Setting Up for React

If you'd like to run React on your own machine, there are a couple of things you'll need to do. This guide assumes *no* prior knowledge of local JavaScript development beyond that already taught in this class. There is *some* assumed knowledge around the terminal/command prompt.

Editor

The editor of choice for this subject (and for a growing number of developers) is VS Code. VS Code is built by Microsoft, and it is freely available, open source and highly extensible. VS Code has first-class support for JavaScript out of the box - you can use intellisense and other features for JS without any extra work. This is the editor that will be "supported" by the tutors.

What's more, because VS Code is built completely on web technologies, it powers the editor on CodeSandbox, so the interface should feel familiar to most of you already.

You can download the installer from https://code.visualstudio.com - it works across macOS, Windows and Linux.

Node

`node` is a server runtime built on the V8 Engine (the JavaScript engine that powers Chrome). It is very widely used in modern web development as it enables front-end developers to build tools that hook into the existing JavaScript toolchain.

`node` powers `create-react-app`, the *de facto* starting point for almost all new React applications today. In addition, it includes the "node package manager" (or NPM) which houses thousands of "packages" that other people have written, which can be included easily in your own code.

Finally, `node` will form the basis of the second part of the assignment in this unit, where you will use it to build the back-end of an application.

Some caveats before installation: Whilst Microsoft is making every effort to get `node` to run more seamlessly on Windows, it is not always straightforward. If you are working with macOS or Linux, you're good to go. If you're on Windows, if at all possible I recommend using the Windows Subsystem for Linux. This will *more or less* allow you to run `node` as a Linux application. You can read more about WSL here.

Windows is less supported because the vast majority of modern web applications run on Linux-based servers. macOS has the advantage of being closely related to Linux, and it is often used by developers so they can have their development environment more closely match their production environment. The unfortunate consequence is that some people write NPM packages *without* Windows in mind

- the file separator on Windows is \, whilst the separator on macOS and Linux is /. Additionally, a lot of documentation will *assume* that you are using macOS or Linux, and tailor the instructions to those platforms, possibly ignoring Windows altogether.

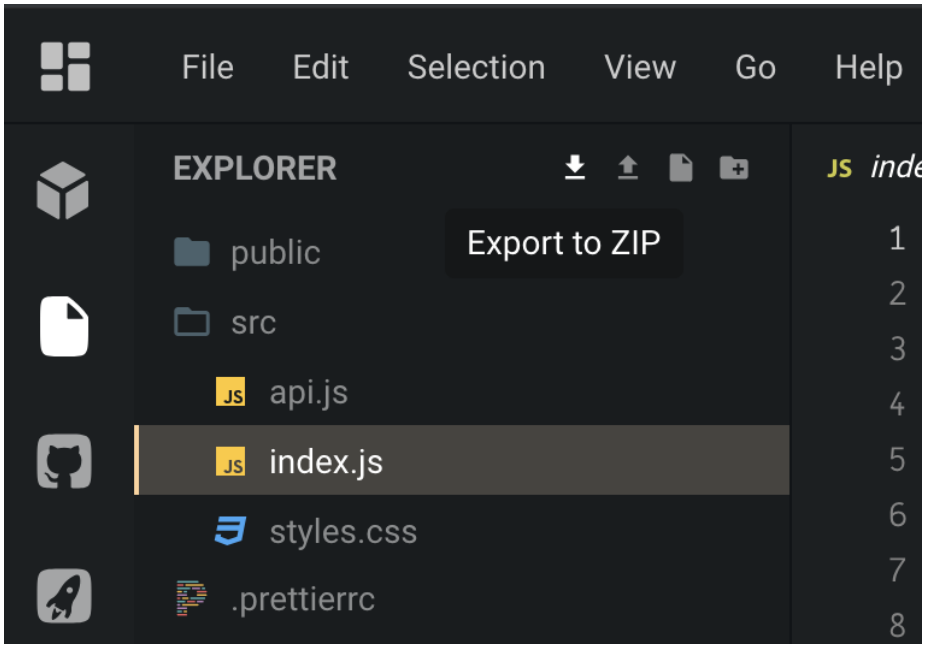
Good packages will take Windows into account, but you may well get bitten. Use WSL if possible.

You can install `node` by going to https://nodejs.org and following the instructions for your platform.

Running a React Application

Once you have an editor and `node` installed, you are more-or-less ready to run your application locally. If you have already built your application with CodeSandbox, they offer the ability to download the application as a `.zip`.

You can download the application by pressing the down arrow next to the Explorer, as shown in the following image:



Once you have the `.zip`, you can extract it to whatever location you like on your file system.

Next, open VS Code, and open the folder you've just extracted from the `.zip`. Before we can run anything or do any work, we must install our *dependencies*. CodeSandbox managed the installation of dependencies for us, but locally, we must install them ourselves. In VS Code, press Ctrl-~ to open a terminal window

inside the editor. Within this terminal window, you can run `npm install`. This will read through the `package.json` file created by CodeSandbox, and find all the dependencies specified, and install them into a `node_modules` folder locally.

Important: `node_modules/` can get *very* large, *very* quickly. Because it is installed the same on every machine, there is no need to keep the folder in Version Control or similar. If you are using `git`, you should add `node_modules` to your `.gitignore`. When submitting assignments, *always* remove the `node_modules` directory first. Please follow this instruction.

