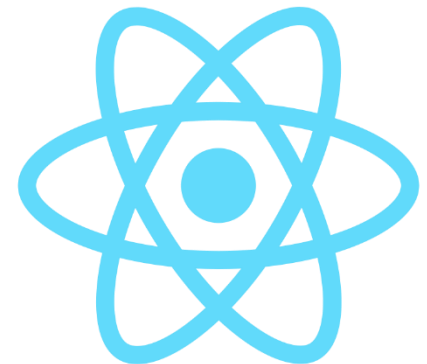


React

Riccardo Cattaneo



Contatore

Prima di andare avanti andiamo a creare un nuovo componente «contatore» per poi vedere alcune nuove funzionalità di React. Per prima cosa creiamo il componente Contatore e dichiariamo una variabile usando useState e passiamo come valore 0, ed all'interno del nostro componente stampiamo il valore della variabile e aggiungiamo 2 bottoni : aumenta e diminuisci.

Bootstrap

Prima di continuare importiamo le librerie di bootstrap all'interno del nostro progetto. Il modo più semplice è scaricare le librerie da riga di comando :

```
npm install - -save bootstrap
```

Dopo di ch  all'interno del file index.js fare l'import con :

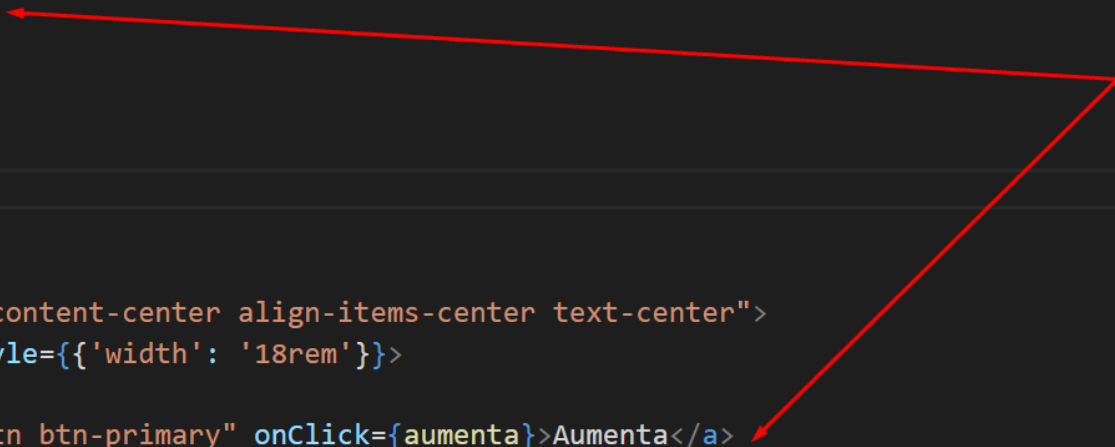
```
import 'bootstrap/dist/css/bootstrap.css';
```

```
const Contatore = () => {  
  
  const [contatore, setContatore] = useState(0);  
  
  return (  
    <div className="container">  
      <div className="row justify-content-center align-items-center text-center">  
        <div className="card" style={{'width': '18rem'}}>  
          <h1>{contatore}</h1>  
          <a href="#" class="btn btn-primary">Aumenta</a>  
          <a href="#" class="btn btn-secondary">Diminuisci</a>  
        </div>  
      </div>  
    </div>  
  )  
}
```

Possiamo ora sviluppare le nostre funzioni in due modi differenti ma che fanno la stessa cosa : il primo modo è mettendo la funzione `setContatore` direttamente nell'evento `onClick` tramite una arrow function.

