The complete React full stack – Coding Revolution Fernando 01/16/21

# (buy - The Complete 2021 Web Development Bootcamp

Npm start, test test. Npm start will initiate a development server to test the code.

Npm run build, npm run eject. When running eject, all the hiding configuration files become visible.

Bundle.js is provided to index.html by React. It includes all the JS code we write.

React.createElement(‘element’,{className},”content”) // React.createElement(‘h1’,{className:”title”},”Hello World”)

From a component, always rturn only a single element. If multiple, wrap it with <div> or <React.Fragment> or just <> </>

Javascript inside jsx can return only a single statement.

Npm install –g create-react-app

Create-react-app <app-name>

If something is sourced into components outside of src folder in React, will not work.



Git code

React events are called synthetic events.

String interpolation is always done using backticks and $

There is no 2 way binding in React. If needed, use ‘state’.

# React.Component

This page contains a detailed API reference for the React component class definition. It assumes you’re familiar with fundamental React concepts, such as [Components and Props](https://reactjs.org/docs/components-and-props.html), as well as [State and Lifecycle](https://reactjs.org/docs/state-and-lifecycle.html). If you’re not, read them first.

### https://reactjs.org/docs/react-component.html

### The Component Lifecycle

Each component has several “lifecycle methods” that you can override to run code at particular times in the process. **You can use**[**this lifecycle diagram**](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)**as a cheat sheet.** In the list below, commonly used lifecycle methods are marked as **bold**. The rest of them exist for relatively rare use cases.

#### **Mounting**

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

* [**constructor()**](https://reactjs.org/docs/react-component.html#constructor)
* [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [**render()**](https://reactjs.org/docs/react-component.html#render)
* [**componentDidMount()**](https://reactjs.org/docs/react-component.html#componentdidmount)

**Note:**

These methods are considered legacy and you should [avoid them](https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html) in new code:

* [UNSAFE\_componentWillMount()](https://reactjs.org/docs/react-component.html#unsafe_componentwillmount)

#### **Updating**

An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:

* [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [shouldComponentUpdate()](https://reactjs.org/docs/react-component.html#shouldcomponentupdate)
* [**render()**](https://reactjs.org/docs/react-component.html#render)
* [getSnapshotBeforeUpdate()](https://reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
* [**componentDidUpdate()**](https://reactjs.org/docs/react-component.html#componentdidupdate)

**Note:**

These methods are considered legacy and you should [avoid them](https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html) in new code:

* [UNSAFE\_componentWillUpdate()](https://reactjs.org/docs/react-component.html#unsafe_componentwillupdate)
* [UNSAFE\_componentWillReceiveProps()](https://reactjs.org/docs/react-component.html#unsafe_componentwillreceiveprops)

#### **Unmounting**

This method is called when a component is being removed from the DOM:

* [**componentWillUnmount()**](https://reactjs.org/docs/react-component.html#componentwillunmount)

#### **Error Handling**

These methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

* [static getDerivedStateFromError()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromerror)
* [componentDidCatch()](https://reactjs.org/docs/react-component.html#componentdidcatch)

### Other APIs

Each component also provides some other APIs:

* [setState()](https://reactjs.org/docs/react-component.html#setstate)
* [forceUpdate()](https://reactjs.org/docs/react-component.html#forceupdate)

### Class Properties

* [defaultProps](https://reactjs.org/docs/react-component.html#defaultprops)
* [displayName](https://reactjs.org/docs/react-component.html#displayname)

### Instance Properties

* [props](https://reactjs.org/docs/react-component.html#props)
* [state](https://reactjs.org/docs/react-component.html#state)

shouldComponentUpdate(nextProps, nextState){

console.log("update- should component update")

console.log("prevstate",this.state.count)

console.log("nextstate",nextState)

if shouldComponentUpdate(nextProps, nextState) return false, no state update will happen and no more rendering. It asks only if the component should be updated. The state will be updated irrespectively.

