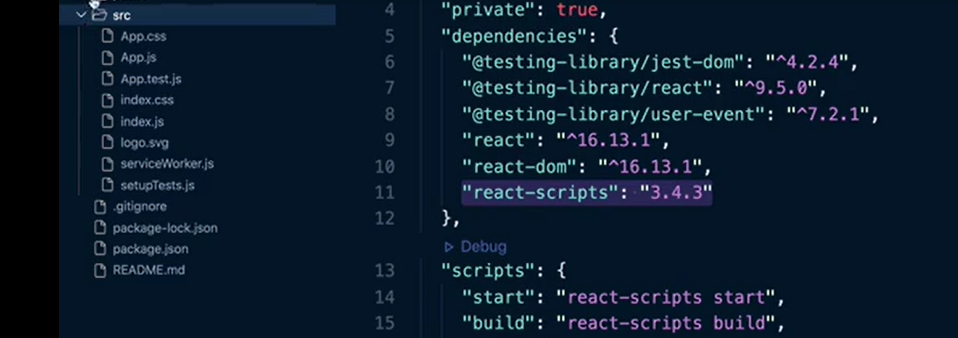
Source: <https://www.linkedin.com/learning/react-js-essential-training/>

* Install Node.js
* Create app using ***npx create-react-app My-Test-App-Name***
* react-scripts referenced in the package.json is the one that is used for different important activities
  + all sorts of tooling like babel and webpack they are configured using react-scripts
  + 
* **src** folder is where were write our application code, and **public** folder contains html files and other public assets, like icons and html files.
* **React.StrictMode**  throws warnings when our code is violating the best practices of react framework.
* **React.createElement –** first parameter is the tag to generate, second is the properties for the element, and third one is the content. In following screenshot style is being passed as one of the properties for the element
* To define a simple component, we use following code

function Header(){

return(

<h1> This is html of the component that we want to return</h1>

);