Il secondo modo, quello classico che conosciamo, dove facciamo più passaggi ma il codice risulta più leggibile, cioè tramite una funzione separata. Ricordiamo che la funzione `setContatore` prende un parametro in **ingresso che rappresenta il valore attuale** del contatore :

```
const Contatore = () => {  
  
  const [contatore, setContatore] = useState(0);  
  
  const aumenta = () => {  
    setContatore(valoreAttuale => {  
      console.log(valoreAttuale);  
      return valoreAttuale + 1;  
    })  
  }  
  
  return (  
    <div className="container">  
      <div className="row justify-content-center align-items-center text-center">  
        <div className="card" style={{width: '18rem'}}>  
          <h1>{contatore}</h1>  
          <a href="#" class="btn btn-primary" onClick={aumenta}>Aumenta</a>  
          <a href="#" class="btn btn-secondary" onClick={() => setContatore(contatore - 1)}>Diminuisci</a>  
        </div>  
      </div>  
    </div>  
  )  
}
```



useState : return value vs functional

Quando chiamo la funzione useState abbiamo detto che mi torna il valore ed una funzione set. Questa funzione set abbiamo detto che serve per poter gestire ed aggiornare in modo autonomo il valore passato a useState. Ad esempio :

```
const [contatore1, setContatore1] = useState(0);  
let contatore2 = 0;
```

Ripetiamo che è la stessa cosa, nel primo caso sto usando la funzione `useState` nel secondo caso sto valorizzando direttamente la variabile.

Quindi qual è la differenza tra i due ? Che `useState` è una funzione che torna il suo valore ed in più torna una funzione che gestisce lo stato della variabile. Quindi se ad esempio cambio il valore della variabile tramite la sua funzione di ritorno, viene aggiornata automaticamente nel componente (e ricordiamo pure che `useState` può essere usata solo all'interno di un componente)


```
const [contatore1, setContatore1] = useState(0);
let contatore2 = 0;

const aumenta = () => {
  setContatore1(valoreAttuale => {
    console.log(valoreAttuale);
    return valoreAttuale + 1;
  })
}

const aumenta2 = () => {
  contatore2++;
  console.log(contatore2);
}

const diminuisci2 = () => {
  contatore2--;
  console.log(contatore2);
}

return (
  <div className="container">
    <div className="row justify-content-center align-items-center text-center">
      <div className="card" style={{'width': '18rem'}}>
        <h2>Contatore 1 : {contatore1}</h2>
        <h2>Contatore 2 : {contatore2}</h2>
        <a href="#" class="btn btn-primary" onClick={aumenta}>Aumenta Cont1</a>
        <a href="#" class="btn btn-secondary" onClick={()=>setContatore1(contatore1 - 1)}>Diminuisci Cont1</a>
        <a href="#" class="btn btn-primary" onClick={aumenta2}>Aumenta Cont2</a>
        <a href="#" class="btn btn-secondary" onClick={diminuisci2}>Diminuisci Cont2</a>
      </div>
    </div>
  </div>
)
```

Una volta vista la differenza tra valorizzare una variabile tramite `useState` e valorizzarla normalmente con il suo valore, vediamo quando una `useState` cambia valore tramite la sua funzione `set` che può farlo in 2 modi : tramite il valore o tramite il ritorno di una funzione.

Per fare un esempio andiamo a vedere la differenza sul bottone Aumenta e Diminuisce. Su aumenta cambiamo direttamente il valore, su diminuisce lo cambiamo tramite il ritorno di funzione, detto anche **functional return**.

```
const [contatore1, setContatore1] = useState(0);
```


```
const aumenta = () => {  
  setContatore1(contatore1+1);  
  console.log(contatore1);  
}
```

```
const diminuisci = () => {  
  setContatore1(valoreAttuale => {  
    console.log(valoreAttuale);  
    return valoreAttuale - 1;  
  })  
}
```

Se proviamo ora il nostro codice effettivamente funziona correttamente in tutte e due i modi. Qual è quindi la differenza tra usare la funzione set **passando direttamente il nuovo valore** oppure usare una **arrow function** che automaticamente prende in ingresso il valore attuale della variabile ?

