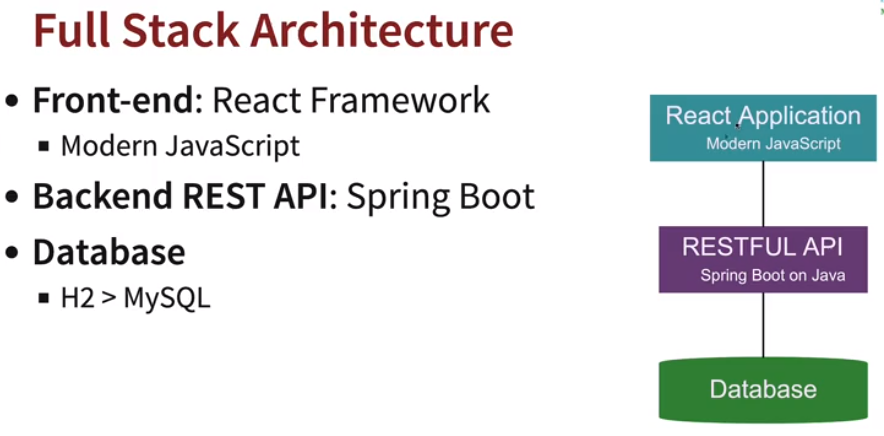
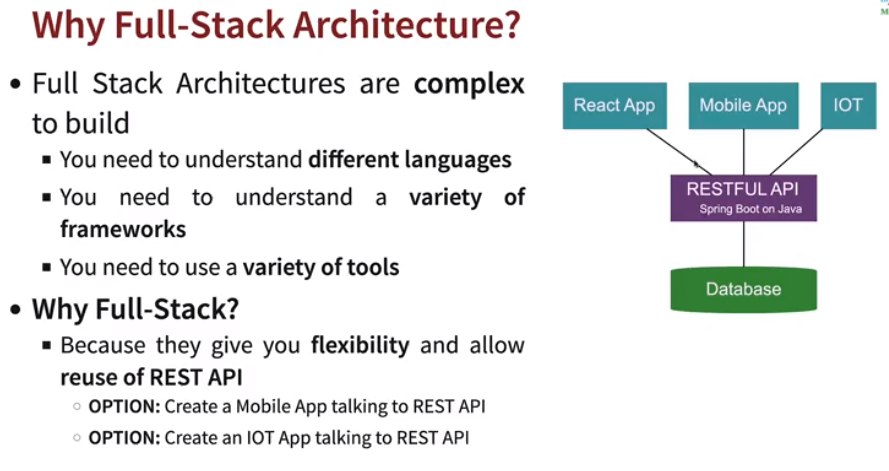
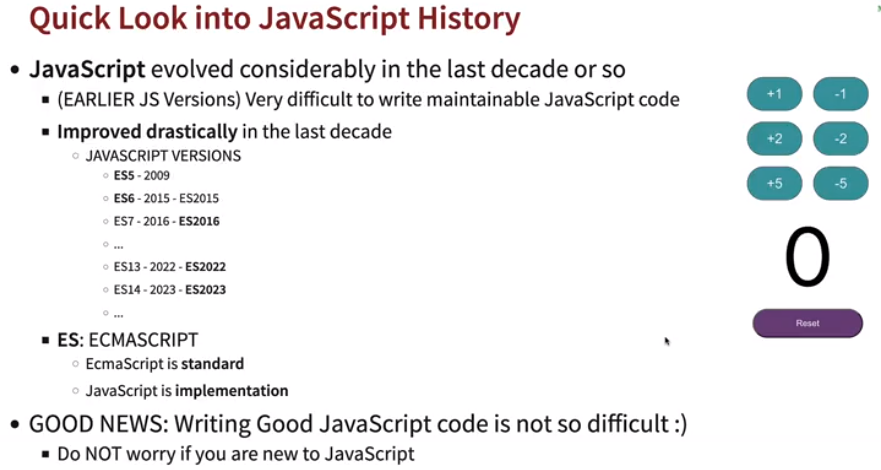
**REACT JS**

* Full Stack development is complex but it allows us to connect different types of applications, like webapplication, RESTApi and IOT applications by reusing the backend.
* It has mainly 3 components Application(React) 🡪 Restful API 🡪 Database.

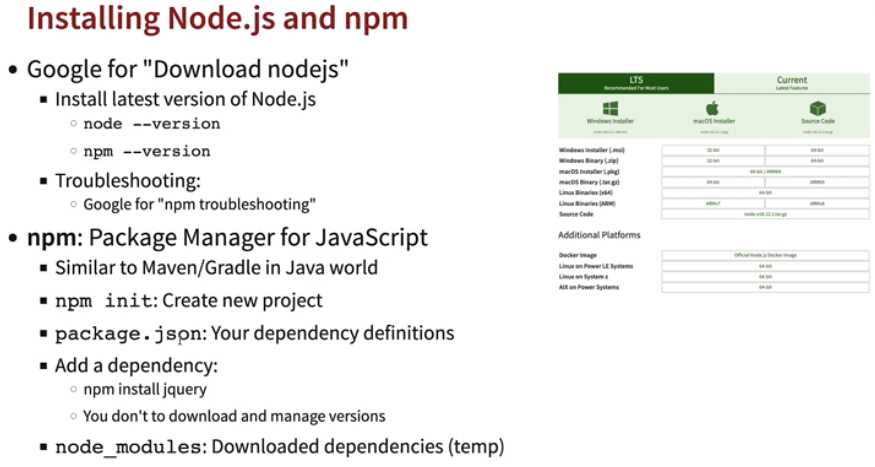
****

****

* **JavaScript:** Earlier writing JavaScript was difficult using DOM (document object model), i.e., manipulating different JavaScript pages.
  + **ES:** ES stands for ECMA Script which is like an interface for JavaScript or standard.
  + **JavaScript:** JS is implementation of ES (ECMA Script).

****

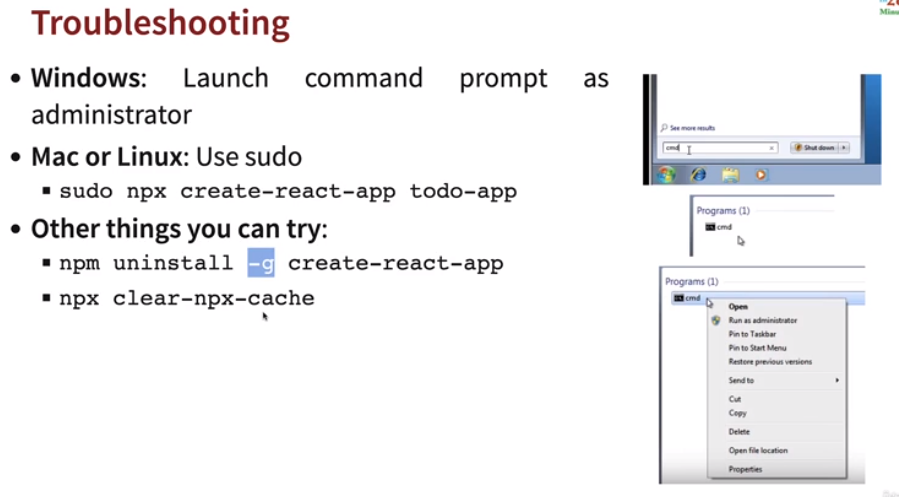
* **Installing Node JS:** <https://nodejs.org/en/download>
* **Node JS:** Node JS is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Just like JRE for the Java programs.
* **NPM:** Node Package Manager 🡪 It's a library and registry for JavaScript software packages. Similar to Maven or Gradle. It is used install, update and delete packages.
* **NPX:** Node Package Executor 🡪 It executes JS packages without installing it.
* **Create new project:** In command prompt type: *npm init.* It will initialize the project and create a package.json, similar to pom.xml which consists of all dependencies definitions.
* **node\_modules:** It is a temp folder where all downloaded dependencies are kept.

****

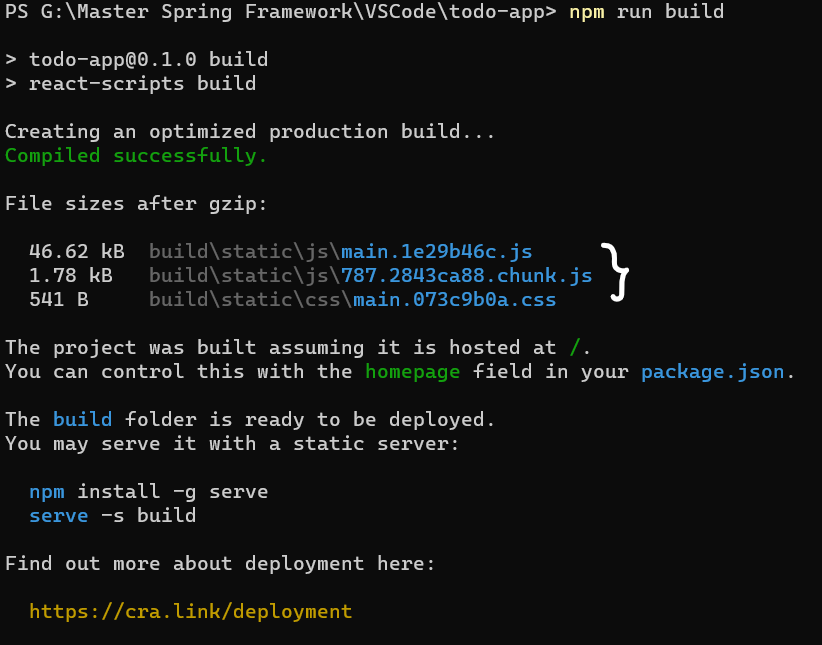
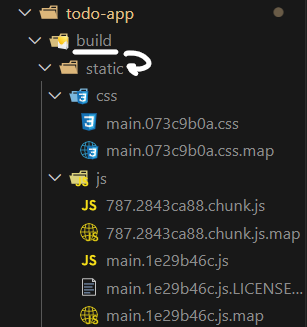
* **React JS:** React JS is used to create Single Page Applications (SPA), which is compatible with Windows, Mac and Linux OS.