}

* We can receive props inside the components like this

function Main(props) {

return(

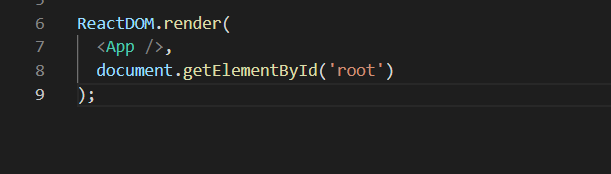
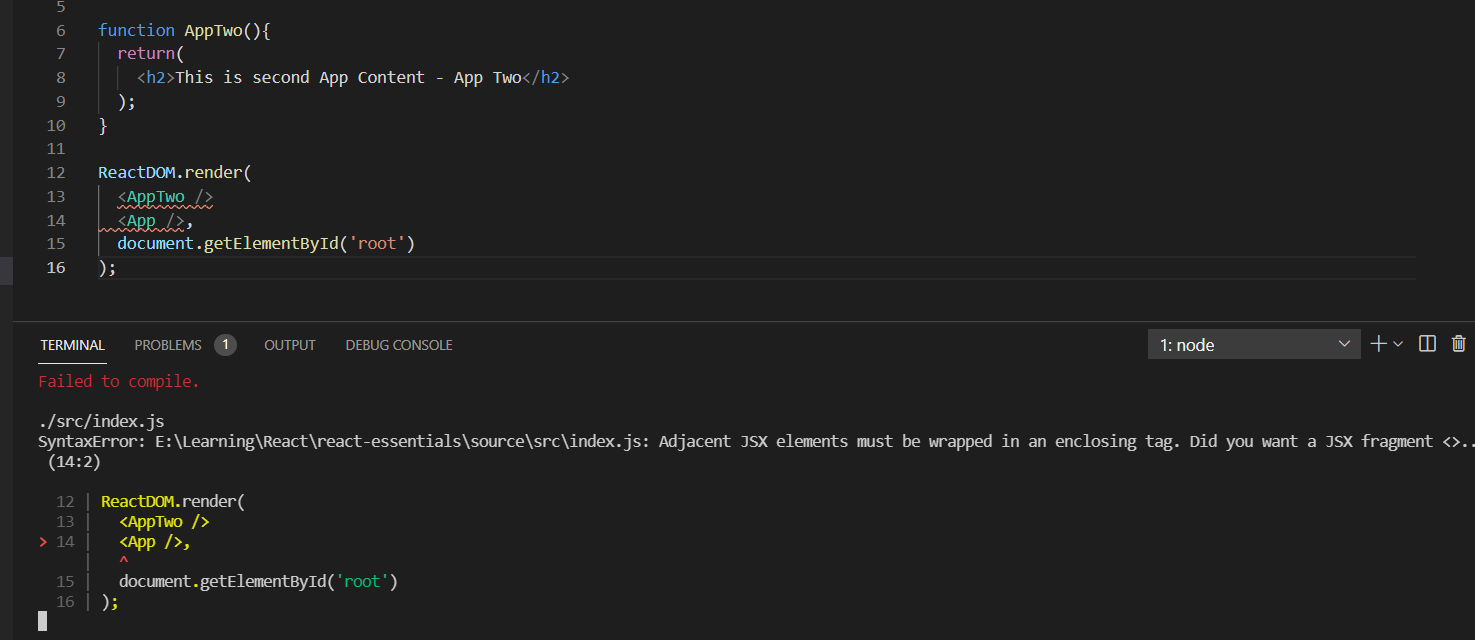
