

Umberto Emanuele

React – Giorno 2

Novembre 2023



Components

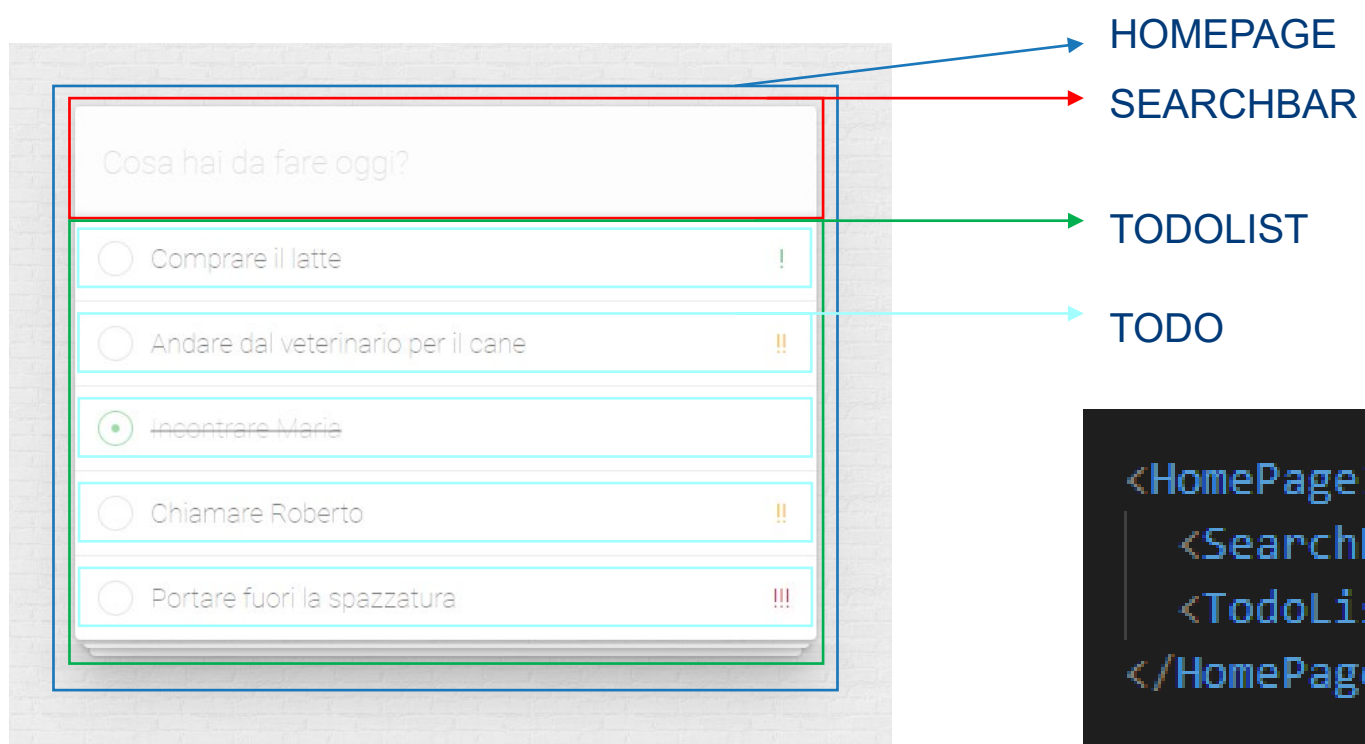
I Componenti ti permettono di suddividere la UI (User Interface) in parti indipendenti, riutilizzabili e di pensare ad ognuna di esse in modo isolato.

<https://it.reactjs.org/docs/components-and-props.html>

Un concetto fondamentale in REACT è la Composizione

La **composizione** è l'operazione di combinare funzioni semplici per creare funzioni complesse.

Idealmente una funzione deve svolgere una sola cosa, così è più facile riutilizzarla più volte all'interno del nostro codice.