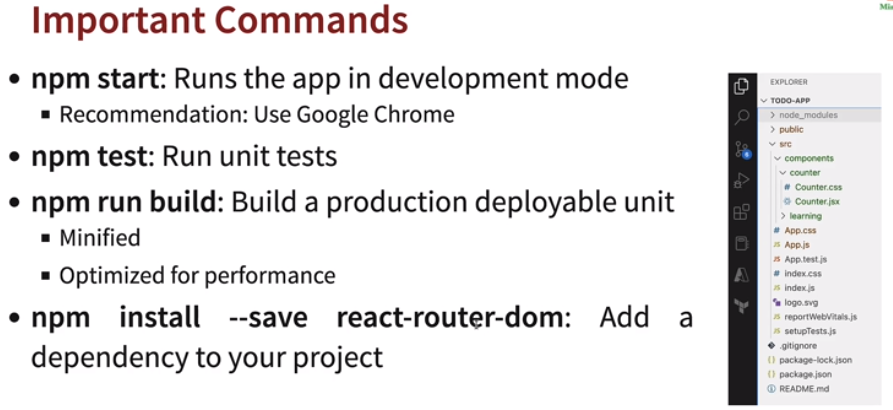
****

****

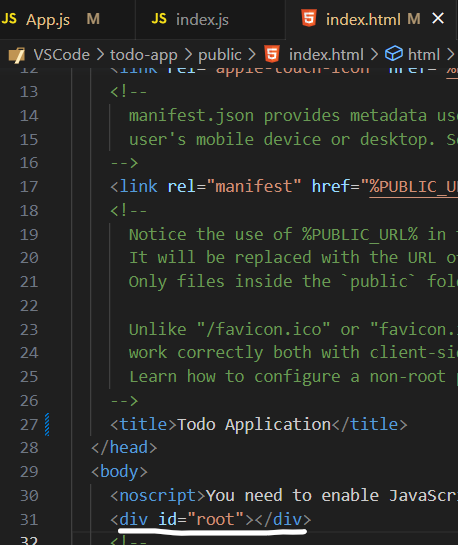
****

* **NPM Commands:** 
  + **npm start:** To start your application in development mode. The changes made in application files will immediately reflect in browser.
  + **npm test:** To run Junit test or test cases defined in application.
  + **npm run build:** Similar to like creating jars for java, it creates minified version of your application code and build files are necessary enough to run application in any environment.
  + **npm install –save <dependency>:** Add a dependency in project.

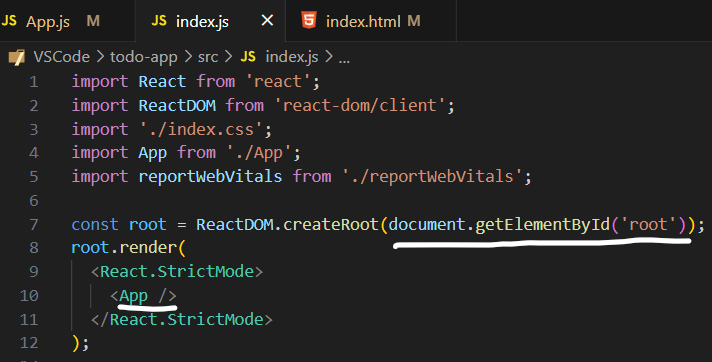
** **

****

* **Folder Structure:** 
  + **Index.html:** It contains a root div which is then mapped as a component in the index.js.

****

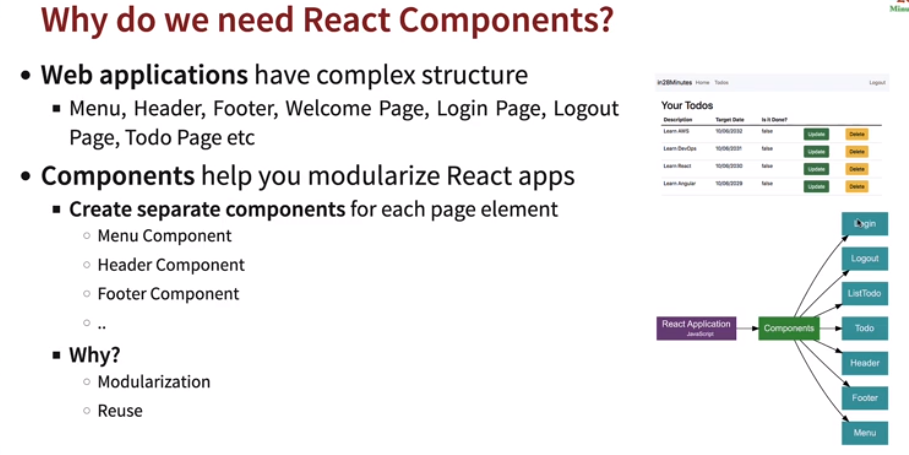
* + **Index.js:** It renders the app (root) component. App component will be the first to be loaded in.

****

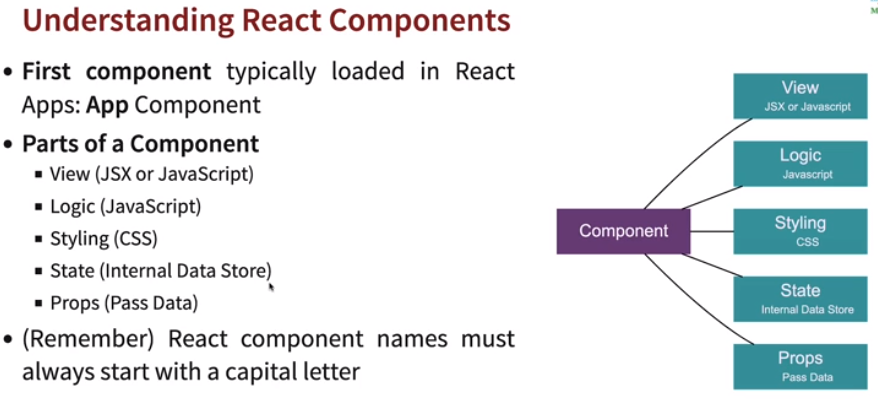
* + **App.js:** It contains the code for app component.

****

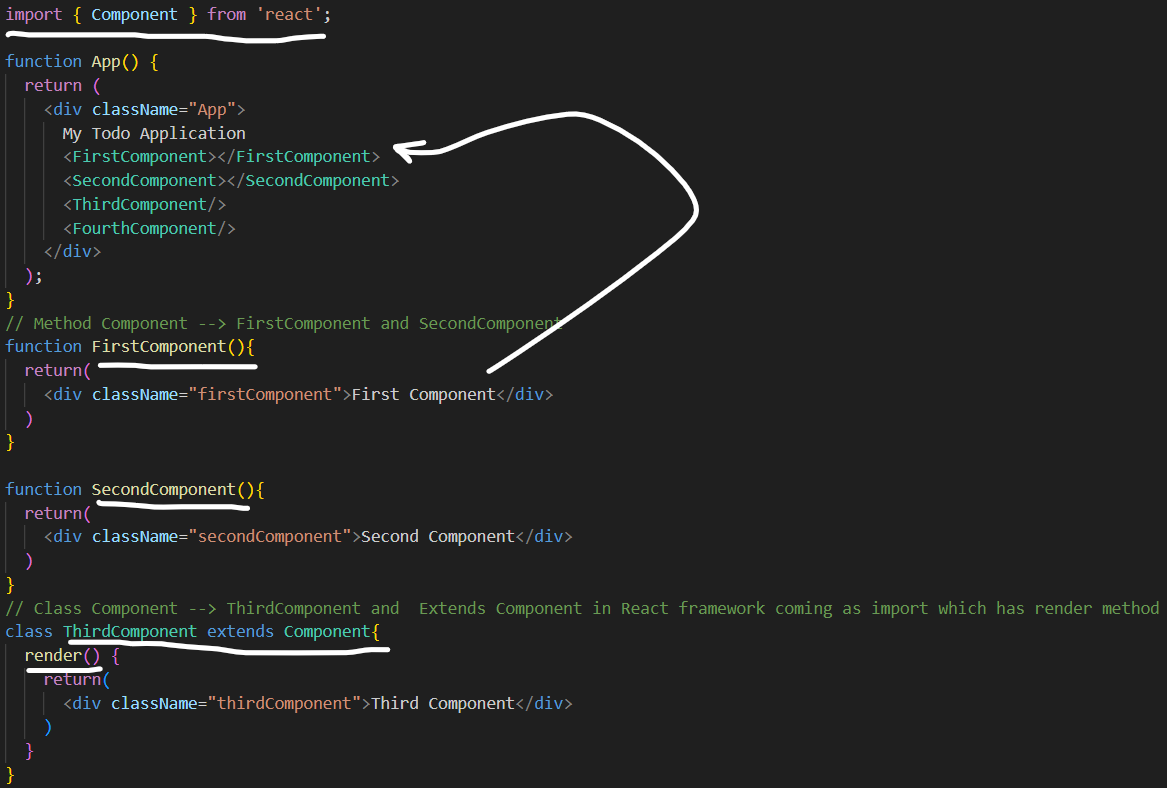
* **React Components:** React components helps in modularizing (separating into different sections) application. Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render().

****

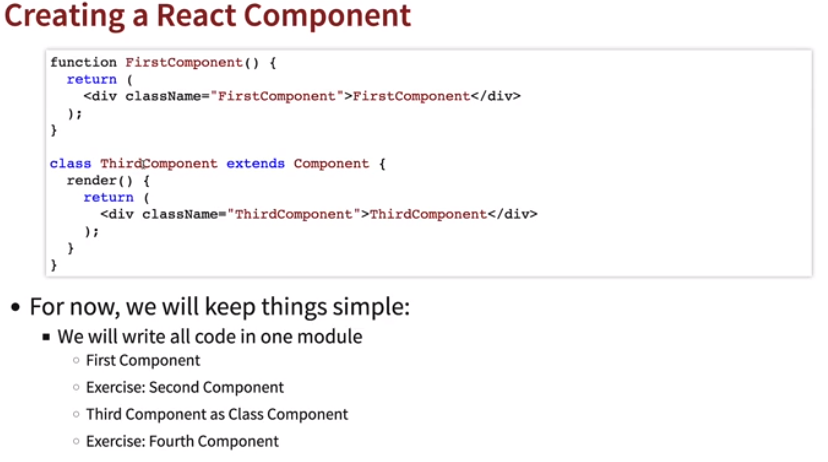
* **First Component loaded in React?** 
  + The first component that is loading in application is *App* component.
  + Components must start with capital letters.
  + All other components created will be child of *App* component.
  + A component can have:-
    - View (JSX or JS)
    - Logic (JS)
    - Styling (CSS)
    - State (Internal Data Store)
    - Props (Pass Data)

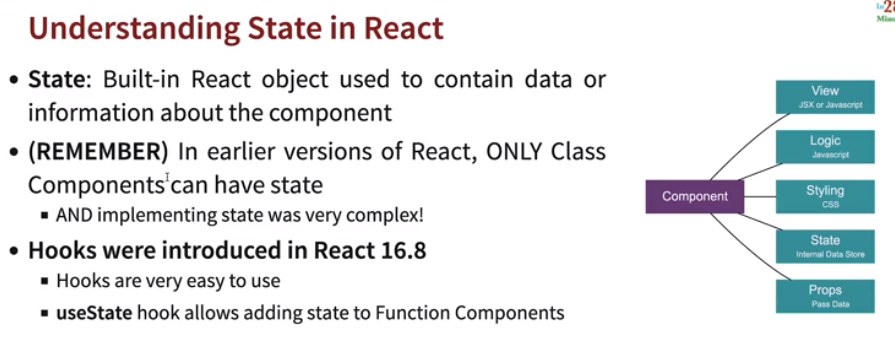
****

* **Creating a component:** Every component should be a child of *App* component or called in *App* component.