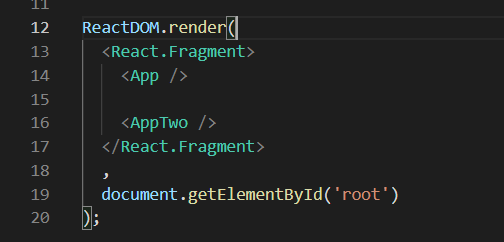
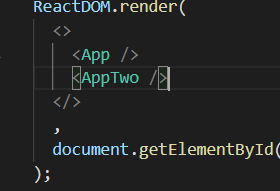
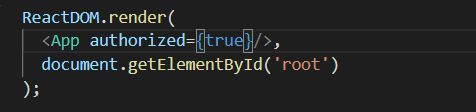
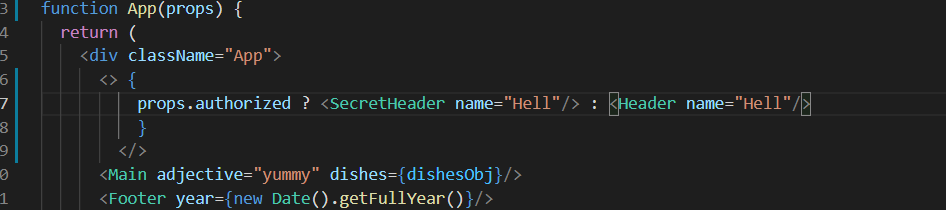
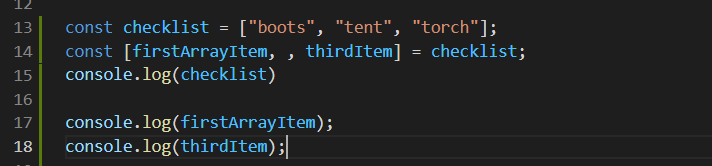
