***Legend***

Here are the ReactJS Hooks described in this document

* useState
* useEffect
* useContext
* useReducer

***Simple State Hook***

The method useState makes possible to define a state variable and a modifier function a the same time.

Here is an example of a counter

|  |
| --- |
| import React, { useState } from 'react';  const HookCounter = () => {      const [count,setCount] = useState(0);      return (        <div>           <div>Current Count is</div>          <div>{count}</div>            <button  onClick={ () => setCount(count+1 )}>Increase</button>           <button  onClick={ () => setCount(count-1 )}>Decrease</button>            </div>      );  };  export default HookCounter; |

***useEffect***

Appears to be used instead of the life cycle of the Class component

[ componentDidMount(), getDerivedStateFromProps , etc]

import React , { useEffect } from 'react';

const FunctionComponentName = () => {

// The use effect is run after each Render():

// Initial and every subsequent Render()

    useEffect( () => console.log('useEffect'));

***Note***: the array function configured in the useEffect will be run after every Render of the page

(Can be a bit too much sometime. See next section to see conditional execution of useEffect)

***useEffect - conditionnal***

Appears to be used instead of the life cycle of the Class component

import React , { useState , useEffect } from 'react';

import AgeSelector from './AgeSelector.js';

const NameInput = () => {

    const [age, setAge] = useState(0);

    useEffect( () => console.log('useEffect called') ,

                [ age ]

    );

Here, the ***useEffect*** hook is configured to change only when there is a change in the state variable ‘***age’***

|  |  |
| --- | --- |
| ***Param Type*** | ***Invoked*** |
| No param | After every Render() |
| [ ] 🡪 Empty bracked | Only after page load  Like the componentDidMount() used to to in the class component |
| [var name] | Only when there is a value change on the variable name |

***useContext***

A way to share information to sub components, instead of using a cascade of props.

An alternative to ReactJS Redux.

Each context should be created in the main class (i.e App.js)

Warning: If more than one contexts to be created, the synthax is much more complex!!!

Pass an general object, having all the info to be shared across all sub components instead.

App.js 🡪

import React from 'react';

import MySubComponent from './components/MySubComponent.js';

const daniel = { first: 'Daniel', last: 'Berger',  age : 51, city: 'Gatineau' };

export const UserInfo = React.createContext();

function App() {

  return (

    <div className="App">

        <UserInfo.Provider value={daniel}>

        <MySubComponent />

        </UserInfo.Provider>

      </div>

  );

}

Note that the subcomponent must be inserted between tags of the name Context variable (i.e UserInfo)

Sub Component.js

import React , { useState , useEffect, useContext } from 'react';

import { UserInfo } from '../App.js';

const MySubComponent = () => {

    const user = useContext(UserInfo);

    const output = (<div>

                        <div>{user.first}</div>

                        <div>{user.last}</div>

                        <div>{user.age}</div>

                        <div>{user.city}</div>

                  </div>);

return (

        <div>

            <div> {output}</div>

        </div>

    );

}

***useReducer***

This is the foundation of the useState, as it manages a class state, but the behavior can be configured.

Note: useReducer has common ideas with Redux (concept of reducer functions, actions to be handled)

import React, { useReducer } from 'react';

const intialValue = 0;

const reducer = (state,action) => {

    switch(action){

        case 'increment':

            return state +1;

        case 'decrement':

            return state -1;

        default:

            return state;

    }

}

const ReducerHookTest = () => {

    const [count,dispatch] = useReducer(reducer,intialValue);

    return (

        <div>

            <div>

                {count}

            </div>

            <div>

                <button   onClick={ () => dispatch('increment') }>increment</button>

                <button   onClick={ () => dispatch('decrement') }>decrement</button>

            </div>

        </div>

    );

};