EPICODE

React Hooks

Gli hooks in React sono una delle features più recenti aggiunte alla libreria.

Sono stati aggiunti con l'intento di combinare la forma più semplice di un componente React (quelli a funzione) con i superpoteri associati normalmente ai componenti a classe (con lo stato interno e i metodi di lifecycle).

Queste funzionalità aggiuntive non sono una rappresentazione 1-to-1 di quelli che già conosci, di solito si usano facilmente e parlando in linea generale, se il tuo componente ha bisogno di una o due variabili di stato o magari una fetch dopo il montaggio iniziale del componente, possono essere una valida alternativa al riscrivere l'intero componente con la più corposa sintassi della classe.

Gli hook più utilizzati sono **useState()** e **useEffect()**, vediamoli nello specifico.

useState è un hook che aggiunge funzionalità al tuo componente a funzione.

Ritorna un array con due variabili: una contenente lo stato e l'altra una funzione "setter" per cambiarne il valore.
Come parametro opzionale può anche accettare il valore iniziale con cui inizializzare lo stato.

```
import { useState } from 'react'

const MyComponent = () => {
    const [counter, setCounter] = useState(0)

    return (
        <>
            <button onClick={() =>
                setCounter(counter + 1)
            >CLICK ME</button>
             <div>{counter}</div>
        </>
    )
}

export default MyComponent
```

Le variabili di stato dichiarate con **useState()** possono contenere qualsiasi valore primitivo, così come array, oggetti ecc..

Ricorda solamente che la funzione "setter" sovrascriverà lo stato attuale della variabile di stato associata, invece che unire i valori al contenuto esistente come faceva il metodo setState() in un componente classe.

useEffect replica le funzionalità del metodo di lifecycle più usate in un componente a classe:

- componentDidMount
- componentDidUpdate
- componentWillUnmount

Può essere eseguito direttamente nel body del tuo componente a funzione, inoltre accetta una funzione di callback come primo parametro e un array di dipendenze facoltativo come secondo parametro.

Il secondo parametro definisce il comportamento di questo hook: la funzione di callback verrà eseguita ogni volta che una delle dipendenze all'interno dell'array cambierà valore.

Se ometterai completamente l'array di dipendenze, la funzione di callback verrà eseguita ad ogni render del componente, esattamente come farebbe **componentDidUpdate**.

Se invece userai un array di dipendenze vuoto come secondo parametro, la funzione verrà chiamata una volta soltanto, dopo il render iniziale. Lo stesso comportamento di **componentDidMount**.

Infine, mettere un return statement esplicito nella tua funzione di callback ti permette di eseguire un'operazione di pulizia prima dello smontaggio (unmount) del componente. Gli utilizzi più classici per questo hook sono la rimozione di timer pendenti, o la chiusura di canali di comunicazione real-time lasciati aperti.
Il lifecycle method che in questo caso l'hook sta replicando si chiama **componentWillUnmount.**

```
import { useEffect } from 'react'

const MyComponent = () => {
  useEffect(() => {
    // come componentDidMount
  }, [])

  useEffect(() => {
    // come componentDidUpdate
  })

  useEffect(() => {
    return () => {
      // come componentWillUnmount
    }
  })

  return (
    <>
      <div>Hello, world!</div>
    </>
  )
}

export default MyComponent
```

```
import { useEffect, useState } from 'react'

const MyComponent = () => {
  const [counter, setCounter] = useState(0)

  useEffect(() => {
    // questo codice verrà eseguito ogni volta
che il valore di counter cambierà!
  }, [counter])

  return (
    <>
      <button onClick={() =>
        setCounter(counter + 1)
      >CLICK ME</button>
      <div>{counter}</div>
    </>
  )
}

export default MyComponent
```