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")

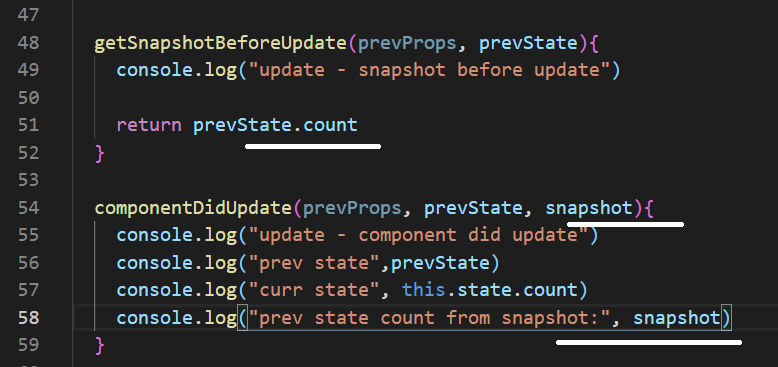
console.log("prev state",prevState)

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

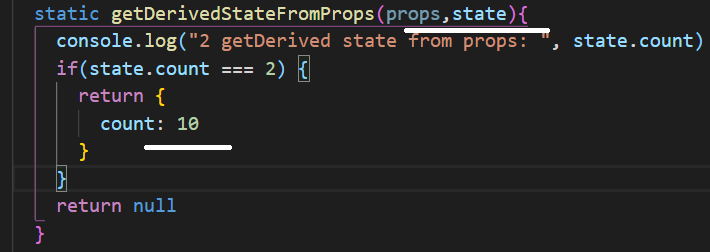
}

componentDidUpdate – gives previous props/states

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

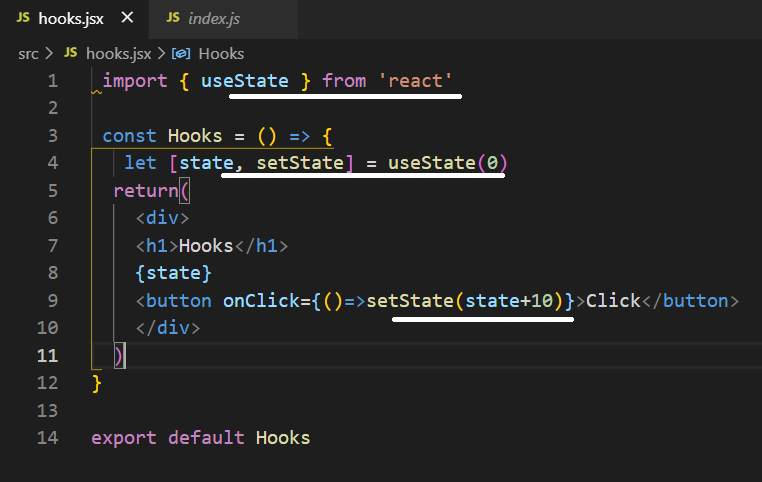


getDerivedStateStateFromProps(props, state) – will provide with the current prop/state before update. This will enable us to modify the state as did below.

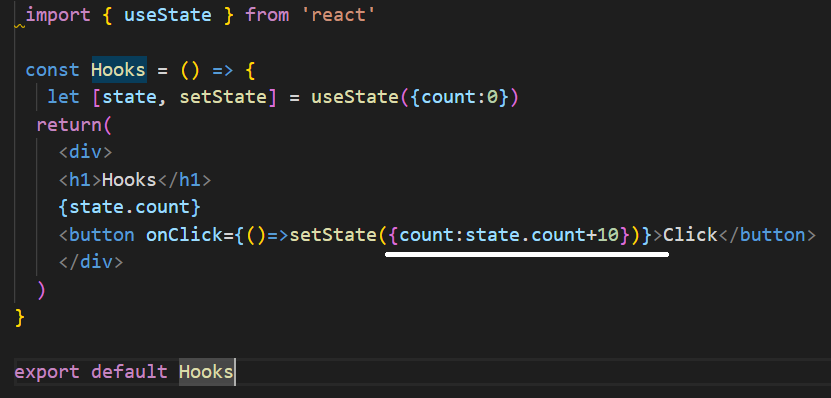


React hooks: <https://reactjs.org/docs/hooks-intro.html>

useState hook:



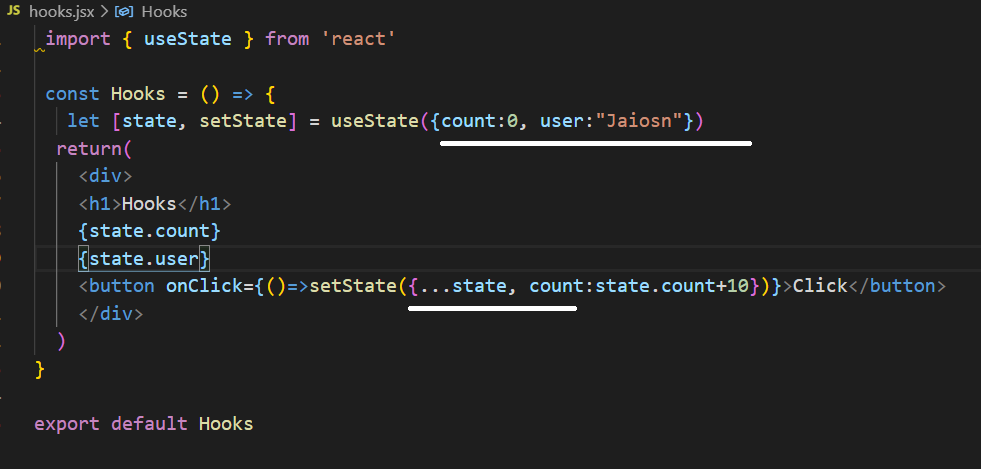
Another form of update:



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

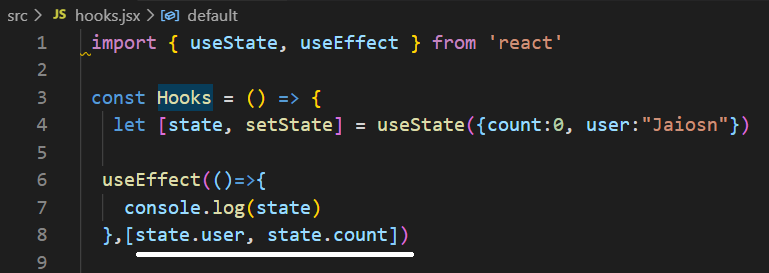
* [Basic Hooks](https://reactjs.org/docs/hooks-reference.html#basic-hooks)
  + [useState](https://reactjs.org/docs/hooks-reference.html#usestate)
  + [useEffect](https://reactjs.org/docs/hooks-reference.html#useeffect)
  + [useContext](https://reactjs.org/docs/hooks-reference.html#usecontext)
* [Additional 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

