**REACT HOOKS**

**Basics Hooks :-**

**useState :-** Returns a stateful value, and a function to update it.

const [state, setState] = useState(initialState);

setState(newState);

**useEffect :-** Accepts a function that contains imperative, possibly effectful code.

the **previous effect is cleaned up before executing the next effect**

Work like componentDidMount

useEffect(() => {

const subscription = props.source.subscribe();

return () => {

// Clean up the subscription

subscription.unsubscribe();

};

});

Work like componentDidUpdate

useEffect(

() => {

const subscription = props.source.subscribe();

return () => {

subscription.unsubscribe();

};

}, [props.source] );

**useContext :- (context API)**

**step1 :- (create context)**

import React, {createContext} from ‘react’

const GlobalSpinnerContext = createContext()

export default GlobalSpinnerContext

**step2 :- (wrap with context provider)**

*const* [state,dispatch] = useReducer(AppReduser,initialstate)

return (

<GlobalSpinnerContext.Provider value={{state , dispatch}}

>

<div className="App">

<GlobalSpinner />

<RandomComments />

</div>

</GlobalSpinnerContext.Provider>

);

}

**Step3 :- (use your context)**

import React, {useContext} from 'react'

import {GlobalSpinnerContext} from '../../context/GlobalSpinnerContext' //import your context

const {isGlobalSpinnerOn} = useContext(GlobalSpinnerContext)

**Additional Hooks :-**

**useReducer :- used for state management . useState is built using useReducer**

const [state, dispatch] = useReducer(reducername, initialstate);

const initialState = {count: 0};

//reducer

function reducer(state, action) {

switch (action.type) {

case 'increment':

return {count: state.count + 1};

case 'decrement':

return {count: state.count - 1};

default:

throw new Error();

}}

**useCallback :- (shouldComponentUpdate)**

const memoizedCallback = useCallback(

() => {

doSomething(a, b);

},

[a, b],

);

useCallback will return a memoized version of the callback that only changes if one of the dependencies has changed .

**useMemo :-**

**useRef :-**

**useImparativeHandle :-**

**useLayoutEffect :-**

**useDebugValue :-**

**REACT REDUX HOOKS**

**useSelactor :- access the state value .**

*const* contacts = useSelector(*state* *=>* state.contactReducer.contects)

**useDispatch :- Dispatch action and change the state value.**

*const* dispatch = useDispatch();

dispatch(getContact(value))

getContact is a action

import { getContact} from '../actions/contectAction'