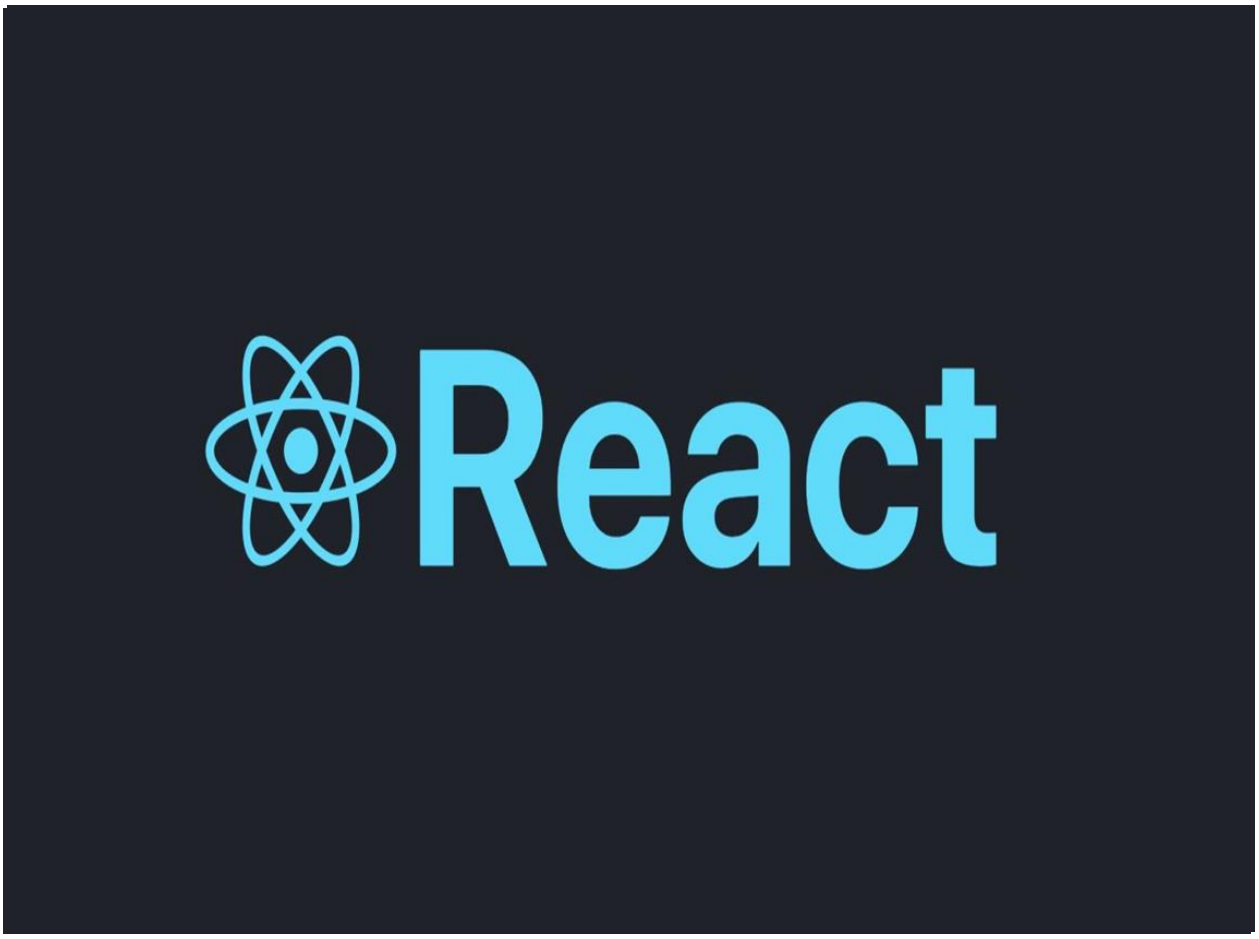

React Notes

Notes help you to start using react

—

By:

Ziad Etman

Create React App. (Zero Config)

Commands

- npx create-react-app appName

- cd appName

- npm start

Component types

1- Class Component “cc”

Import react, { Component } from “react”;

class Name extends Component{

state=[……]; -> Here we put the dynamic data we want to use in the dynamic JSX

…

Here we can put the Functions

…

render() {

… any needed vars or expressions …

return ……;

