# Setting up a project to be built as an SPA

© Prashanth Puranik, www.digdeeper.in

## Basic project setup

- Create the project folder. Rest of steps have to be run from terminal(s) opened in the project folder.
- Create package.json

```
npm init -y
```

- For a Git-based project, make sure you avoid pushing node_modules (any any other folders like dist) to the online Git repository (GitHub, BitBucket etc.) by creating a `.gitignore` file. You can easily create one for a Node JS-based project (one where dependencies are managed by npm etc.) using the Node package which is also called `gitignore`. Since this tool is required only once to setup the `.gitignore` file, you can run it using `npx`.

```
npx gitignore node
```

- Install `http-server` server

```
npm i -D http-server
```

- Create the project structure with required folders and files ( `public/index.html`, `src/ts`, `src/scss`, `src/ts/index.ts` )
- Write the start script in `package.json`

```
"scripts" : {
  ...,
  "start": "http-server ./public -c-1 -p 4000 --proxy http://localhost:4000?"
}
```

- Serve the app

```
npm start
```

- Create files and folders as per need within the `src` folder

## Setting up Webpack, TypeScript, Sass

- Install dependencies

  ```
  npm i -D typescript sass webpack webpack-cli ts-loader style-loader css-loader sass-loader html-w
  ```

- Create a TypeScript configuration file `tsconfig.json`

  ```
  ./node_module/.bin/tsc --init
  ```

- Set up the `tsconfig.json` to have these options

  ```
  {
  "compilerOptions": {
    "target": "es5",
    "lib": [ "ES2015", "DOM" ],
    "module": "es2015",
    "sourceMap": true,
    "rootDir": "./src/ts",
    "outDir": "./dist",
    "noEmitOnError": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "skipLibCheck": true,
    "moduleResolution": "node"
  },
  "exclude": [
        "node_modules"
   ]
  }
  ```

- Add webpack.config.js with configuration for entry, output, module rules, resolving extensions, injecting bundle into HTML etc.

  ```
  const path = require( 'path' );
  const HtmlWebpackPlugin = require( 'html-webpack-plugin' );
  module.exports = {
    mode: 'development',
    entry: './src/ts/index.ts',
    output: {
        path: path.join( __dirname, 'public/dist' ),
        filename: '[name].bundle.js',
        clean: true
    },
    module: {
        rules: [
            {
                test: /\.s?css$/i,
                use: [
                    'style-loader',
                    'css-loader',
                    'sass-loader'
                ]
            },
            {
                test: /\.ts$/i,
                use: 'ts-loader',
  ```

```
                exclude: /node_modules/
            }
        ]
    },
    resolve: {
        extensions: [ '.ts', '.js' ]
    },
    plugins: [
        new HtmlWebpackPlugin({
            filename: './index.html',
            template: './public/index.html',
            inject: true,
            chunks: [ 'main' ]
        })
    ]
};
```

- Setup compilation using webpack ( `build` script in package.json)

  ```
  "scripts" : {
    ...,
    "build": "webpack"
  }
  ```

- Include the bundle in `index.html` . Make sure to use the `defer` attribute if the script tag is in the head section of the HTML page. This makes sure that the script executes only after the DOM has been created and HTML elements are thus available.

  ```
  <script src="dist/js/main.bundle.js" defer></script>
  ```

- Build the bundle everytime you make code changes

  ```
  npm run build
  ```

## Setting up the SPA using MVC architecture

- Structure app as MVC - models, view (templates in `index.html` ), controller classes for every page
- Setup `index.ts` to manage the SPA routing. Setup the routes as per the templates and controller classes created.

## Setting up Unit Testing using Jest (with Babel)

First setup Babel which is required for TS compilation, by Jest. Then setup Jest.

### Setting up Babel

- Install Babel

  ```
  npm i -D @babel/cli @babel/core
  ```

- Install the required Babel plugins and presets

  ```
  npm i -D @babel/plugin-transform-regenerator @babel/plugin-transform-runtime @babel/plugin-propos
  ```

- Create a `.babelrc` file

  ```
  {
    "plugins": [
        "@babel/plugin-transform-regenerator",
        "@babel/plugin-proposal-class-properties",
        "@babel/plugin-transform-runtime"
    ],
    "presets": [
        "@babel/preset-env",
        "@babel/preset-typescript"
    ]
  }
  ```

## Setting up Jest

- Install `babel-jest` for Jest to work with Babel which helps compile TS to JS files.

  ```
  npm i babel-jest
  ```

- Install Jest related dependencies. However we shall install version 28.x.x of `jest` as `jest-environment-jsdom` seems to work with v28 of Jest only currently (please check if this is the case or not currently). `@types/jest` if a TypeScript type definition file. It helps VSCode understand functions like `test`, `describe` etc. of Jest.

  ```
  npm i -D jest@28 jest-environment-jsdom @types/jest
  ```

- Create a `jest.config.js` file by running the following command. Select options as mentioned below.

  ```
  ./node_modules/.bin/jest --init
  ```

  **NOTE**: Use backslashes on Windows if you use Cmd prompt instead of Git Bash
- Choose the following from the choices
  ✓ Would you like to use Jest when running "test" script in "package.json"? › n ✓ Would you like to use Typescript for the configuration file? … no
  ✓ Choose the test environment that will be used for testing › jsdom (browser-like)
  ✓ Do you want Jest to add coverage reports? … yes
  ✓ Which provider should be used to instrument code for coverage? › babel
  ✓ Automatically clear mock calls, instances, contexts and results before every test? › y
- Add this line to `jest.config.js` - this helps transform test .js and .ts files using babel-jest

  ```
  transform: {
    "\\.[jt]s$": "babel-jest"
  }
  ```