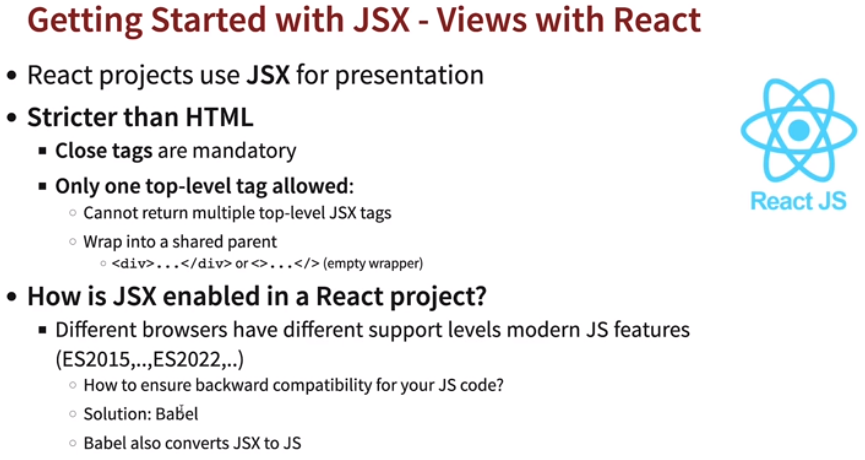
****

* In earlier versions of *React < 16.8* only *Class Components had the state* (some data) whereas onwards *React >= 16.8 Function Components also have states* as *Hooks* were introduced.

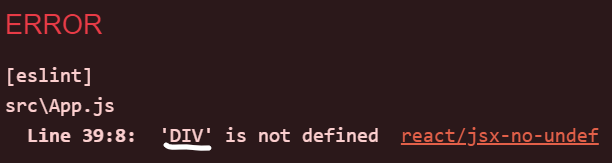
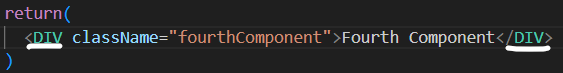
****

****

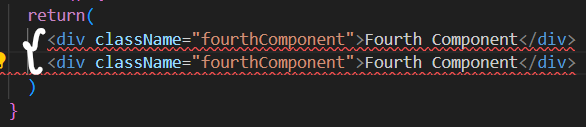
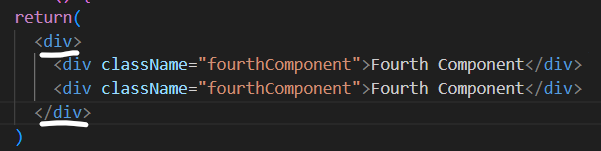
* **JSX:** JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

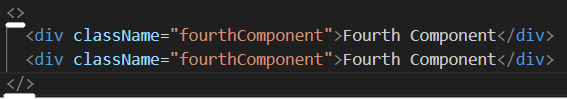
****

* + Stricter than HTML. Case of components should be capital case or elements needs to be small case.

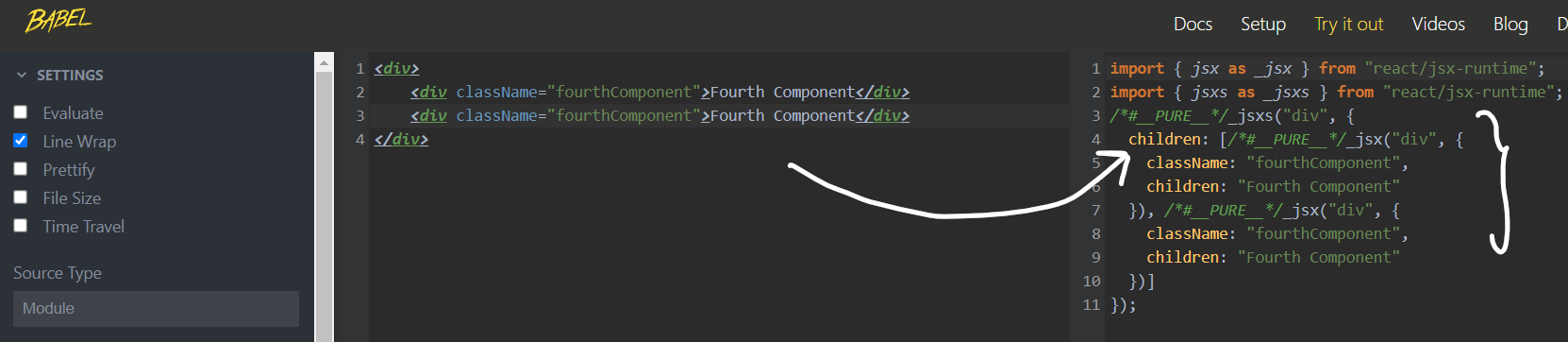
** **

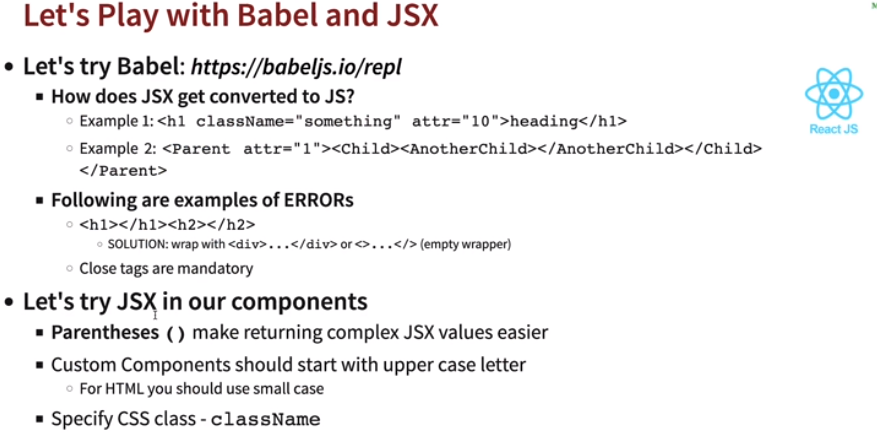
* + Only one Top level tag is allowed, otherwise if we have multiple tags in the component we need to use <></> or <div></div> to enclose.

** **

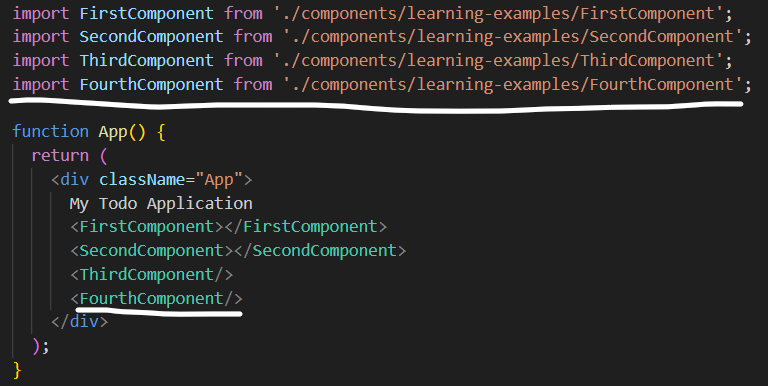
****

* **Babel:** Browsers now uses new JS (ES2022, ES2023….), to ensure compatibility with older versions of browsers, Babel is used. Babel is responsible for converting your JSX to JS.

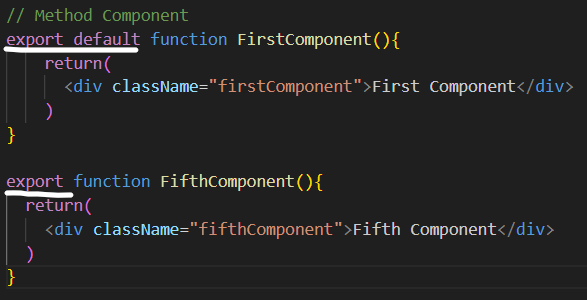
****

****

* **Exporting component from different files:** 
  + Import it in *App.js* as *import <Component> from ‘<Location>’;*

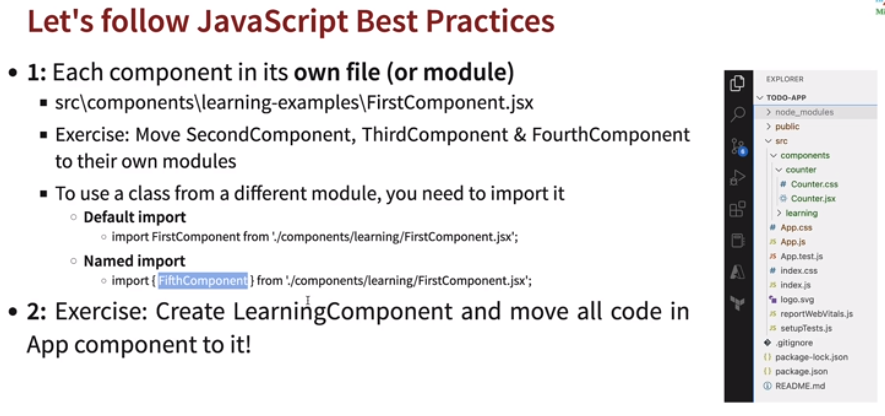
****

* + Use *export/export default* in the JSX file.