Suddividere in componenti

```
<HomePage>  
  <SearchBar />  
  <TodoList />  
</HomePage>
```

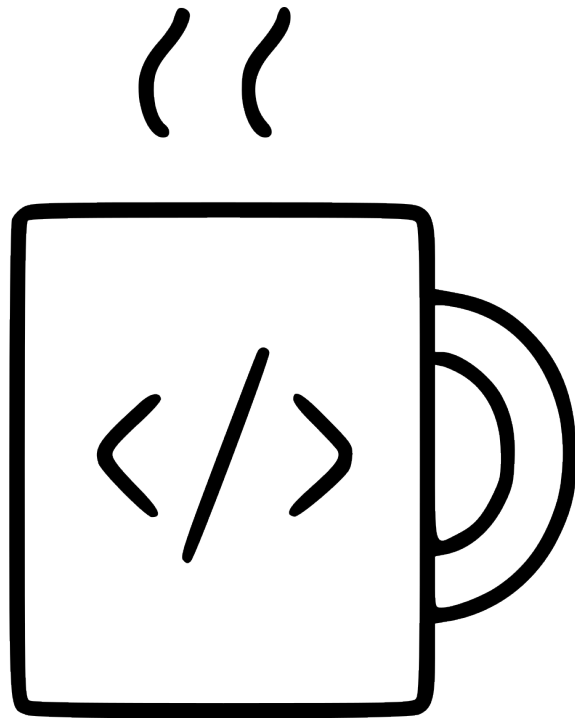
Creare un Component

- **Functional Components:** definire un componente tramite una funzione
- **Class Components:** definire un componente usando ES6
 - deve avere almeno un metodo *render()*
 - dovremmo inoltre utilizzare la keyword **extends** per indicare un vincolo
 - di ereditarietà fra il componente e la "classe" base **React.Component**

Differenza tra Functional Component e Class Component.

I functional component:

- Sono normalissime funzioni
- Sono semplici da riutilizzare e testare
- Vengono utilizzati principalmente per i componenti che non hanno una complicata logica interna e sono creati con lo scopo di strutturare l'interfaccia grafica dell'applicazione
- Hanno funzionalità limitate
- Non è possibile definire uno stato interno del componente
- Non è necessario utilizzare la keyword `this` all'interno di questi componenti
- Non sono presenti i cosiddetti "lifecycle hooks"



PAUSA

Ci vediamo alle ore 11.17

Il modo più semplice di definire un componente è quello di scrivere una funzione JavaScript:

```
function Ciao(props) {  
  return <h1>Ciao, {props.nome}</h1>;  
}
```

Puoi anche usare una classe ES6 per definire un componente:

```
class Ciao extends React.Component {  
  render() {  
    return <h1>Ciao, {this.props.nome}</h1>;  
  }  
}
```


Quando una **Class Component** non deve salvare degli **stati** si può utilizzare una **Functional Component**, altrimenti è preferibile utilizzare una classe.

```
src > component > JS titolo.js > Titolo > render
1  import React from 'react';
2
3  class Titolo extends React.Component {
4    render() {
5      //console.log(this.props);
6      return <h1 className = 'abc' style = {{backgroundColor: 'black'}}>{this.props.titolo}</h1>
7    }
8  }
9
10 export default Titolo;
```

Una **Class Component** che ha solo il metodo **render()** può essere riscritta come una **Functional Component**

```
src > component > JS titolo.js > ...
1  import React from 'react';
2
3  function Titolo(props){
4    return <h1 className = 'abc' style = {{backgroundColor: 'black'}}>{props.titolo}</h1>
5  }
6
7  export default Titolo;
```

Una **Functional Component** non ha la keyword **this** ma prende gli argomenti all'interno delle **props** passate come parametro.

