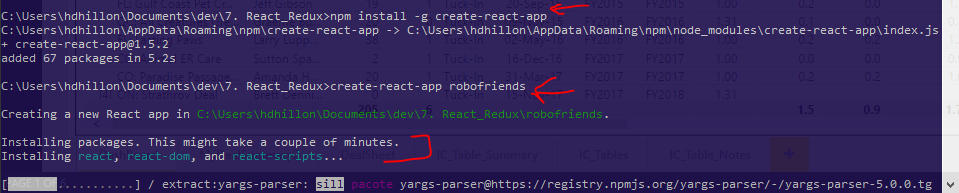
**REACT + REDUX NOTES**

**GENERAL**

* React is an NPM package for DOM manipulation
* Underneath the hood React lets you use the HTML-like syntax (JSX) to create a “virtual dom”, which they compare the real dom to and update parts that changed in the virtual dom in the real dom.

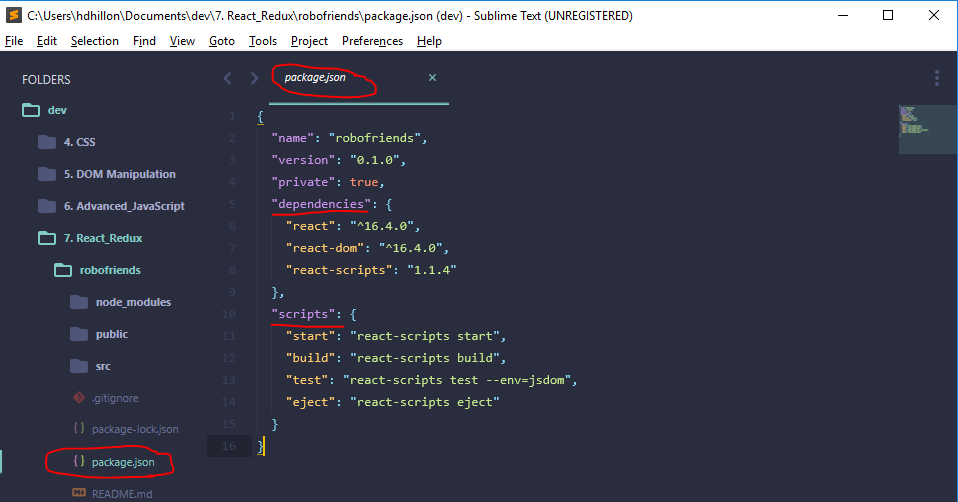
**CREATING A REACT APP**

* Create react app with all files in place – **global** package from npm to be installed
  + Contained web-pack, babel, linten, debugging
  + Creates infrastructure so can start writing the app
* Command line:
  + npm install -g create-react-app
* Creating the app command line:
  + create-react-app [nameofapp]
  + create-react-app robofriends
* This then creates a folder called robofriends with require infrastructure (modules)

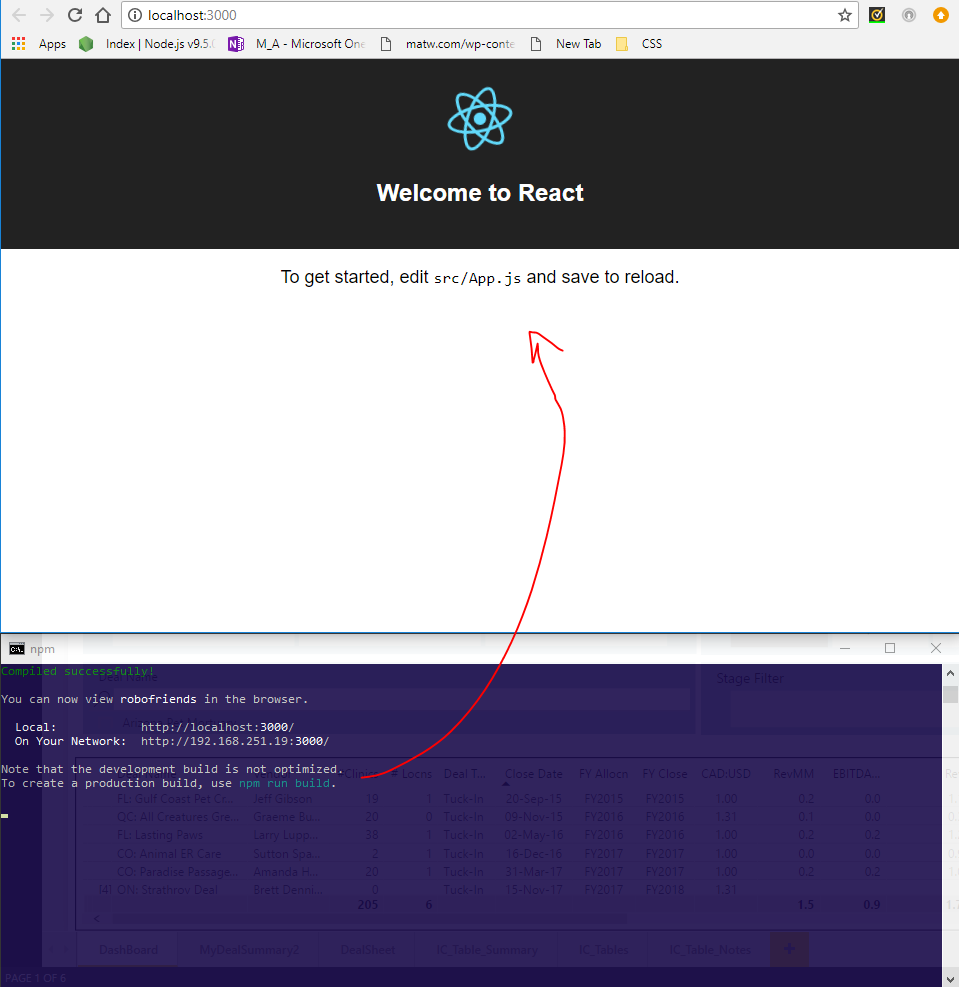




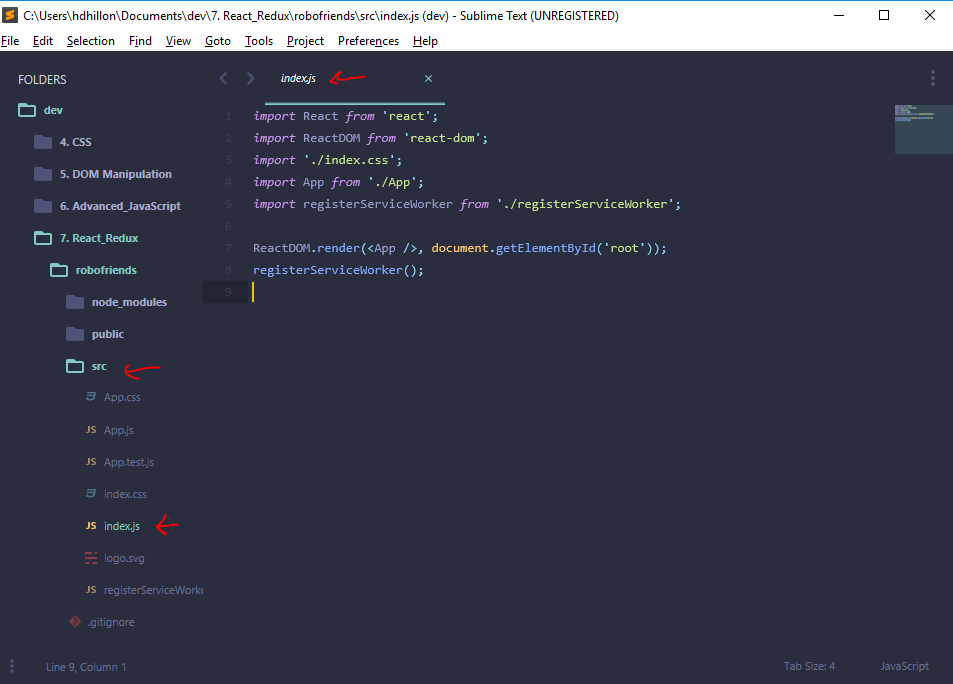
* If you look at package.json file – looks like they have added dependencies and scripts for us