}

}

export default Name;

2- Function Component

"sfc"(stateless function component)

```
const Name = () => {
        return ......;
}
export default Name;
```

Notes

- To Edit the state with a new value connection use setState(object with the edited values of state)

- To use This in Event Listener:
  bind it in the constructor
  Or write it as an arrow function and use this in it directly (Best option)

- To use EventListener in JSX:
  call it like this -> onClick={ this.clickHandler } without ()

- To pass a parameter to the event listener:
  create another function and call the EL in it with the params passed (passed to the EL call, not the fun)
  And call it in JSX as ref (mean without ())
  Or
  call the EL like this onClick={ this.EL.bind(this, params) }
  Or write the arrow function and call the EL inside and pass the parameters normally
  OnClick={() => this.EL(params)} (best option)

Props

- Any value passed to the component like

<Component name=... title=... /> can be used in this component using props

In this example there is props.name , props.title

- All attributes appear in props except key

- we can pass whole element or component in the component and call it using children

Pass it like this <Comp> ...children... </Comp>

State Vs. Props

State: local data owned by that specific component

Props: data passed from parent component

If the Component doesn't use state and use only render .. we can turn it into "sfc"

BUT!! Don't forget to pass props as a parameter and delete any use of "this"

Component life Cycle

1- **Mounting** -> Create phase

constructor() -> initialization

render()

componentDidMount() -> it is used to call the **backend** server

2- **Updating**

Re-render

componentDidUpdate() -> here we can access the old and new values of state and props using prevProps, prevState for the old vals

3- **Unmount**

componentWillUnmount() - > here we can do something just before deleting the component

Hooks

Make us able to use all react features in function components

Advantages:-

- No need for this keyword
- No need to manually bind event handlers
- Allow for shared logic encapsulation
- Make it cleaner to build complex components

useState()

- Takes initial value of state to use it

```
const [ state, setState ] = useState(init val);
```

- To call it on some event

```
onClick = { () => setState(newValue) }
```

- We can setState based on prevState

```
setState( (prevState) => ..... )
```

- The val of state can be object To edit specific element in the object we do this

```
setState( { ...state, element : state.element } )
```

useEffect()

Cases:-

1- without dependency array

useEffect(Arrow function) Executes after every render

2- with empty dependency array

useEffect(Arrow function, [ ]) Executes just after first render

3- with dependency array

useEffect(Arrow function, [...,...])

Executes after first render and after any change in the dependencies.

useInterval()

Takes 2 parameters

1- function to be executed

2- waiting time before re-executing the function

clearInterval() -> the interval must be cleared so no unexpected actions happen

To clear it

useEffect(() => {

.....

Function implementation

const interval = setInterval(function, 1000);

return () => {clearInterval(interval)};

}, [...., ...]);

- If you want to use this function just in useEffect .. write it inside useEffect

Routing

To render different components based on the URL

There are 2 versions V5, V6 each one of them have its way

V5

Import { BrowserRouter, Routes, Route } from “react-router-dom”

To use it:-

```
<BrowserRouter>
<Routes>
<Route path=”/” element={ <Home/> }/>
        <Route path=”/about” element={<About/>}/>
    </Routes>
</BrowserRouter>
```

- If we want to add shared element in all routers we put this component in BR(BrowserRouter) outside Routes

- to navigate between routes, we use Link instead of a

```
<Link to=”path”> Link text </Link>
```

V6

This makes us able to deal with APIs

To use it :-

Add to the imports : createReactBrowser, createRoutersFromElements, RouterProvider

```
const router = createBrowserRouter(
CreateRoutesFromElements(
<Route path="/" element={ <RootLayout/> }>
                    <Route index (this means path "/") elements={<Home/>}/>
                    <Route path="about" element={<About/>}/>
                </Route>
        )
);
function RootLayout() {
return (
<>
                    Components to be shown at the top always like navbar
                    <Outlet/> (in Execution this will be replaced by the routes)
                    Components to be shown at the top always like navbar
</>
        );
}
 function App() {
return <RouterProvider router={router}/>}
```