****

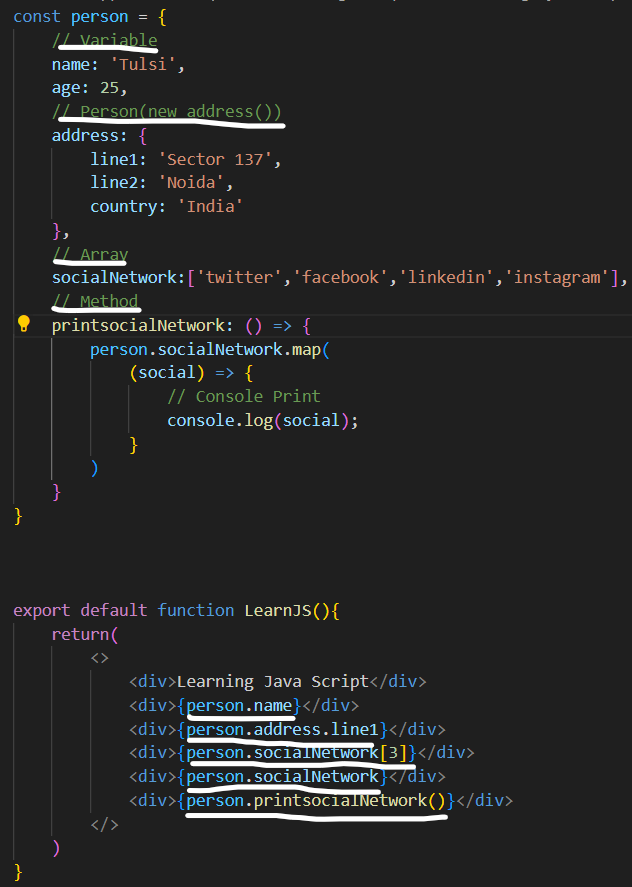
* + By default there’s only one *export default is allowed.* If we need to import another export component which isn’t an *export default,* while importing we need to use ***{}***and if ***{}*** is not used then only *export default will import.*

****

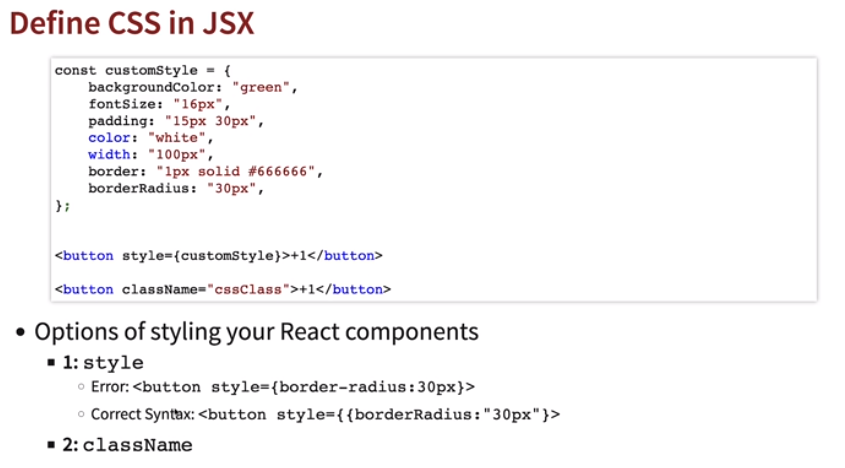
****

* **Learning JS:**

****

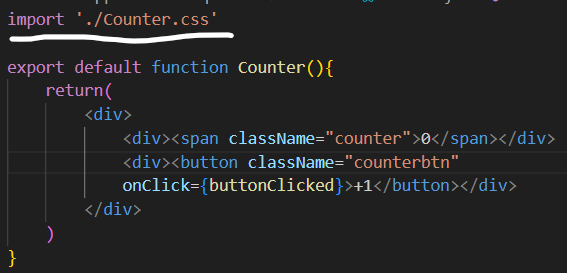
****

* **Styling (CSS):** In JSX if we define style inside JSX or element we need to use ***{{<style>}}***. Otherwise it will not work or other way is to store it in Java object and access as style variable.Also ***–*** are not allowed in CSS property so we need to use Capital first letter. Ex: *font-size: 🡪 fontSize:*

****

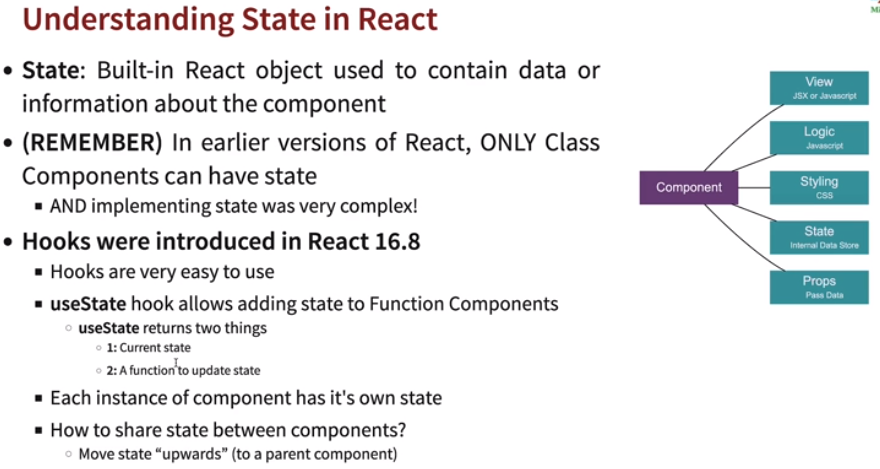
** **

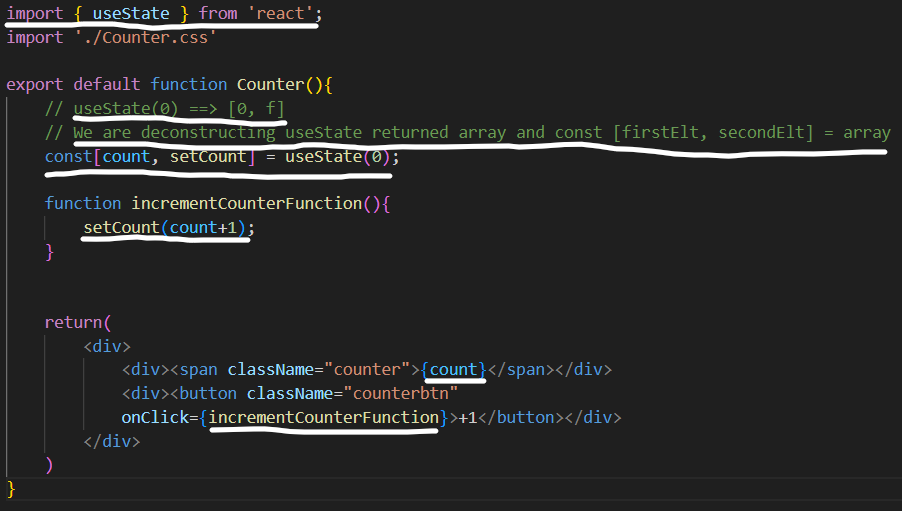
* + Importing from another CSS file, just need to add *import ‘<file>’.*

** **

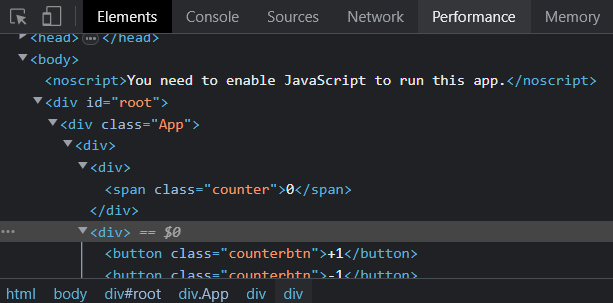
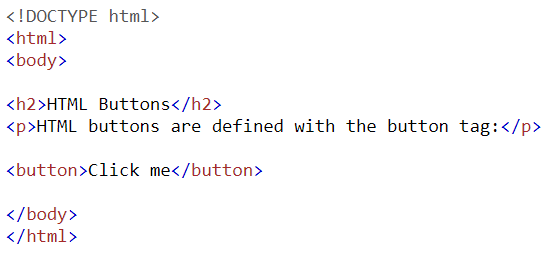
* **State:** Built in React objects to contain data or information of a component. The *useState* Hook can be used to keep track of *strings, numbers, booleans, arrays, objects, and any combination of these.* A *useState* will return or have array of 2 things, one is current state/value and other is function for changing its value.

****

****

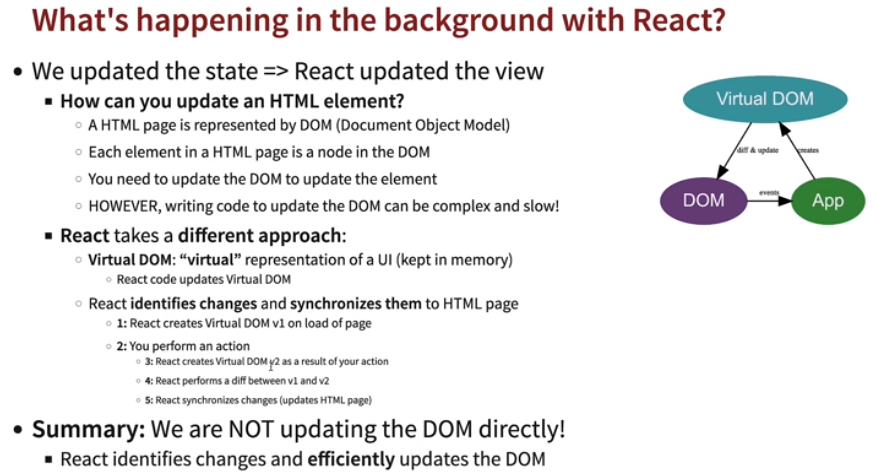
****

* **DOM (Document Object Model):** The Document Object Model (DOM) is a language-independent model made up of objects representing the structure of a document. HTML is one language for writing such documents.