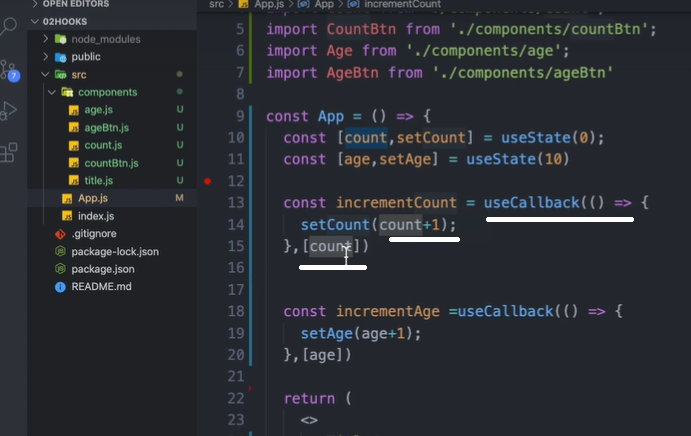
useEffect executes a call back function. If the 2nd parameter is omitted, the function will be executed during create, update and delete life-cycle. If an empty bracket is given, its run only during the mount and unmount. If any state object is given inside the bracket, its updated whenever that state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



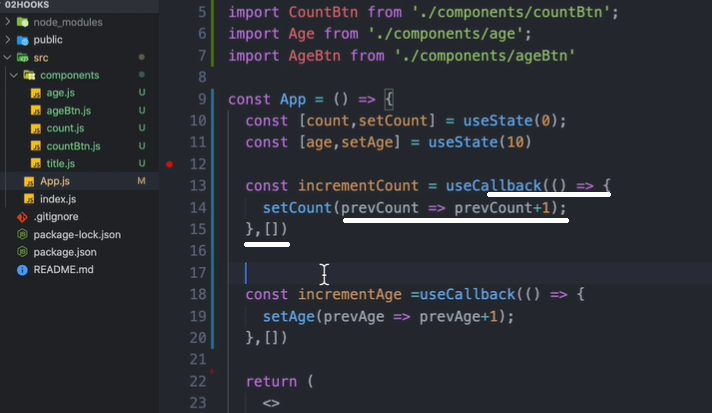
In memory, the functions if a component are different in each re-render.

useCallback is going to cache the function in memory.

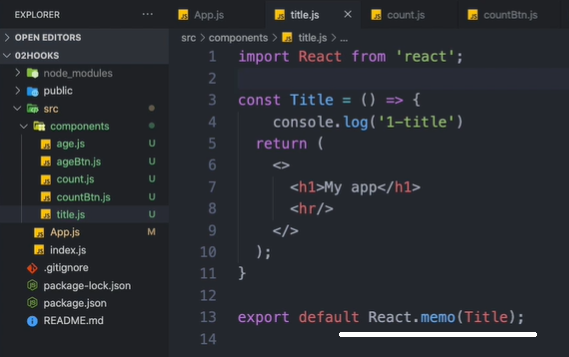
Dependency:



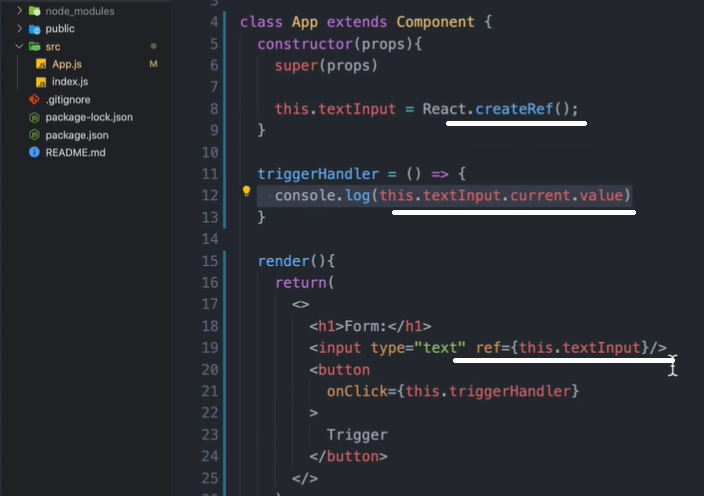
No dependency:



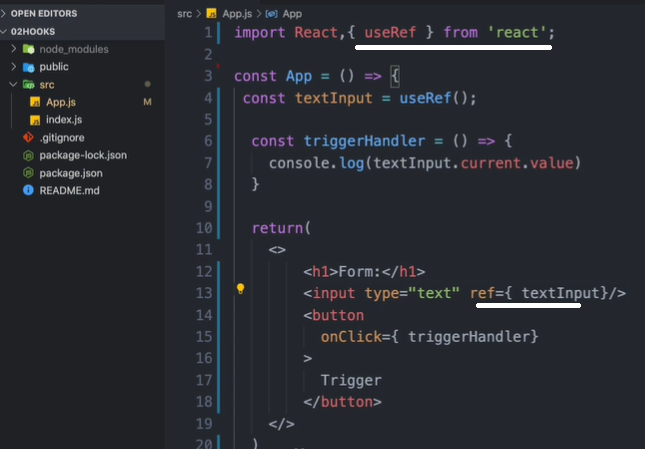
React.memo – if the component doesn’t receive any chaned props, the child component will not be re-renderd.



createRef:

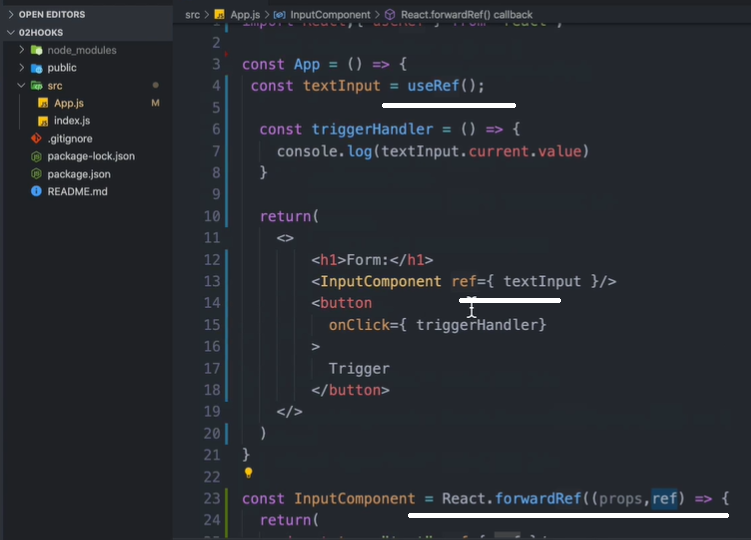


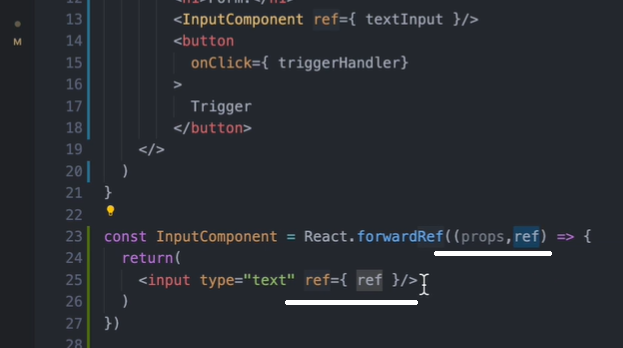
useRef:



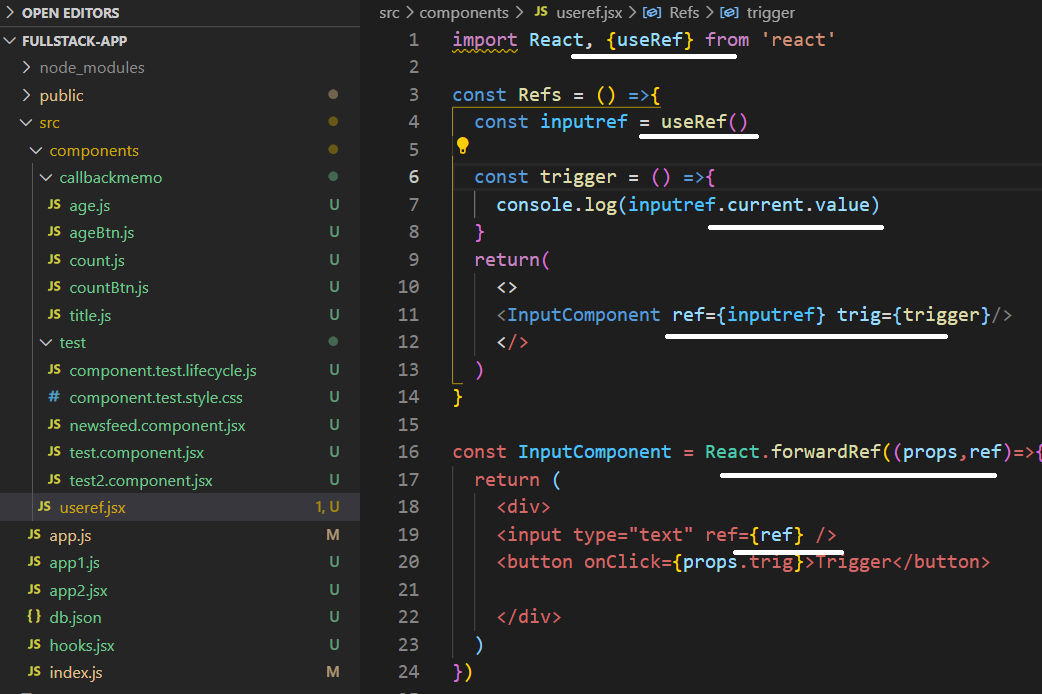
References are not properties, cant send down to child components.

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





Program:



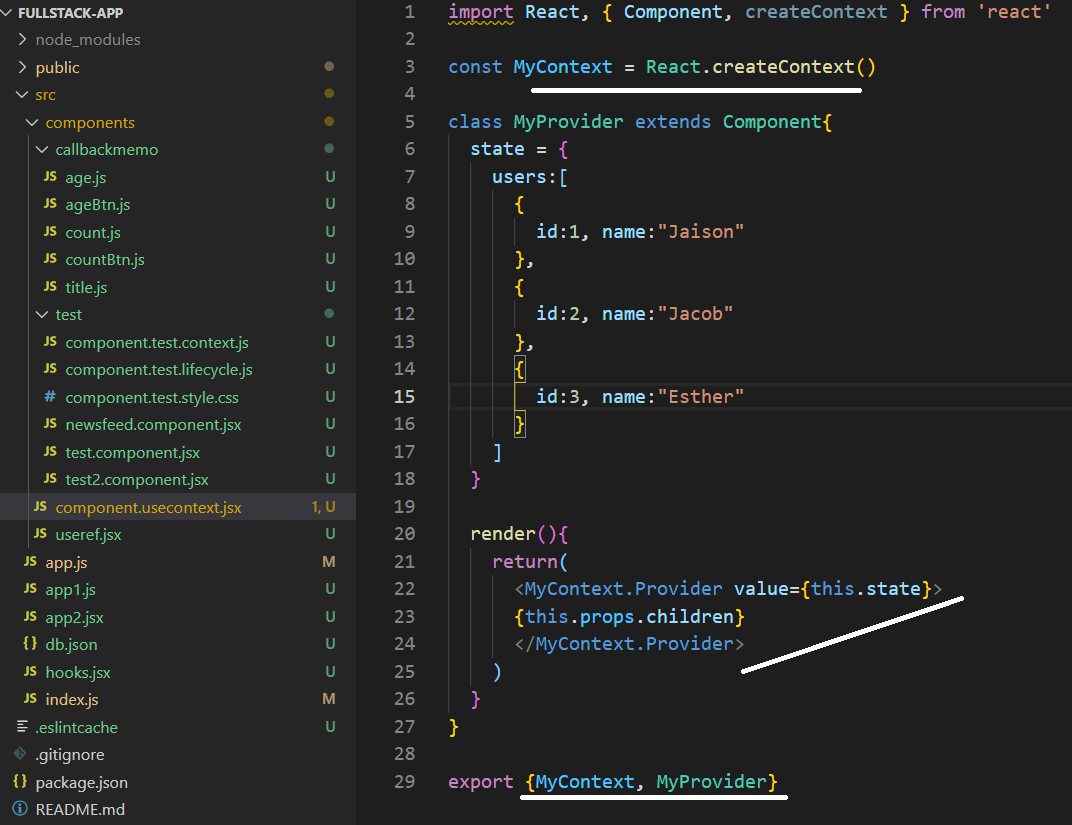
Context allow as to store and retrieve data and functions from a centralised place.

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

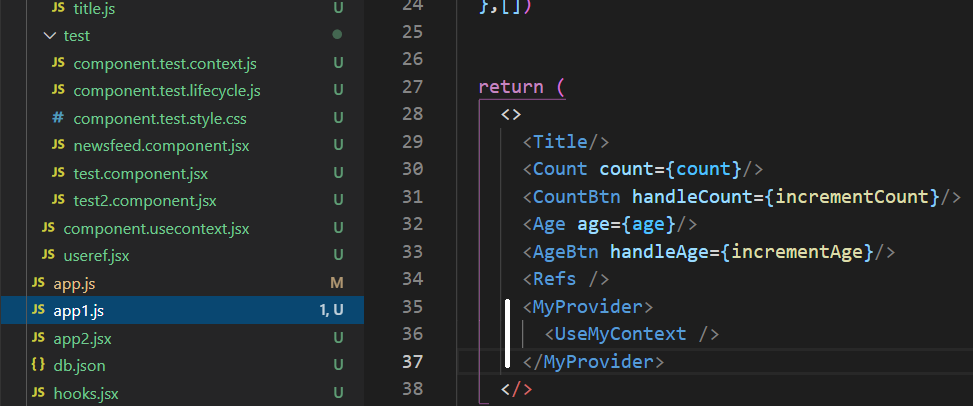
In the main app, import the provider and wrap it around the components which need the context.

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

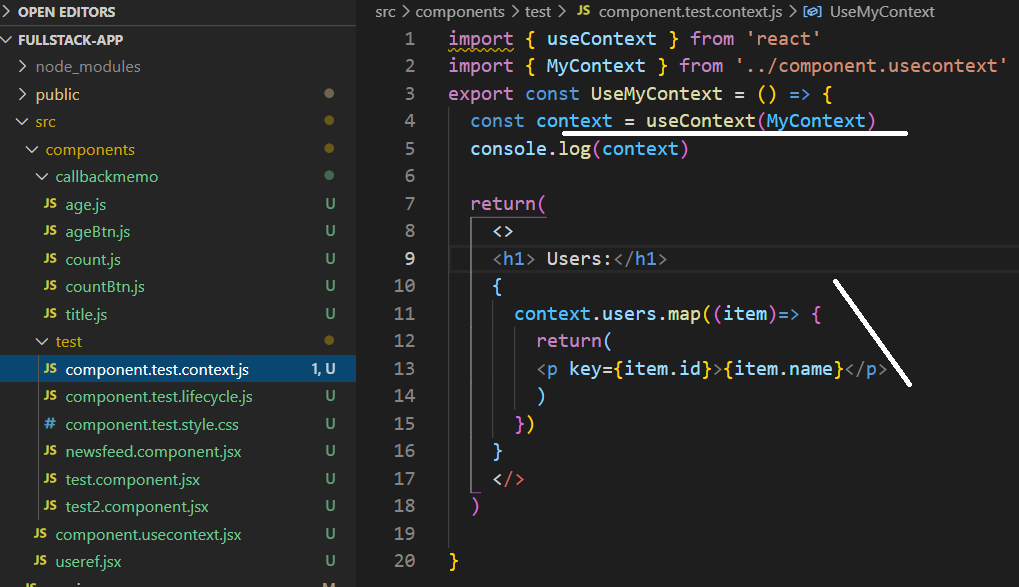
Class context provider:



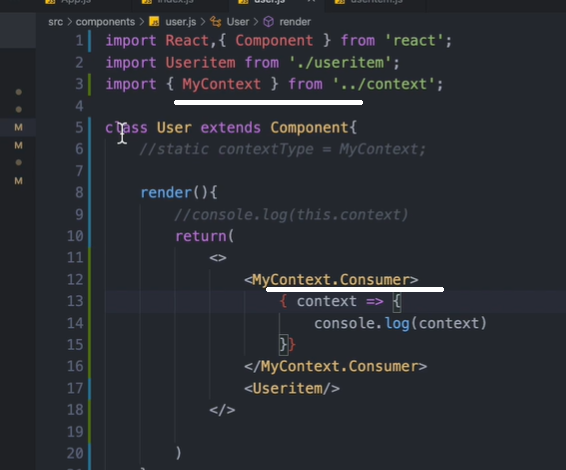
Configuring the provider in the main app:



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",

To access the context from a class based component, use the ‘static’ property:

Static contextType = MyContext – staticType is a reserved word. It gives the ‘context’ word.

To install fonts, go to fonts.google.com search for the font, select the font, go to select this style, select the link, paste it in index.html inside the public folder.

Npm install react-bootstrap bootstrap

Directly import ‘bootstrap/dist/css/bootstrap.min.css’ into the app.js

Npm install react-toastify

import { ToastContainer, toast } from 'react-toastify'

in the main app:

import 'bootstrap/dist/css/bootstrap.min.css'

import "react-toastify/dist/ReactToastify.css";

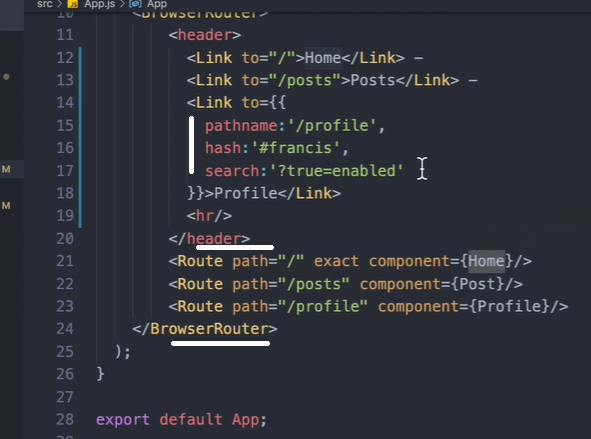
**React Router:**

Npm install react-router-dom

The BrowserRouter class knows the url history.

Import { BrowserRouter, Route, Link } from ‘react-router-dom

Link is for putting the ‘link’ into the url. When the route detets the link, it loads the appropriate component.



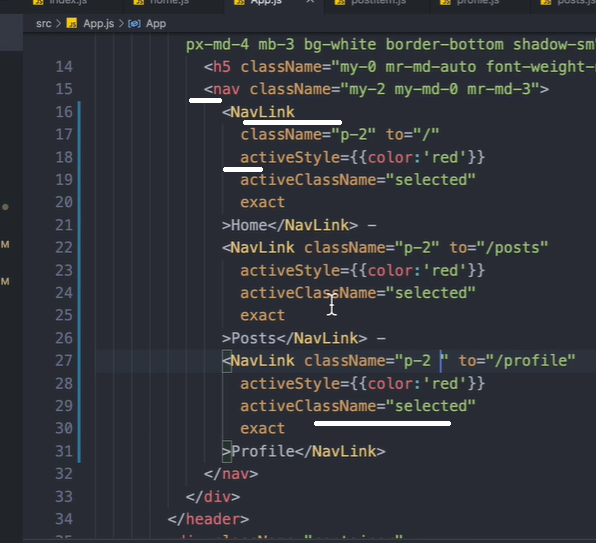
By default, the Route will pass props to the components. It comes from React Router. Location, history, match.

For child links, use <Link to={{ pathname:`${props.match.url}/posts`}}

Dynamic params are accessed via props.match.params.id posts/:id

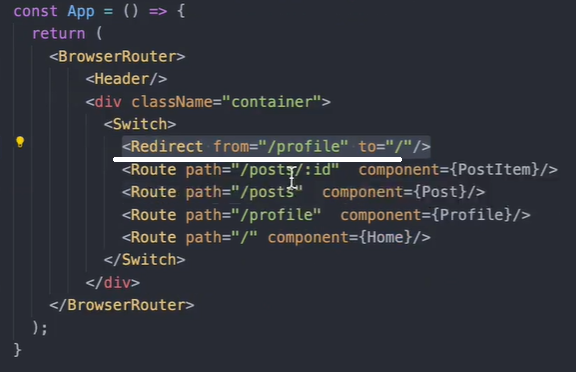
HashRouter and MemoryRouter - - hashrouter shows # in the url. Memoryrouter shows no links in the url. These are given in the same place as the BrowserRouter.

NavLink:

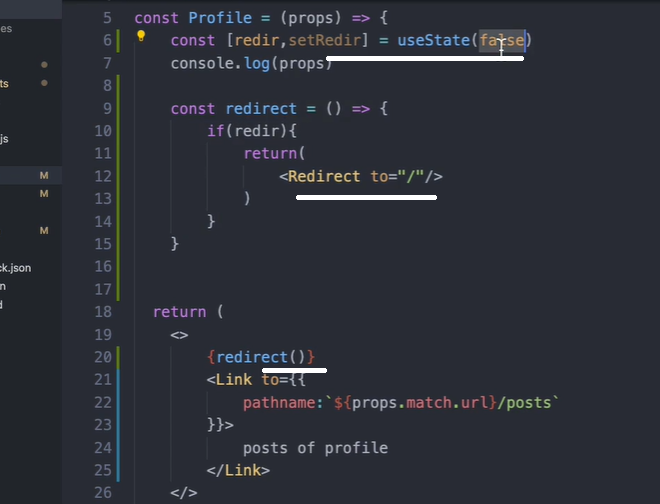


Switch will attempt find specific path for the url. Once it finds, then stops.