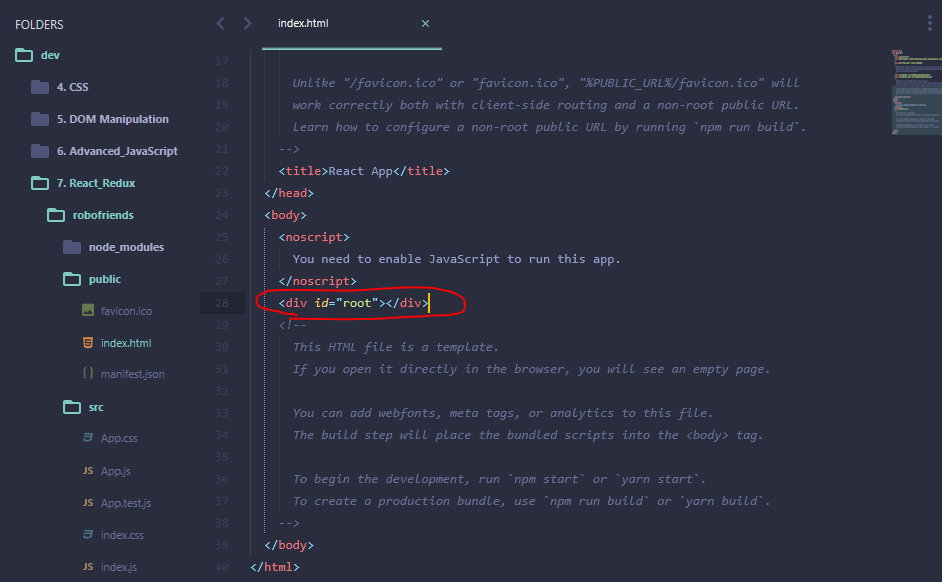
* Run the “start” script in command line: “npm run start” 🡪 starts up our server



* **What has create-react-app created for us?**
  + README
  + Package.json (see above) – dependencies to discuss later
  + Scripts – all use react-scripts 🡪 this is what makes create-react-app awesome. Instead of installing web-pack and babel, react scripts does everything for us and uses latest scripts
    - Allows for building really fast apps
    - If you want to customize react-scripts you can “eject” and customize app
  + Package-lock.json file
    - Automatically created
    - Makes sure version #s of dependencies are locked in so if given to anyone else, makes sure versions work 100% of the time
  + Public folder
    - Index.html
    - Manifest.json 🡪 new feature that allows ppl to download short-cut to desktop and have an icon to it
    - Favicon.ico 🡪 little icon in browser tab
  + src folder
    - this is where all the “magic” of react happens
    - index.js is the main script file. Looking at this script file we have the following:

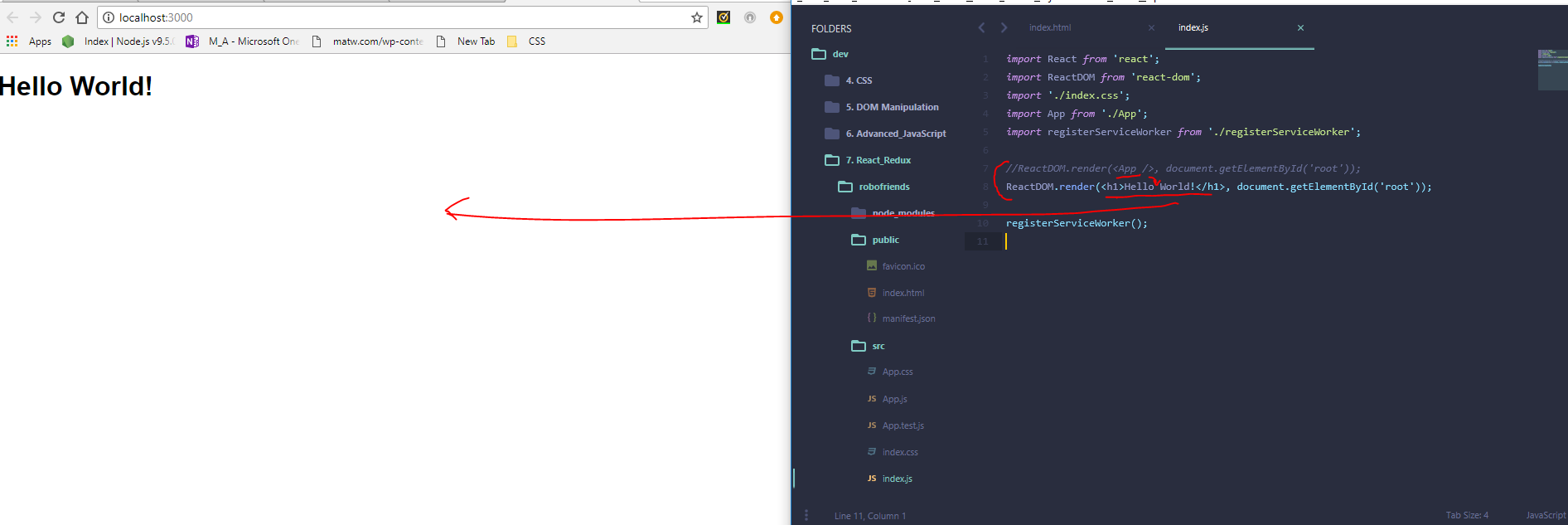


* + - * import various scripts (import react from ‘react’ etc.)
      * REACTDOM.render(<App />, document.gelementByID(‘root’)); 🡪 *we are getting an of element with ID “root” (from index.html file) and rendering <App /> there*
        + If we go to index.html under the public folder – you will see within the body there is only one <div id = “root”></div> and nothing in between it. Yet, when you go to browser (see above), you clearly see a page with a React Icon and “Welcome to React” text, etc.

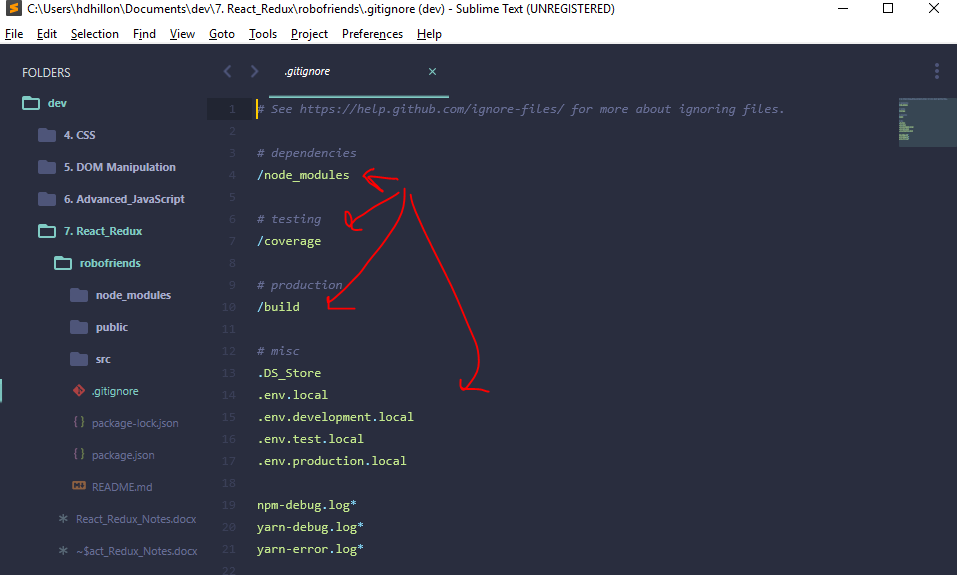


* + - * + <App /> -- is jus a way of saying this is a “React component” and I want this rendered.

You can even replace <App /> with <h1>Hello World</h1>



* + .gitignore
    - Before we run git and push project to github – git will check this file to look for anything that shouldn’t be pushed to repo
      * E.g., node modules folder, coverage foler, testing folder, etc. 🡪 automatically generated for us.