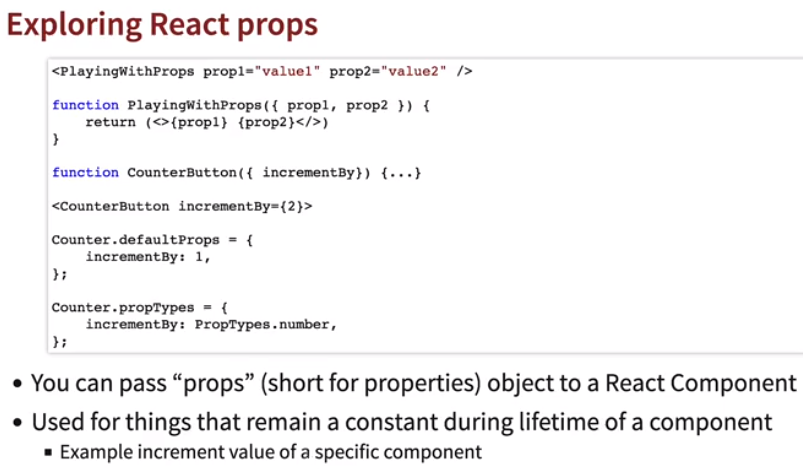
** **

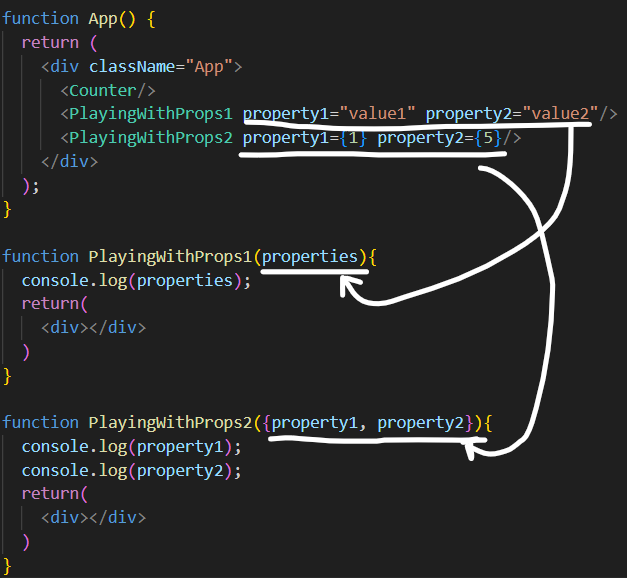
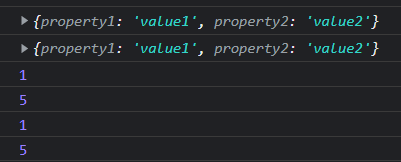
**DOM HTML**

* **How React updates HTML so fast or what’s happening in the background?**
  + In earlier approaches, we used to write the code to find the specific elements in HTML and write code to update its values.
  + React keeps the *virtual DOM in memory* when we make changes, React compares the changes b/w *v1(old vDOM)* and *v2(new vDOM)* and updates DOM with differences b/w *v1* and *v2*. React identifies changes and synchronizes with HTML pages. This makes React much more efficient.

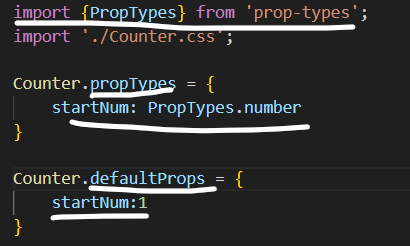
****

* **Props:** Props is short for properties are the values that remain constant and never change throughout. *<Component property1=”<val1>” property2=”<val2>”…../>.* Properties return an object of these values *(properties) 🡪 returns object of props*, or if we want to access values directly instead from object we need to have *{property1, property2…..}* defined in components.

****

** **

* Here we’re setting the properties attribute by specifying it’s default type should be number and default value should be *1*.
  + *Import {propTypes} from ‘prop-types’;*.
  + Then use as 🡪 *<Component>.propTypes = {<prop>: <attr>}.*

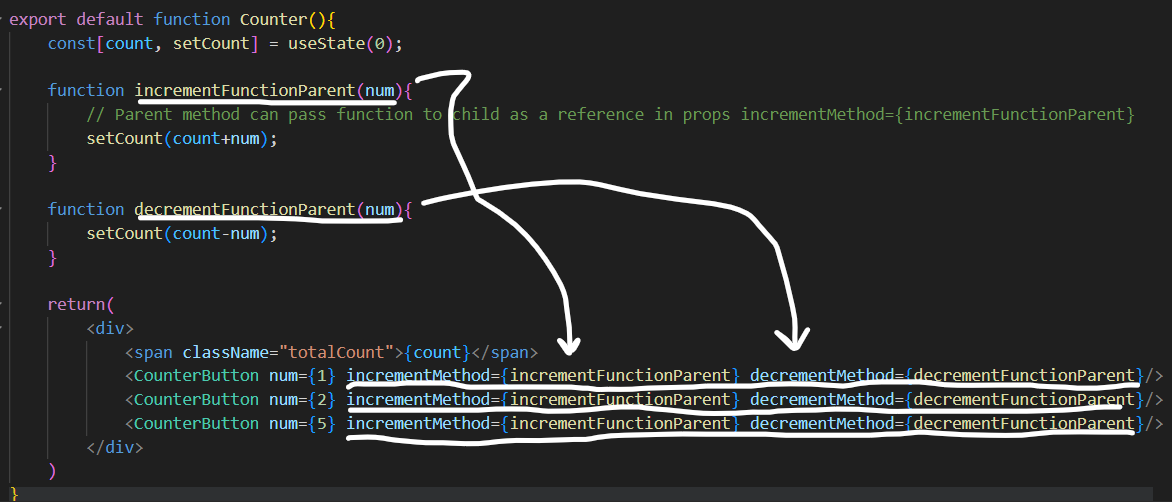
****

* In case we change the propType to some other let’s say String we’ll get error in console.

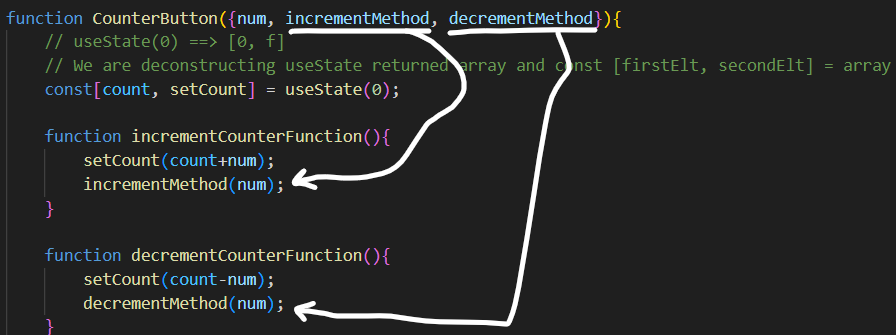
****

****

* **Passing Parent method to child:** Parent method can be passed as props into Child Method, or it can be directly called using *()=><ParentFunction(<arg>)>.*

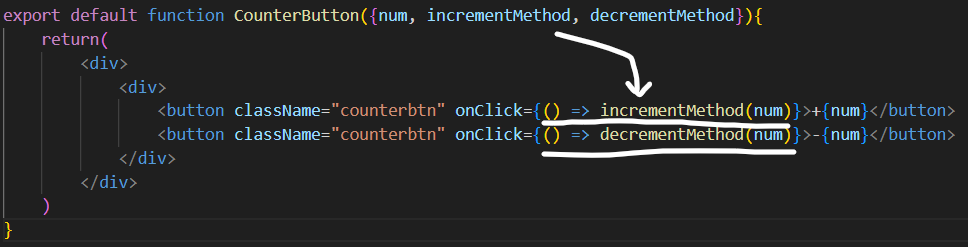
****

**Parent Class**

****

**Child Class**

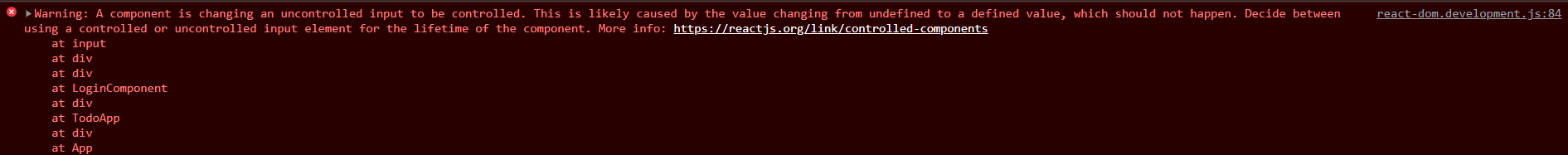
**Or**

****

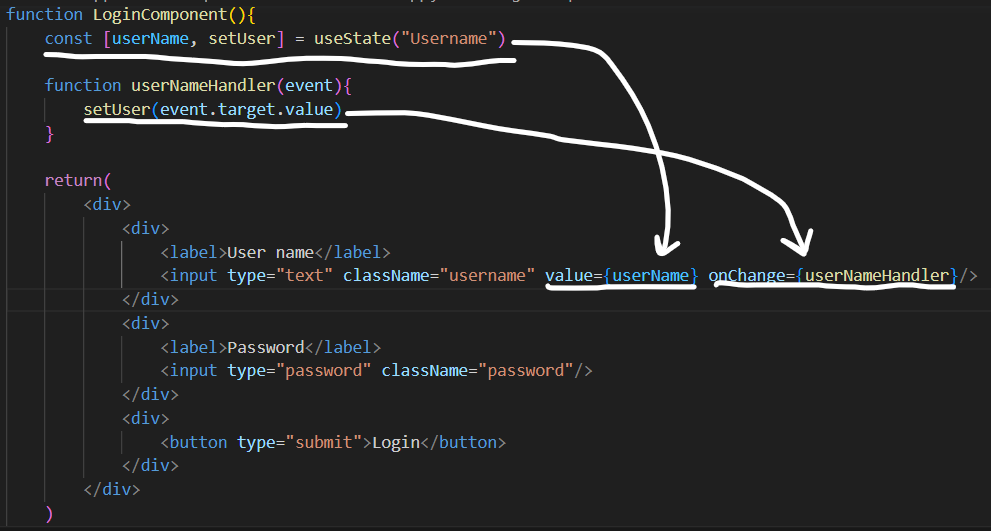
**Child Class**

* **Controlled Component:** Controlled components in React are those in which form data is handled by the component’s state.
* **Uncontrolled Component:** Uncontrolled components are those for which the form data is handled by the DOM itself. It refers to the fact that these components are not controlled by React state.

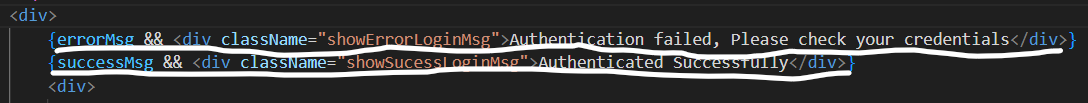
****

****

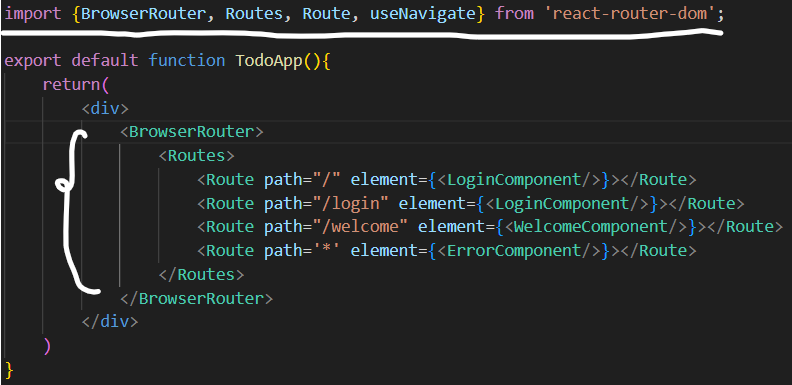
* + In the below code, since putting *value=”<def\_val>”* in *input field,* in React the component value should be in synch with the DOM value. By this we’re changing from uncontrolled input to controlled input and values are changing from undefined to defined value. We are in this case using React *Hook* *useState* and mapping it to the onChange event in the fields.