- Change the `exclude` section of `tsconfig.json` to have this ignore test related files

  ```
  {
  "compilerOptions": {
    ...
  },
  "exclude": [
        "node_modules",
        "**/*.test.ts",
        "src/ts/mocks/*.ts",
        "src/ts/setupTests.ts"
   ]
  }
  ```

### Adding stubs for CSS/SCSS module imports

- CSS and SCCS imports that are enabled in a JS/TS file by Webpack, will fail when run through Jest tests. We need to provide "stubs" for it. A stub substitutes import of some module(s) with other modules(s). We have a Node package called `identity-obj-proxy` that helps set up a stub. Through this setup we can have Jest import a dummy JS/TS file wherever a CSS/SCSS file import is encountered (this package can be used to setup stubs for other kinds of files as well, but in this app we include only SCSS files apart from TS files).
- First install the required package

  ```
  npm i -D identity-obj-proxy
  ```

- Next add this option to `jest.config.js`

  ```
  module.exports = {
    ...,
    moduleNameMapper: {
        "^.+\\.(css|scss)$": "identity-obj-proxy"
    },
    ...
  }
  ```

### Running the tests

- Add the `test` script to package.json which reports test coverage as well.

  ```
  "scripts": {
    ...,
    "test": "jest --coverage"
  },
  ```

- Run tests from time-to-time to check if your code changes are not breaking any existing functionality. Fix any breaking code.

  ```
  npm test
  ```

### Setting up mocking for HTTP requests using msw

- Since Jest runs in a Node environment, fetch API is not available. We install a *polyfill* for fetch (polyfill is a script that adds a mising functionality in a runtime environment).

  ```
  npm i -D whatwg-fetch
  ```

- We *mock out* the backend, and make sure the real backend is not requested in API calls. For this an HTML5 *service worker* based package called `msw` may be used.

  ```
  npm i -D msw
  ```

- Create a `ts/mocks` folder with handlers.js, server.js, and any mock data
- Create `ts/setupTests.js` for code that starts, stops the mock server, and clears any HTTP request mock overrides before/after a test file runs

  ```
  import 'whatwg-fetch';
  import server from './mocks/server';
  beforeAll( () => server.listen() ); // runs before the first test in a test file runs
  afterEach( () => server.resetHandlers() ); // runs after each test in a test file runs
  afterAll( () => server.close() ); // runs after the last test in a test file runs
  ```

- This file be set up to run before every test using this setting in `jest.config.js`

```
module.exports = {
  ...,
  setupFilesAfterEnv: [
      './src/ts/setupTests.ts'
  ],
  ...
}
```

- Add handlers for various fetch API requests made in the code you test in `handlers.js` . Make sure to send response that mimics the data sent, for the API request, by the actual backend. Choose appropriate response status code, response body etc.

## Setting up and using ESLint

- Install ESLint

  ```
  npm i --save-dev eslint
  ```

- Since we are setting up ESLint for linting TypeScript files as well, we need to install the following (not required if we choose to lint only for JavaScript files in the next step).

  ```
  npm i --save-dev @typescript-eslint/parser @typescript-eslint/eslint-plugin
  ```

- Create the ESLint configuration file.

  ```
  ./node_modules/.bin/eslint --init
  ```

  Choose the following options during setup.

  ```
  @eslint/create-config
  Ok to proceed? (y) y
  ✔ How would you like to use ESLint? - __To check syntax and find problems__
  ✔ What type of modules does your project use? - __JavaScript modules (import/export)__
  ✔ Which framework does your project use? -  __None of these__
  ✔ Does your project use TypeScript? - __Yes__
  ✔ Where does your code run? - __browser__
  ✔ What format do you want your config file to be in? - __JSON__
  The config that you've selected requires the following dependencies:
  @typescript-eslint/eslint-plugin@latest @typescript-eslint/parser@latest eslint@latest
  ? Would you like to install them now with npm? - __Yes__
  ```

  **NOTE**: Use backslashes on Windows if you use Cmd prompt instead of Git Bash

- Create a `.eslintignore` file. Make sure to ignore all code except the ones you created as part of the app. This shall be the contents of the file.

```
node_modules
public/dist
jest.config.js
webpack.config.js
*.test.ts
```

**Reference**: Check https://eslint.org/docs/user-guide/configuring/ignoring-code#ignorepatterns-in-config-files for file path patterns (glob patterns) to specify files and folders to ignore

- Add a lint script to package.json. Also add one to automatically fix where possible the lint issues

  ```
  "scripts" : {
    ...,
    "lint": "eslint . --ext .ts",
    "lint:fix": "eslint . --ext .ts --fix",
    ...
  }
  ```

- Run ESLint

  ```
  npm run lint
  ```

  Make necessary changes to the `rules` section in `.eslintrc.json` file as per your team's standards. A sample set of rules is shown below for reference.

  ```
  {
    ...,
    "rules": {
        "comma-dangle": ["error", "never"],
        "import/extensions": 0,
        "import/no-unresolved": 0,
        "indent": [ "error", 4 ],
        "keyword-spacing": 0,
        "space-in-parens": [ "error", "always" ]
    }
  }
  ```

## References

- Quick Webpack set up for Single Page Applications
- Jest website
- Getting started with Jest
- Babel website
- Webpack website
- Webpack guide
- Linting in TypeScript using ESLint and Prettier
- ESLint website
- Configuring ESLint rules
- Sass overview
- TypeScript handbook
- Getting started with Mock Service Worker

© Prashanth Puranik, www.digdeeper.in