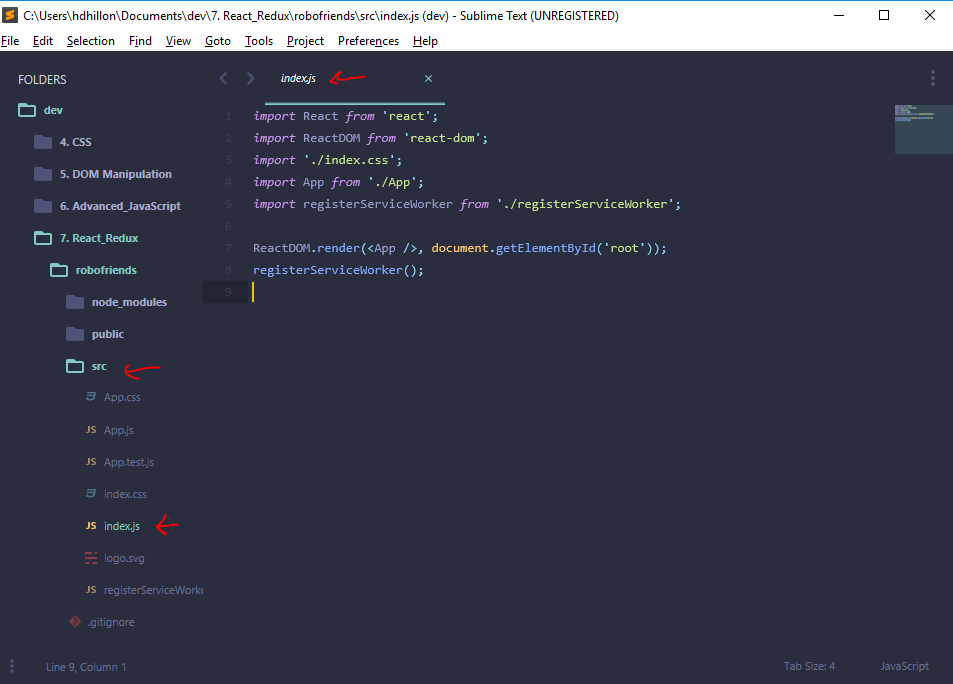


* Sublime Text change from “JavaScript” type to “Babeljs”

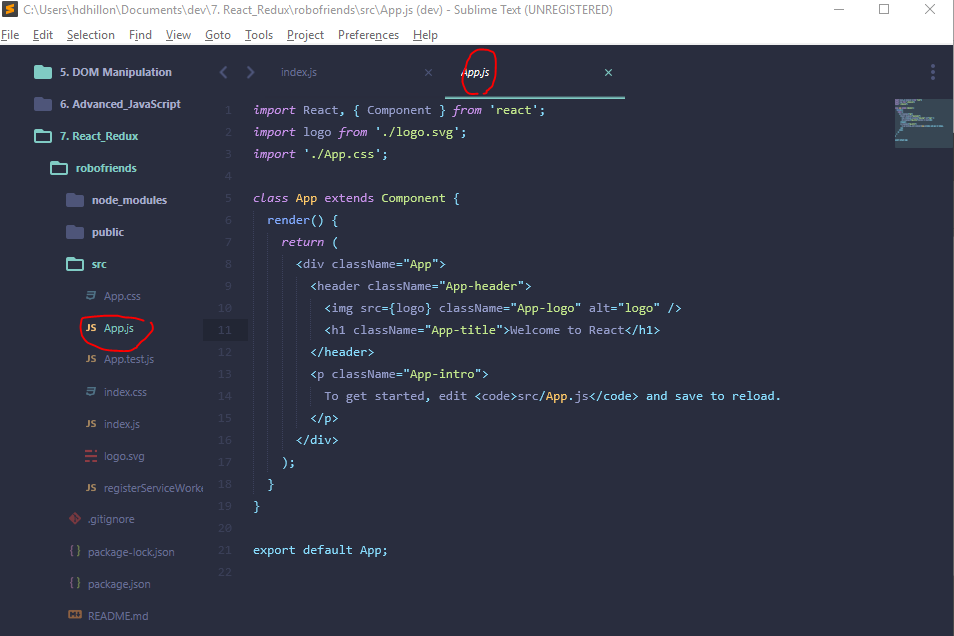
**BUILDING A COMPONENT**

**Overview of how Components and File Structure Fit Together**

* **Go back to index.js file**



* + Note ‘import’ works for us here bc react uses ‘web-pack’ to do the bundling for us (i.e., therefore no need to use browserify and ‘require’ syntax)
  + Import ReactDOM 🡪 used for websites (i.e., DOM). You can use something like ReactNATIVE that renders to mobile phones
  + Import ‘./index.css’
    - Allows us to add CSS not just for one file, but each different component
      * E.g., we have
        + App.js and App.css
        + Index.js and index.css
      * Note the ./index.css’ 🡪 means it’s in the same directory
  + Registerserviceworker
    - New feature that allows our app to become faster and potentially work off line – won’t concern ourselves with for now
  + Main lines we care about:
    - *import App from './App';* 🡪 even though extension not used, by default, looking for file named “App.js” in the same folder
    - *ReactDOM.render(<App />, document.getElementById('root'));*
    - *ReactDOM.render(<App />, document.getElementById('root'));*
  + Go to **App.js file** – we find (change type to Babel > Javscript Babel)



* + Importing here – { Component } from ‘react’ is using destructuring. Destructuring allows us to prevent typing “React.Component” all the time and just use “Component”.
  + class App extends Component { … }
    - Component rule: must render something and the way it renders is it “returns” and HTML piece of the website
    - So we are creating our own custom HTML tags that we can just add
    - In order to render multiple lines of JSX (i.e., the <div><header><p> tags), need to wrap all of them in the “return” statement with ( )’s.

**Put it together -- Building a basic “Hello World” Component**

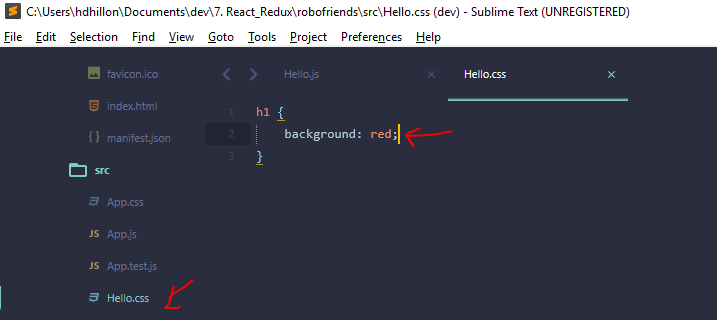
* Start with index.js file and replace App component with Hello Component
  + Note – I hve also added greeting = “Harman” and date = “Thursday” as these values will be considered “props” “properties” of the Hello component and will be passed when into the Hello.js file (to be created) and “Hello” function (to be created – will create the tags that will be rendered).



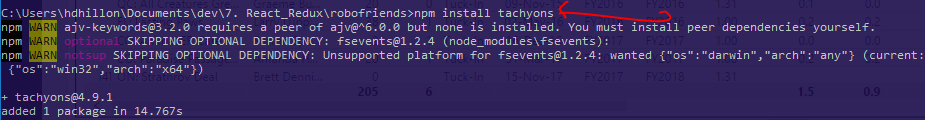
* Create Hello.js file with appropriate imports as well as ‘export default Hello;’ at the end of .js file
  + Note, if we were returning more than one function, instead of having ‘export default Hello;’ **[look up how to export multiple functions from script file to be used in another file]**



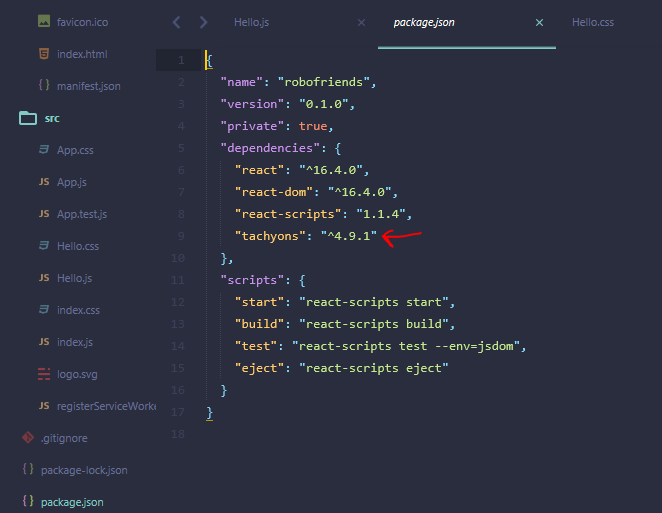
* **Adding styling (CSS)** 
  + Add CSS standard styles with a separate file = Hello.css
    - Create the file and apply style to <h1>



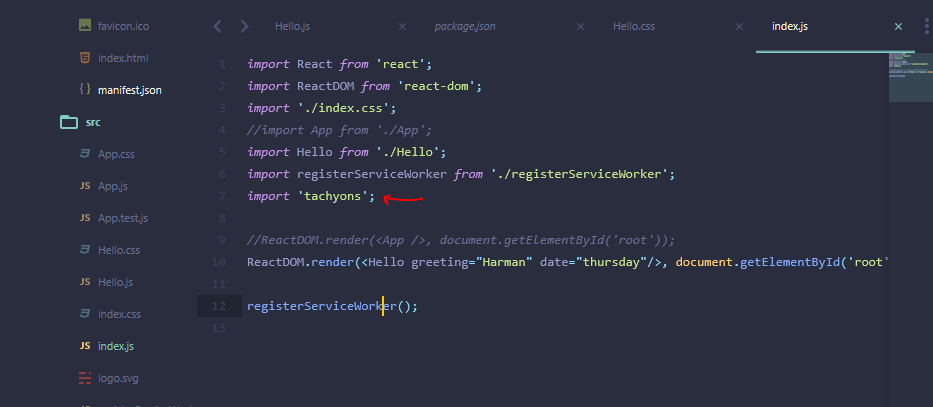
* + Add CSS styles using Tachyons package (like Bootstrap) 🡪 command line: npm install tachyons