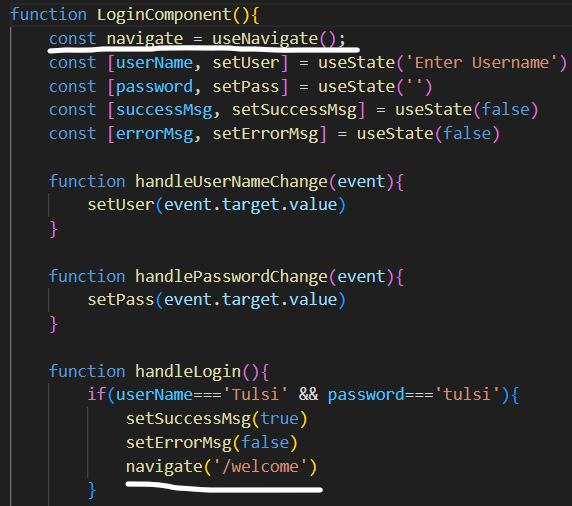
****

* *true && ‘<Str>’ 🡪 String*
* *false && ‘<Str>’ 🡪 false*

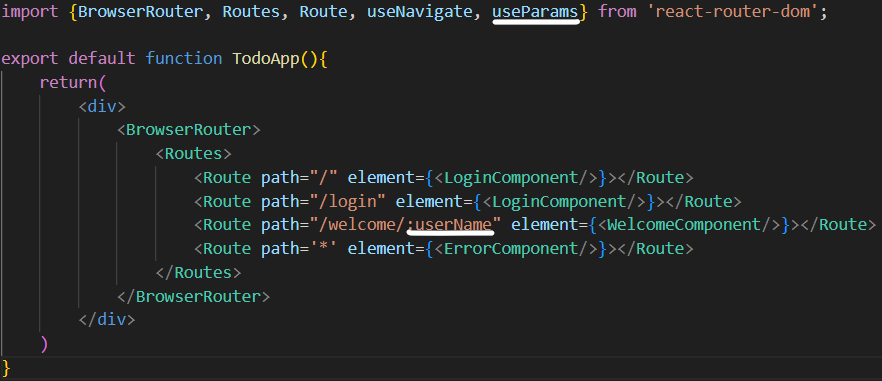
**

* **React Router DOM:** In the application, React Router is utilized to define various routes.

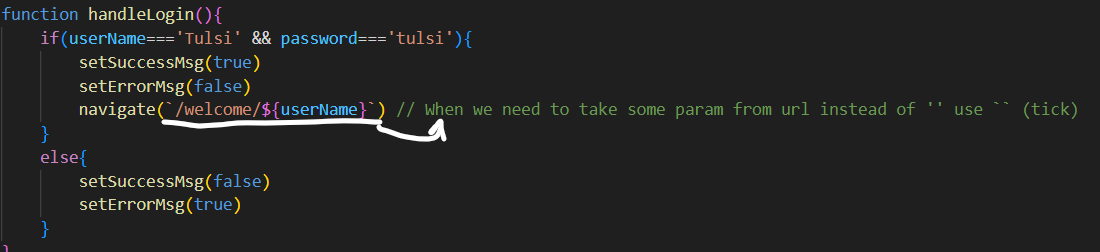




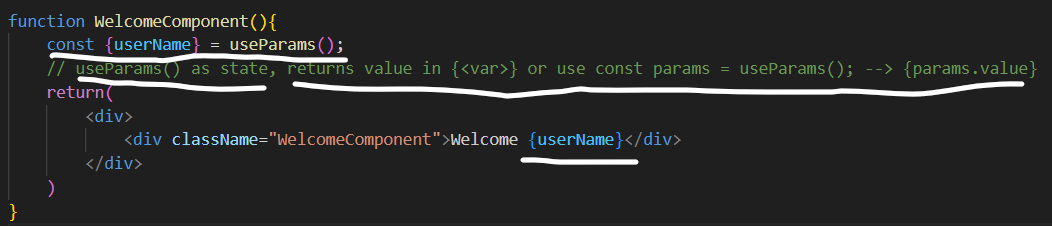
* + *npm install react-router-dom* 🡪 Install package
  + *BrowserRouter 🡪* It stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.
  + *Routes 🡪* It looks through all its child routes to find the best match and renders that branch of the UI.
  + *Route 🡪* It has a *path=”<url>”* and *element=”{<Component/>}”*. It is binding an URL to a component.
  + *Naviagte 🡪* It changes the current location to the location specified when it is rendered. *useNavigate()* is a state.
* **useParams:** *useParams* is used to get some parameters passed in url and pass it to a component.
  + Import *useParams* and in *Route* mention parameter as ***:***



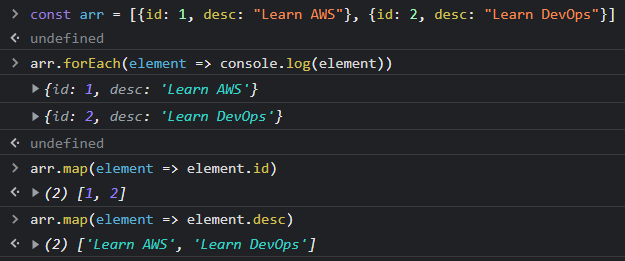
* + In *navigate* instead of **‘’** (single quotes) use **``** (tick) and mention param as *${}*



* + *useParams()* returns an object of value in url, either we can access value directly by *const{param} = useParams();🡪 {params}* or *const params = useParams() 🡪 {params.value}* (object.value).



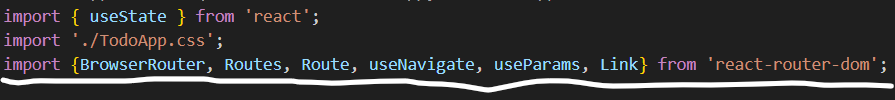
* **Array Map and forEach:** 
  + *arr.map(element => element.<var>) 🡪* It will get the list of values specific to a field.
  + *arr.forEach(element => console.log(<ele>) 🡪* It will get or print all the values.



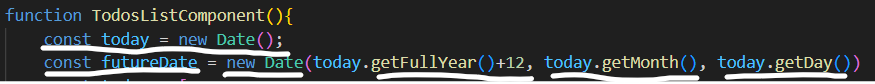


* **React Link:** React link is imported with React router and is similar to *<a href=””></a>* but the difference is *<a></a>* when used loads whole page, where using *<Link to=””></Link>* it redirects to next component instead of loading whole page.



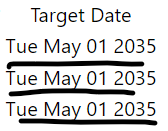


* **Date:** 
  + *const <tvar> = new Date(); 🡪* Get today’s date
  + *const <fvar> = new Date(<tvar>.getFullYear()) 🡪* Get current Year
  + *const <fvar> = new Date(<tvar>.getMonth()) 🡪* Get current Month
  + *const <fvar> = new Date(<tvar>.getDay()) 🡪* Get current Day of week



* + *<fvar>.toDateString() 🡪* Convert date to String format.





* **Bootstrap:** Bootstrap is used for styling.
  + *npm install bootstrap 🡪* To install bootstrap in node modules
  + *import ‘bootstrap/dist/css/bootstrap.css’ 🡪* To import bootstrap in components.
* **Context:** A context is shared state between all components. Context is primarily used when some data needs to be accessible by many components at different nesting levels.
  + *Create a context 🡪 import { createContext } from "react";*
    - *Const <state> = createContext()*
  + *Share context across all other children ({children}) components and create a state to be shared.*

*function AuthProvider({children}){*

*const[number, setNumber] = useState(10);*

*return(*

*<div>*

*<AuthContext.Provider value = {number}>*

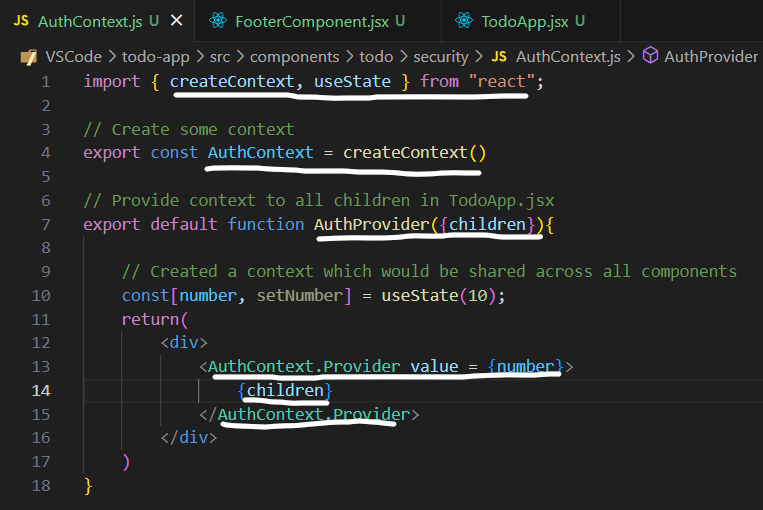
*{children}*

*</AuthContext.Provider>*

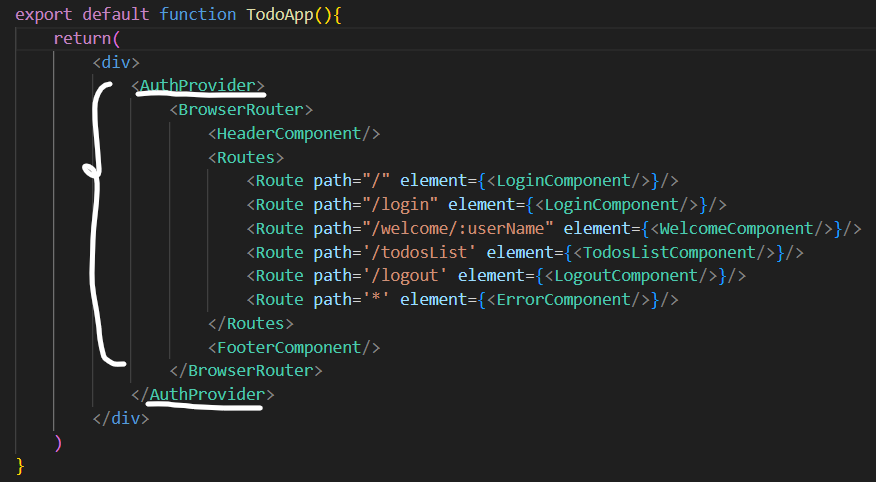
*</div>*

*)*

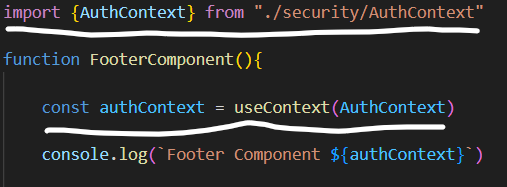
*}*

**

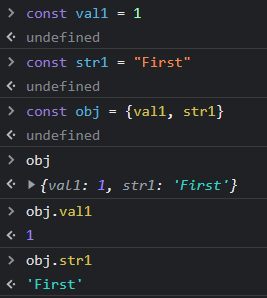
* *Wrap AuthProvider across all children components.*