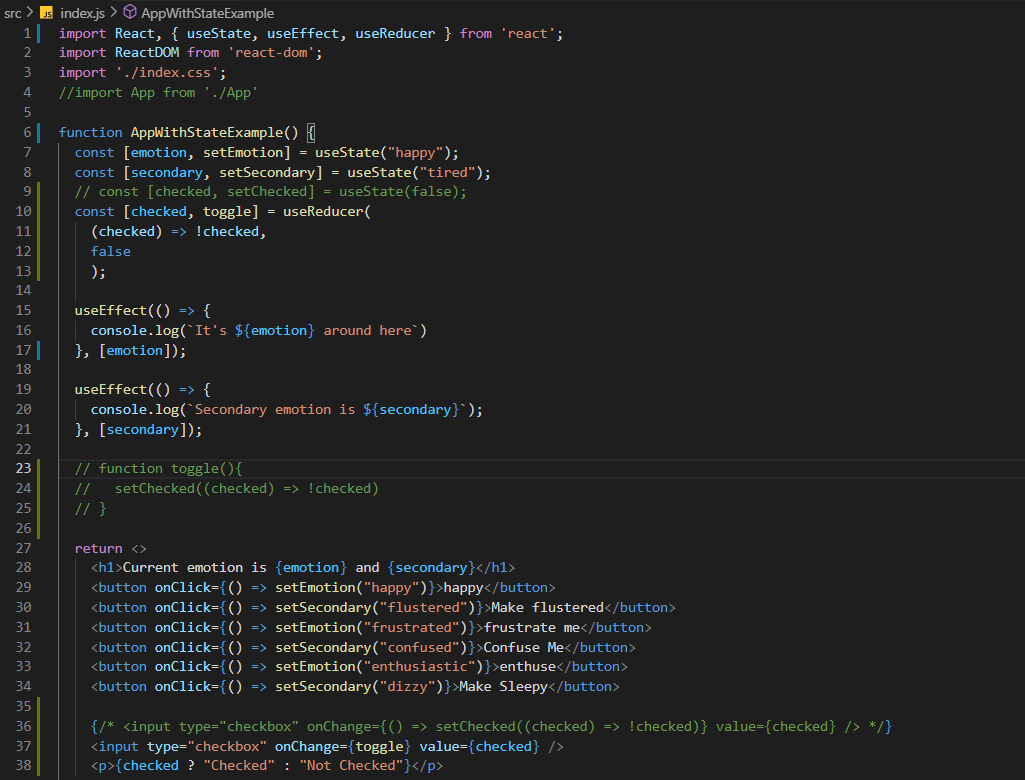
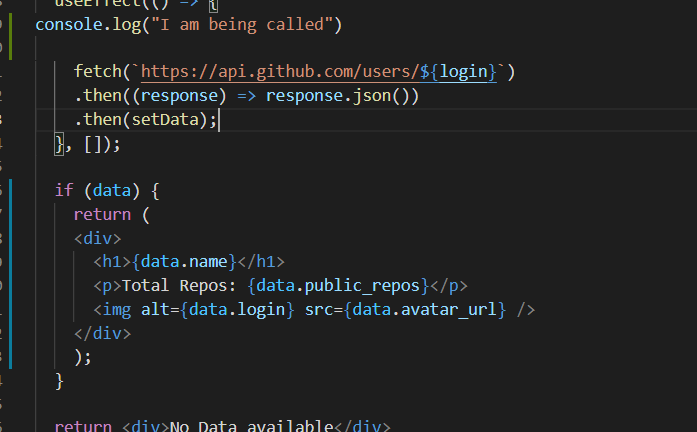
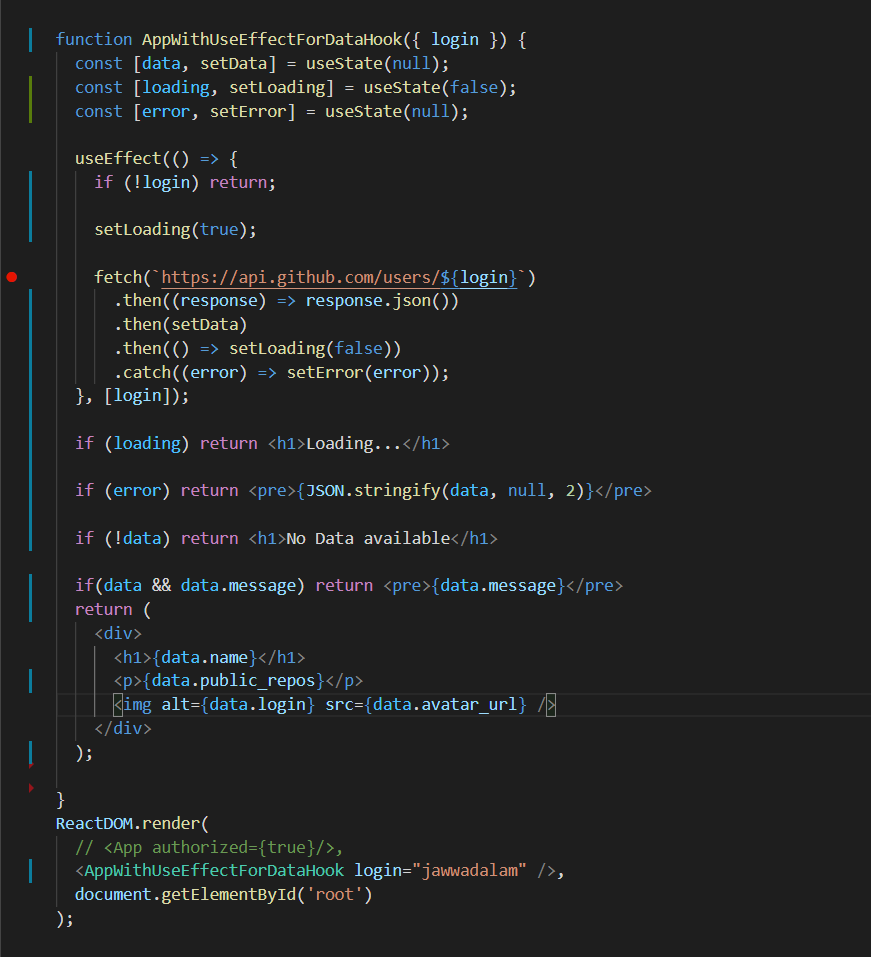
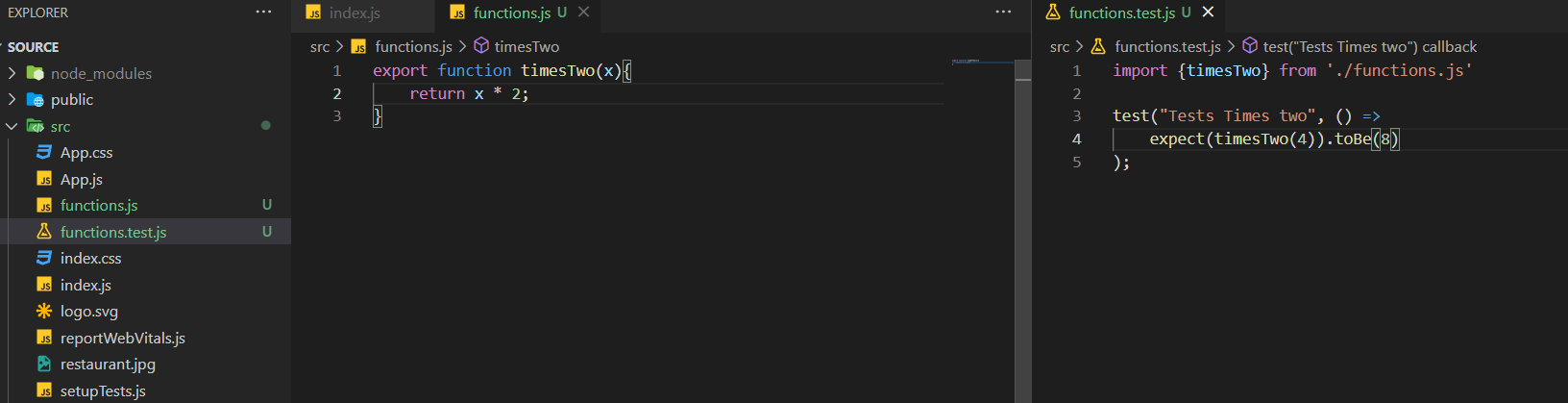
<p>Hello {props.name} </p>