* + This adds tachyons to package.json file



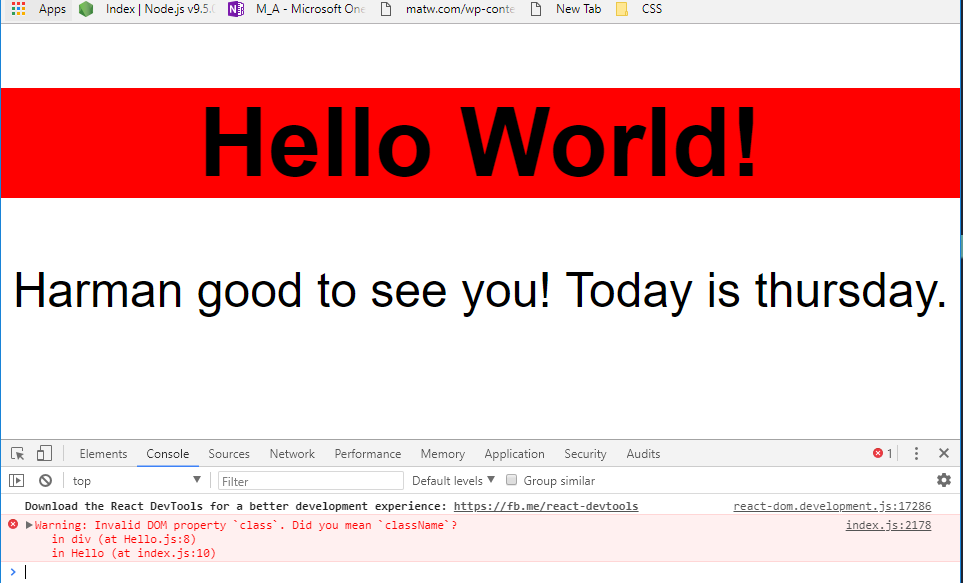
* + To **use** tachyons
    - go to index.js file and import ‘tachyons’;



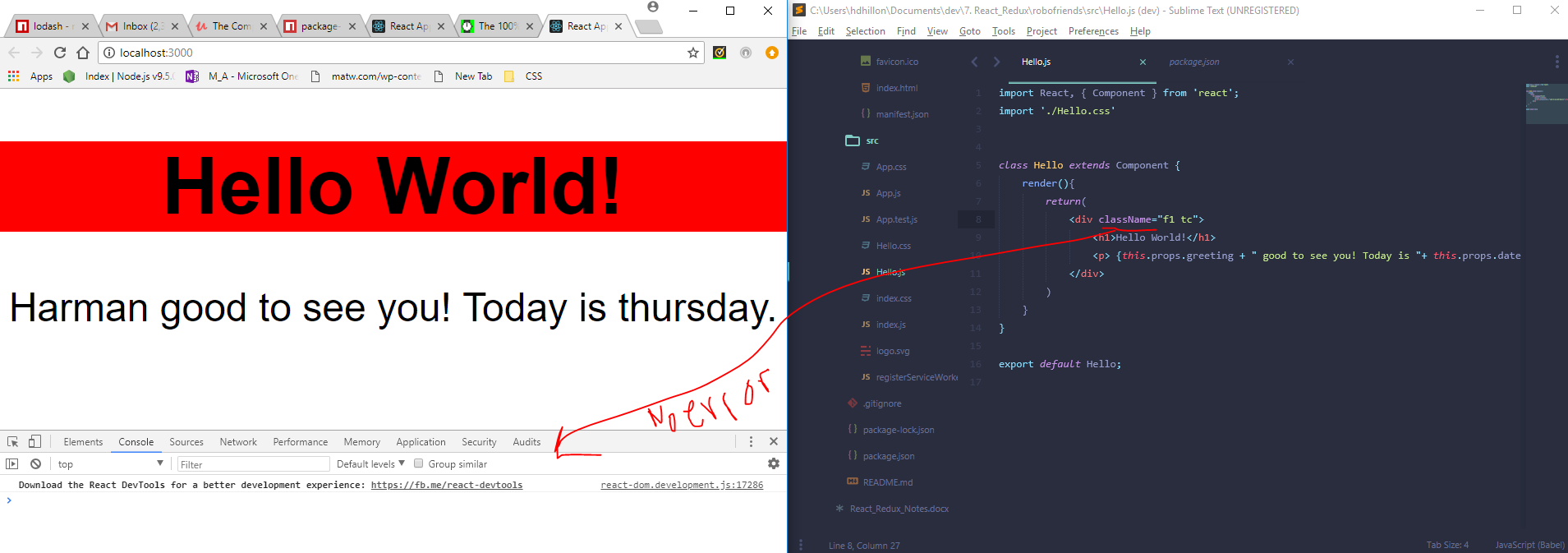
* + - go to Hello.js file and add “className” to div tag where we want to apply *(note – google Tachyons if want ot figure out what type of formatting, etc. to use)*



* + - **Note – directly above, we used class = “f1 tc”.** On the browser, this appears to work, but there is an error. This is because you cannot use “class” as the tags within the “return” function are not REALLY HTML, they are JSX, which is HTML-like.

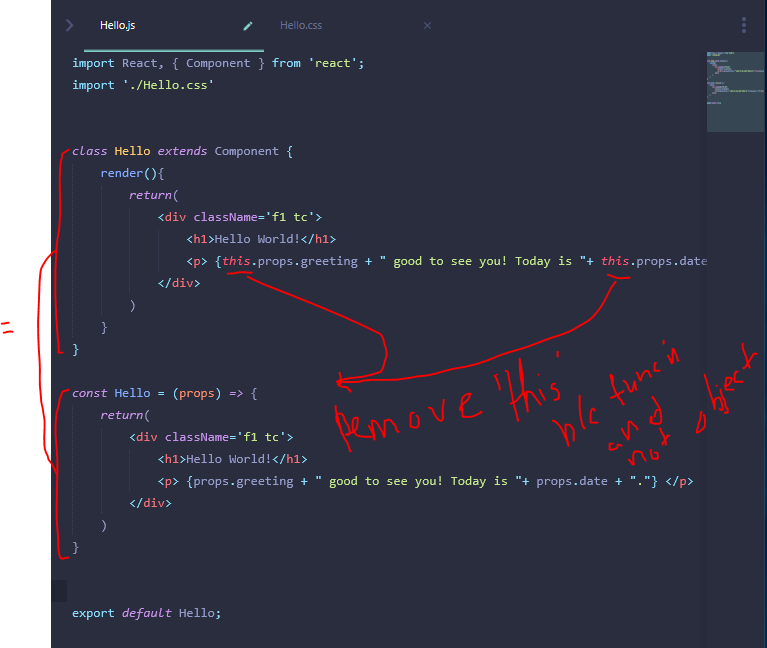


* JSX is a form of javascript and in javascript the term “class” is a reserved name so we must use className = “f1 tc” to avoid the error.