Redirect redirects the url from the url to the url:



Redirect using function: change the state ‘redir’ using a button click using ‘setRedir’



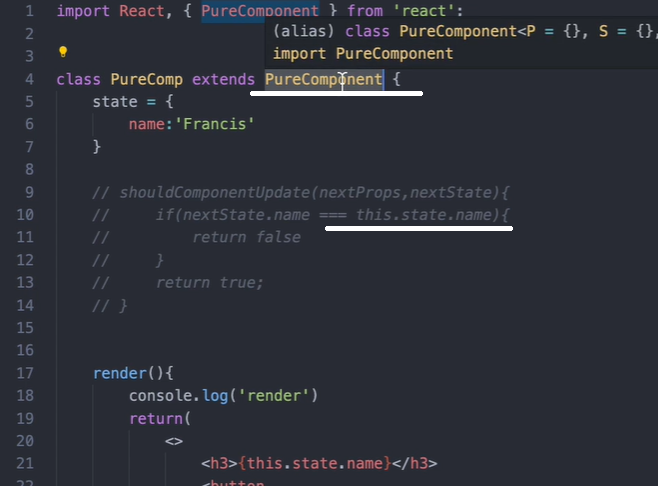
Redirect using props.history.push



Using default Route for 404 cases where the url is not valid. Route has a render method

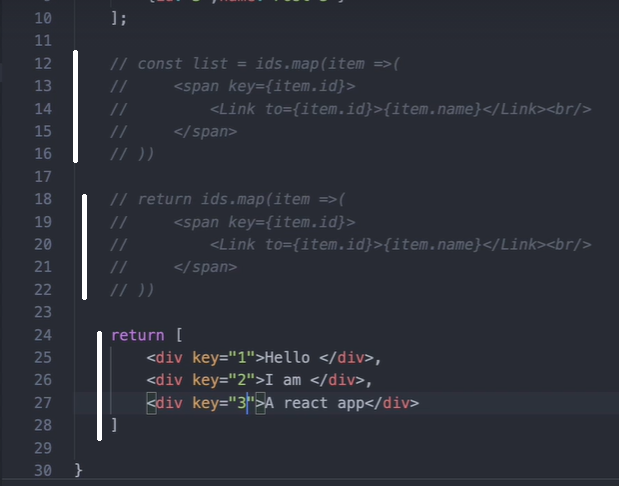


Pure components checks to see if there is any changes to the props and state before rendering the component. If no changes, it ignores the rendering of the component. Like shoudComponentUpdate returns false if not needed.



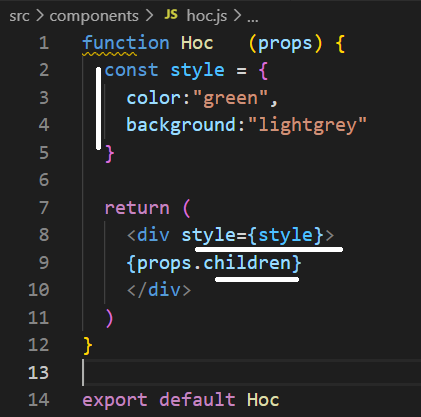
For functional components, if there is no change to the state, automatically will not re-render.

In 3 ways, we can list an array into the component:



An exporting component object can be an arrow function with a constant declaration or a normal function.

A HOC component declaration:



An hoc can be a component for user authentication.

Transition in React is Transition group, motion and spring.

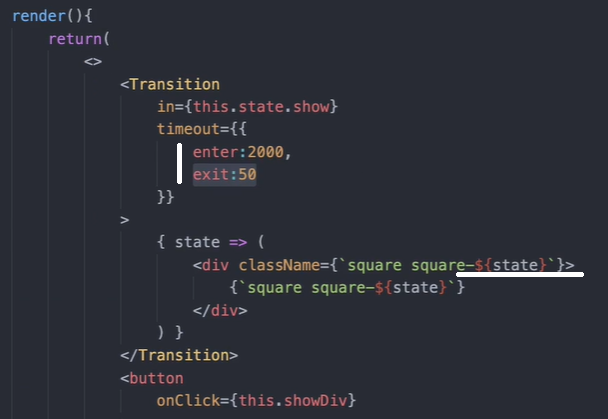
Npm install react-transition-group

Transition has 4 states. Entering, entered, exiting, exited.

Import Transition from ‘react-transition-group’

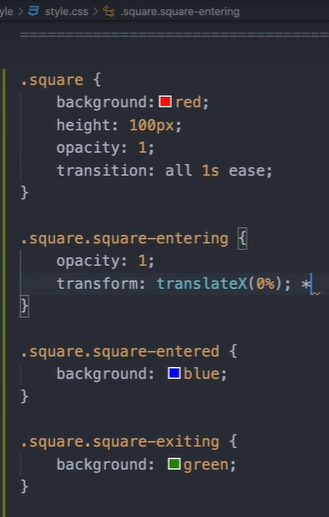


Changing class with state interpolation on css



Can make enter={false} like this inside transition.

Css

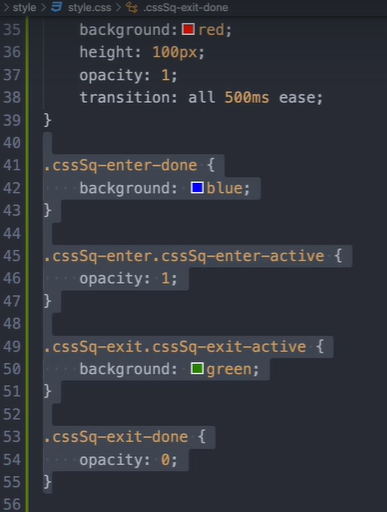


https//:Reactcommunity.org

import CSSTransition from ‘react-transition-group’



Css



react-awesome-reveal // animatecss // <https://github.com/FormidableLabs/react-animations>

**REDUX** – A predictable state container for JS apps

The provider from ‘redux-react’ should wrap the entire app from the top level. It’s the bridge connecting React to redux

A promise example.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Promise</h2>

<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>

<h1 id="demo"></h1>

<script>

let myPromise = new Promise((rsolv, rjct) =>{

let jj = "Jaison"

if (jj === "Jaison") rsolv("Success, resolved")

else rjct("Error, failed")

})

myPromise.then((value) => {

document.getElementById("demo").innerHTML = value},

(value) => {

document.getElementById("demo").innerHTML = value},)

</script>

</body>

</html>

---

The applyMiddleware function from ‘redux’ is to use 3rd party middleware functions for redux

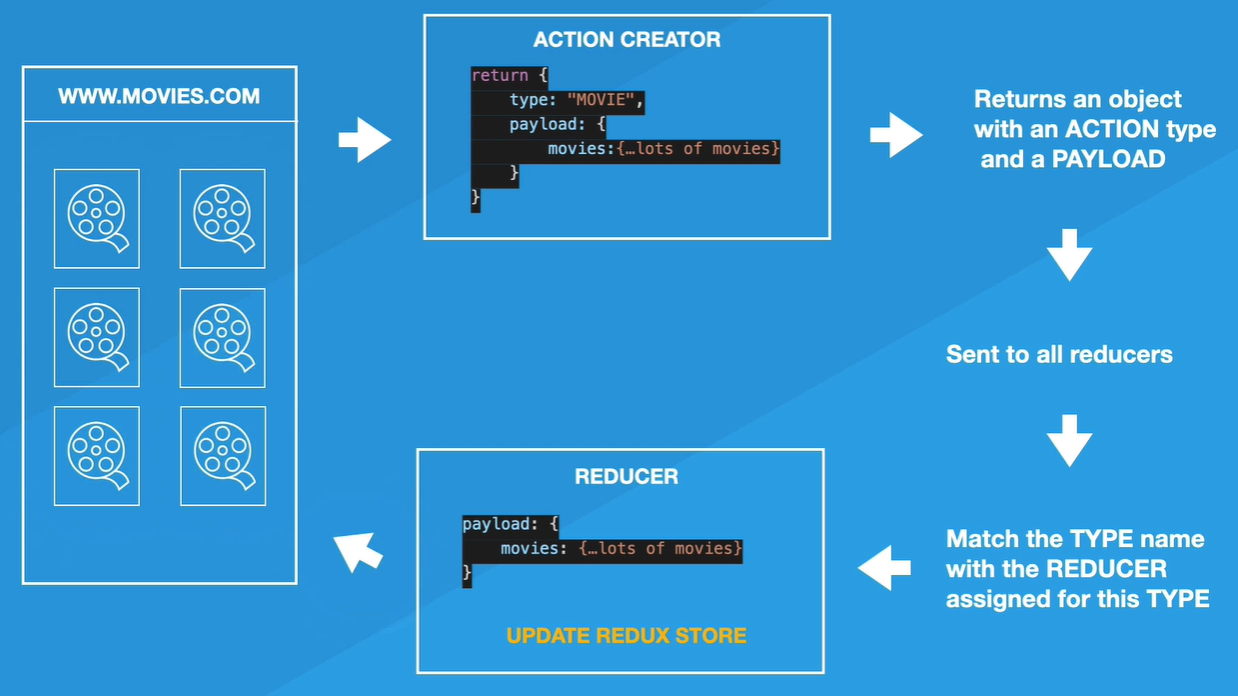
Const createStoreWithApplyMiddleware = applyMiddleware()(createStore)