La differenza è che il **functional return tiene traccia del valore** e lo aggiorna in maniera corretta. Per capirlo modifichiamo il nostro esempio ed aggiungiamo la funzione `setTimeout` cioè il cambio valore avviene dopo 2 secondi e non subito :

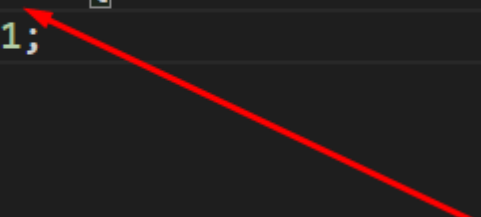
```
const Contatore = () => {  
  
  const [contatore1, setContatore1] = useState(0);  
  
  const aumenta = () => {  
    setTimeout(function(){  
      setContatore1(contatore1+1);  
      console.log(contatore1);  
    }, 2000)  
  }  
}
```



In questo caso ad ogni click cambia il valore, ma ogni 2 secondi. Quindi se clicco dopo 2 secondi non ci sono problemi. Ma cosa succede se clicco velocemente per 10 volte ? Succede che non prende tutti e 10 i click in quanto non rimane traccia di tutto quello che è successo mentre attendeva i 2 secondi.

Questo problema viene risolto e viene gestito automaticamente dal **functional return** di useState in questo modo :

```
const aumenta = () => {  
  setTimeout(function(){  
    setContatore1(valoreAttuale => {  
      return valoreAttuale + 1;  
    });  
    console.log(contatore1);  
  }, 2000)  
}
```



Per questo motivo normalmente tutti gli useState vengono gestiti con una functional return.

useEffect

useEffect è un Hook che è responsabile di gestire tutte le azioni che avvengono al di fuori del componente.

Per fare un esempio andiamo a creare un nuovo componente ed importiamo lo `useEffect` così come abbiamo fatto per `useState` ed anche in questo caso è utilizzabile solo dentro un componente :

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

const EsempioUseEffect = () => {

  useEffect(() => {
    console.log('ho chiamato una useEffect');
  })

  return (
    <div>
      <h1>useEffect</h1>
    </div>
  )
}

export default EsempioUseEffect
```


La **prima regola** che possiamo evincere da questo esempio è che `useEffect` viene chiamato DOPO che viene fatto il render del componente, praticamente viene chiamato ogni volta che usiamo il nostro componente.

Questo possiamo verificarlo subito andando a stampare dopo dello `useEffect` un altro log e vediamo che invece lo stampa prima :

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

const EsempioUseEffect = () => {

  useEffect(() => {
    console.log('ho chiamato una useEffect');
  })

  console.log('sono al di fuori di useEffect');

  return (
    <div>
      <h1>useEffect</h1>
    </div>
  )
}

export default EsempioUseEffect
```

► XHR finished loading: GET "<http://localhost:3000/79f6b5e...hot-update.json>".

