REACT HOOKS

1

1. Initialization

1-What are hooks

Hooks are new feature addition in react version 16.8 which allow us to use react feature without having to write a class.

Ex: State of a component

Hooks don’t work inside classes

2-Why Hooks?

Reason Set 1:

Understand how this keyword works in JavaScript

Remember to bind event handlers in class components

Classes don't minify very well and make hot reloading very unreliable

Reason Set 2:

There is no particular way to reuse stateful component logic

HOC and render props patterns do address this problem

Makes the code harder to follow

There is need a to share stateful logic in a better way

Reason Set 3:

1-Create components for complex scenarios such as data fetching and subscribing to events

Related code is not organized in one place

2-Ex: Data fetching - In componentDidMount and componentDidUpdate

Ex: Event listeners — In componentDidMount and componentWillUnmount

3-Because of stateful logic — Cannot break components into smaller ones 3

3-NoteWorthy Points

React version 16.8 or higher

Completely opt in

Hooks don’t contain any breaking changes and the release is 100% backwards-compatible.

Classes won't be removed from React

Can’t use Hooks inside of a class component

Hooks don't replace your existing knowledge of React concepts

Instead, Hooks provide a more direct API to the React concepts you already know

4-!!!!!!!!!!!!! Summary

Hooks are a new feature addition in React version 16.8

They allow you to use React features without having to write a class

Avoid the whole confusion with this' keyword 3

Allow you to reuse stateful logic

Organize the logic inside a component into reusable isolated units

RFCE to create a functional component

1. UseState
   * 1. Rules of Hooks

1-"Only Call Hooks at the Top Level”

Don’t call Hooks inside loops, conditions, or nested functions

2-"Only Call Hooks from React Functions”

Call them from within React functional components and not just any regular

JavaScript function

* + 1. UseState Hook

1. import React ,{useState} from 'react'
2. function App() {
3. const [count,setCount]=useState(0)
4. return (
5. <div className="App">
6. Count:
7. <button onClick={()=>setCount(count+1)}>{count}</button>
8. </div>
9. );
10. }
11. export default App;
12. UseState with previous state
13. import React ,{useState} from 'react'
14. function App() {
15. const [count,setCount]=useState(0);
16. const AddFive=()=>{
17. for(let i=0;i<5;i++){
18. //setCount(count +1) c'est pas la bonne manniere
19. // il ajouter 1 seulement une seule fois pas 5 il faut
20. //faire comme e bas pour que ca marche
21. setCount(prev=>prev+1)
22. }
23. }
24. return (
25. <div className="App">
26. Count:
27. <button onClick={()=>AddFive()}>{count}</button>
28. </div>
29. );
30. }
31. export default App;
32. UseState with Object Hook

import React, { useState } from "react";

function App() {

  const [name, setName] = useState({ firstName: "", lastName: "" });

  return (

    <form>

      <input

        type="text"

        value={name.firstName}

        onChange={(e) => setName({ ...name, firstName: e.target.value })}

      />

      <input

        type="text"

        value={name.lastName}

        onChange={(e) => setName({ ...name, lastName: e.target.value })}

      />

      <h2>Your first name is - {name.firstName}</h2>

      <h2>Your last name is - {name.lastName}</h2>

      <h2>{JSON.stringify(name)}</h2>

    </form>

  );

}

1. useState With Array
2. import React, { useState } from "react";
3. function App() {
4. const [items, setItems] = useState([]);
5. const addItem = () => {
6. setItems([
7. ...items,
8. {
9. id: items.length,
10. value: Math.floor(Math.random() \* 10) + 1,
11. },
12. ]);
13. };
14. return (
15. <div>
16. <button onClick={addItem}>Add a number</button>
17. <ul>
18. {items.map((item) => (
19. <li key={item.id}>{item.value}</li>
20. ))}
21. </ul>
22. </div>
23. );
24. }

6. Summary UseState

The useState hook lets you add state to functional components.

In classes, the state is always an object.

With the useState hook, the state doesn’t have to be an object.

The useState hook returns an array with 2 elements.

The first element is the current value of the state, and the second element is a state setter

function.

New state value depends on the previous state value? You can pass a function to the setter function.

When dealing with objects or arrays, always make sure to spread your state variable and then

call the setter function.

1. UseEffect

## Introduction

The Effect Hook lets you perform side effects in functional components.

It is a close replacement for componentDidMount, componentDidUpdate and componentWillUnmount .