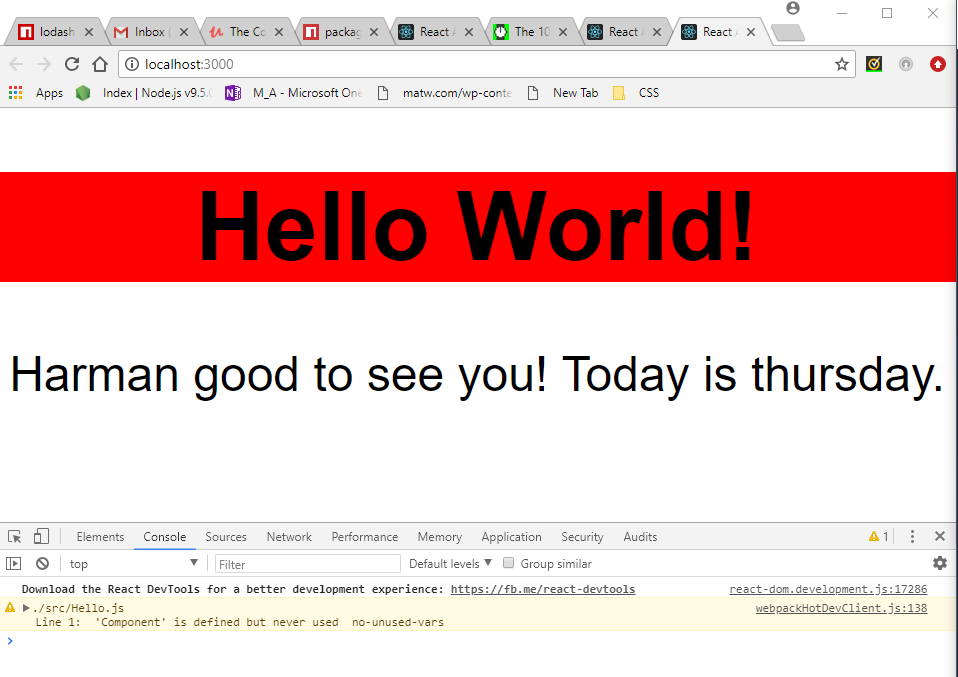


**REACT IS JUST JAVASCRIPT FUNCTIONS**

This is really just a function (although we are working with it as a class). The above class definition (object) is the same as the function definition below.



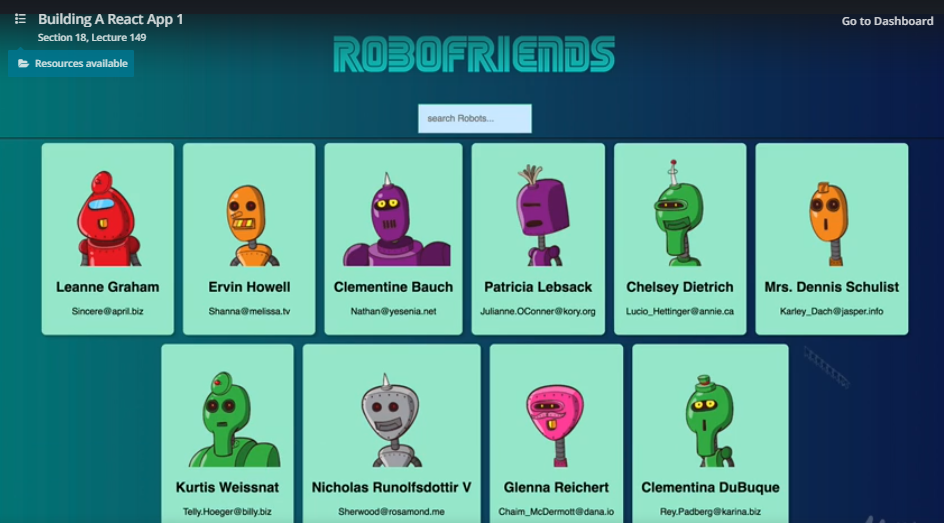
Still works (after commenting out the class definition):



**BUILDING ROBOFRIENDS APP: USING MULTIPLE COMPONENTS**

Want to build following site – what are the components?

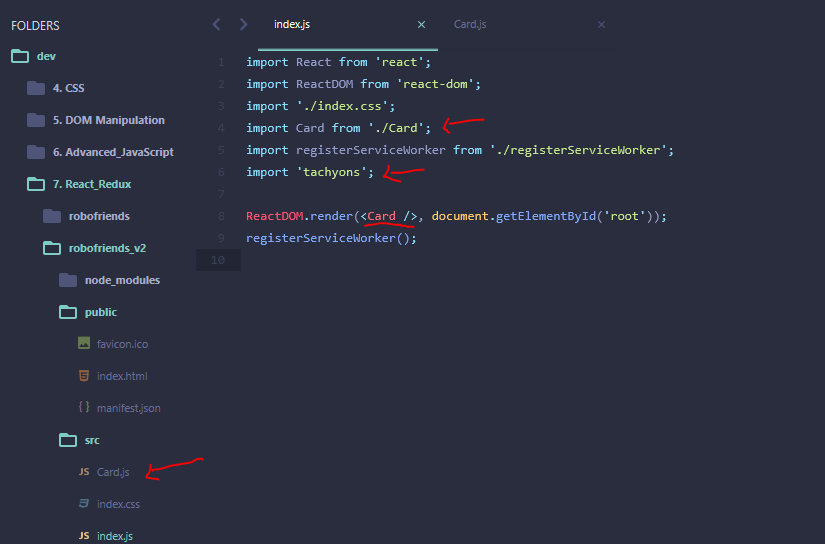
* Cards with info in them 🡪 if we can build one of them we can just copy all of the components and add them to the page
* Title
* Searchbox



**(1) Build Cards Component**

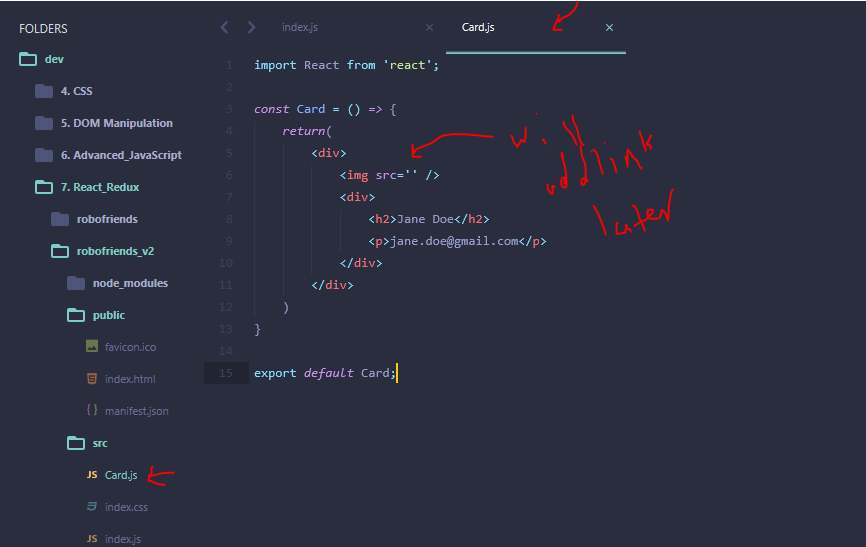
* created a new react app to keep things clean: see “robofriends\_v2”
  + command line: create-react-app robofriends\_v2
* also, let’s delete4 files we won’t need: App.js, App.css, App.test.js, logo
* add tachyons package again using NPM as it was a local install (see above): npm install tachyons

Update index.js file as follows and create Card.js file:

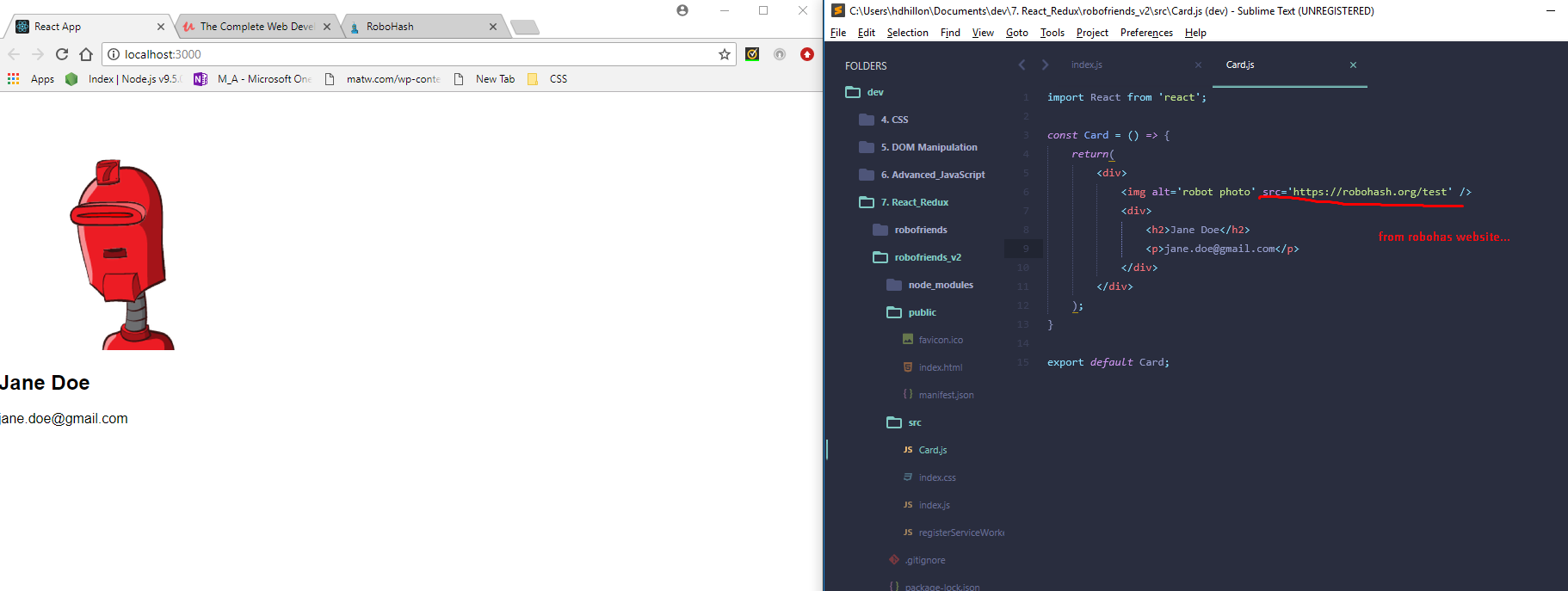


Using function syntax (for now, instead of doing react way) create Card component:

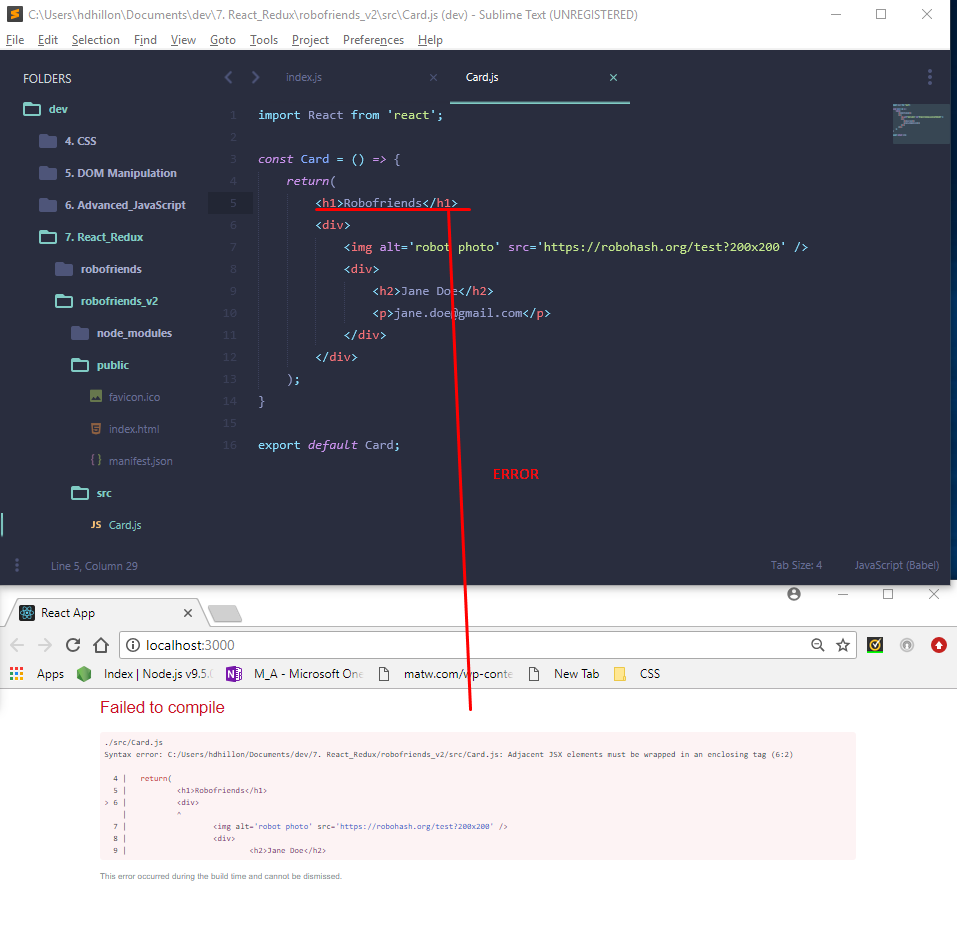
* Note – we still need the ‘import React’ here so our program understands JSX (the HTML-like syntax).



Using API add robopictures into <img src> tag:

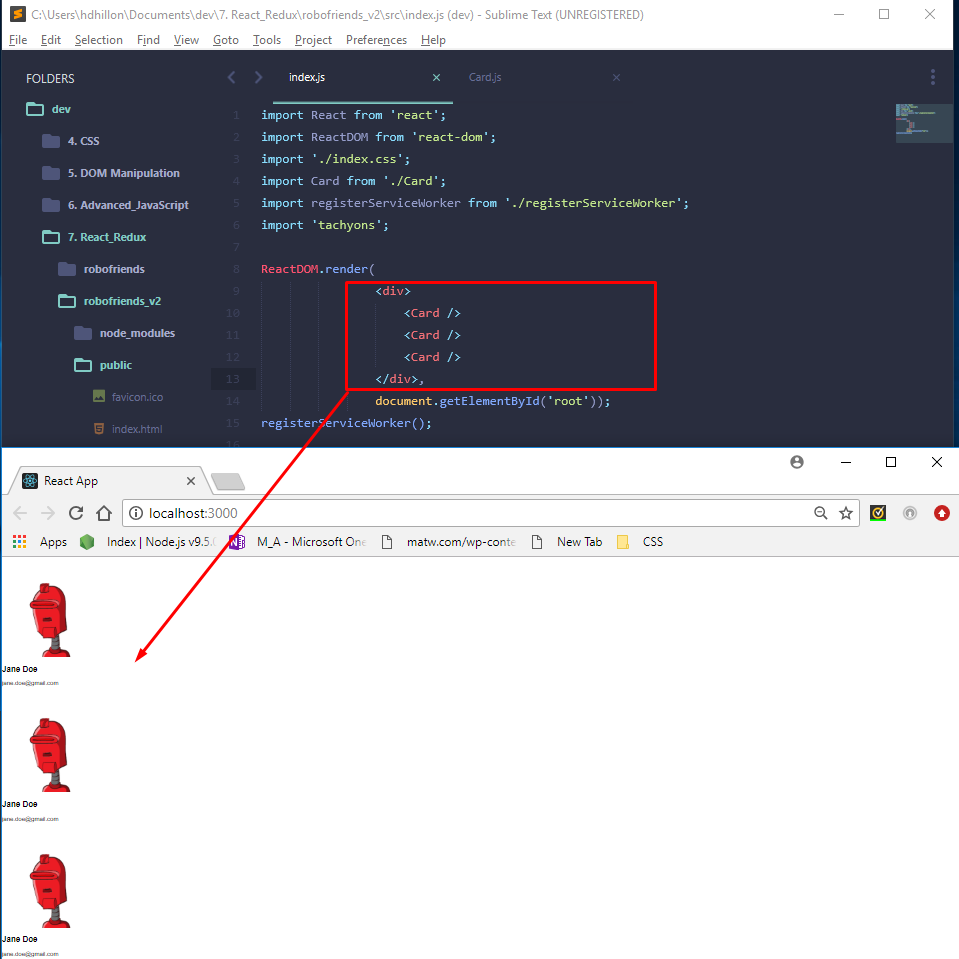


**Note:** Say we wanted to add a title and added an <h1> outside of our div … we would get an error. This is because react components are only permitted to return one element (i.e., cannot return an <h1> AND a <div> tag).