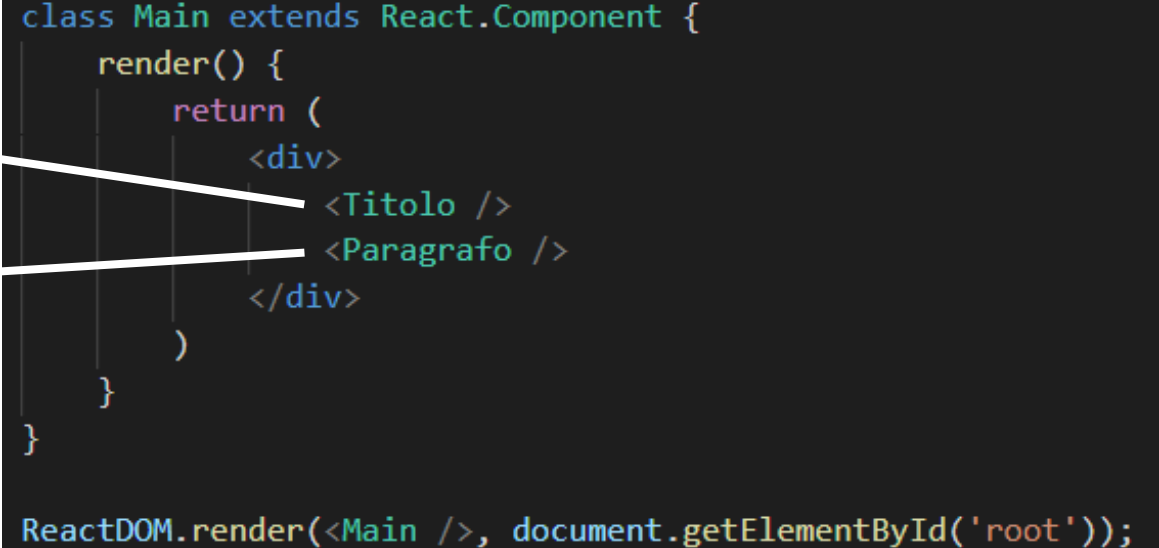
Come creare un Class Component in React

```
class Lista extends React.Component {  
  render(){  
    const elementi = ['Primo', 'Secondo', 'Terzo'];  
  
    return (  
      <div>  
        <h1>Lista</h1>  
        <ol>  
          {elementi.map((e, index) => <li key={index}>{e}</li> )}  
        </ol>  
      </div>  
    )  
  }  
}  
  
ReactDOM.render(<Lista />, document.getElementById('root'));
```

Riutilizzo di Component in React

```
class Titolo extends React.Component {  
  render() {  
    return <h1>Titolo</h1>  
  }  
}  
  
class Paragrafo extends React.Component {  
  render() {  
    return <p>Prova Paragrafo</p>  
  }  
}
```

```
class Main extends React.Component {  
  render() {  
    return (  
      <div>  
        <Titolo />  
        <Paragrafo />  
      </div>  
    )  
  }  
}  
  
ReactDOM.render(<Main />, document.getElementById('root'));
```



I componenti possono essere **statici** o **dinamici** e visualizzare i contenuti passati, tramite i **props** del componente.

Questo rende i componenti improvvisamente molto più utili poiché diventano generici e riutilizzabili.

Un componente può ricevere un numero illimitato di props e il loro valore può essere una stringa, un numero, un oggetto, un array... anche una funzione

L'oggetto **props** non dovrà mai essere modificato all'interno del nostro componente.

Le props, passate al nostro componente, devono essere considerate come delle costanti.

Possono essere usate all'interno del componente come **proprietà di sola lettura**

Ad esempio, il codice seguente renderizza il messaggio "Ciao, Sara" nella pagina:

```
function Ciao(props) {  
  return <h1>Ciao, {props.nome}</h1>;  
}  
  
const elemento = <Ciao nome="Sara" />;  
ReactDOM.render(  
  elemento,  
  document.getElementById('root')  
);
```


Ricapitoliamo cosa succede nell'esempio:

1. Richiamiamo `ReactDOM.render()` con l'elemento `<Ciao nome="Sara" />`.
2. React chiama a sua volta il componente `Ciao` con `{nome: 'Sara'}` passato in input come props.
3. Il nostro componente `Ciao` ritorna un elemento `<h1>Ciao, Sara</h1>` come risultato.
4. React DOM aggiorna efficientemente il DOM per far sì che contenga `<h1>Ciao, Sara</h1>`.

Per esempio, possiamo creare un componente `App` che renderizza `Ciao` tante volte:

```
function Ciao(props) {  
  return <h1>Ciao, {props.nome}</h1>;  
}  
  
function App() {  
  return (  
    <div>  
      <Ciao nome="Sara" />  
      <Ciao nome="Cahal" />  
      <Ciao nome="Edite" />  
    </div>  
  );  
}  
  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
);
```

Estrarre Componenti

Non aver paura di suddividere i componenti in componenti più piccoli.