Now, with your dependencies installed, you can run your application. In that same terminal, run `npm start`. This will run the script defined in the `package.json` with the name `start`. This will spawn a server to host your application while you work on it. After you've executed this command, go to the URL as printed in your terminal (probably something like `http://localhost:3000`) and you'll see your application. You may also find that your default browser opens automatically.

Creating a New React App

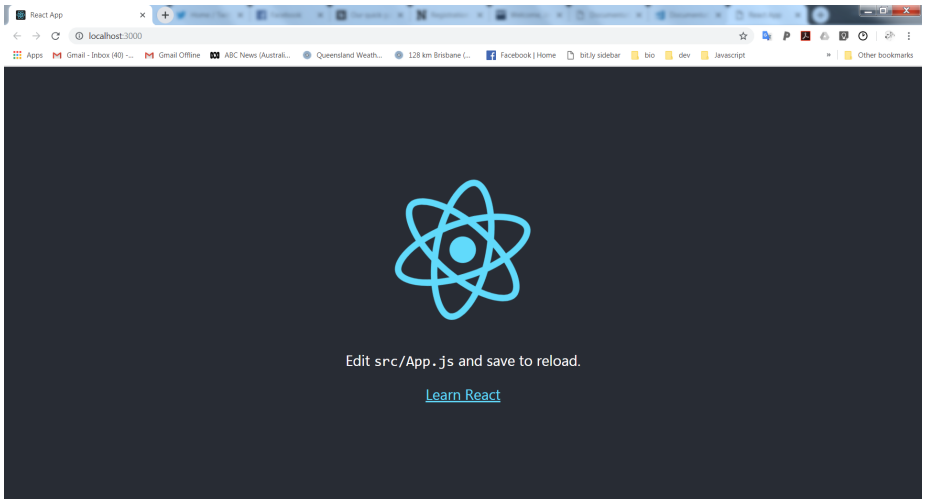
If you want to create a *new* react application on your local machine, the easiest way to start is to use `create-react-app`. This is a 'bootstrapper' that is built and maintained by the React core team at Facebook. It is the most common way for people to begin to develop a React application, because it abstracts away the need for you to set up a build configuration and worry about compilers and so on. For the majority of applications, you won't need access to the underlying infrastructure of your builds, so you can go ahead with `create-react-app`.

Open a terminal or command prompt window and navigate to the directory in which you want to create a project. Note: this should be the *parent* of the eventual project directory. For example, if you want your project to live in `/home/name/project/my-app` (or `C:\Users\name\projects\my-app`), you should go to `/home/name/projects` (or `C:\Users\name\projects`).

Next, run `npx create-react-app my-app` to create an application named `my-app` in the `my-app` directory. You can of course choose whatever application name you like.

`npx` is a tool that ships with `node` (and `npm`) - it essentially downloads and runs whatever you specify as an argument without installing it globally. It's helpful for things like command-line utilities that you want to use, but don't necessarily need to keep around.

Once `create-react-app` has finished, you can again open the directory in VS Code and work with the application. Helpfully, `create-react-app` installs the dependencies on app creation, so there is no need to follow the `npm install` step we used above when migrating from CodeSandbox. As before, you can launch the app by opening a terminal and running `npm start`. You should see something like this:



You can read more about `create-react-app` here - they have instructions and details in more depth if you need them.

The Nitty Gritty

If you *really* want to dive deeper into how a modern build system works, how the JavaScript compilers work, and how to take full control of your application's build process, I've included some links below. It is well beyond the scope of this class (nothing we cover will require anything more that `create-react-app`), but it is very interesting.

Warning: Do *not* try this with your application without making a backup or starting a new project for this explicit purpose. Most of it *cannot* be undone. There are ways to break the build system completely. Anything you do will be isolated to the project you're currently working with, but you should make sure that your assignment and prac folders and code are *not* used.

`create-react-app` is *still* a great place to start exploring. I recommend that you create a new application as outlined above, and then run `npm run eject` - this will run a script provided by `create-react-app` which will remove the `create-react-app` helper scripts and instead scaffold out an application that builds using webpack. You can see, for a start, that your list of dependencies inside `package.json` has grown considerably.

If you look inside `package.json`, you'll see a key with a name of `babel` - `babel` is the compiler used to turn your non-standard JavaScript into something more standard, like turning `JSX` back into function calls. In fact, it's far more than that, because it can be extended to support almost anything. You can read more about `babel` here, and this is well worth the effort if you want to understand the whole ecosystem.

Inside the newly generated `config` folder, you'll see a file called `webpack.config.js`.

`webpack` is the build tool that takes all your JavaScript files and combines them into one file, ready for sending to the browser[1]. This is often seen as overly confusing, but the Webpack team has spent a lot of time making it simpler and improving the standard of their documentation. You can see the result of their efforts here.

There is plenty more to dig into, but that is necessarily left as an exercise for the reader. The biggest piece of advice would be to:

> Focus on how to build great applications

Almost every company you go on to work at will already have a build system set up - your job will be to build out applications. Webpack, Babel, et al. will come with experience.

[1] This is a gross oversimplification - webpack does so much more than this, and can in fact generate a whole lot of files. It is essentially responsible for figuring out how to get the code you've written to 'bundle', ready to be sent to the browser.