**Starting up:**

<https://www.youtube.com/watch?v=_ZTT9kw3PIE>

Install node, run ‘npm install -g create-react-app’ to get the latest Facebook-released model. Use this as a boilerplate.

**Our React build-chain:**

**Webpack:**

* When you write code, you do it in multiple files – it just makes sense.
* When you serve it to a user, you don’t want them to have to download 100 files – that’s 100 separate network requests. Webpack just combines all of them, minifies them, jzips them, and provides them for download.
* It’s just an asset pipeline for nodejs.

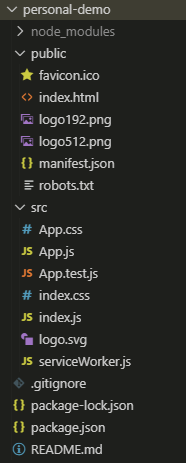
**ESLint:**

* Obvs.

**Babel:**

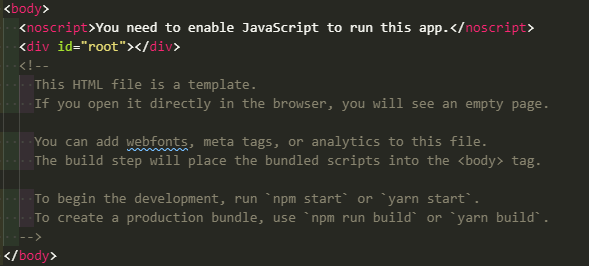
* Obvs again – just lets you write modern JavaScript and have it compile to ES6.
* JSx also – which we use in React, and which is important for creating components.

**Folder Structure:**



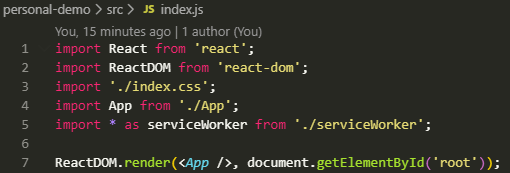
* Root contains misc. configuration settings.
* Public contains favicons and other publicly visible things.
* SRC contains all of your JS, CSS, etc.

**Index.html:**



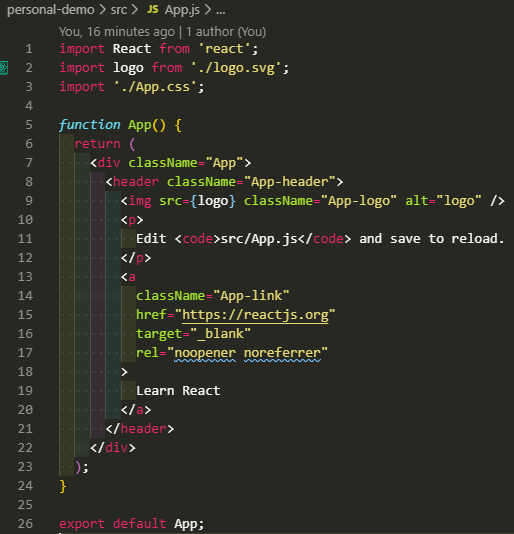
* The goal of React is to build HTML elements in the code and inject them into the correct location within ‘root’, not to build them manually in the html file.

**Index.js**



* We’re importing the JSS directly into the JS, not the HTML file. This is a feature of Webpack and allows us to live reload while developing.
* We’re also pulling in app.js, which is boilerplate react-style html – the ReactDOM.render function is injecting the HTML we have in App() into the root element of our document 😊.

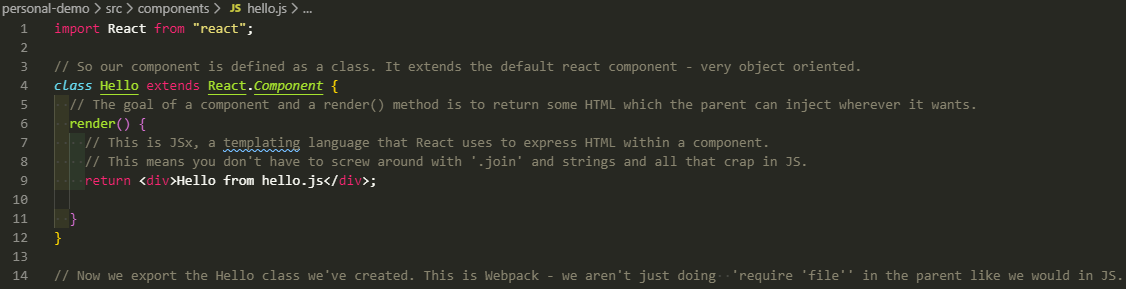
**App.js**



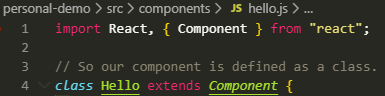
**Our First Component:**

* React is all about creating components – a small part of a UI that you can reuse and isolate.
* If you look at a site you can generally tell the logical way components have been split up.