);

}

* To set style property of a component we use JSX syntax with {} to pass in the objects like following, note first curly bracket is to start JSX object notation and brackets inside is Javascript object notation for passing in style object. Here we are not using css text-align, instead we use camel case because if use text-align this will break JSX

<ul style={{textAlign: “left”}}> </ul>

* 
* For lists, unique key is important. One way of doing that is assigning id of the map function inside the component to generate the unique key, and other way is to generate the unique id outside the component, part of the array being passed for lists, so this method is more reliable
* 
* We can use images in our components like we do text. First you have import the images from your src folder like import imageName from ‘./imageFileName.jpg’. Note imageName is something that you can use inside curly brackets inside your JSX syntax
* React.Fragment. When we have to render or return more than one elements from a React component, like for example inside ReactDOM.render method, usually we have an App object and then we have the target html element like this:
  + 
  + What if we want to add another AppTwo here…we get error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX Fragment
  + 
  + We can fix this error using React.Fragment
  + 
  + Or we can use simplified Fragment syntax like following
  + 
* Conditional Formatting: To use conditional formatting, we can use Fragment syntax <> {props.authorized ? <AuthorizedHeader /> : <DefaultHeader /> } </> like for example:
* 
* And 
* We can de-structure an array using e.g. const [firstArrayItem, secondArrayItem, andThirdArrayItem] = objArrayOfItems:
* 
* Similarly we can de-structure the objects as well. But note for Object de-structuring, we do not use the Square brackets that we use for Array de-structuring. we use {}, which are curly brackets
* 
* **useState** hook: for useState hook use
  + we import it from react using ***import { useState } from react***
  + then the useState is an array that we de-construct using array de-construction mechanism.. inside our component like **let [emotion, setEmotion] = useState(“happy”);**
  + Note the type of state object could be anything other than string as well
  + Then we are able to call the setEmotion to update the emotion state value, and we can use the emotion variable to display values or use anywhere inside the component
  + 
* useEffect: is the hook that we can use to react to side affects of things happening around component but those are not related to component rendering. Like for example if we want to console log whenever there’s change of state, we can use useEffect
  + useEffect function receives two parameters, first is the function to invoke when there’s an effect,
  + and second is the dependencies array to create dependencies on the objects for which this effect should be used. For example if we have two states that this component handles, we can add two useEffects and filter/tie them with specific state change, like in following code, we are tying two different effects to two different state items:
  + 
  + Note: if we do not pass dependency array parameter, the useEffect will be called everytime… and if we pass useEffect empty Dependencies Array like useEffect(() => {console.log(“message”)}, []); this effect will be called once only
* useReducers: they let you define functions that let you reduce the function calls, like for toggle behavior of a checkbox, you can use the useState hook as well, but with useReducer, you can provide the toggle anonymous function behavior while defining the reducer and then you can use that toggle with your checkbox, onChange event and also that will be called automatically to set the state, e.g.:
  + 
* To Fetch Data from API: we can use useEffect and invoke an API using fetch(url).then((response) => response.json()).then(setData); here setData is state setting function
  + 
* And we can use the data passed into parsed to show individual UI elements like following:
  + 
* For the fetch api, before call you can use loading and error state fields to determine and set states related to loading and error. While loading state is true, you can return the content related to loading, if error, you can show error, else show main data, like following:
  + 
* Testing React: We can use jest with crate-react-app. For that the convention is that you should have file ending with **test.js** , for example function.**test.js**.
  + You import the functions to the test file, and define test with first parameter is name of test, second parameter is the test function, which contains for example ***expects(functionName(param)).toBe(expectedValueToMatch)***
  + 
  + You use **npm test** command to run the tests