REACT

Sommario

[1.CONCETTI CHIAVE 3](#_Toc108595583)

[1.1. Introduzione a JSX 3](#_Toc108595584)

[1.1.1. Separazione delle responsabilità anziché separazione delle tecnologie 3](#_Toc108595585)

[1.1.2.Incorporare espressioni in JSX 4](#_Toc108595586)

[1.1.3.JSX è un espressione (si possono utilizzare costrutti come if e for) 4](#_Toc108595587)

[1.1.4.Specificare gli attributi con JSX(virgolette per le stringhe e parentesi graffe per le espressioni) 4](#_Toc108595588)

[1.1.5.Specificare figli in JSX 4](#_Toc108595589)

[1.1.6.JSX previene attacchi XSS 4](#_Toc108595590)

[1.1.7.JSX rappresenta oggetti 4](#_Toc108595591)

[1.2.Renderizzare elementi 4](#_Toc108595592)

[1.2.1.Renderizzare un elemento nel DOM(nodo DOM root) 4](#_Toc108595593)

[1.2.3.Aggiornare un elemento renderizzato 4](#_Toc108595594)

[1.3.Componenti e Props 5](#_Toc108595595)

[1.3.1.Funzioni e classi componente 5](#_Toc108595596)

[1.3.2.Renderizzare un componente 5](#_Toc108595597)

[1.3.3.Comporre componenti(react fragment) 5](#_Toc108595598)

[1.3.4. Estrarre componenti 5](#_Toc108595599)

[1.3.5.Le Props sono in sola lettura 5](#_Toc108595600)

[1.4.State e lifecycle 6](#_Toc108595601)

[1.4.1.Localstate di una classe 6](#_Toc108595602)

[1.4.2.Aggiungere metodi lifecycle ad una classe 7](#_Toc108595603)

[1.4.3.Set state 8](#_Toc108595604)

[1.4.4.Update state objects 8](#_Toc108595605)

[1.5.Gestione degli eventi 9](#_Toc108595606)

[1.5.1.Prevent default on submit 9](#_Toc108595607)

[1.5.2.Interruttore che cambia il suo stato in risposta all’event click 9](#_Toc108595608)

[1.6.Renderizzazione condizionale 10](#_Toc108595609)

[1.6.1.Variabili elemento 10](#_Toc108595610)

[1.6.2.Condizione if inline e operatore logico && 10](#_Toc108595611)

[1.6.3.Condizione if else con operatore condizionale(operatore ternario) 10](#_Toc108595612)

[1.7.Liste e Chiavi 11](#_Toc108595613)

[1.8.Forms 12](#_Toc108595614)

[1.9.Spostare lo stato 13](#_Toc108595615)

[1.9.1.Spostare lo stato “in alto” 13](#_Toc108595616)

[1.10.Composizione vs ereditarietà 14](#_Toc108595617)

[1.10.1.Contenimento 14](#_Toc108595618)

[1.10.2.Specializzazioni 14](#_Toc108595619)

[2.GUIDE AVANZATE 15](#_Toc108595620)

[2.1.Accessibilità 15](#_Toc108595621)

[2.2.Code splitting 15](#_Toc108595622)

[2.3.Context 16](#_Toc108595623)

[2.3.1.Quando usare Context 16](#_Toc108595624)

[2.3.2.Il Provider Pattern 16](#_Toc108595625)

[3.API DI RIFERIMENTO 17](#_Toc108595626)

[4.HOOKS 18](#_Toc108595627)

[4.1.Introduzione agli hooks 18](#_Toc108595628)

[5.TESTING 19](#_Toc108595629)

[6.PRINCIPI DI DESIGN 20](#_Toc108595630)

[RIFERIMENTI 21](#_Toc108595631)

# 1.CONCETTI CHIAVE

## Introduzione a JSX

### Separazione delle responsabilità anziché separazione delle tecnologie

[Rethinking best practices](https://www.youtube.com/watch?v=x7cQ3mrcKaY)

#### Build component, not templates

#### Template encourages a low separation of concerns

#### “View model” tightly couples to display logic

#### Template separate tecnologies, not concerns

#### Symptoms that your front-end technology is underpowered(e.g. inventing a lot of new concepts that already exists in JavaScript)