<Provider store={ createStoreWithApplyMiddleware())>

Redux works with actions and reducers.

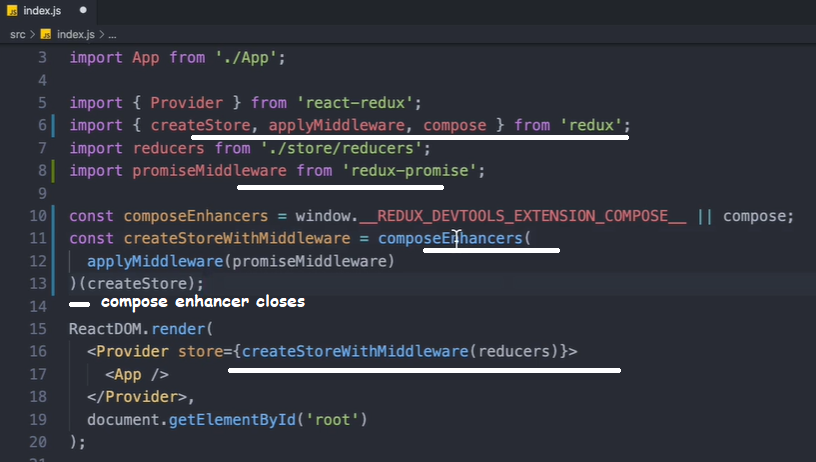
React-redux process flow:

Application dispatch an action to the action creator. Action creator executes the action and return an object with type and payload. Reducer recieves the type and update store with the payload.



Redux-promise is a middleware. Its like redux-thunk

Import promiseMiddleware from ‘redux-promise’ – in the main component where ‘redux’ is imported



‘compose’ helps the browser extension ‘devtools’ to work efficiently.

An ‘action’ is a function that returns an object with a type and pyload.

The flow: 1) an action function is created and exported that returns an object with type and payload.

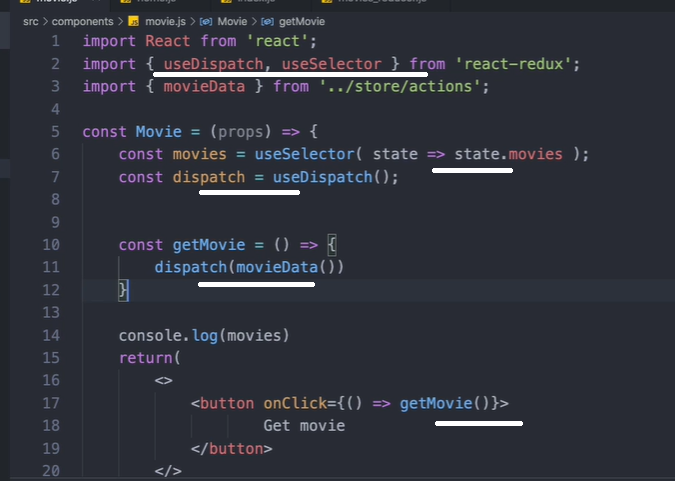
2) the reducer is created and exported. 3) the combineReducers imports the reducer function.

4) the combineReducers is imported in the main app. 5) the Home component imports the action function. It imports ‘connect’ from ‘react-redux’. The Home component dispatches the action, which the reducer handles and updates the store. Upon changes to the data, React re-renders the component.

combineReducers is a function itself.

Connect – gives access to state for the component through mapStateToProps.

When using redux hooks in funcitonal components, import { useDispatch, useSelector } from ‘react-redux’. The useSelector behave like mapStateToProps like in class.



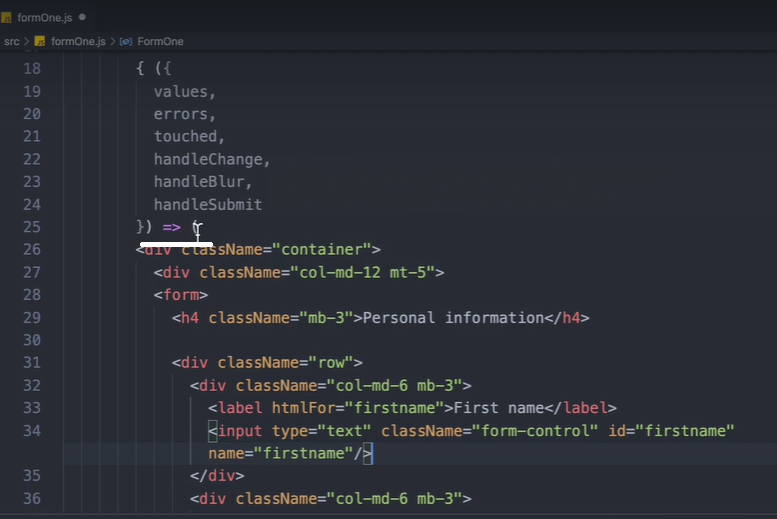
Formik is a library for managing Forms. Formik.org

Schema validation with Yup.

Npm install formik yup

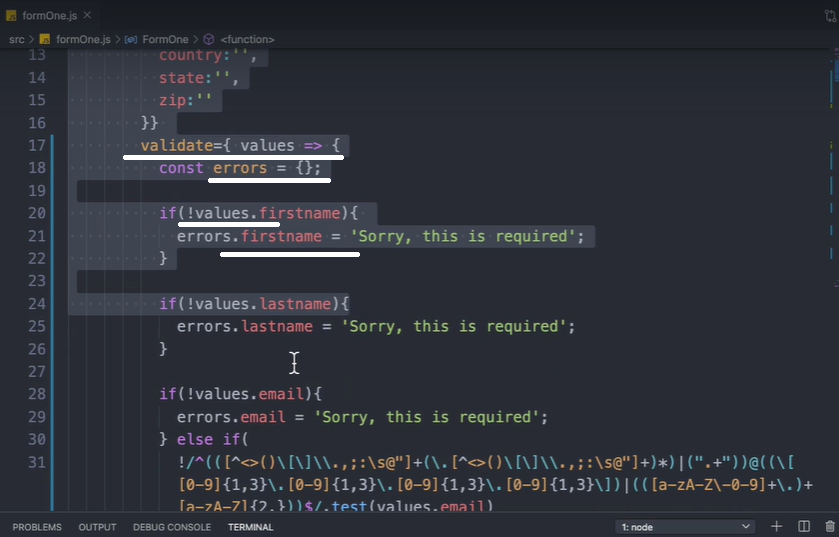
* 1. Create a formik element as wrapper. 2) inside the wrapper, create an arrow function. In that function, take Formik methods as objects. 3) inside the function return, provide the form elements. 4) Inside Formik element, pass fom element names as properties for initalValues. 5) assign/connect form element values with Formik property and form element methods with Formik methods.