When the state/prop of a parent component changes, all its child components are ‘updated’.

componentDidUpdate(prevProps, prevState){

console.log("update - component did update")

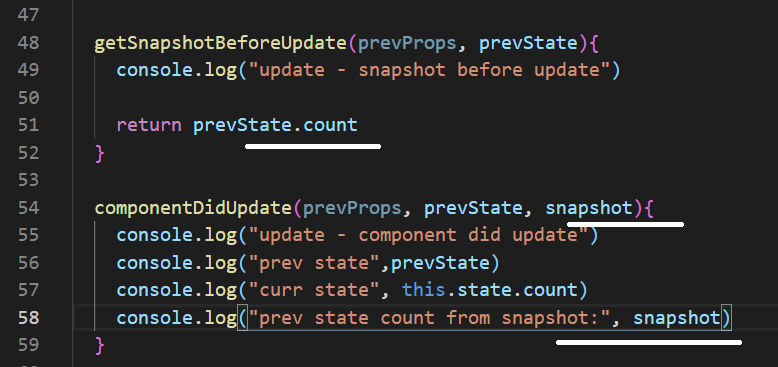
console.log("prev state",prevState)

console.log("curr state", this.state.count)

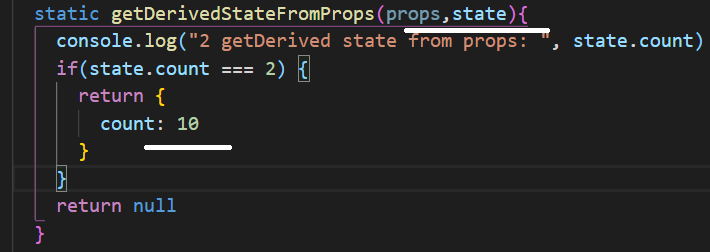
}

componentDidUpdate – gives previous props/states

getSnapshotBeforeUpdate(prevProps, prevState) can return any props/state value, which can be used as 3rd argument in componentDidUpdate(,,snapshot).

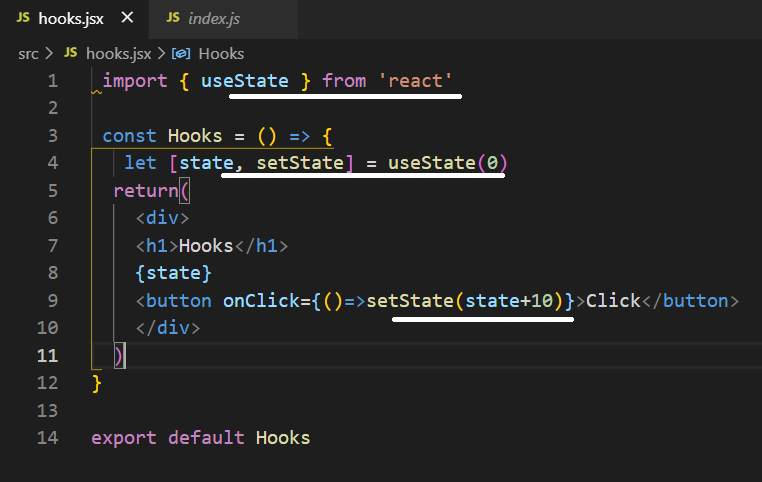


getDerivedStateStateFromProps(props, state) – will provide with the current prop/state before update. This will enable us to modify the state as did below.

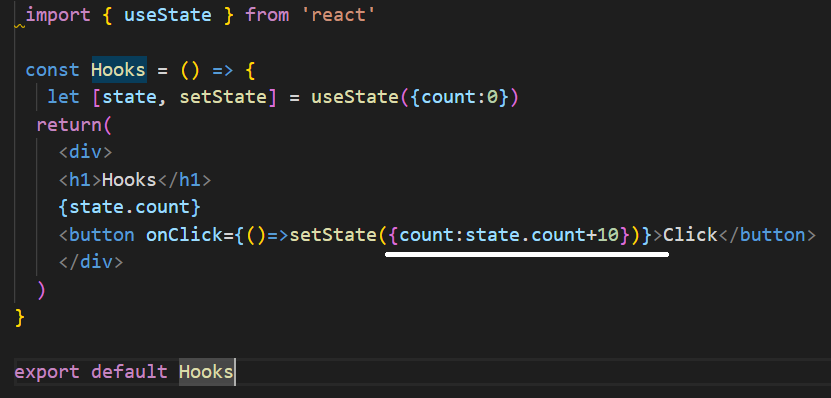


React hooks: <https://reactjs.org/docs/hooks-intro.html>

useState hook:



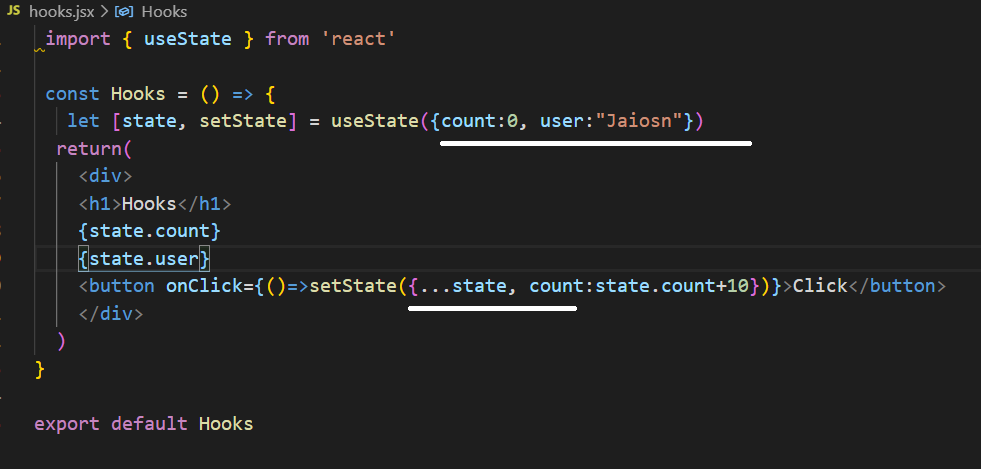
Another form of update:



https://reactjs.org/docs/hooks-reference.html

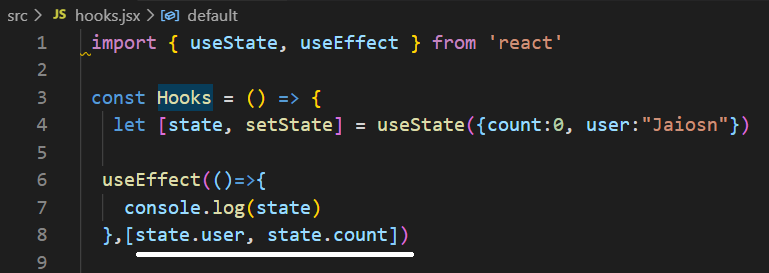
* [Basic Hooks](https://reactjs.org/docs/hooks-reference.html#basic-hooks)
  + [useState](https://reactjs.org/docs/hooks-reference.html#usestate)
  + [useEffect](https://reactjs.org/docs/hooks-reference.html#useeffect)
  + [useContext](https://reactjs.org/docs/hooks-reference.html#usecontext)
* [Additional Hooks](https://reactjs.org/docs/hooks-reference.html#additional-hooks)
  + [useReducer](https://reactjs.org/docs/hooks-reference.html#usereducer)
  + [useCallback](https://reactjs.org/docs/hooks-reference.html#usecallback)
  + [useMemo](https://reactjs.org/docs/hooks-reference.html#usememo)
  + [useRef](https://reactjs.org/docs/hooks-reference.html#useref)
  + [useImperativeHandle](https://reactjs.org/docs/hooks-reference.html#useimperativehandle)
  + [useLayoutEffect](https://reactjs.org/docs/hooks-reference.html#uselayouteffect)
  + [useDebugValue](https://reactjs.org/docs/hooks-reference.html#usedebugvalue)

useState will not merge data like setState. To override this, use the spread operator as below:



useReducer can be effieciently used when multiple values are entered inside the state object

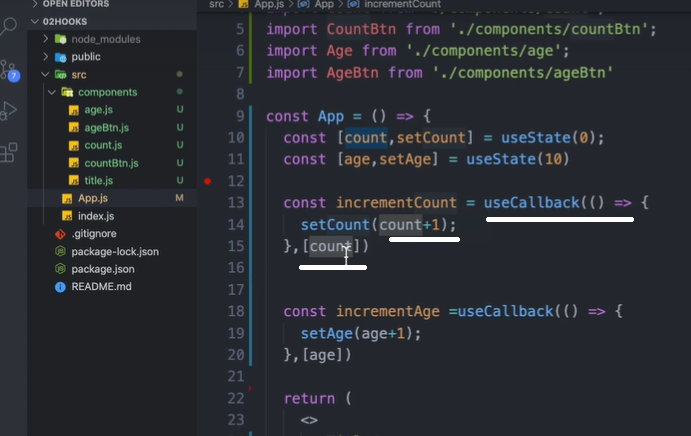
useEffect executes a call back function. If the 2nd parameter is omitted, the function will be executed during create, update and delete life-cycle. If an empty bracket is given, its run only during the mount and unmount. If any state object is given inside the bracket, its updated whenever that state changes.