Ad esempio, considera questo componente `Commento`:

```
function Commento(props) {  
  return (  
    <div className="Commento">  
      <div className="InfoUtente">  
        <img className="Avatar"  
          src={props.autore.avatarUrl}  
          alt={props.autore.nome}/>  
        <div className="InfoUtente-nome">  
          {props.autore.nome}  
        </div>  
      </div>  
      <div className="Commento-testo">  
        {props.testo}  
      </div>  
      <div className="Commento-data">  
        {formatDate(props.data)}  
      </div>  
    </div>  
  );  
}
```

Un componente scritto in questo modo, con **codice molto annidato**, è difficile da modificare.

Per lo stesso motivo, **non si possono riutilizzare** con facilità parti dello stesso.

Procediamo quindi ad **estrarre** qualche componente.

```
function Avatar(props) {  
  return (  
    <img className="Avatar"  
      src={props.utente.avatarUrl}  
      alt={props.utente.nome}  
    />  
  );  
}
```

```
function Commento(props) {  
  return (  
    <div className="Commento">  
      <div className="InfoUtente">  
        <Avatar utente={props.autore} />  
        <div className="InfoUtente-nome">  
          {props.autore.nome}  
        </div>  
      </div>  
      <div className="Commento-testo">  
        {props.testo}  
      </div>  
      <div className="Commento-data">  
        {formatDate(props.data)}  
      </div>  
    </div>  
  );  
}
```

Il componente **Avatar** non ha bisogno di sapere che viene renderizzato all'interno di un Commento.

Può essere riutilizzato in più parti della nostra App.


Estrarre componenti può sembrare un'attività pesante ma avere una tavolozza di componenti riutilizzabili ripaga molto bene nelle applicazioni più complesse.

Una buona regola da tenere a mente è che se una parte della tua UI viene usata diverse volte (Bottone, Pannello, Avatar) o se è abbastanza complessa di per sé (App, StoriaFeed, Commento), allora questi componenti sono buoni candidati ad essere riutilizzabili.

Riutilizzo di Component in React

```
class Titolo extends React.Component {  
  render() {  
    return <h1>Titolo</h1>  
  }  
}  
  
class Paragrafo extends React.Component {  
  render() {  
    return <p>Prova Paragrafo</p>  
  }  
}
```

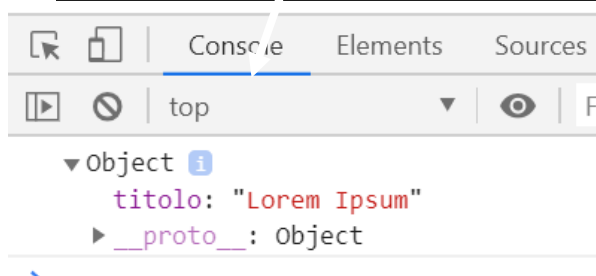
```
class Main extends React.Component {  
  render() {  
    return (  
      <div>  
        <Titolo />  
        <Paragrafo />  
      </div>  
    )  
  }  
}  
  
ReactDOM.render(<Main />, document.getElementById('root'));
```



Utilizzo delle **props** per rendere i **componenti** indipendenti.

(Le parentesi graffe dicono semplicemente a JSX che abbiamo del JavaScript che vogliamo valutare.)

```
class Titolo extends React.Component {
  render() {
    console.log(this.props);
    return <h1>{this.props.titolo}</h1>
  }
}
```