**

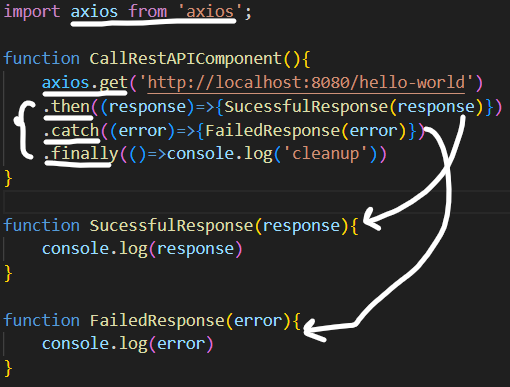
* *Use AuthContext in any other component.*

**

* **AuthProvider:** Just like a dataProvider, an authProvider is an object that handles authentication and authorization logic.
* **Setting object in JS:**

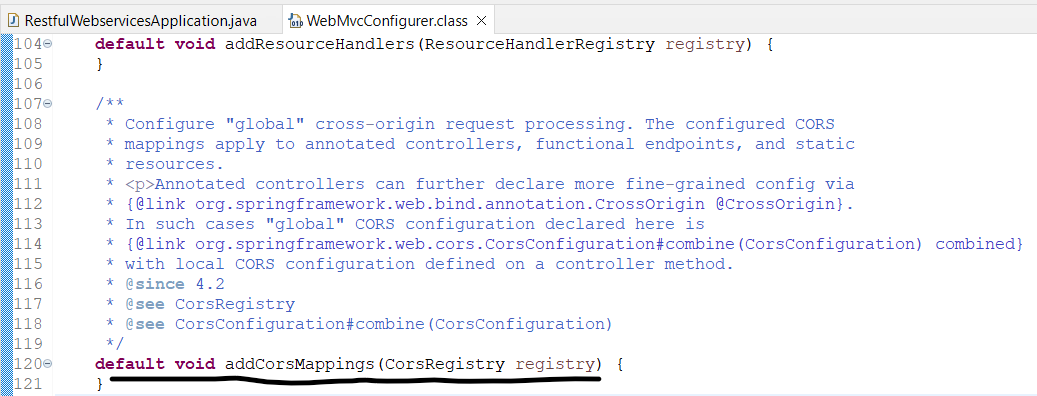
**

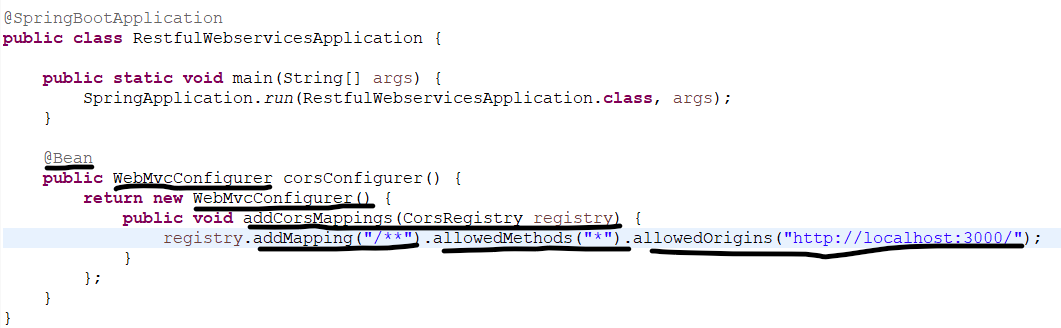
* **Asynchronous vs Synchronous:** *Asynchronous* is a non-blocking architecture, so the execution of one task isn't dependent on another. Tasks can run simultaneously. *Synchronous* is a blocking architecture, so the execution of each operation is dependent on the completion of the one before it.
* **Axios:** *Axios* is a framework used to called backend APIs. *Axios* is a promise-based HTTP client that lets you handle asynchronous HTTP requests.
  + *npm install axios*
  + *import axios from 'axios';*



* **Promise:** A Promise is an object that represents an asynchronous operation.
* **Behind the background Axios:** When we use *axios* to make a call to restAPI, an *asynchronous* call happens, which returns a *promise object. Axios* then the response came from API if it’s successful we need to handle it using *.then((response) => {<>})*. If it fails or in error status *.catch((error) => {<>})* is used to handle errors. *Finally* is executed at last irrespective of status, *.finally(() => {<>}).*
* **CORS (Cross Origin Requests):** Websites that have the combination of the same scheme, hostname, and port are considered *same-origin*. Everything else is considered *cross-origin*.

By default *CORS* is blocked in Spring boot so if we need to add allow *CORS* we need to use a implementation *WebMvcConfigurer 🡪 Override method addCorsMapping()* with our own configuration.



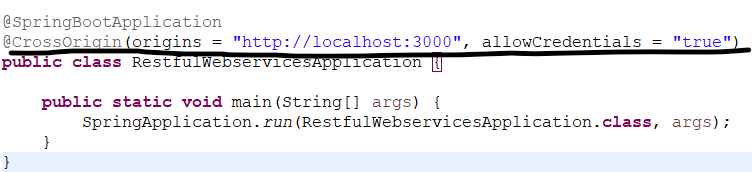


In above we created a *@Bean* object of *WebMvcConfigurer* return its object inside that object we *override* with our configuration for *registry.*

*.addMapping(“/\*\*”) 🡪* For all urls from cross origin.

*.allowedMethods(“\*”) 🡪* Types of methods allowed (GET, PUT, POST).

*.allowedOrigins(“<url>”) 🡪* Allow specific origin url requests.



Or we can do the same using *@CrossOrigin(origins = "<url>", allowCredentials = "true")*

1. In old restAPI project, created a Basic Authentication for your restAPI, disable/comment in your *SecurityConfiguration --> filterChain()* to address *CORS issue.*

*//All requests incoming should be authenticated*

*//http.authorizeHttpRequests(auth -> auth.anyRequest().authenticated());*

2. If in case want to still keep basic auth, pass in Axios the basic auth like this in React Application to address *CORS issue*. *btoa()* method creates a Base64-encoded ASCII string from a binary string.

*var username = '<User>';*

*var password = '<Password>';*

*var credentials = btoa(username + ':' + password);*

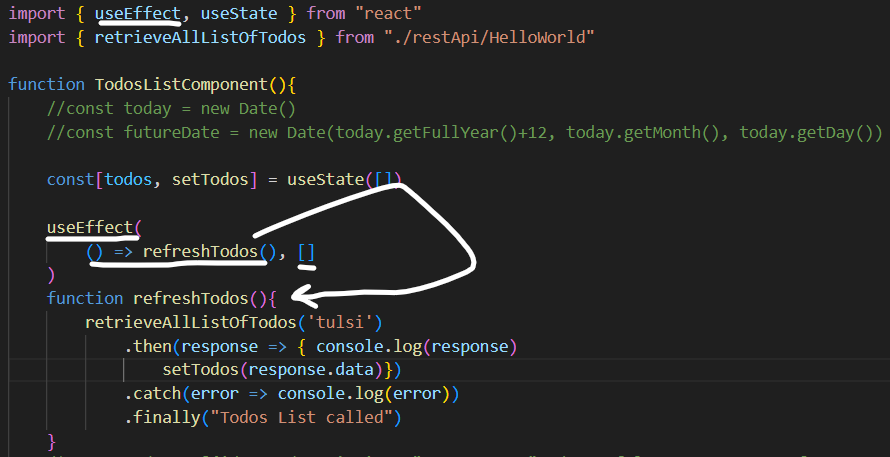
*var basicAuth = 'Basic ' + credentials;*

*axios.get('http://localhost:8080/hello-world', { headers: { 'Authorization': basicAuth } })*

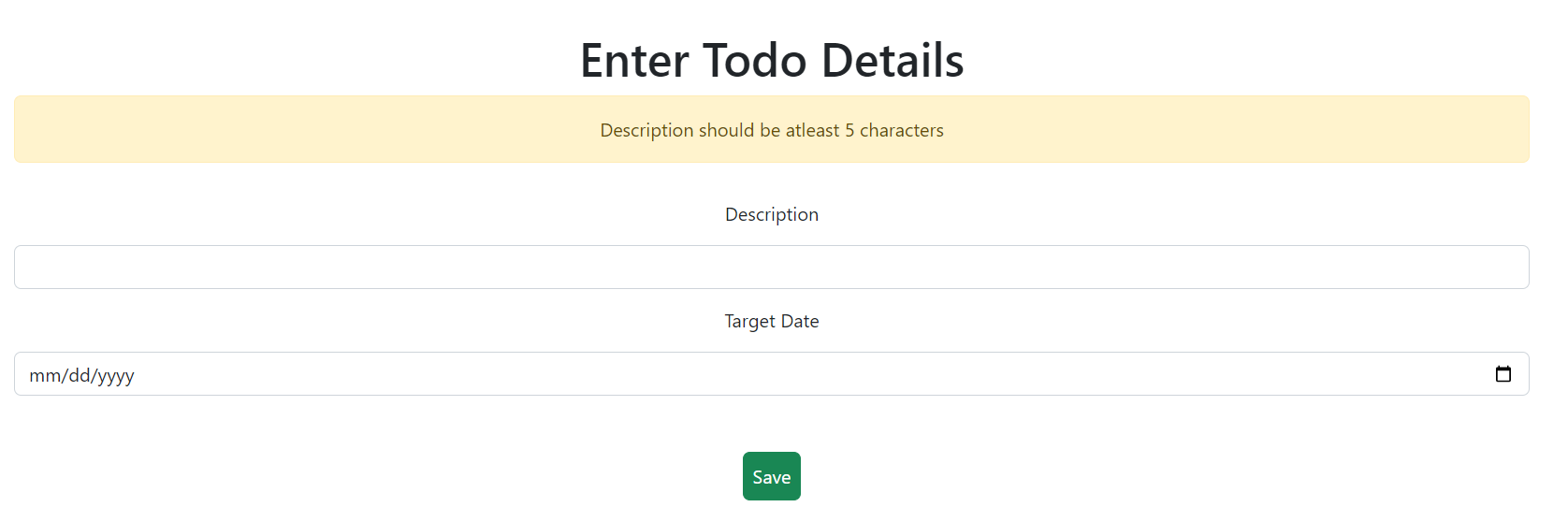
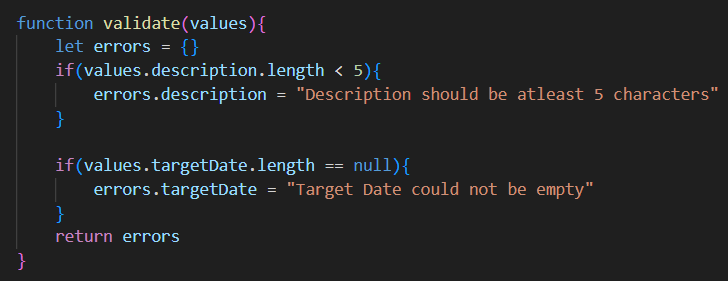
In the below code we refactored our *axios* calls to a separate js file. *Axios* provides a *create* method where we can configure our common baseUrl. In case we need to pass some parameter in the url we need to use *`` (tick) instead of ‘’ (single quotes)* and *path variable* with *${<var>}*. After that all we need to is import the method from js files in components and call the method.