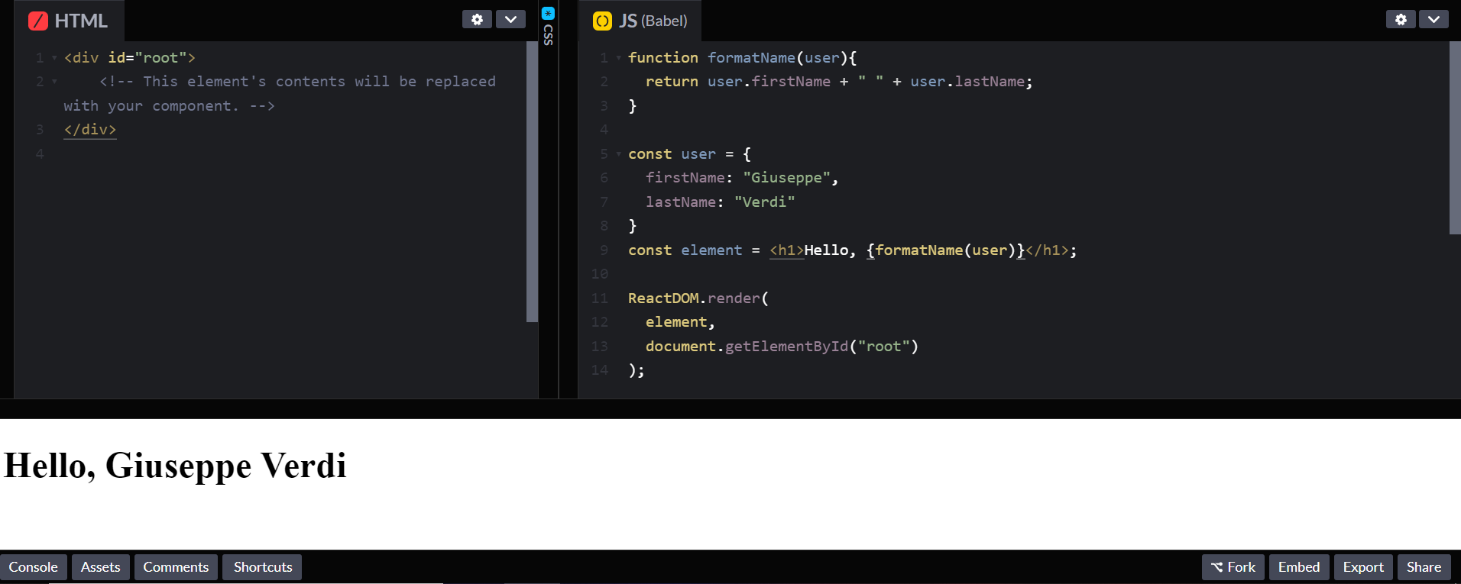
#### Components are reusable

#### Components are unit testable

#### Components are composable

#### The virtual DOM let us do fun things

### 1.1.2.Incorporare espressioni in JSX



### 1.1.3.JSX è un espressione (si possono utilizzare costrutti come if e for)

### 1.1.4.Specificare gli attributi con JSX(virgolette per le stringhe e parentesi graffe per le espressioni)

### 1.1.5.Specificare figli in JSX

### 1.1.6.JSX previene attacchi XSS

### 1.1.7.JSX rappresenta oggetti

## 1.2.Renderizzare elementi

### 1.2.1.Renderizzare un elemento nel DOM(nodo DOM root)

### 1.2.3.Aggiornare un elemento renderizzato

Gli elementi React sono **immutabili**. Con la conoscenza che abbiamo, l’unico modo di aggiornare la UI è quello di creare un nuovo elemento e di passarlo a **ReactDOM.render().**

****

## 1.3.Componenti e Props

I **componenti** ti permettono di suddividere la UI in parti indipendenti riutilizzabili e di pensare ad ognuna di esse in modo isolato.

Concettualmente, i **componenti sono come funzioni JavaScript**: accettano in input **dati arbitrari**(sotto il nome di “**props**”) e ritornano elementi React che descrivono cosa dovrebbe apparire sullo schermo.