!!!!!!!!!!!!!! Si on veut faire l’apple d’une function dans le UseEffect c’est mieux de cree cette finction dans le useEffect

We can use many useEffect in just one React hook function

1. useEffect after every render
2. const [count, setCount] = useState(0);
3. useEffect(() => {
4. document.title = `You clicked ${count} times`;
5. });
6. //UseEffect Run After every Runder
7. return (
8. <div>
9. <button onClick={() => setCount(count + 1)}>Click {count} times</button>
10. </div>
11. );
12. useEffect Conditionally run effects

Applying the Effect after every Render sometimes Creates many problems so we need to use Conditionnally Effect

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

function App() {

  const [count, setCount] = useState(0);

  const [name, setName]=useState('');

  useEffect(() => {

    document.title = `You clicked ${count} times`;

  },[count]);

  //UseEffect Run After every Count Change (Props or state change)

  return (

    <div>

      <input type='text' value={name} onChange={e=>setName(e.target.value)}/>

      <button onClick={() => setCount(count + 1)}>Click {count} times</button>

    </div>

  );

}

1. useEffect Run Effect only Once
2. import React, { useState, useEffect } from "react";
3. function App() {
4. const [x, setX] = useState(0);
5. const [y, setY]=useState(0);
6. const logMousePosition=e=>{
7. console.log("Mouse event")
8. setX(e.clientX);
9. setY(e.clientY)
10. }
11. useEffect(() => {
12. console.log("UseEffect alled")
13. window.addEventListener('mousemove',logMousePosition);
14. },[]);
15. //UseEffect Run Effect only once Like ComponentDidMount
16. return (
17. <div>
18. {`X:"${x}"    Y:"${y}"`}
19. </div>
20. );
21. }

5) UseEffect with cleanup

componentWillUnmount() Method

In this article, we are going to see how to execute a function when the component is deleted from the DOM tree.

This method is called during the unmounting phase of the React Lifecycle, i.e., before the component is destroyed or unmounted from the DOM tree. This method is majorly used to cancel all the subscriptions that were previously created in the componentWillMount method.

Never call this.setState() inside the componentWillUnmount method as this component is never going to be re-rendered again.

componentWillUnmount() Method To UseEffect

* In a class we define componentWillUnmount Like this

componentWillunmount(){

window.removeEventListener( 'mousemove', this.logMousePosition)

}

* In a React Hook Function

HookMouse.js

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

export default function HookMouse() {

  const [x, setX] = useState(0);

  const [y, setY] = useState(0);

  const logMousePosition = (e) => {

    console.log("Mouse event");

    setX(e.clientX);

    setY(e.clientY);

  };

  useEffect(() => {

    console.log("UseEffect alled");

    window.addEventListener("mousemove", logMousePosition);

    return()=>{

        console.log("Component unmouning Code")

        window.removeEventListener("mousemove", logMousePosition)

    }

    //It will executed when the component will unmount Mine kaymoute Makayb9ach f Dom

  }, []);

  //UseEffect Run Effect only once Like ComponentDidMount

  return <div>{`X:"${x}"    Y:"${y}"`}</div>;

}

App.js

import "./App.css";

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

import HookMouse from "./Component/HookMouse";

function App() {

  const [display, setDisplay] = useState(true);

  return (

    <div>

      <button onClick={() => setDisplay((prev) => !prev)} />

      {display && <HookMouse />}

    </div>

  );

}

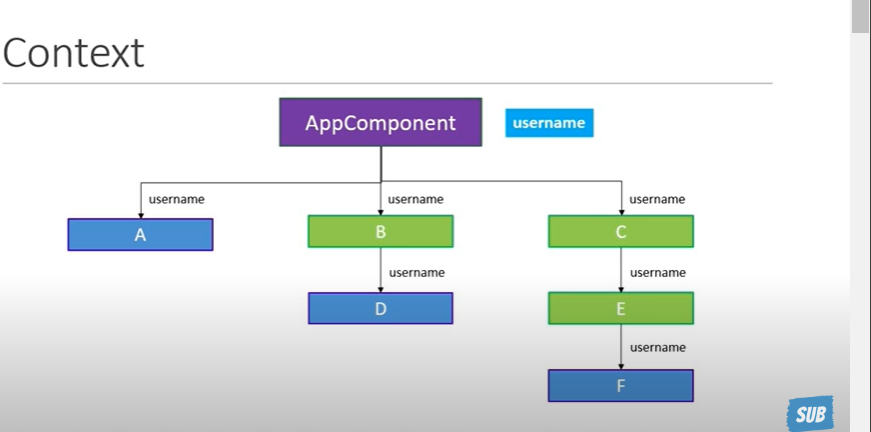
export default App;

1. UseContext

Context provides a way to pass data through the component tree without having to pass props down manually at every level

Le cas normal en passe username proprs de composent père jusqu’à le composant fille mais avec useContext on peux passe le props directement d’un composent a l’autre composant.

