Umberto Emanuele

# React – Giorno 5

Gennaio 2023


## React Hooks

**Gli Hooks sono stati aggiunti in React 16.8. Ti permettono di utilizzare state ed altre funzioni di React senza dover scrivere una classe.**

https://reactjs.org/docs/hooks-reference.html

# PRO

- ✓ Iniziare ad usare i React Hooks è facile…
- ✓ Molti sviluppatori preferiscono utilizzare i componenti funzioni invece delle classi (più semplici)
- ✓ Perfetti per creare applicazioni veloci o per fare esempi (sono molto usati anche nelle documentazioni proprio per questo)
- ✓ OTTIMI per disaccoppiare la logica nei React Components!
- ✓ Potrebbero diventare il futuro dell'intera libreria React


# CONTRO

- ✓ … ma padroneggiarli a dovere è un’altra storia
- ✓ Non sostituiscono al 100% le funzionalità dei componenti di classe (almeno non facilmente)
- ✓ Una volta che l'applicazione cresce, potrebbero diventare più difficili da leggere e mantenere rispetto alle classi
- ✓ Non sono ancora spinti al 100% dal team di sviluppo di React (la documentazione ufficiale spiega ancora tutto con le classi, gli hook vengono introdotti separatamente, in un capitolo diverso)

# React Hooks

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

function Esempio() {
  // ...
}
```

**Cos'è un Hook?** Un Hook è una speciale funzione che ti permette di "agganciare" funzionalità di React. Per esempio, `useState` è un Hook che ti permette di aggiungere lo state React nei componenti funzione. Impareremo altri Hooks più tardi.

**Quando dovrei utilizzare un Hook?** Se scrivi un componente funzione e capisci di aver bisogno dello state, prima avresti dovuto convertirlo in classe. Adesso puoi utilizzare un Hook dentro il componente funzione esistente. Entriamo subito nel dettaglio!

# React Hooks


```
class Esempio extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      contatore: 0
    };
  }
```

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

function Esempio() {
  // Dichiara una nuova variabile di stato, che chiameremo "contatore"
  const [contatore, setContatore] = useState(0);
```

# React Hooks

## Usare l'Hook State

```jsx
import React, { useState } from 'react';

function Esempio() {
  // Dichiara una nuova variable di stato, che chiameremo "contatore"
  const [contatore, setContatore] = useState(0);

  return (
    <div>
      <p>Hai cliccato {contatore} volte</p>
      <button onClick={() => setContatore(contatore + 1)}>
        Cliccami
      </button>
    </div>
  );
}
```

## variabili di stato multiple

```jsx
function EsempioConMoltiState() {
  // Dichiara le variabili di stato!
  const [eta, setEta] = useState(42);
  const [frutto, setFrutto] = useState('banana');
  const [todos, setTodos] = useState([{ testo: 'Impara gli Hooks' }]);
  // ...
}
```

## Esempio Classe Equivalente

```jsx
class Esempio extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      contatore: 0
    };
  }

  render() {
    return (
      <div>
        <p>Hai cliccato {this.state.contatore} volte</p>
        <button onClick={() => this.setState({ contatore: this.state.contatore + 1 })}>
          Cliccami
        </button>
      </div>
    );
  }
}
```


# useState()

**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.

# Leggere lo Stato

Quando vogliamo visualizzare l'attuale contatore in una classe, leggiamo `this.state.contatore`:

```
<p>Hai cliccato {this.state.contatore} volte</p>
```

In una funzione, possiamo usare direttamente `contatore`:

```
<p>Hai cliccato {contatore} volte</p>
```

## Aggiornare lo Stato

In una classe, dobbiamo chiamare `this.setState()` per aggiornare lo stato `contatore`:

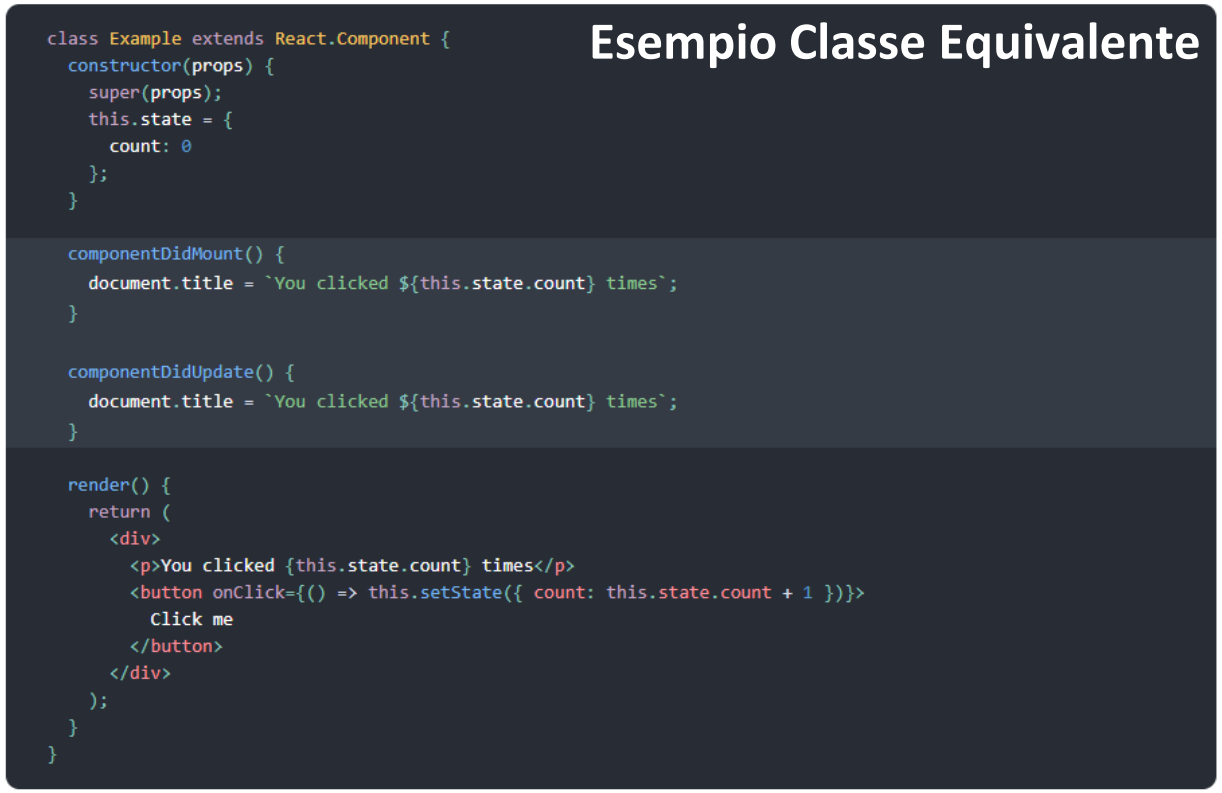
```
<button onClick={() => this.setState({ contatore: this.state.contatore + 1 })}>
  Cliccami
</button>
```

In una funzione, abbiamo `setContatore` e `contatore` come variabili quindi non abbiamo bisogno di `this`:

```
<button onClick={() => setContatore(contatore + 1)}>
  Cliccami
</button>
```

# React Hooks


**useEffect** equivale a **componentDidMount** e **componentDidUpdate** il return equivale al **componentWillUnmount**



# useEffect()

useEffect replica le funzionalità del metodo di lifeclycle più usate in un componente a classe:
**componentDidMount** – **componentDidUpdate** - **componentWillUnmount**
Accetta una funzione di callback come primo parametro e un array di dipendenze facoltativo come secondo parametro. Se ometterai completamente l'array di dipendenze, la funzione di callback verrà eseguita ad ogni render del componente, esattamente come farebbe il componentDidUpdate.
Se invece userai un array di dipendenze vuoto come secondo parametro, un array vuoto senza nessuna dipendenza, la funzione verrà chiamata una volta solamente, dopo il render iniziale, proprio come il componentDidMount.
Infine, mettere un return statement esplicito nella tua funzione di callback ti permette di fare pulizia, qualsiasi codice debba essere eseguito prima dell'unmount (smontaggio) del componente stesso andrà lì, così come se avessimo usato il componentWillUnmount per cancellare ad esempio un interval.