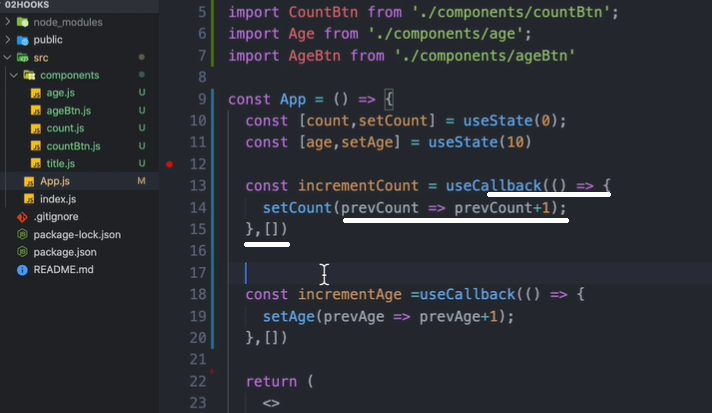
In memory, the functions if a component are different in each re-render.

useCallback is going to cache the function in memory.

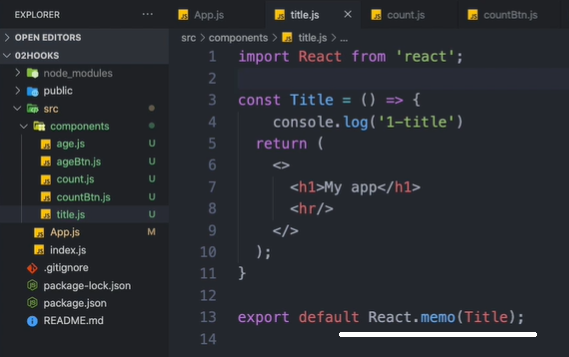
Dependency:



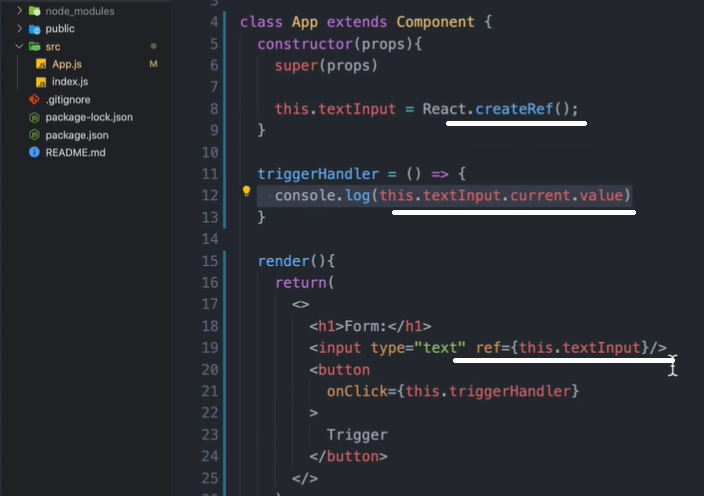
No dependency:



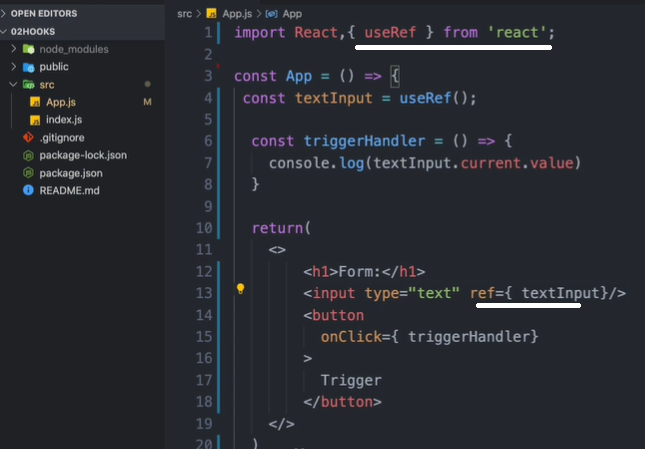
React.memo – if the component doesn’t receive any chaned props, the child component will not be re-renderd.