Get email format check from regex github.

Form validation:



Validation Code:

validate={ values => {

const errors = {};

if(!values.firstname){

errors.firstname = 'Sorry, this is required';

}

if(!values.lastname){

errors.lastname = 'Sorry, this is required';

}

if(!values.email){

errors.email = 'Sorry, this is required';

} else if(

!/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/.test(values.email)

){

errors.email = 'Invalid email';

}

return errors;

}}

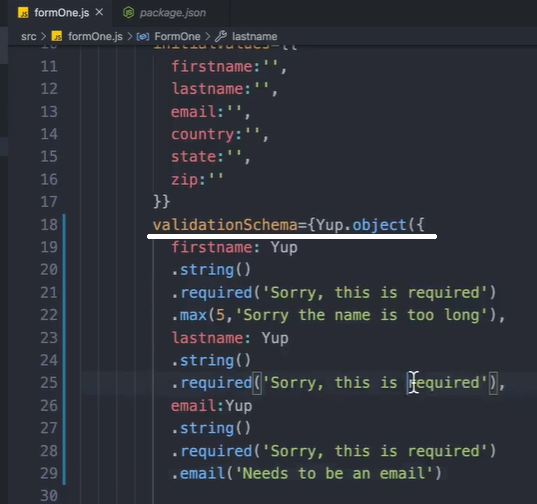
--------------------------------------

Error input on the form on condition.



Import \* as Yup from ‘yup’

Its only for validation of the elements.



Code (Yup):

import React from 'react';

import { Formik } from 'formik'

import \* as Yup from 'yup'

const Play2 = () => {

return (

<Formik

initialValues={{

firstname:'',

lastname:'',

email:'',

country:'',

state:'',

zip:''

}}

**validationSchema={Yup.object({**

**firstname: Yup**

**.string()**

**.required('Sorry, this is required')**

**.max(5,'Sorry the name is too long'),**

**lastname: Yup**

**.string()**

**.required('Sorry, this is required'),**

**email:Yup**

**.string()**

**.required('Sorry, this is required')**

**.email('Needs to be an email')**

**})}**

onSubmit={( values )=>{

console.log(values)

}}

>

{ ({

values,

errors,

touched,

handleChange,

handleBlur,

handleSubmit

}) => (

<div className="container">

<div className="col-md-12 mt-5">

<form onSubmit={handleSubmit}>

<h4 className="mb-3">Personal information - Play2</h4>

<div className="row">

<div className="col-md-6 mb-3">

<label htmlFor="firstname">First name</label>

<input

type="text"

className="form-control"

id="firstname"

name="firstname"

value={values.firstname}

onChange={handleChange}

onBlur={handleBlur}

/>

{ errors.firstname && touched.firstname ?

<span>{ errors.firstname }</span>

:null}

</div>

<div className="col-md-6 mb-3">

<label htmlFor="lastname">Last name</label>

<input

type="text"

className="form-control"

id="lastname"

name="lastname"

value={values.lastname}

onChange={handleChange}

onBlur={handleBlur}

/>

{ errors.lastname && touched.lastname ?

<span>{ errors.lastname }</span>

:null}

</div>

</div>

<div className="mb-3">

<label htmlFor="email">Email</label>

<input

type="email"

className="form-control"

id="email"

name="email"

placeholder="you@example.com"

value={values.email}

onChange={handleChange}

onBlur={handleBlur}

/>

{ errors.email && touched.email ?

<span>{ errors.email }</span>

:null}

</div>

<div className="row">

<div className="col-md-5 mb-3">

<label htmlFor="country">Country</label>

<select

className="custom-select d-block w-100"

id="country"

name="country"

value={values.country}

onChange={handleChange}

>

<option value="">Choose...</option>

<option value="US">United States</option>

<option value="CA">Canada</option>

<option value="NL">Netherlands</option>

</select>

</div>

<div className="col-md-4 mb-3">

<label htmlFor="state">State</label>

<select

className="custom-select d-block w-100"

id="state"

name="state"

value={values.state}

onChange={handleChange}

>

<option value="">Choose...</option>

<option value="california">California</option>

<option value="toronto">Toronto</option>

<option value="utrech">Utrech</option>

</select>

</div>

<div className="col-md-3 mb-3">

<label htmlFor="zip">Zip</label>

<input

type="text"

className="form-control"

id="zip"

name="zip"

value={values.zip}

onChange={handleChange}

/>

</div>

</div>

<hr className="mb-4"/>

<button className="btn btn-primary btn-lg btn-block" type="submit">

Submit

</button>

</form>

</div>

</div>

)}

</Formik>

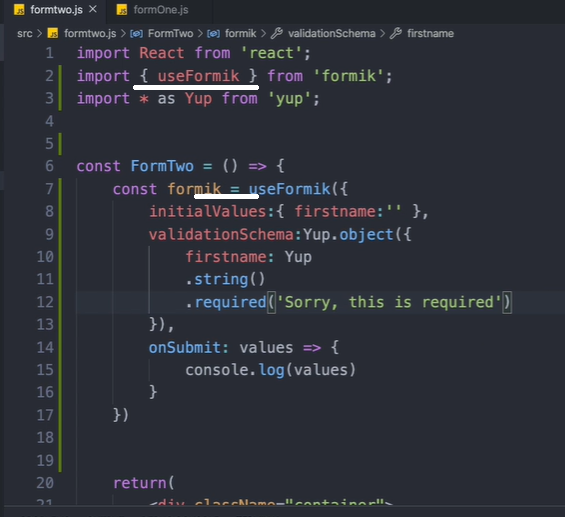
);

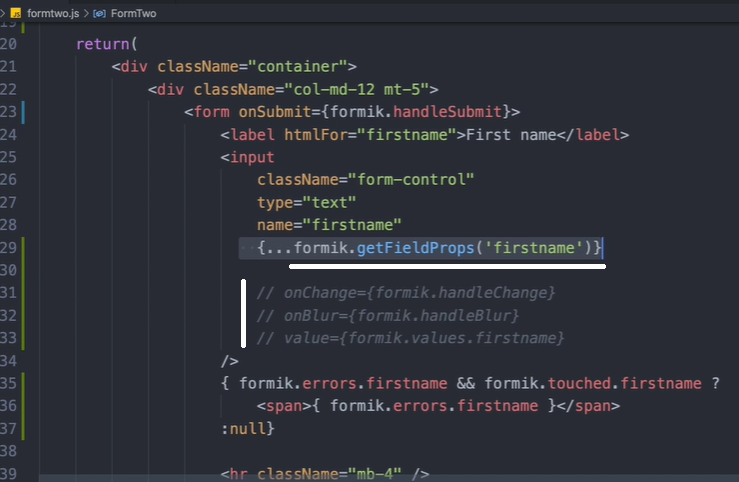
}

export default Play2;

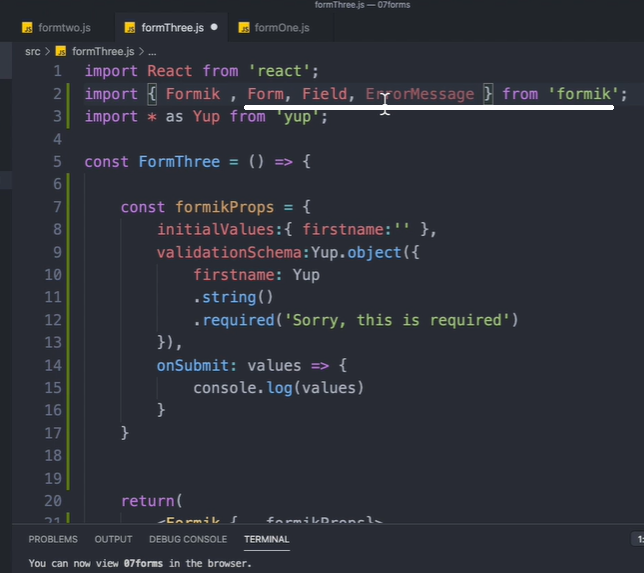
Formik can be used inside class and functional components.

