Blog Reference

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

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 on first run. Then it will 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, click the green "Launch Chrome" 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

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.

If you haven't installed "serve" yet, do so by

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

Then, to actually run the production server, do

```
$ serve -s dist
```

Inside the root folder of the app / from the VSCode terminal.

Clean Everything

To delete the node_modules folder, dist folder etc. and thereby do a cleanup of build, VSCode > Terminal menu > Run Task > Clean All

Why?

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

The create-react-app way

The create-react-app way to create a TypeScript React Application is

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

Or

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

More details in Part 0: The "magic" way to do this below

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 have seen developers commit entire node_modules folders into source control. That is terrifying! And I think the mystery behind “create-react-app” is part of what contributes to this kind of behavior(??). So I figured, I’ll learn how to do this from scratch, and having done that, I am documenting my observations in this tutorial / blog post / document.

License

Part 0: The “magic” 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`

Part 2: Adding TypeScript

Install relevant types

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

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 @types/jest
    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 @types/jest
    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.

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!

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

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
```