createRef:

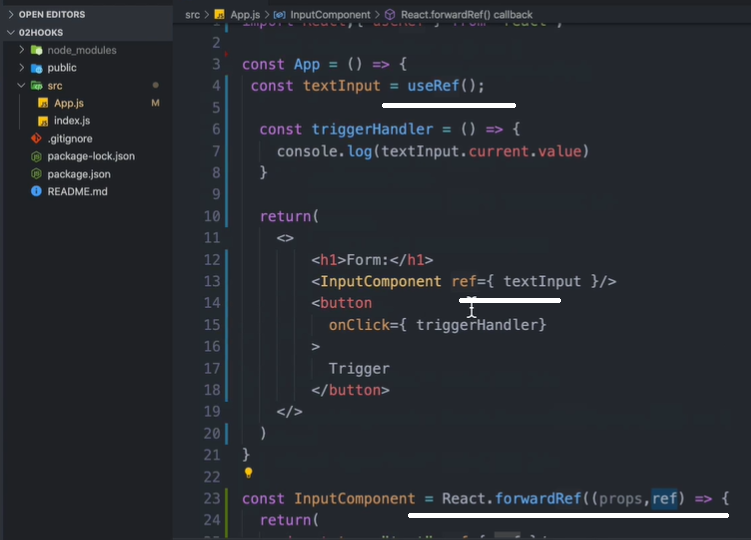


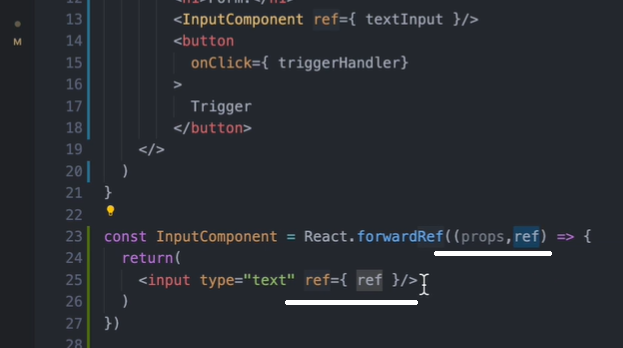
useRef:



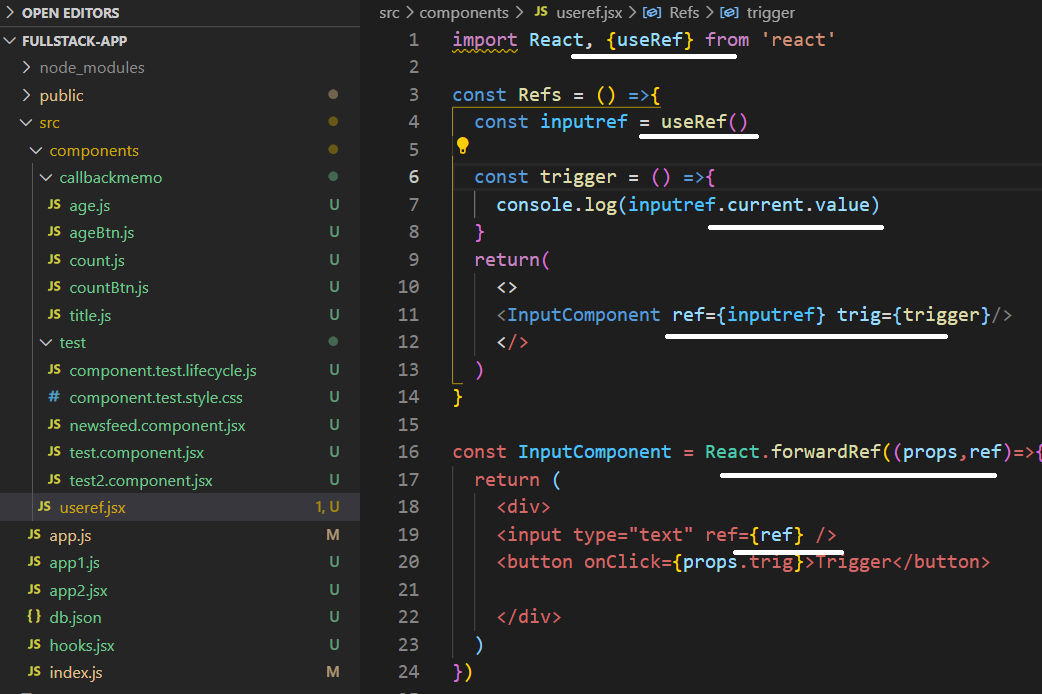
References are not properties, cant send down to child components.