sono al di fuori di useEffect

ho chiamato una useEffect

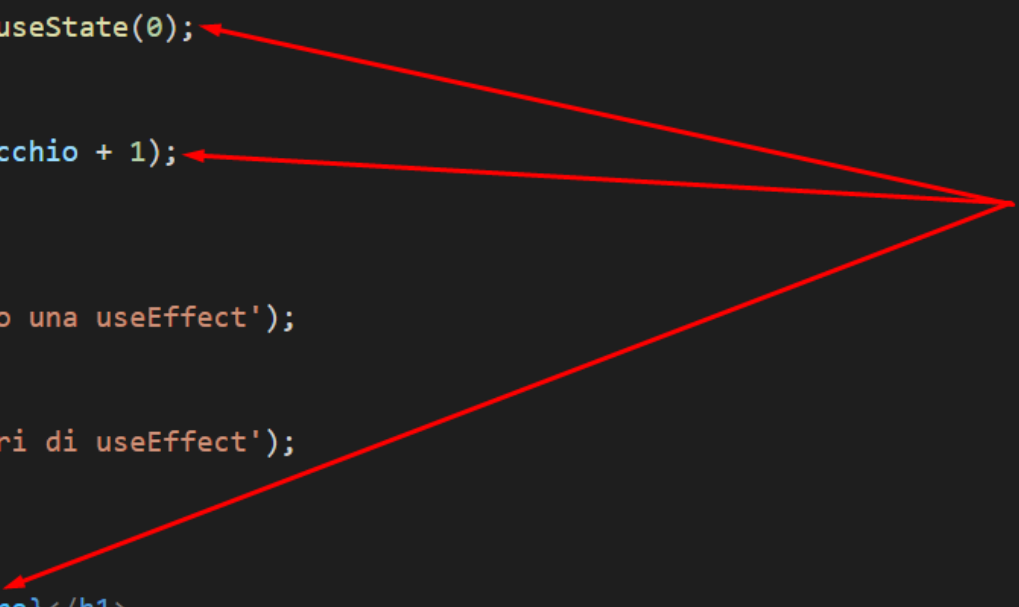
>

Per fare un esempio andiamo a cambiare il titolo della pagina utilizzando l'oggetto document di javascript (**document.title**).

Ricordiamoci che **useEffect viene chiamato ogni volta che il nostro componente subisce un render**, quindi se modifichiamo qualcosa al nostro componente verrà automaticamente eseguito anche la funzione useEffect (questo grazie alle useState) :

Riprendiamo l'esempio precedente dove con un click e l'utilizzo di useState vado ad incrementare un numero :

```
const EsempioUseEffect = () => {  
  
  const [valore, setValore] = useState(0);  
  
  const aumenta = () => {  
    setValore(vecchio => vecchio + 1);  
  };  
  
  useEffect(() => {  
    console.log('ho chiamato una useEffect');  
  });  
  
  console.log('sono al di fuori di useEffect');  
  
  return (  
    <div>  
      <h1>useEffect {valore}</h1>  
      <button onClick={aumenta}>Aumenta</button>  
    </div>  
  )  
}  
  
export default EsempioUseEffect
```



useEffect 15

Aumenta

Chrome DevTools Console interface showing the following components:

- Top Bar:** Elements, Console, Sources, Network, Performance, Memory, Security, Application, Lighthouse. Right side: 3 warnings, settings gear, and a vertical ellipsis.
- Left Sidebar:**
 - 38 messages
 - 38 user messages
 - No errors
 - 3 warnings
 - 35 info
 - No verbose
- Console Panel:**
 - Buttons: top, eye icon, Filter, Default levels, No Issues, 3 hidden.
 - Settings:
 - ☐ Hide network
 - ☒ Preserve log
 - ☐ Selected context only
 - ☒ Group similar messages in console
 - ☒ Log XMLHttpRequests
 - ☒ Eager evaluation
 - ☒ Autocomplete from history
 - ☒ Evaluate triggers user activation
 - Log messages (15 visible):
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect
 - ho chiamato una useEffect
 - sono al di fuori di useEffect

All'interno degli Hooks possiamo inserire anche le condizioni ricordando che verranno eseguite solo dopo il render :

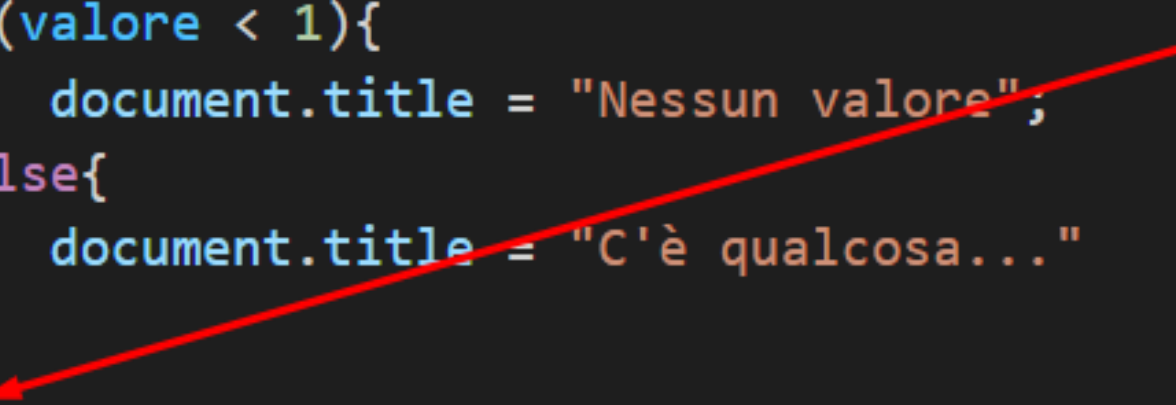
```
useEffect(() => {  
  console.log('ho chiamato una useEffect');  
  if(valore < 1){  
    document.title = "Nessun valore";  
  }else{  
    document.title = "C'è qualcosa..."  
  }  
});
```

useEffect secondo parametro

Di default il nostro useEffect verrà eseguito ad ogni render del nostro componente. Ma c'è un modo per decidere quando eseguire il nostro useEffect, e cioè attraverso il secondo parametro che opzionalmente accetta useEffect che è un array.

