Webpack and ES6,7.. on Clients

The Webpack Tutorial

This exercise will use some of the guides from Webpack, to set up an example project which will illustrate (some of) the use cases for Webpack. Please note that this exercise "jumps" a little bit in the original tutorial, so not all screen-shots will be exactly as in the original tutorial. This should hopefully not be a problem.

All the headlines below refer to a section in the Webpack guide

Getting Started

Complete *all* steps in the getting started section from the Webpack-guides.

Asset Management

Complete the *Setup, Loading CSS* and *Loading Images* sections from Asset Management to verify that we can bundle "things" other than code (actually, everything, for which there is a loader).

Output Management

Complete everything in this section

Using source maps

Open index.html in Chrome → open developer tools → Open the Sources-tab. Convince yourself that the original code we wrote is "hidden" inside main.js, but is in no way available in an easy to read form. This really sucks when we need to debug. Read the section about source maps, and add the single required line to your webpack.config.js to make your app "debuggable".

Rebuild, and open index.html in Chrome again. Verify that we now have a "debuggable" version of index.js available.

Development → Using Watch Mode

Complete everything in this section

Development → Using Webpack-dev-server

What you have done, up until now to test your application was to open the index.html file directly. Obviously, for a real web-app, we would like to test it via a server running on localhost. If would also we nice to have a feature, somehow similar to what we are used to with a Create-react-app generated app (you can get it almost exactly like that if you also complete the section Hot Module Replacement)

Complete the steps in the section Using Webpack-dev-server

Using node with Babel

By now, you must have noticed that node comes with it's own module system (`require` and `module.export`), while JavaScript, from ES6, includes its own module system (`import, export`).

This can be annoying for a FullStack JavaScript project since client code, written for example with React, can use `import` and `export` (due to Babel), while backend code must use `require` and `module.export`. Luckily however, we can use Babel with our backend code as well, and use newer JavaScript features in our code.

1) Create a new folder with a package.json file for this project (`npm init -y`)

2) Add a JavaScript file **index.js** to this folder

3) Install the required dependencies for what we are about to do:

```
npm install @babel/core @babel/node @babel/preset-env --save-dev
```

The first two imports will allow us to use babel from the command-line (and even easier, from our package.json). The last one, will allow us to use different upcoming JavaScript features.

4) Now in the same folder, create a configuration file for babel **.babelrc** and add this to the file:

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

5) In the `script section` in your **package.json**, add this entry: `"start": "babel-node index.js"`

This is how you MUST call the code we will write below (`npm start`). Calling **index.js** from a terminal with just node will NOT work, you must call your code with babel-node.
Hint: You could call babel-node like this from the terminal:

```
        node node_modules/@babel/node/bin/babel-node index.js
```

The cool thing about doing it from inside package.json is that here, it will automatically use your `node_modules` folder.

Now we are ready to use the new features. Remember from last week, that to use fetch from node, we needed to import a polyfill first.

6) Import `node-fetch` (`npm install node-fetch`) and use it like last week, but import it with an ES6 `import`, INSTEAD of `require`.

7) Now, also to summarize things from last week, remember that, when we looked at the Promise-API, there was a method [Promise.any](), clearly marked as experimental, and with the note that it was in stage-4 of the TC39 process (September 2020). Lets see if we can use this method with the current version of node, using what we have done above.

Add this line at the top of index.js: import "core-js/proposals/promise-any and add code to demonstrate, first `Promise.all(..)` as you did last week, followed by an example that uses the new method `Promise.any`

React, without Create-react-app

Up until now, and also in the future, unless you have compelling reasons not to do it, you have used create-react-app to create your React start code.

Before we start, let's see what we really get from using create-react-app:

a) Create a new react project, called *deleteme* like this: `npx create-react-app deleteme`

Open the project in vs-code, and observe the content, especially the package.json file. Hopefully, this should all be very familiar.

`Create-react-app` gives you a lot for "free", but also adds a few restrictions to what you can do, so in some cases, you need to give up on what you got from `create-react-app`. To get a setup similar, but much more complex, to what we did in the previous exercise, you can eject from a `create-react-app` generated project (if you do, there is no going back).

Let's try that:

b) In the root of the project you just created type: `npm run eject`

c) In VS-code, observe all the changes (in package.json, webpack.config.js etc.)

This should convince you that, unless there are compelling reasons for not to do it, just use create-react-app to setup your React infrastructure.

d) Delete this project, it has served its purpose.

e) 🟨 Creating a React/babel setup from scratch

There is an alternative, to use an ejected `create-react-app` project, and that is to create the setup by yourself, and ONLY add those features we actually need.

Complete this tutorial https://www.valentinog.com/blog/webpack/#getting-started-with-webpack to see how we can do that, and see how we can use babel in our Webpack setup.

Use the project created above

Locate some of your third-semester React Projects and see whether you can move the react code into the project created above, instead of relying on the original create-react-app created project.

For the rest of this semester, if you need to build a React-client, I suggest you use the project created above.