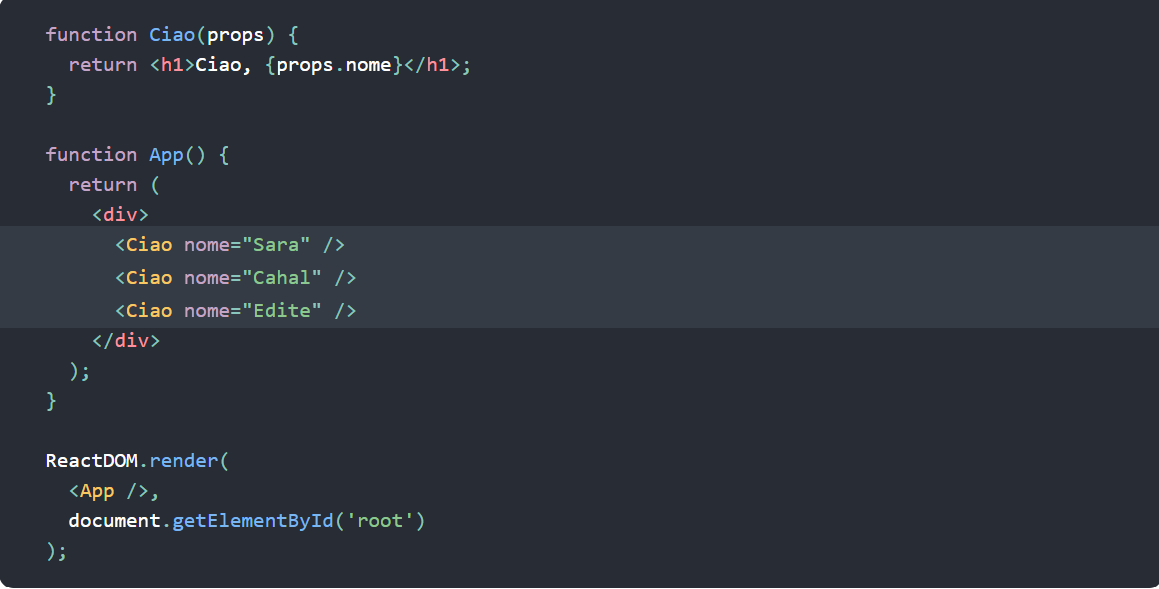
### 1.3.1.Funzioni e classi componente

### 1.3.2.Renderizzare un componente



### 1.3.3.Comporre componenti(react fragment)

Normalmente, le nuove applicazioni React hanno un singolo componente chiamato App nel livello più alto che racchiude tutti gli altri componenti. Deve stare tutto dentro un div senno react te lo da come errore.