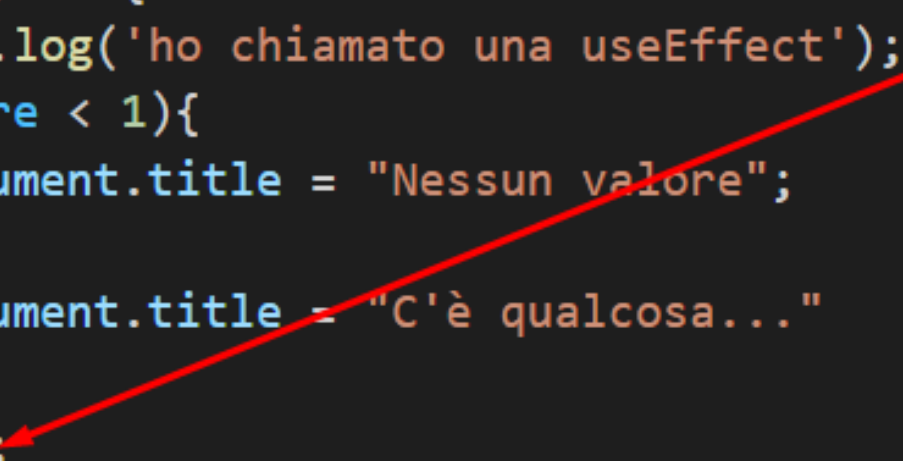
Se lascio **l'array vuoto** sto dicendo che il nostro useEffect deve essere eseguito una sola volta :

```
useEffect(() => {  
  console.log('ho chiamato una useEffect');  
  if(valore < 1){  
    document.title = "Nessun valore";  
  }else{  
    document.title = "C'è qualcosa..."  
  }  
}, []);
```



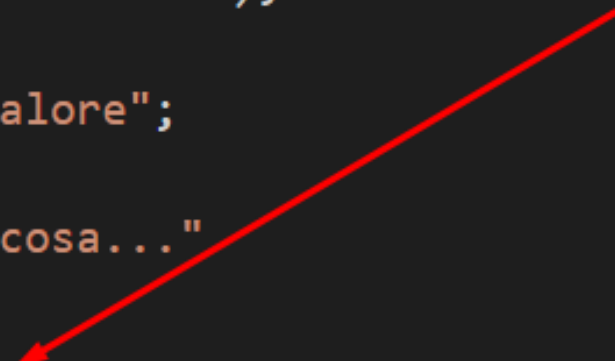
Oppure posso passare un parametro del componente, serve ad indicare che alla variazione di quel parametro `useEffect` verrà eseguito :


```
useEffect(() => {  
  console.log('ho chiamato una useEffect');  
  if(valore < 1){  
    document.title = "Nessun valore";  
  }else{  
    document.title = "C'è qualcosa..."  
  }  
},[valore]);
```



NB la funzione `useEffect` può essere presente più di una volta all'interno di un componente. Opzionalmente può tornare una function chiamata **Cleanup** che ha di particolare che viene chiamata prima del render successivo.

```
useEffect(() => {  
  console.log('ho chiamato una useEffect');  
  if(valore < 1){  
    document.title = "Nessun valore";  
  }else{  
    document.title = "C'è qualcosa..."  
  }  
  
  return( () => { console.log("Eseguo un po di pulizia") })  
},[valore]);
```



Esercizio 1

Realizzare un componente che gestisce un cronometro. Impostare una variabile a zero e mostrarla a video. Aggiungere 3 bottoni : Start Stop e Reset.