Using React.forwardRef – to use the ‘ref’ as property:





Program:



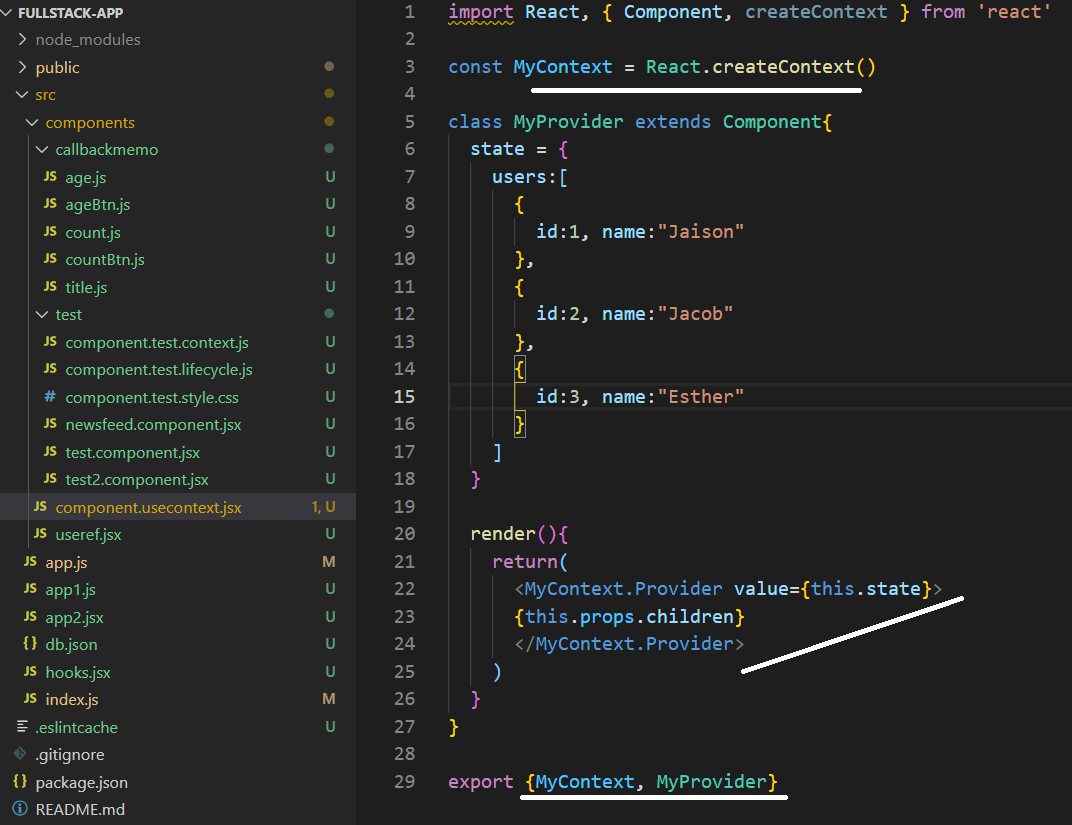
Context allow as to store and retrieve data and functions from a centralised place.

To use context, optionally in a class, declare the context, provider. Create state inside the class and pass it to the provider as value.