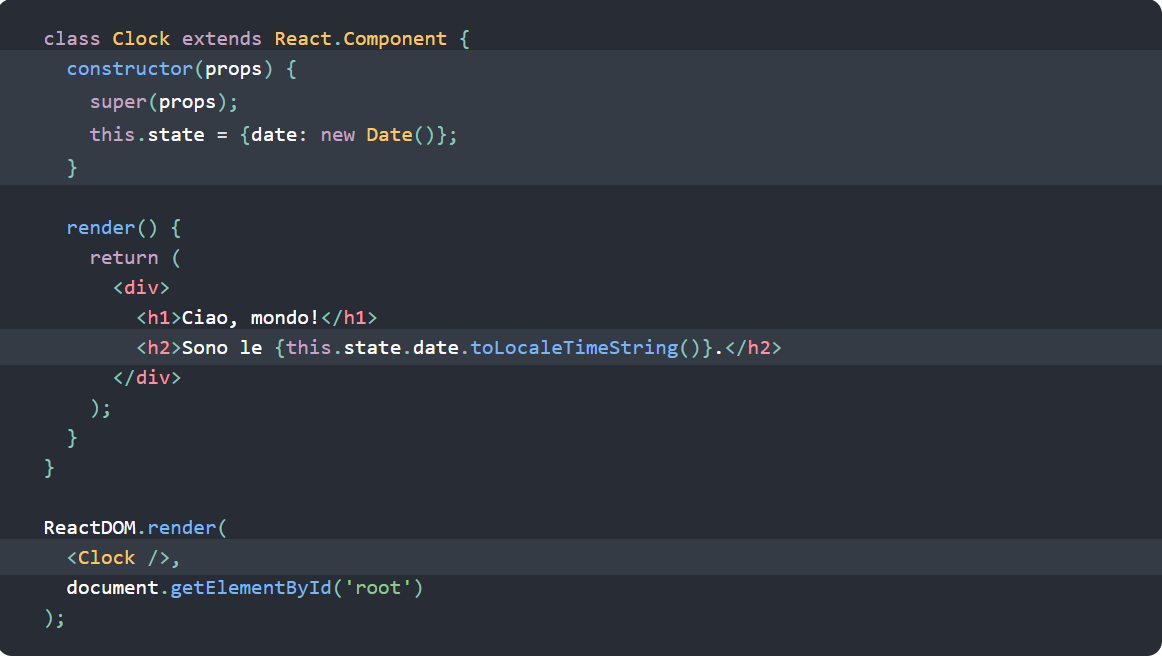
### 1.3.4. Estrarre componenti

### 1.3.5.Le Props sono in sola lettura

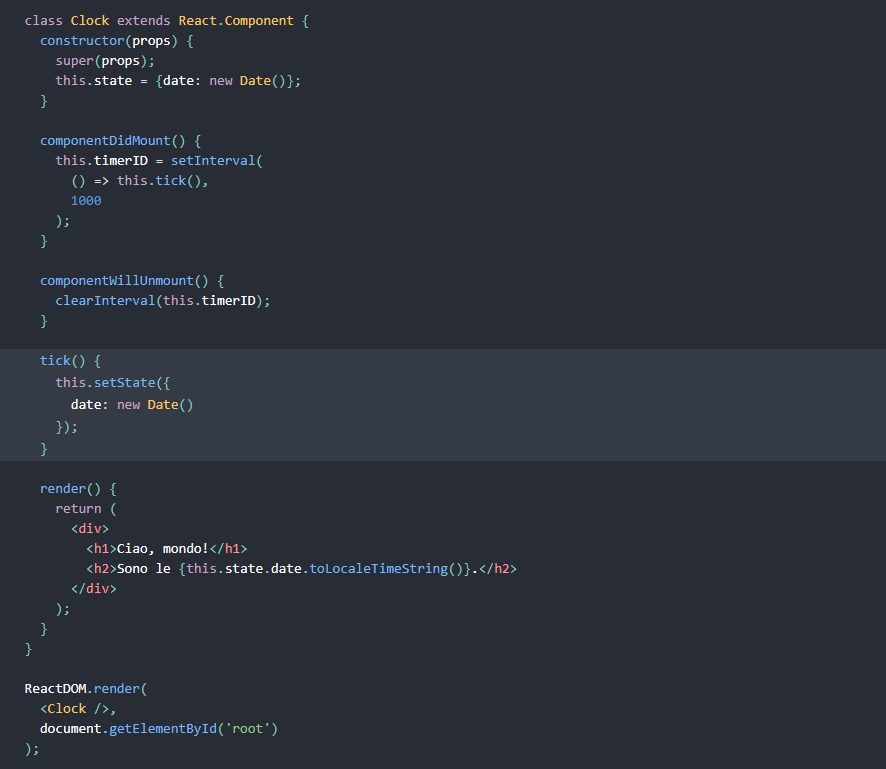