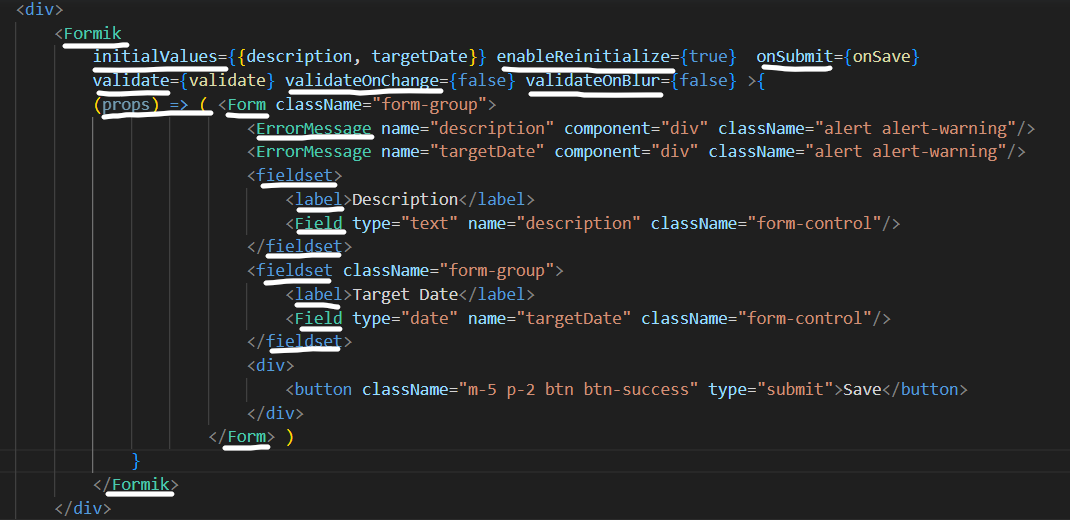


* **useEffect:** The *useEffect* Hook allows you to perform side effects in your components. Some examples of side effects are: fetching data, directly updating the DOM, and timers. *useEffect* allows to load the data in component once component is ready.
  + In *useEffect* we need to specify when to load the data, otherwise it will keep loading data again and again so, we need to *[]* to load at start only.

****

* **Formik:** *Formik* is third party React form library. It provides basic form programming and validation. It is based on controlled component and greatly reduces the time to do form programming.
  + - *npm install formic*
    - *import { Formik, Form, Field*, *ErrorMessage* *} from ‘formik’;*
* **Inside <Formik></Formik>:** 
  + We need to import *Formik, Form, Field* and *ErrorMessage* from the package *'formik'.*
  + *Formik* provides initialization of field values using *initialValues={{<val1>, <val2>}} 🡪* Initial field values of the form.
  + Using *enableReinitialize={true} 🡪 Formik* should reset the form if *initialValues* changes.
  + *onSubmit={<func>} 🡪* On submit of form call a function.
  + *Validate={<func>} 🡪* Validation function is called for checks, by default it checks at every *onChange* and *onBlur.* So we need to set *validateOnChange* and *validateOnBlur* as false.
  + *(props) 🡪 <Formik>* is a component that helps you with building forms. It uses a render props pattern or return values in props.
  + The *<fieldset>* tag is used to group related elements in a form. The *<fieldset>* tag draws a box around the related elements.
  + The *<form>* element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.
  + In *validate* function, created an empty *errors = {}* which will be returned once all validations are done. This function is taking *values* from the *Formik.*

****

****

**==** is used for comparing two variables irrespective of the variable's datatype.

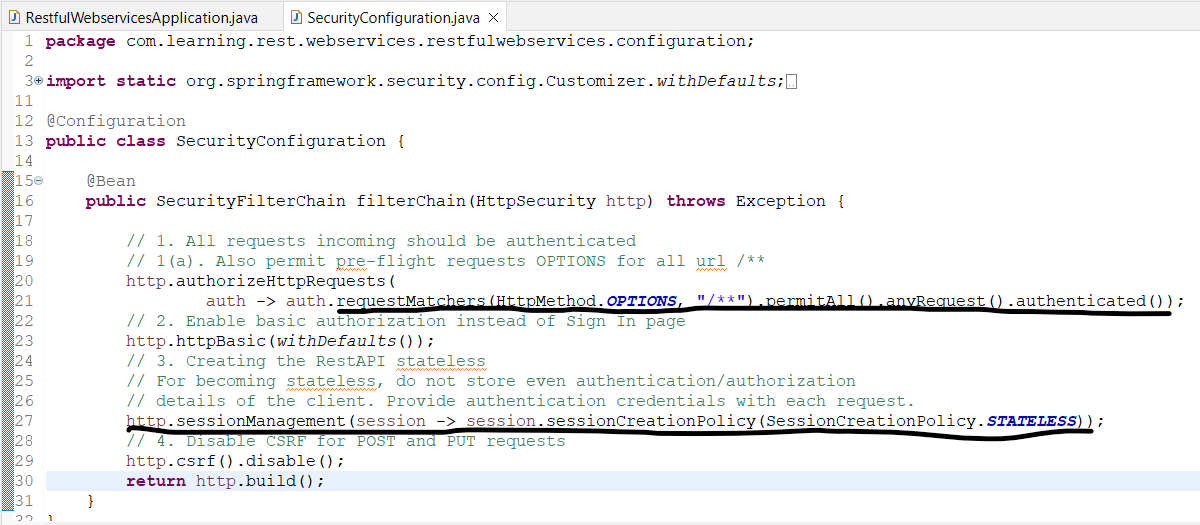
**===** is used for comparing two variables, but this will check the strict type, which means it will check the datatype and compare two values.

* **SessionCreationPolicy.STATELESS:** To make API stateless i.e., not store any kind of authentication and independent for each request.
* **OPTIONS:** A CORS preflight request is a CORS request that checks to see if the CORS protocol is understood and a server is aware using specific methods and headers.
  + Add in SecurityConfiguration of Spring Boot RestAPI, otherwise will get pre-flight errors even though calls from React has headers with basic token.

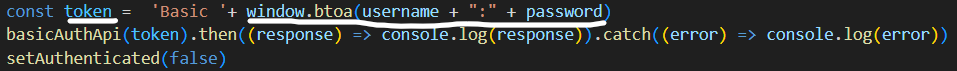




* + - *.requestMatchers(HttpMethod.OPTIONS, "/\*\*").permitAll()*

**

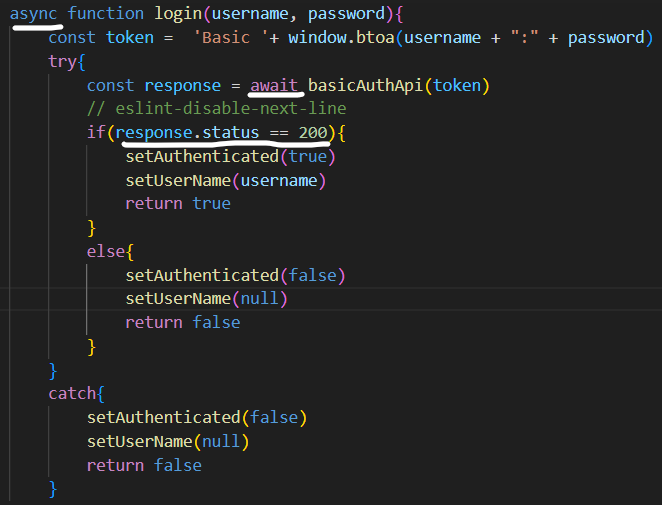
* **BTOA:** The *btoa()* method encodes a string in base-64.
* **Creating Basic Auth Token:** *const token = 'Basic '+ window.btoa(username + ":" + password)*

**

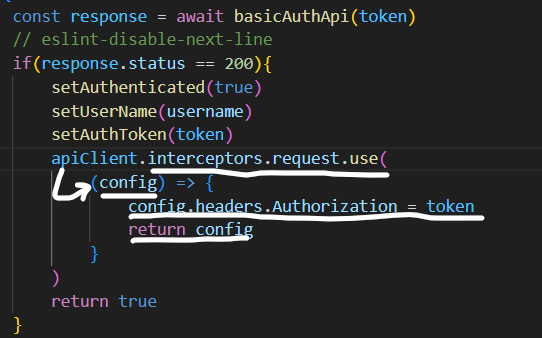
* **Async and Await:** The keyword *'async'* before a function makes the function return a *promise*, always. And the keyword *await* is used inside *async* functions, which makes the program wait until the promise is resolved, otherwise if we don’t use *asynch* with a function then next statement will be executed immediately. *await* will only execute next line once *promise* comes back or *async* completed it’s execution. All functions calling *async* also need to be *asynch* and wait for *promise* to be returned.







* **Interceptors:** *Interceptors* can be used to *add headers, modify requests, handle errors*, and much more.
  + In below code, *apiClient* is js file where all API calls are configured, *apiClient.interceptors.request.use 🡪* All the requests going forward use the following configuration.
  + *Config.headers.Authorization = token 🡪* Set the token in the headers and return the configuration back to API, all API calls will have headers added to them.



* **Basics of JWT:** Basic (Base-64) can be decoded by anyone for authorization, hence we can use JWT.