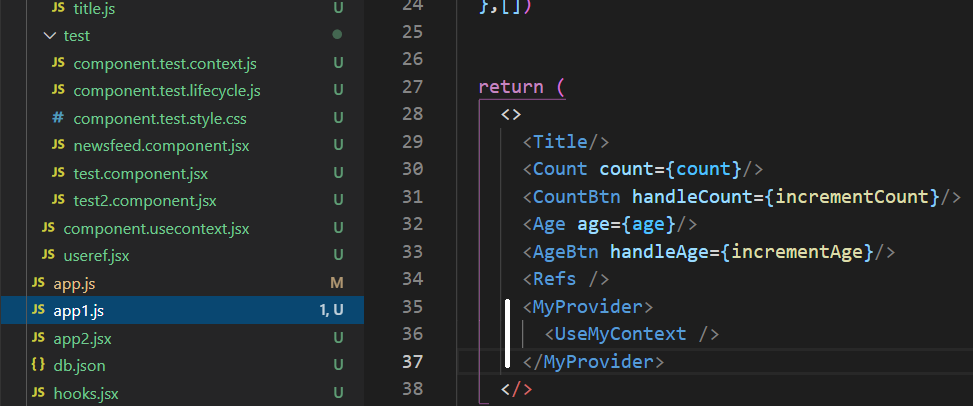
In the main app, import the provider and wrap it around the components which need the context.

In the sub component, import the context and use ‘useContext’ to access the state or context.

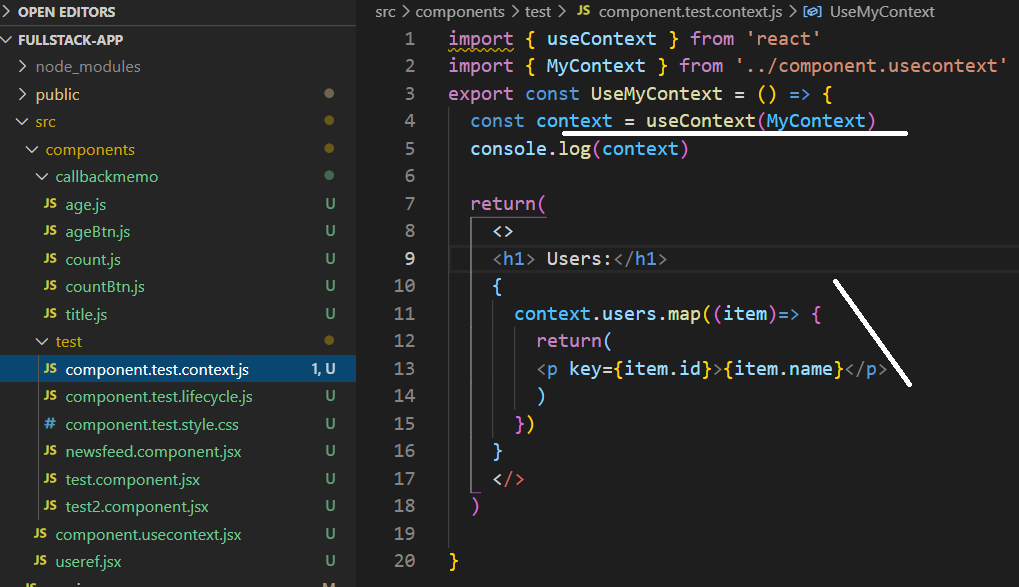
Class context provider:



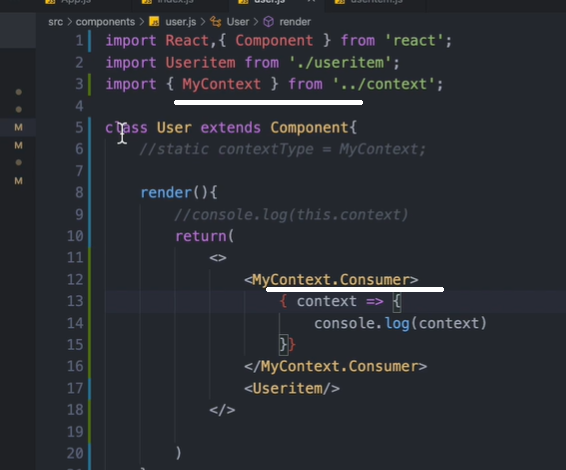
Configuring the provider in the main app:



Using the context in the component:



Context consumer – better to use function when using the consumer.



To change the development server port in package.json

    "start": " set PORT=8000 && react-scripts start",

\*star