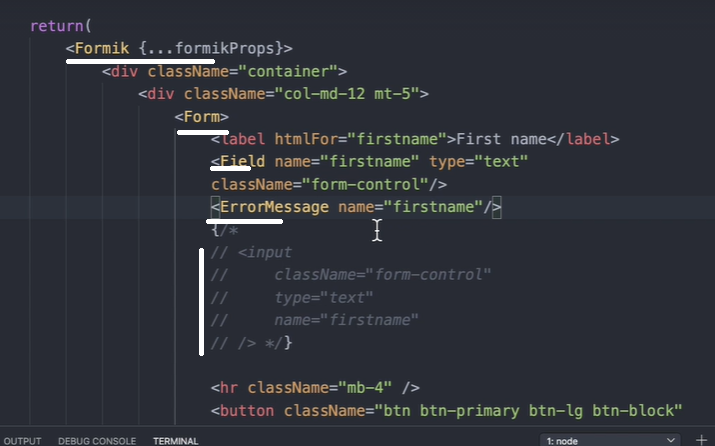
Formik hook. Import { useFormk } from ‘formik’





Formik components: import { Formik, Form } from ‘formik’

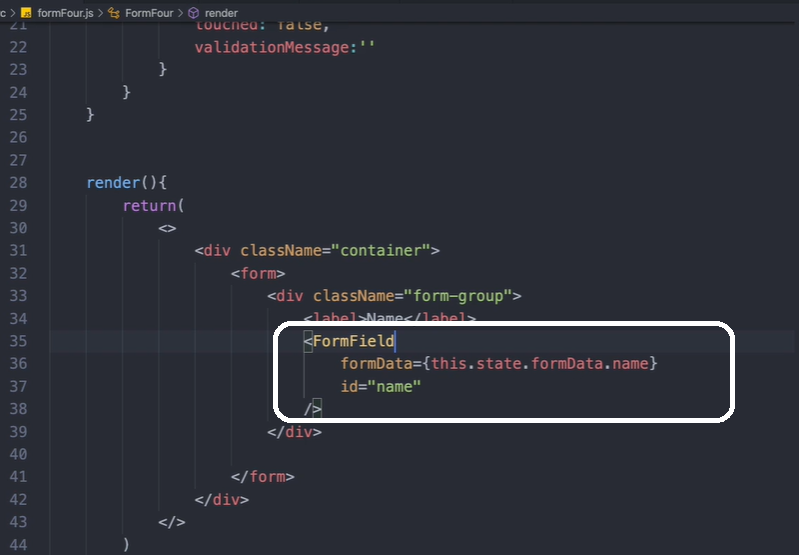




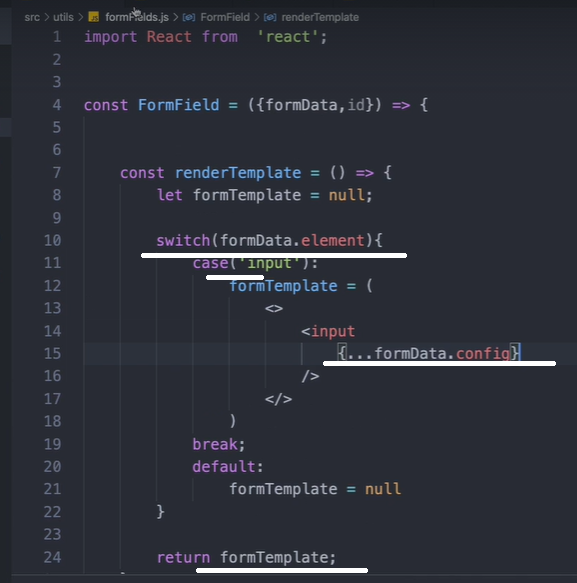
Formik.org

Custom state and form: this is a class component



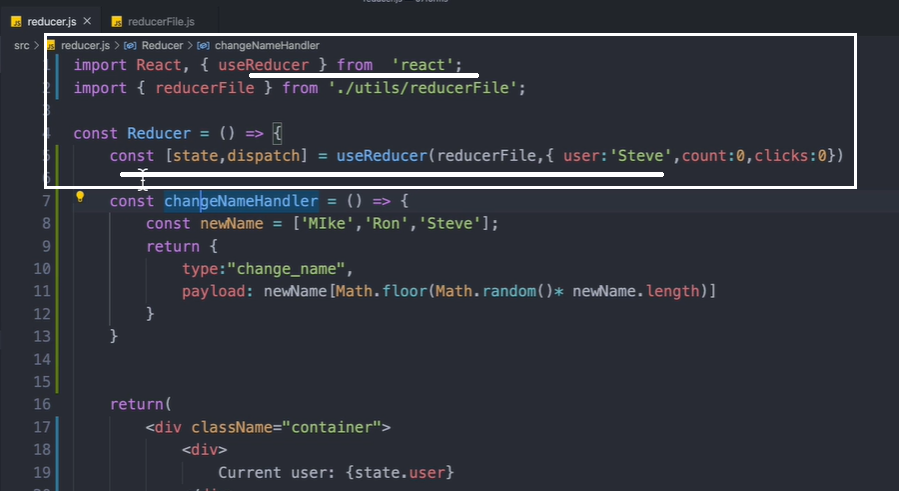


This is a utility component for generating the Form.



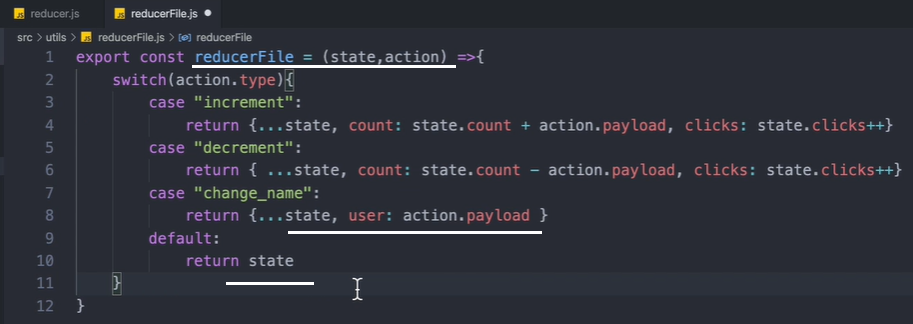
Call the ‘formTemplae’ inside the return as { formTemplate }

useReducer:





Reducer logic seperated as a function:



\*star