## 1.4.State e lifecycle



### 1.4.1.Localstate di una classe



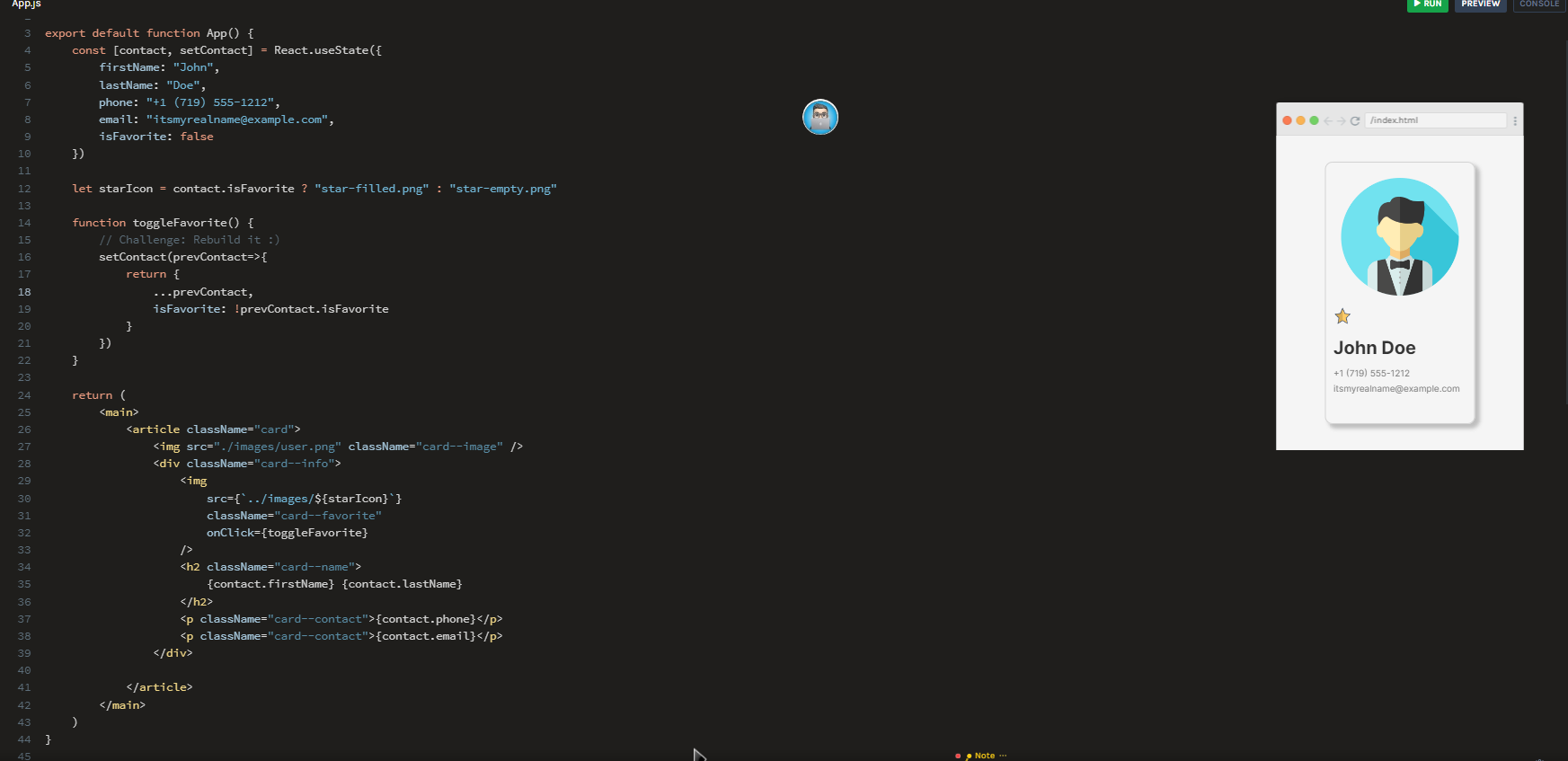
### 1.4.2.Aggiungere metodi lifecycle ad una classe