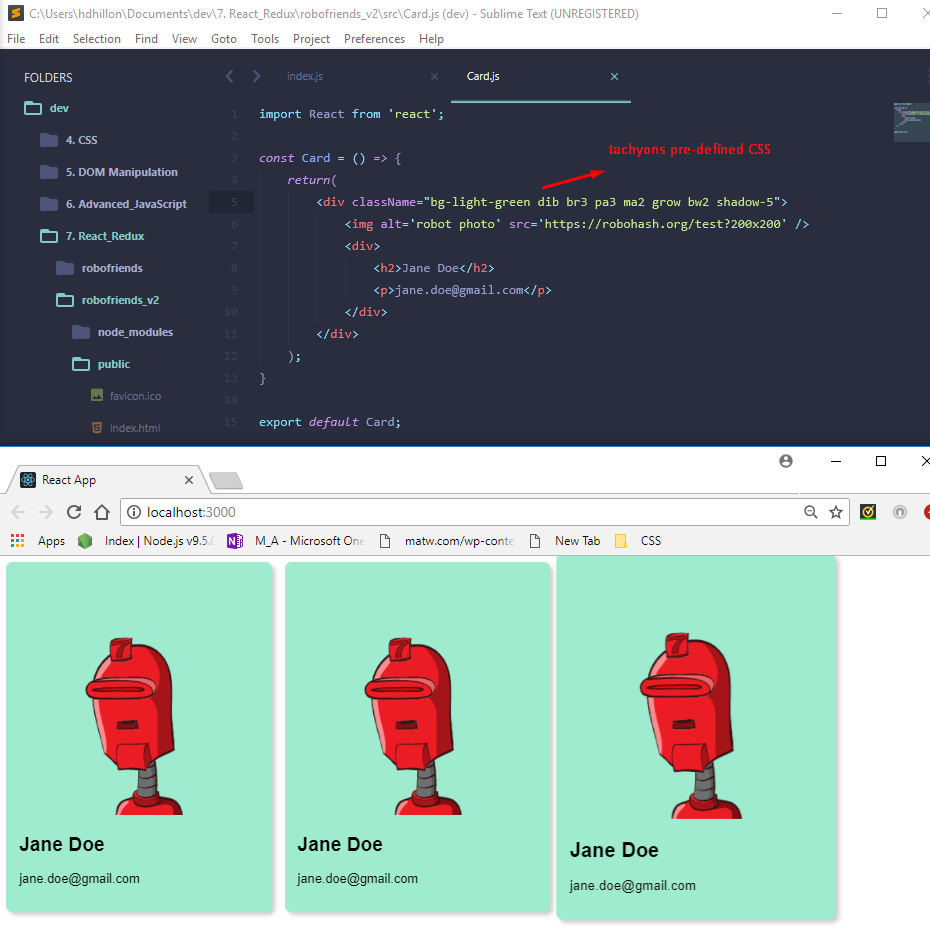


**What if we wanted multiple card components on our website (i.e., multiple users)?**

* Manual way (for now): add multiple card components to index.js file wrapped under a <div> tag (i.e., as this is JSX)

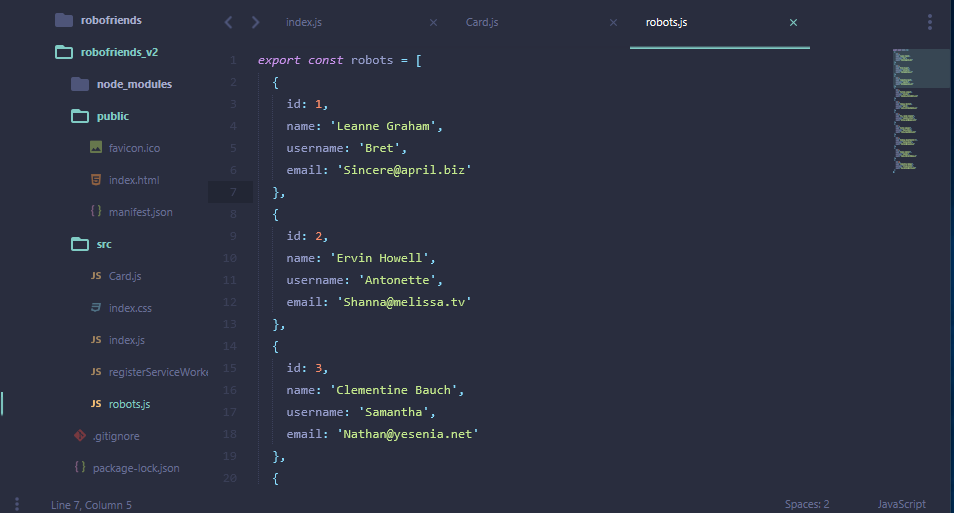


**User tachyons to add styling:**

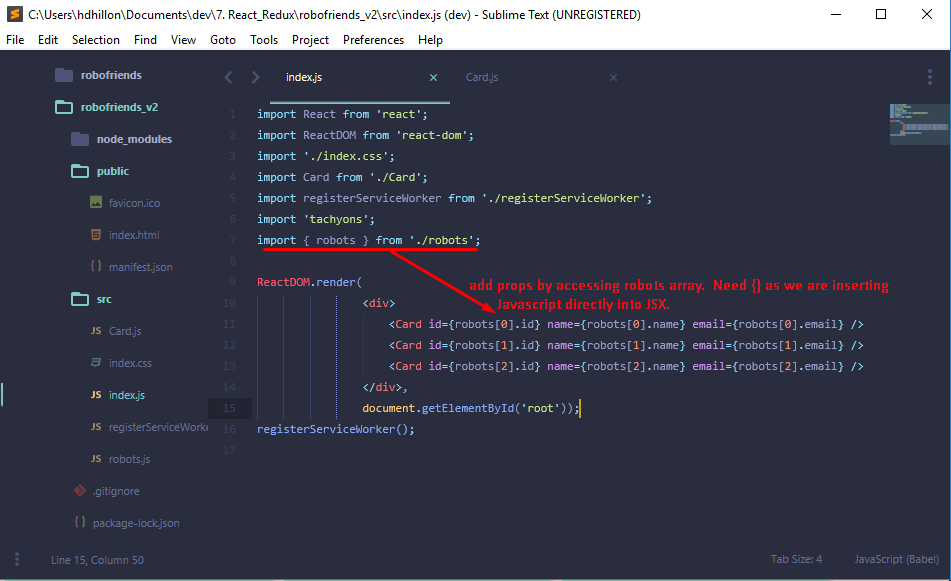


**In order to make the card names dynamic (i.e., not all “Jane Doe”) we need to use props to pass through different names.** We need: (1) a file with various robot names/attributes; (2) import that file into our code and display the dynamic info.

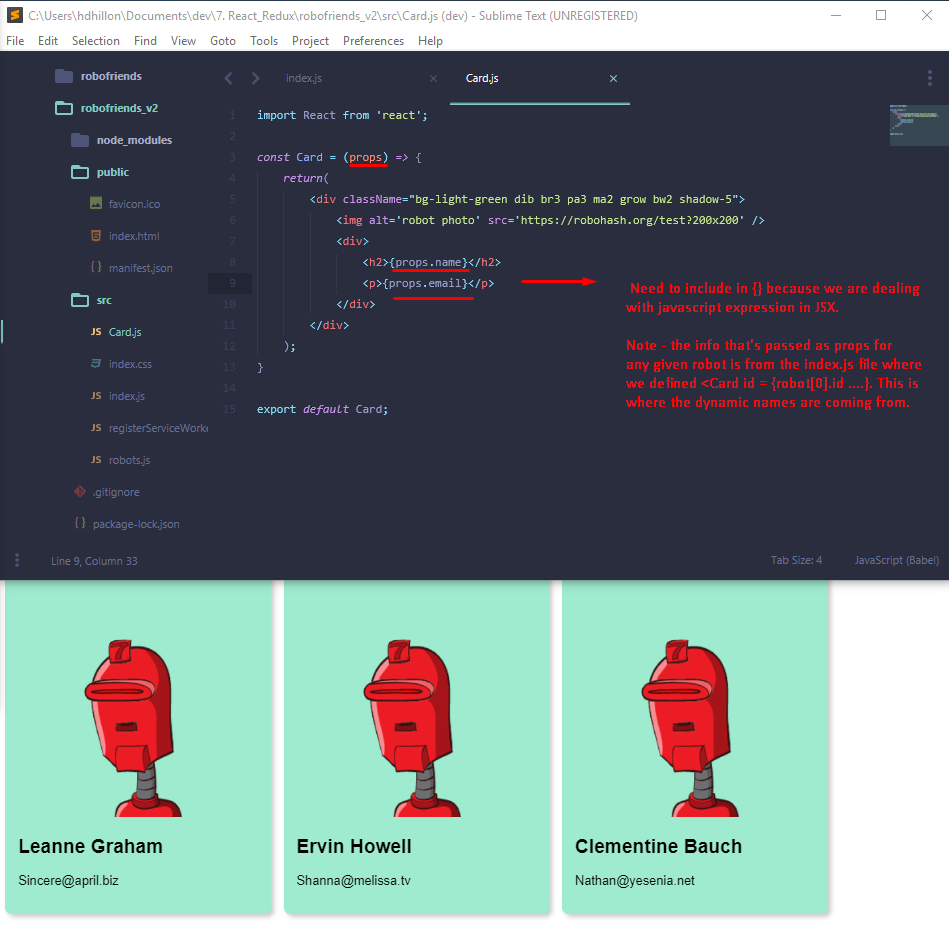
* Create a separate .js file with javascript objects in an array called robots with the robofriends info…



* Import info of robot.js into index.js by destructuring the robots array and add info to pass to Card component using props

