpack-plugin": "^5.5.0",
    "serve": "^13.0.2",
    "style-loader": "^3.3.1",
    "ts-loader": "^9.2.6",
    "webpack": "^5.67.0",
    "webpack-cli": "^4.9.2",
    "webpack-dev-server": "^4.7.3"
  }
}
```

Using this from the command line

Now once you git clone the repository, you can directly run the dev server by

```
$ npm start or $ npm run start
```

And you can directly run the production app by

```
$ npm run serve
```

You can also build production without running by

```
$ npm run build
```

To clean everything

```
$ npm run clean
```

Updated tasks.json

Now that we have updated, npm scripts, the VSCode > Terminal > Configure Tasks option will allow you to create wrapper tasks around these – you will see these options, if you clear out your existing task.json before trying.

So using that, we update our tasks.json to look like this:

```
{
    // Reference: https://code.visualstudio.com/docs/editor/tasks
     "version": "2.0.0",
     "tasks": [
          {
               "type": "npm",
               "script": "install",
               "problemMatcher": [],
               "label": "npm: install",
               "detail": "install dependencies from package"
          },
          {
               "type": "npm",
               "script": "serve",
               "problemMatcher": [],
               "label": "npm: serve",
               "detail": "npm run build && serve -s dist"
          },
          {
               "type": "npm",
               "script": "clean",
               "problemMatcher": [],
               "label": "npm: clean",
               "detail": "node clean-all.js"
          },
          {
               "type": "npm",
               "script": "build",
               "group": {
                    "kind": "build",
                    "isDefault": true
               },
               "problemMatcher": [],
               "label": "npm: build",
               "detail": "npm install && webpack --config
webpack.config.js --mode production"
          },
          {
               "type": "npm",
               "script": "start",
```

```
                "problemMatcher": [],
                "label": "npm: start",
                "detail": "npm install && webpack serve --mode
development --hot --open"
            }
        ]
}
```

Using this in VSCode

Now, to run the development server, do

VSCode > Terminal > Run Task > npm: start

After you have development server running, you can debug using Chrome/Edge/Chromium as before.

To serve the production build, do

VSCode > Terminal > Run Task > npm: serve

To build production without serving

VSCode > Terminal > Run Task > npm: build

To clean everything

VSCode > Terminal > Run Task > npm: clean

Part 6: Setting up Jest Unit testing

Figuring this out was a pain in the backside, given there are a lot of ways to do unit testing, and many folks don't try to integrate with VSCode debugging but just leave it at `npm test`.

Install Dependencies

$ npm install -D ts-node ts-jest @jest/types

Add a jest.config.ts file like below

- The moduleNameMapper avoids issues with importing css or data files inside react components
- The test environment makes sure we can use document.***() html DOM methods

```
import type {Config} from '@jest/types';
```

```
const config: Config.InitialOptions = {
  verbose: true,
  preset: "ts-jest",
  testEnvironment: "jsdom",
  moduleNameMapper: {

"\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a
|aac|oga)$": "<rootDir>/src/tests/__mocks__/fileMock.ts",
    "\\.(scss|sass|css)$": "<rootDir>/src/tests/__mocks__/styleMock.ts"
  }
};
export default config;
```

Add the mock files referenced above

Make a “tests” folder under src. Tests will be in this folder. Make a “__mocks__” folder under it for these files.

Reference: https://jestjs.io/docs/webpack

fileMock.ts

```
const mockResult = "file-stub";
export default mockResult;
```

styleMock.ts

```
const mockResult = {};
export default mockResult;
```

Update your package.json test script

It should now be:

```
"test": "npm run build && jest",
```

Add App.test.tsx file

Place this file in src/tests. Contents could be something like:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "../App";

describe("App", () => {
    let container: HTMLDivElement;
```

```
    beforeEach(() => {
        container = document.createElement("div");
        document.body.appendChild(container);
        ReactDOM.render(
            <App heading="Hello React" body="Hope you have a ton of
fun!"/>,
            container
        );
    });

    afterEach(() => {
        document.body.removeChild(container);
        container.remove();
    });

    // These tests below are rather trivial and flaky
    // There are here only to verify that the template supports unit
testing

    it("containes heading Hello React", () => {
        const h1Element = document.getElementsByTagName("h1");
        expect(h1Element.item(0)?.firstChild?.textContent).toBe("Hello
React");
    });

    it("contains paragraph Hope you have a ton of fun!", () => {
        const pElement = document.getElementsByTagName("p");
        expect(pElement.item(0)?.firstChild?.textContent).toBe("Hope
you have a ton of fun!");
    });
});
```

This is using DOM APIs for the test. This is perhaps inefficient, and you could find better frameworks for unit testing.

Run the test using npm

At this point, if you run `npm test` **or** `npm run test`, your test should run and pass.

Integrate with VSCode deugging

Go to the “Run and Debug (CTRL+SHIFT+D)” section on the far left. In the drop down under the menu, choose “Add Configuration”. This will take you to launch.json, and show a menu for the configuration to add. Select “Jest: Default jest configuration”

This default configuration will work for us. However, you might want to give it a better “name”. I chose, “Run tests using Jest”.

Additionally, I added a `"preLaunchTask": "npm: build"` to this configuration, so if the application is not yet built, it will build it before testing.

Note: You can do similarly for the browser launch tasks for debugging, so you don't have to run the start task before debugging. I don't want to do this, so I won't forget to quit the development server after debugging.

Now, you can set a breakpoint in App.test.tsx, and hit the "Run tests using Jest" button in "Run and Debug" section. Your tests will run, and your breakpoint will hit.

References

Webpack with TypeScript: https://webpack.js.org/guides/typescript/

JavaScript App: https://dev.to/riddhiagrawal001/create-react-app-without-create-react-app-2lgd

VSCode tasks: https://code.visualstudio.com/docs/editor/tasks

Debugging: https://www.youtube.com/watch?v=tC91t9OvVHA

Resolving "document" reference / DOM APIs in TypeScript: https://stackoverflow.com/questions/41336301/typescript-cannot-find-name-window-or-document

- tsc -t ES2016 --lib "ES2016","DOM" ./your_file.ts

Using VSCode with Chromium Snap: https://askubuntu.com/questions/1048540/using-vs-code-with-chromium-snap

OS Specific NPM Scripts: https://stackoverflow.com/questions/45082648/npm-package-json-os-specific-script and https://bugzilla.redhat.com/show_bug.cgi?id=1735361

Configuring Jest: https://jestjs.io/docs/configuration

Working around imported CSS in Jest:

- https://jestjs.io/docs/webpack
- https://stackoverflow.com/questions/39418555/syntaxerror-with-jest-and-react-and-importing-css-files

NPM Notes

If you don't know the specific version, you can do @latest to the package
To update react-scripts in the app

- `$ npm install react-scripts@latest`

To list all globally installed stuff

- `$ npm ls -g`

Example

```
root@mate-2110x64-play:/home/ubuntu# npm ls -g
/usr/local/lib
├── @types/node@17.0.10
├── corepack@0.10.0
├── n@8.0.2
├── npm@8.4.0
├── serve@13.0.2
└── yarn@1.22.17
```

The difference between npm install, npm install -g, and npm install -D

- `$ npm install` : installs locally under dependencies. Used for dev and production.
- `$ npm install -D` : installs locally under devDependencies. Used only for dev.
- `$ npm install -g` : installs globally

See package.json

Copyright (C) 2022 Varghese Mathew (Matt) | Distributed under GNU GENERAL PUBLIC LICENSE Version 3