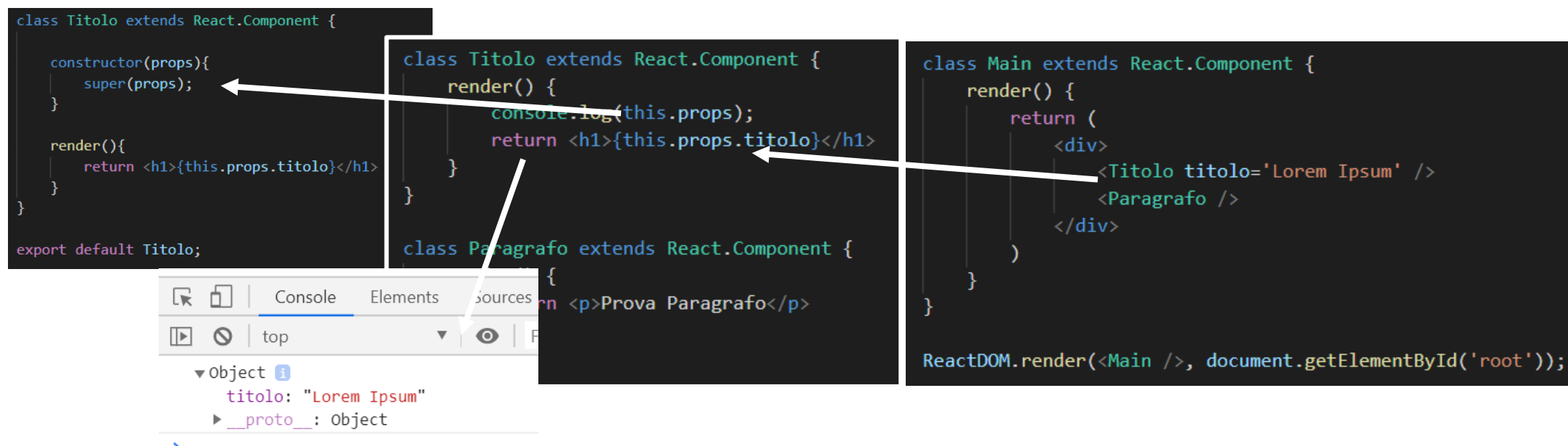
```
class Main extends React.Component {
  render() {
    return (
      <div>
        <Titolo titolo='Lorem Ipsum' />
        <Paragrafo />
      </div>
    )
  }
}

ReactDOM.render(<Main />, document.getElementById('root'));
```

La parola chiave **this** fa riferimento alla classe del componente, e **props** è un oggetto che contiene tutti i valori passati. Nel nostro caso, l'oggetto props contiene solo una voce, ma è possibile aggiungerne altri.

Utilizzo delle **props** per rendere i **componenti** indipendenti.

(Le parentesi graffe dicono semplicemente a JSX che abbiamo del JavaScript che vogliamo valutare.)



The image shows a code editor with three files: `Titolo`, `Paragrafo`, and `Main`. The `Titolo` component has a `constructor(props)` and a `render()` method that returns `<h1>{this.props.titolo}</h1>`. The `Paragrafo` component has a `render()` method that returns `<p>Prova Paragrafo</p>`. The `Main` component has a `render()` method that returns a `<div>` containing `<Titolo titolo='Lorem Ipsum' />` and `<Paragrafo />`. A browser console window is open, showing the props object passed to the `Titolo` component: `{ titolo: 'Lorem Ipsum' }`. Arrows indicate the flow of props from the `Main` component to the `Titolo` component.

```
class Titolo extends React.Component {
  constructor(props){
    super(props);
  }
  render(){
    return <h1>{this.props.titolo}</h1>
  }
}
export default Titolo;
```

```
class Titolo extends React.Component {
  render() {
    console.log(this.props);
    return <h1>{this.props.titolo}</h1>
  }
}
```

```
class Paragrafo extends React.Component {
  render() {
    return <p>Prova Paragrafo</p>
  }
}
```

```
class Main extends React.Component {
  render() {
    return (
      <div>
        <Titolo titolo='Lorem Ipsum' />
        <Paragrafo />
      </div>
    )
  }
}
```

```
ReactDOM.render(<Main />, document.getElementById('root'));
```

Console log: `{ titolo: 'Lorem Ipsum' }`

La parola chiave **this** fa riferimento alla classe del componente, e **props** è un oggetto che contiene tutti i valori passati. Nel nostro caso, l'oggetto **props** contiene solo una voce, ma è possibile aggiungerne altri.