# React Hooks

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

const [state, setstate] = useState([initialState])
```

Array [ [], dispatchAction() ]

```
 // ComponentDidMount, ComponentDidUpdate
 // return -> ComponentWillUnmount
 // [array di dipendenze]
  useEffect(() => {
    effect
    return () => {
      cleanup
    }
  }, [input])
```

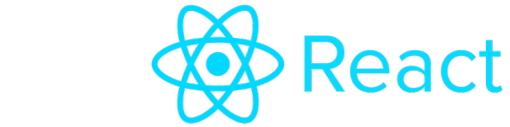
```
hooks > reacthooks > src > JS App.js > ...
import React, {useEffect, useState} from 'react';
import './App.css';

function App() {
  //const [state, setstate] = useState([initialState])
  const myState = useState([]);
  console.log(myState); //Array [ [], dispatchAction() ]

  // ComponentDidMount, ComponentDidUpdate
  // return -> ComponentWillUnmount
  useEffect(() => {
    effect
    return () => {
      cleanup
    }
  }, [input])

  return (
    <div className="App">
      <header className="App-header">
```

# React Hooks



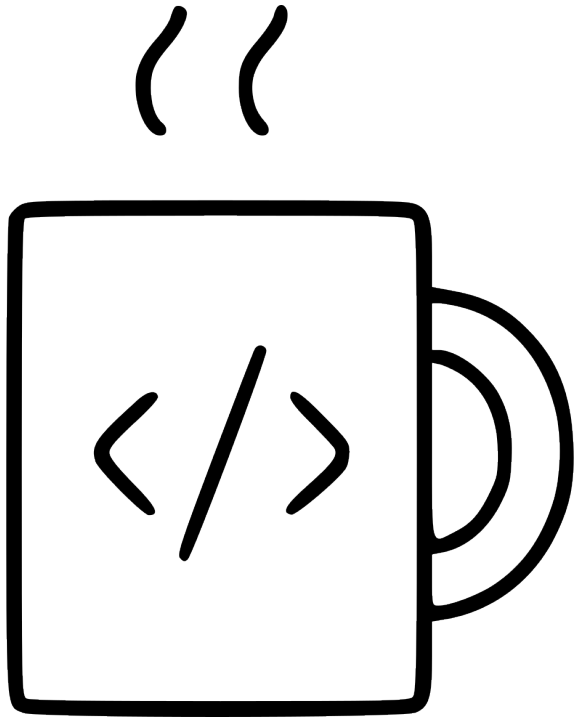

```jsx
import React, {useEffect, useState} from 'react';
import './App.css';

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users').then(response => response.json()).then(json => setUsers(json))
    return () => { }
  }, [])

  return (
    <div className="App">
      <header className="App-header">
        <ul>
          {users.map( user => <li key={user.id}>{user.name}</li> )}
        </ul>
      </header>
    </div>
  );
}