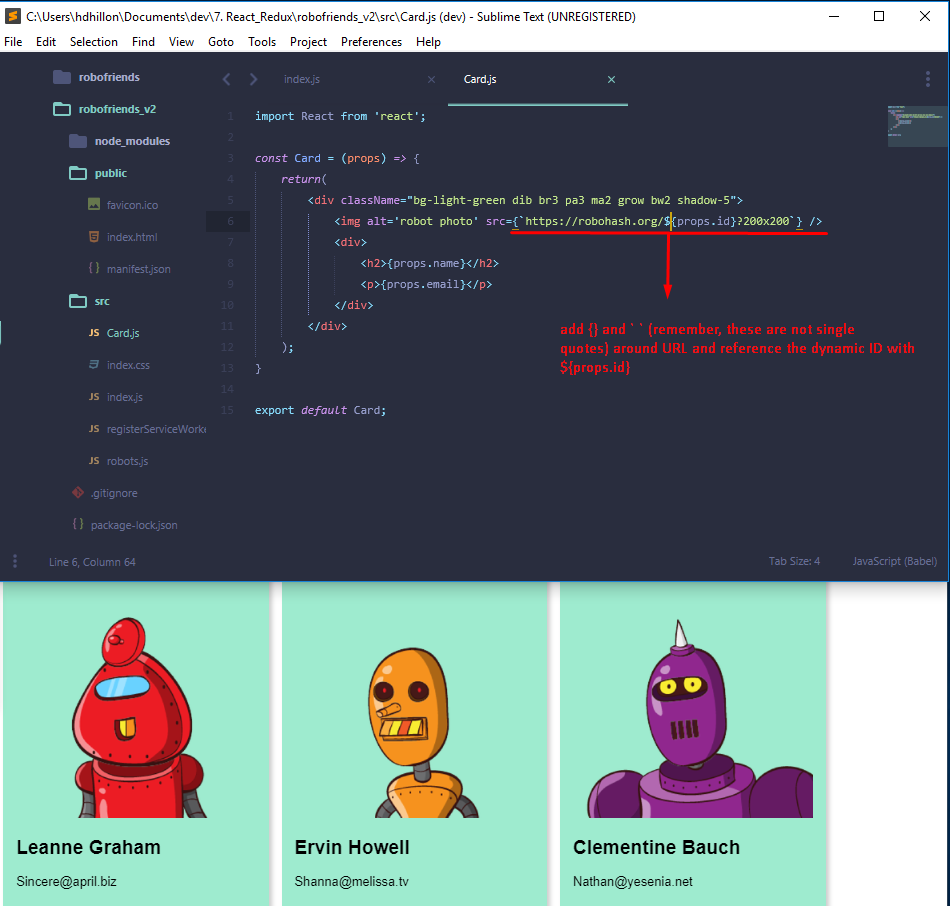


* Update Card.js file so it accepts and displays props that are being passed from the <Card> component tag in the index.js file

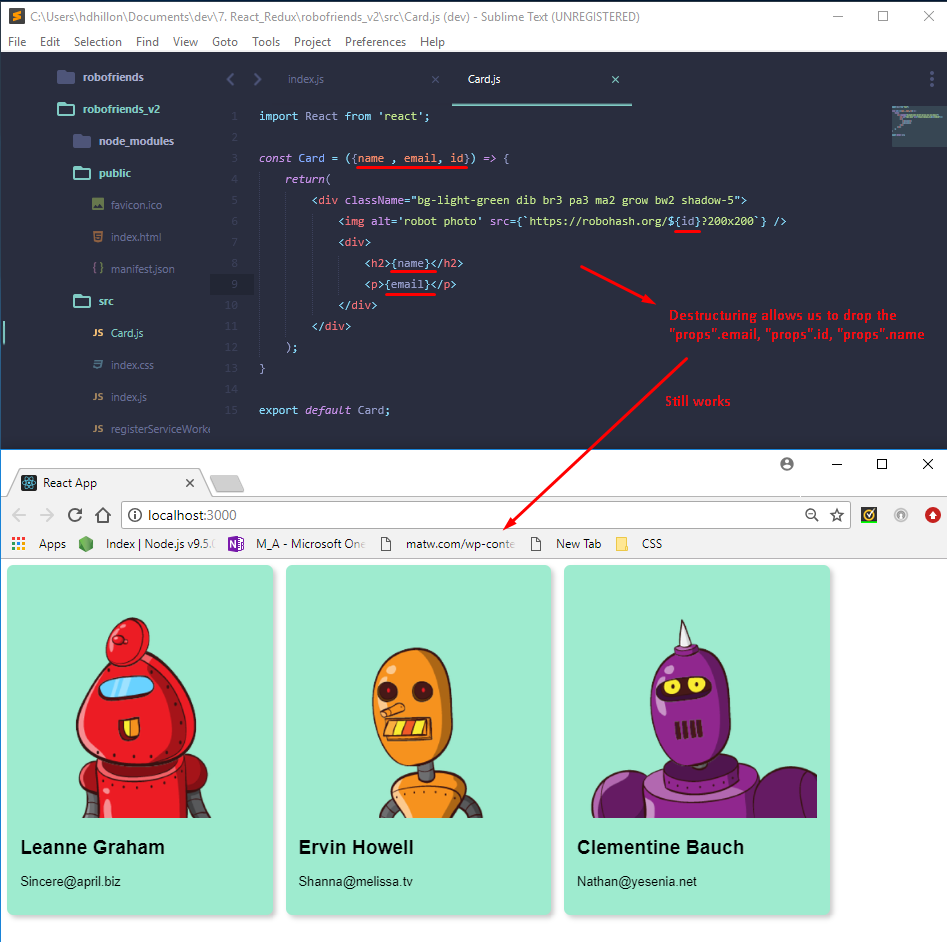


**In order to make the card pictures dynamic (i.e., not all the red robot) we can rely ‘template strings’ in Javascript to enter validate expression directly into the API url.**

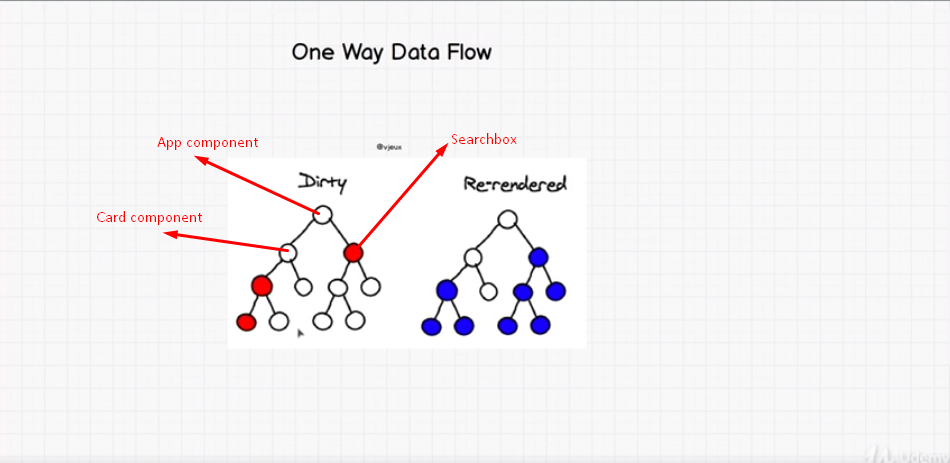
* **Note –** recall in **template strings**, in order to escape the string and add a dynamic javascript expression, need to escape he string using ${javascript expression here}.



* **Destructuring –** clean up the above code using destructuring of props. This let’s us drop the “props.” references



**Understanding the One-Way Data Flow – Parent and Child Components**



* Above shows a potential data flow – we would have a “parent” component (e.g. App) with children components (e.g., Card, Search Box, etc.) which then have their own children components.
  + In order to combine the multiple “card components” on the view (without listing them out one-by-one – see directly below), we may want to combine them with a single component “CardList” component as their parent and make the “Card” component a child of CardList.



* Updating Code to Create a Parent “CardList” Component of which “Card” will now be child component:
  + Create new CardList.js file
  + import Card from ‘./CardList/js’
  + Delete Card.js import
  + Update render so robots array is passed to “robots” as props



* + In CardList.js file
    - Import child Card component (i.e., import Card from ‘./Card’)
    - create function called CardList that destructures props into { robots }
    - move the three <card Id = {robot[0].id} ….> tags from index.js file into CardList function
      * Note – this is how the Card.js (child component) and CardList.js (parent component) are linked for the timebeing and props are passed to Card component.
  + don’t forget to export CardList