Programmatically Navigate

useNavigate()

```
const navigate = useNavigate();

const handleClick = () => {

//logic

navigate(path)

}
```

Protected Route

Be used if there are pages can't be accessed except if logged in user or any condition

Forms

- you can build form in normal way but you must take care of onSubmit -> create its handler

- the values of the elements must be stored in the state so we can use them …. to sync values on submit you use

    name=”state element name” in html element and
    value={this.state.element}

- you must handle change too otherwise you won’t be able edit the form elements

- in onSubmit you must firstly write e.preventDefaults() so the page doesn’t render every time we submit

- to handle Change

```
changeHandler = e => {

//clone

Let state = { ...this.state };

//Edit

State[e.currentTarget.name] = e.currentTarget.value;

//set State

this.setState(state);

}
```

Validation

- to handle validation, you can use joi library or

We add “errors{}” in state

Create validate fun

```
Validate = () => {
const errors = {};
If(this.state.username.trim()===””)
errors.username=”username is required”;
If(this.state.password.trim()===””)
errors.username=”password is required”;
//set state
This.setState({errors});
        return object.keys(errors).length===0 ? Null : errors;
};
handleSubmit = e => {
e.preventDefault();
const errors = this.validate();
If(errors) return;
//else call backend and continue your logic
};
```

Calling Backend

The best place to call the backend is in componentDidMount()

As example to get data from JsonPlaceholder

```
componentDidMount() {
const promise = fetch('url of end point');
const res = promise.then(res => res.json());
Res.then(data => console.log(data));
}
```
(this get all data as json and output them as object)

Promise: it is returned result of calling any async function

- You can deal with backend using fetch or any other library like Axios

To do the same with Axios

```
async componentDidMount(){
const { data } = await axios.get(EP url);
}
```

- to test dummy data related to your app using json file … you need to use Json-Server

To use it (after installing) open terminal in the json file folder and run this command:

npx Json-Server –watch jsonfilename.json

Dealing with data

1- GET (list all elements)

```
async componentDidMount(){
//call backend data
const { data } = await axios.get(EP url);
//set state
this.setState( { stateName : data } );
}
```

2- POST (add new element)

In submit handler

```
await axios.post(EP url, obj(the new element));
```

3- GET (specific element by ID)

To get ID

```
const id = this.props.match.params.id;
await { data } = await axios.get( EP url + id );
```

4- PUT (edit element)

In submit handler

First delete id in the obj you are editing, so no conflict happens.

```
delete obj.id;

await axios.put(EP url + id, obj);
```

5- PATCH (edit element)

The same way as put ... but just send fields that need update

6- DELETE

```
await axios.delete( EP url + id );

Then edit the state .. as example

const products = this.states.filter( p => p.id !== product.id );

this .setState( {products} );
```

Pessimistic Vs. Optimistic

Pessimistic (slow app)

1- Call Backend

2- Change State

This can be better used while testing and developing to see the results of the execution in a better way

Optimistic (fast app)

1- Change State

2- Call Backend

After checking everything it's better to use this way so the app appears faster

Prop Drilling

sending info by props to many components that may not need it
happens because of Lifting State up so specific component - maybe in the same level- can reach it by props when it be in a shared parent

To solve this problem, we can use context API and useContext() hook.

useReduser()

It does the same job as useState()

It is better to use useReducer() in cases when we have many related states in the component and a handler for each of them

Using it here will make the code look cleaner as you will create just 1 handler for the related states and put the logic of each case in it

React.memo()

When you perform some action on a child component the whole component re-render even the child components that have not been changed

As example

<Parent>

<Child1 />

<Child2 />

</Parent>

If you perform action on child2 .. the whole component(Parent) re-render including child1 that have not been touched

To solve this and avoid unnecessary render we use React.memo()

It check the passed props if there is no changes (depending on the old props) it doesn't render

To use it you write it in the export line of the child component

export default React.memo(ComponentName);

useCallback()

Allow you to cache the function definition between renders

It takes 2 parameters

1- Callback function -> the function we want to cache its reference

2- Array of dependencies -> contains the variables that relate to the function

- you should use it with React.memo() to make it useful

useMemo()

Cache the result of any heavy calculation between renders

React Toastify

Allow you to add nice toast notification to your app.