export default App;
```



# PAUSA

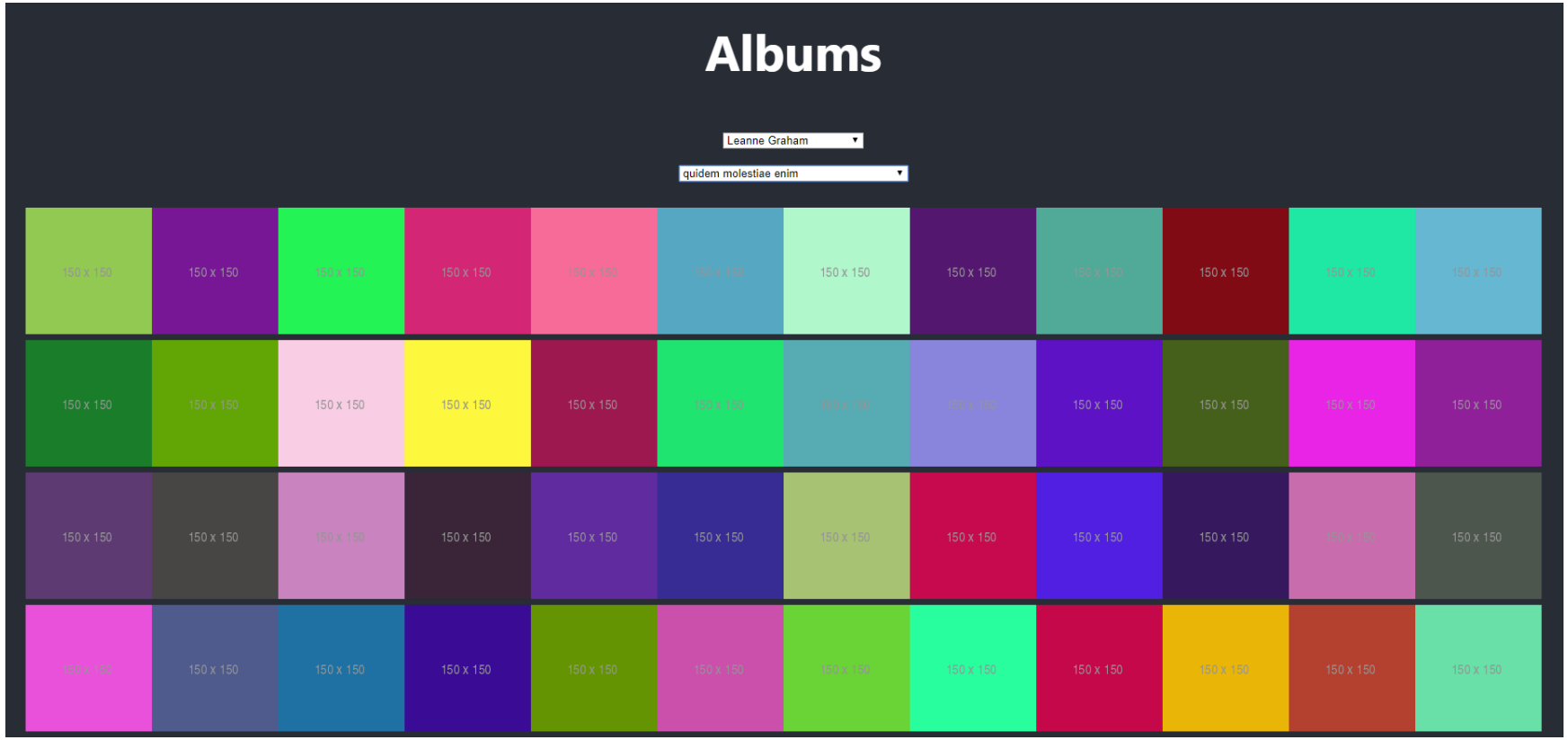
Ci vediamo alle ore 11.20

## useRef

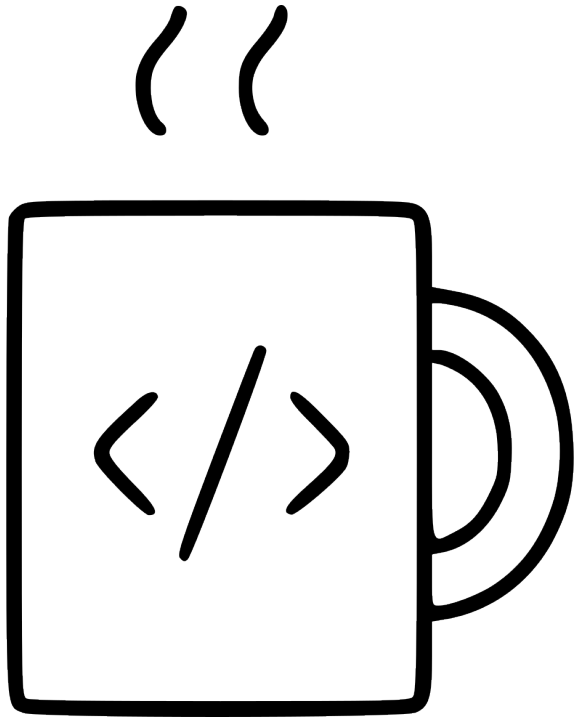
```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

inputEl.current.value

# React Hooks




Progetto reacthook



# PAUSA

Ci vediamo alle ore 14.00

challenge
NETWORK

*shaping the skills of tomorrow*

challengenetwork.it