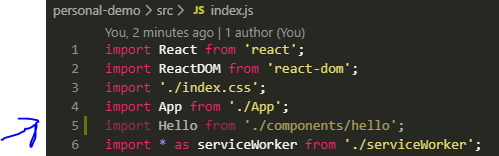
1. We’ll create a components folder in src (where it’s tracked by Webpack)
2. We create a file called ‘hello.js’.
3. We import React from the “react” package.
4. We create a component by defining a class which will extend the default React component. So it’s all very similar to object oriented programming.
5. The goal of a component is to have a render method and return some HTML inside that method.



Note that if we specifically require ‘Component’ we don’t need to use React.Component, as you’d expect!



Now we can import it:



Remember that index.js is the main JavaScript file run when the page is loaded. It pulls in compoents and injects them into the html file, which won’t have anything in it originally.

So: We create component classes with a render() function that returns HTML and has an export line so that index.js can import it.

And here within index.js we’re performing that injection operation, mounting a component in a specific location on the page.



Really cool point – because react actually injects HTML into a dom element, you can write a component and inject it somewhere without swapping everything over – you can have a small part of your page be react-based, and the rest old cruddy js!

**Props**

*Injecting dynamic variables into a component and building reusable objects:*

We use props as arguments for components. We add key/value pairs and pass that information into the component as we load it up. Here we’re passing a string, but it could a function, a string, an object, anything.

**Index.js:**

**Hello.js:**

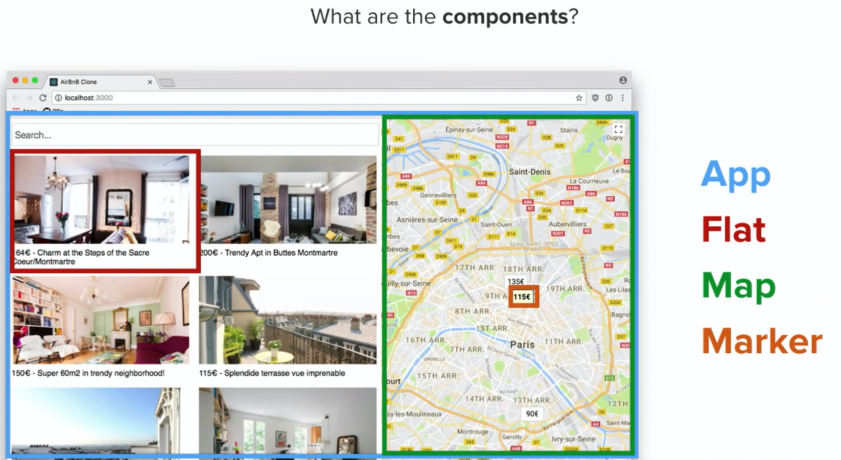


* Note that in JSx we use curly braces to say ‘here is some JavaScript’.
* So this is equivalent to ‘create a new class of type Hello with argument of ‘Nick’ of firstName.
* We can also do it by defining variables:



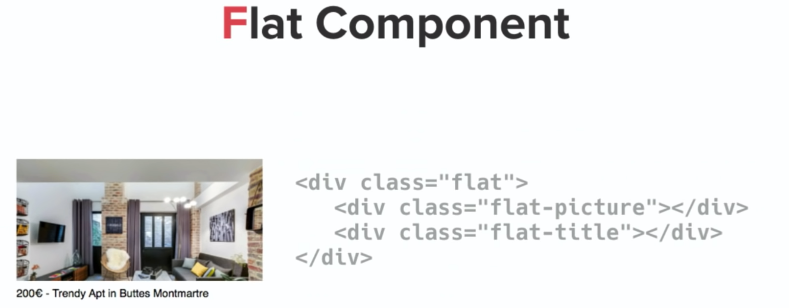
* And we can instantiate multiple by building up a more complex HTML object:

**Replicating Air B&B:**

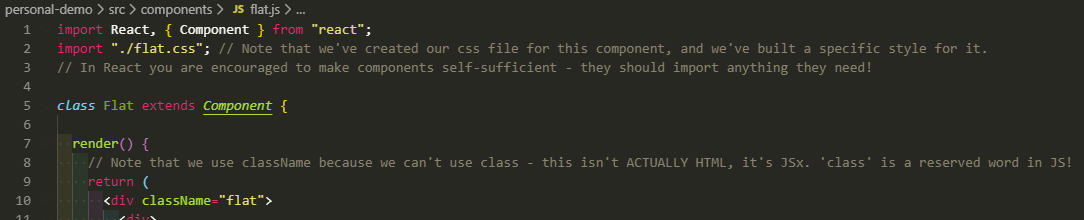


1. The search bar.
2. The flats as cards.
3. The map.
4. The main container – the **app**.

**The Flat Component:**



1. We start by creating a component in our components folder. We need to import React, create a class that extends component, create a render() function, and export it.



1. We create a CSS file and import it – in React you’re encouraged to make components as self-sufficient as possible, so import your own .css 😊 .
2. We’ll also need data from the external world (AN API) – we are going to assume that we’ll connect to an API of some kind and populate our card with data from that.

https://youtu.be/\_ZTT9kw3PIE?t=2800