Organizzare componenti su file.

```
> node_modules
> public
✓ src
  ✓ component
    JS main.js
    JS paragrafo.js
    JS titolo.js
    JS index.js
  .gitignore
  debug.log
  package-lock.json
  package.json
  README.md
```

```
src > component > JS titolo.js > ...
1  import React from 'react';
2
3  class Titolo extends React.Component {
4    render() {
5      console.log(this.props);
6      return <h1>{this.props.titolo}</h1>
7    }
8  }
9
10 export default Titolo;
```

```
src > component > JS paragrafo.js > ...
1  import React from 'react';
2
3  class Paragrafo extends React.Component {
4    render() {
5      return <p>Prova Paragrafo</p>
6    }
7  }
8
9  export default Paragrafo;
```

```
src > component > JS main.js > ...
1  import React from 'react';
2  import Titolo from './titolo';
3  import Paragrafo from './paragrafo';
4
5  class Main extends React.Component {
6    render() {
7      return (
8        <div>
9          <Titolo titolo='Lorem Ipsum' />
10         <Paragrafo />
11        </div>
12      )
13    }
14  }
15
16 export default Main;
```

```
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import Main from './component/main';
4
5
6  ReactDOM.render(<Main />, document.getElementById('root'));
```

Applicare CSS alle applicazione REACT

Applicare CSS alle applicazione REACT



```
> node_modules
> public
▼ src
  > component
  # index.css
  JS index.js
  .gitignore
  debug.log
  package-lock.json
  package.json
  README.md
```

```
src > component > JS titolo.js > ...
1  import React from 'react';
2
3  class Titolo extends React.Component {
4    render() {
5      console.log(this.props);
6      return <h1 className='abc'>{this.props.titolo}</h1>
7    }
8  }
9
10 export default Titolo;
```

```
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import Main from './component/main';
4  import './index.css';
5
6  ReactDOM.render(<Main />, document.getElementById('root'));
```

```
src > # index.css > .abc
1  .abc {
2    color: red;
3  }
```

In HTML le classi CSS si applicano ai tag della pagina attraverso la keyword "**class**", In React per associare una classe CSS ad un tag è utilizzato "**className**".

Uno dei motivi per cui è stato scelto di differenziare è il fatto che per le applicazioni React era già prevista la keyword class.

Applicare CSS alle applicazione REACT



```
src > component > JS titolo.js > ...
1  import React from 'react';
2
3  class Titolo extends React.Component {
4    render() {
5      console.log(this.props);
6      return <h1 className = 'abc' style = {{backgroundColor:'black'}}>{this.props.titolo}</h1>
7    }
8  }
9
10 export default Titolo;
```

Applicare CSS alle applicazione REACT



Materialize

A modern responsive front-end framework based on Material Design

GET STARTED

UPGRADE FROM 0.100.2

Release: 1.0.0

<https://materializecss.com/>

MUI

MUI is a lightweight CSS framework that follows Google's Material Design guidelines

<https://www.muicss.com/>



React Materialize

Welcome to the React Materialize docs.

<http://react-materialize.github.io/>

Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

Get started

Download

Currently v4.3.1



<https://getbootstrap.com/>

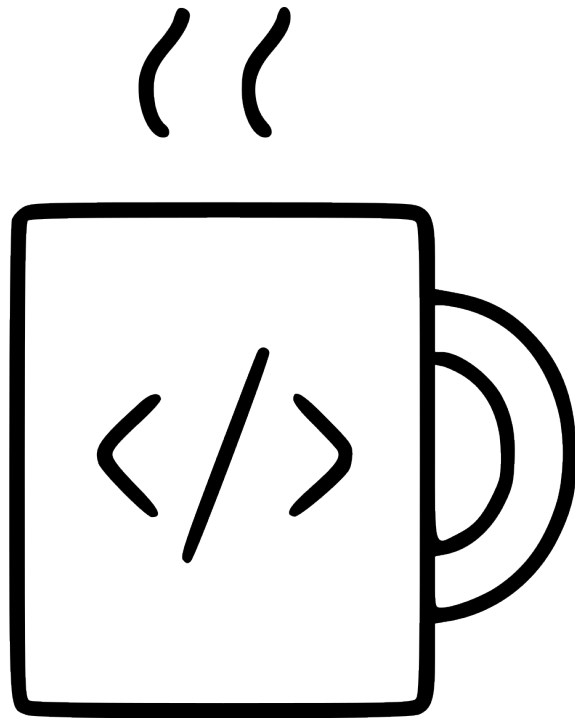


MATERIAL-UI

React components for faster and easier web development. Build your own design system, or start with Material Design.

GET STARTED

<https://material-ui.com/>



PAUSA

Ci vediamo alle ore 14.00



shaping the skills of tomorrow

challengenetwork.it