Le cas normale sans utiliser UseContext



It’s so simple you can see this Example:

App.js

import "./App.css";

import React from "react";

// import HookMouse from "./Component/HookMouse";

import ComponentC from "./Component/ComponentC";

export const UserContext=React.createContext()

export const ChannelContext=React.createContext()

function App() {

  return (

    <div>

      <UserContext.Provider value={'Vishwas'}>

      <ChannelContext.Provider value={'codeEvolution'}>

      <ComponentC/>

      </ChannelContext.Provider>

      </UserContext.Provider>

    </div>

  );

}

export default App;

ComponentF.js

import React,{ useContext} from 'react'

import { UserContext ,ChannelContext } from '../App'

export default function ComponentF() {

    const user=useContext(UserContext)

    const channel=useContext(ChannelContext)

  return (

    <div>{user} {channel}</div>

  )

}

1. useReducer

1.Introduction

useReducer is a hook that is used for state management

useReducer is related to reducer functions

useReducer(reducer,initialState)

reducer(currentState,action)

It is an alternative to useState

useState is built using useReducer

When to useReducer vs useState?

UseReducer is more simpler than using React-Redux

Reduce in JS

Une image contenant texte

Description générée automatiquement

Une image contenant table

Description générée automatiquement

Example UseReducer is just Like Redux

Simple State and action

import React, { useReducer } from "react";

import { UserContext, ChannelContext } from "../App";

export default function ComponentC() {

  const initialState = 0;

  const reducer = (state, action) => {

    switch (action) {

      case "increment":

        return state + 1;

      case "decrement":

        return state - 1;

      case "reset":

        return initialState;

      default:

        return state;

    }

  }

  const [count,dispatch]=useReducer(reducer,initialState)

  return (

    <div>

     <div>Count -{count}</div>

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

     <button onClick={()=>dispatch("decrement")}>n9sse</button>

     <button onClick={()=>dispatch("reset")}>Rde l9lawi ki kane</button>

    </div>

  );

}

Complex state and action

import React, { useReducer } from "react";

import { UserContext, ChannelContext } from "../App";

export default function ComponentC() {

  const initialState = { firstCounter: 0, secondCounter: 10 };

  const reducer = (state, action) => {

    switch (action.type) {

      case "increment":

        return { ...state, firstCounter: state.firstCounter + action.value };

        case "decrement":

          return {...state,firstCounter:state.firstCounter - action.value};

      case "reset":

        return initialState;

      default:

        return state;

    }

  };

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

  return (

    <div>

      <div>Count -{count.firstCounter}</div>

      <button onClick={() => dispatch({ type: "increment", value: 1 })}>

        Zide

      </button>

      <button onClick={() => dispatch({type:"decrement",value:1})}>n9sse</button>

      <button onClick={() => dispatch({type:"reset",value:1})}>Rde l9lawi ki kane</button>

    </div>

  );

}

2.Multiple Reducer

We can use Multiple UserReducer like multiple state with usining the same reducer function or different reducer function

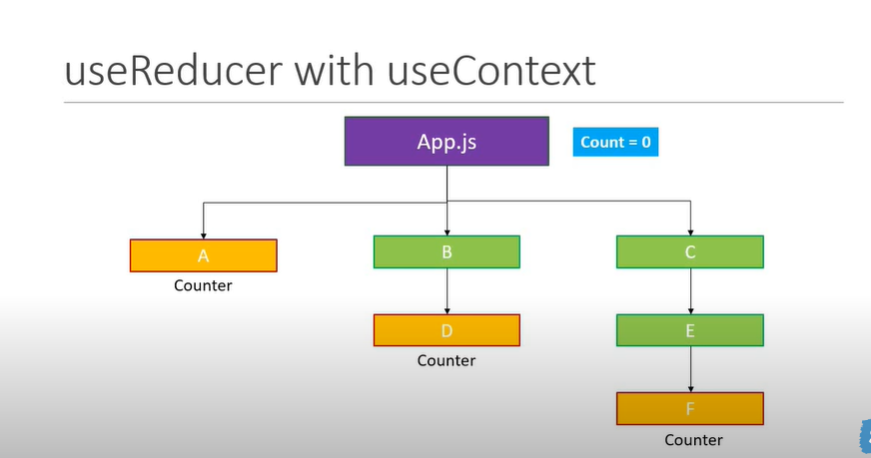
3.UseReducer with useContext

useReducer — Local state management

Share state between components — Global state management

useReducer + useContext

In This example we need to use the same counter State in many component without using Props



App.js

import "./App.css";

import React, { useReducer } from "react";

