Webpack and Babel

Front-end Developer tools

RE: `create-react-app`

- To this point, we have been using `create-react-app` for all of our React workshops and assignments because it easily allows us to develop React applications without the need for manually building a server ourselves.

- CRA uses webpack-dev-server to serve your React application and enable the hot reloading upon code update that we all know and love.

- While using CRA does have a lot of benefits, and is generally an acceptable way to approach starting a React project that you may be building, we should keep some things in mind.

RE: `create-react-app`

- If we build out an API by creating our own server and endpoints, connecting it to a CRA project during development takes an extra step.

- Our own server would have to run concurrently with the server running our React application, and we would have to proxy any requests through our React application to hit our self-made server.

- I.e. - We'd have two separate localhosts running on different ports, requiring us to route our requests across these ports.

RE: `create-react-app`

- More importantly, though, is that all of the configuration for setting up a React application is taken care of for us, which prevents us from understanding how to put one together ourselves should the need arise.

- We can gain access to these configuration files by ejecting our CRA, which can be helpful taking more granular control of our React application.

- But, to even be able to do something like that, we should understand what a couple of the more significant tools are and how they work.

What is Webpack?

- Webpack is a tool you can install using `npm`.
- It takes your source JavaScript files using `require` and `module.exports`, and produces a single, bundled JavaScript that the browser understands.
- While it is not exclusive to React and has many use cases, Webpack is extremely helpful for us because it can create one large .js file, typically named `bundle.js`, that contains all of our client (React/JS) code.

What is Webpack?

- To use, you can `npm install --save-dev webpack`

- Side note: `--save-dev` will save Webpack as a dev dependency, meaning when you deploy to production webpack would be ignored as a dependency. But why?

- Webpack is a build tool that we'll typically only need while we're developing our app. When we deploy our app to a production server, we may already have everything bundled and ready to go, so we don’t need to install Webpack on the production server. Instead, we can add a special flag (like `npm install --production`) to signal that the install should ignore dev dependencies.

Using Webpack

- By using Webpack to bundle our self-made React application, we can then serve a single index.html file with a script tag to load our bundle.js and any stylesheets that we require to render our application in the browser.

- In doing so, we can ensure that we can properly route our information through our own server by serving up the index.html file that we will be integrating our React application into.

Using Webpack

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

- Create a webpack.config.js file in the root directory of your project

- View this link to see a simple configuration example and understand what each part of the config object is doing: https://repl.it/@johnnybee4e/webpackConfigExample

- View the Webpack docs for further explanation: https://webpack.js.org/configuration/

Using Webpack

- To create a bundle.js file, you can create scripts in your package.json that looks like this:

```
{
  ...
  "scripts": {
    // running build-client will run webpack,
    // creating a new/updated bundle.js file in your public directory
    "build-client": "webpack",
    // running build-client-watch will start weback in watch mode,
    // which will create a new bundle every time you save changes
    // within your client folder
    "build-client-watch": "webpack -w",
    // running start-dev will start webpack in watch mode
    // as well as your server with nodemon, which is essentially
    // a watch mode for node.
    "start-dev": "npm run build-client-watch & nodemon server/app.js"
  },
  ...
}
```

What is Babel?

- Babel is a JavaScript compiler
- According to the docs: *“Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browser environments”*
- Not every browser will support the most recent advances in ECMAScript standards (ES6/7+), so we use Babel to transform our syntax from the code that we are used to writing into a browser-compliant version of JavaScript.

Using Babel

- `npm install --save-dev @babel/core`
  - `@babel/core` provides the core functionality for Babel

- Create a .babelrc file in your project directory

- In it, create a JSON object with the key “presets” pointing to an array of strings representing the Babel presets that you want to use:

```
{
  “presets”: [“@babel/react”, “@babel/env”]
}
```

Using Babel

- To use the presets, be sure to npm install whichever ones you require for your project. Some examples:
  - `@babel/preset-env`: allows you to use the latest JavaScript without needing to micromanage which syntax transforms are needed by a target environment
  - `@babel/preset-react`: Uses several babel plugin options to convert JSX syntax.
- Make sure to also grab `babel-loader` to use with your webpack.config.js