// import HookMouse from "./Component/HookMouse";

import ComponentA from "./Component/ComponentA";

import ComponentD from "./Component/ComponentD";

import ComponentF from "./Component/ComponentF";

export const CountContext = React.createContext();

function App() {

  const initialState = 0;

  const reducer = (state, action) => {

    switch (action) {

      case "increment":

        return state + 1;

      case "decrement":

        return state - 1;

      case "reset":

        return initialState;

      default:

        return state;

    }

  };

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

  return (

    <CountContext.Provider

      value={{ countState: count, countDispatch: dispatch }}

    >

      <div>

        {count}

        <ComponentA/>

        <ComponentD />

        <ComponentF />

      </div>

    </CountContext.Provider>

  );

}

export default App;

ComponentD

import React, { useContext } from 'react'

import { CountContext } from '../App';

export default function ComponentA() {

    const countContext = useContext(CountContext);

  return (

    <>

      <div>ComponentA -{countContext.countState}</div>

     <button onClick={()=>countContext.countDispatch("increment")}>Zide</button>

     <button onClick={()=>countContext.countDispatch("decrement")}>n9sse</button>

     <button onClick={()=>countContext.countDispatch("reset")}>Rde l9lawi ki kane</button>

    </>

  )

}

ComponentD

import React, { useContext } from "react";

import {  CountContext } from "../App";

export default function ComponentD() {

    const countContext = useContext(CountContext);

    return (

      <>

        <div>ComponentD -{countContext.countState}</div>

       <button onClick={()=>countContext.countDispatch("increment")}>Zide</button>

       <button onClick={()=>countContext.countDispatch("decrement")}>n9sse</button>

       <button onClick={()=>countContext.countDispatch("reset")}>Rde l9lawi ki kane</button>

      </>

    )

}

ComponentF

import React, { useContext } from "react";

import { CountContext } from "../App";

export default function Component() {

  const countContext = useContext(CountContext);

  return (

    <>

      <div>ComponentF -{countContext.countState}</div>

     <button onClick={()=>countContext.countDispatch("increment")}>Zide</button>

     <button onClick={()=>countContext.countDispatch("decrement")}>n9sse</button>

     <button onClick={()=>countContext.countDispatch("reset")}>Rde l9lawi ki kane</button>

    </>

  );

}

3.Fetching data using useReducer

import "./App.css";

import React, { useReducer, useEffect } from "react";

import axios from "axios";

export const CountContext = React.createContext();

const initialState = {

  loading: true,

  error: "",

  post: {},

};

const reducer = (state, action) => {

  switch (action.type) {

    case "FETCH\_SUCCESS":

      return {

        loading: false,

        post: action.payload,

        error: "",

      };

    case "FETCH\_ERROR":

      return {

        loading: false,

        post: {},

        error: "Something went wrong",

      };

    default:

      return state;

  }

};

function App() {

  const [state, dispatch] = useReducer(reducer, initialState);

  useEffect(() => {

    axios

      .get("https://jsonplaceholder.typicode.com/todos/3")

      .then((response) => {

        dispatch({ type: "FETCH\_SUCCESS", payload: response.data });

      })

      .catch((error) => {

        dispatch({ type: "FETCH\_ERROR" });

      });

  });

  return (

    <>

      {state.loading ? "loading" : state.post.title}

      {state.error ? state.error : null}

    </>

  );

}

export default App;

3.useState vs useReducer

Une image contenant table

Description générée automatiquement

1. useCallback && useMemo

Une image contenant texte

Description générée automatiquement

1. useRef

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

This example is the common use cases to use useRef

In this example we want the user input field name to be focus

import "./App.css";

import React, { useRef, useEffect } from "react";

function App() {

     const inputRef=useRef(null)

   useEffect(()=>  {

      //focus the input Element

     inputRef.current.focus()

     //React will send the ref current property to the corresponding Dom node

   },[])

  return (

    <>

     <input ref={inputRef} type='text'/>

    </>

  );

}

export default App;

Some problem can slove using useRef

Une image contenant texte

Description générée automatiquement

Solution using useRef

import "./App.css";

import React, { useRef, useEffect,useState } from "react";

function App() {

     const [timer,setTimer]=useState(0)

  const inervalRef=useRef()

     useEffect(()=>{

      inervalRef.current=setInterval(()=>{

        setTimer(prevTimer=>prevTimer+1)

      },1000)

      // return()=>{

      //   clearInterval(inervalRef.current)

      // }

     },[])

  return (

    <>

      Hook Timer -{timer}

      <button onClick={()=>clearInterval(inervalRef.current)}>Clear Hook Timer</button>

    </>

  );

}

export default App;