### 1.4.3.Set state

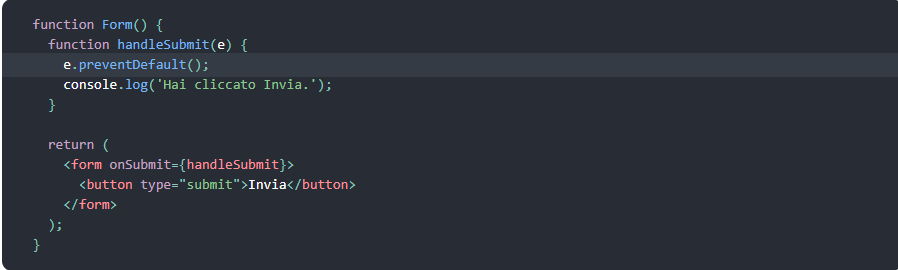


### 1.4.4.Update state objects

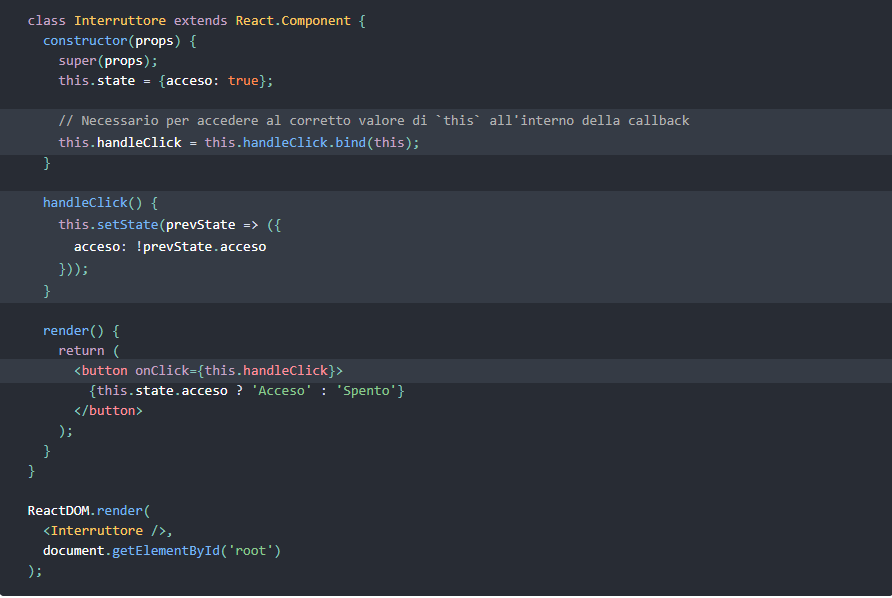


## 1.5.Gestione degli eventi

### 1.5.1.Prevent default on submit

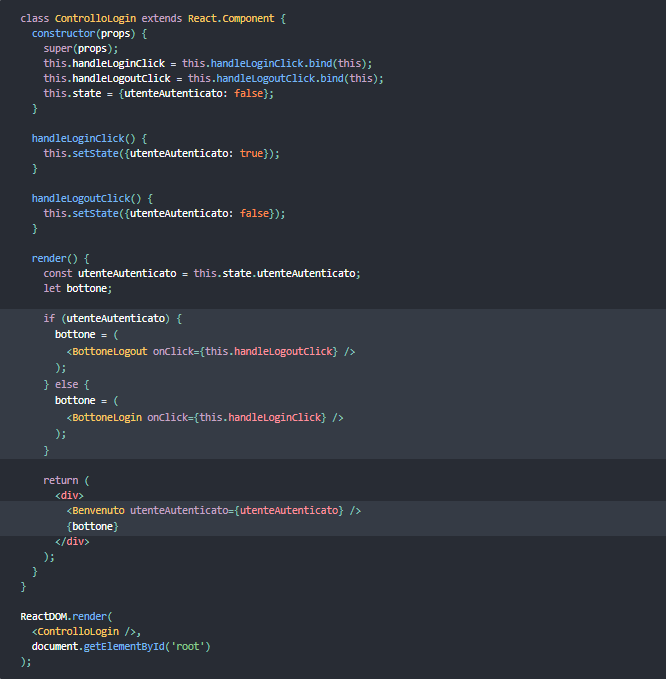


### 1.5.2.Interruttore che cambia il suo stato in risposta all’event click

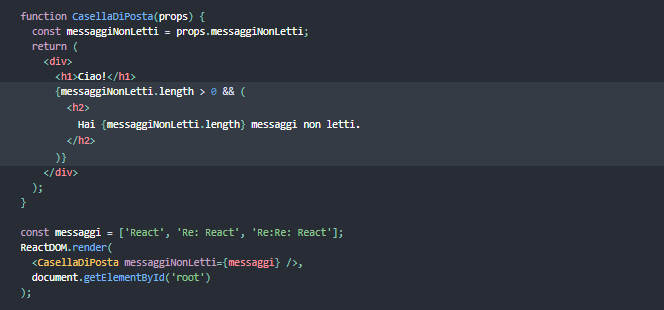


## 1.6.Renderizzazione condizionale

### 1.6.1.Variabili elemento



### 1.6.2.Condizione if inline e operatore logico &&

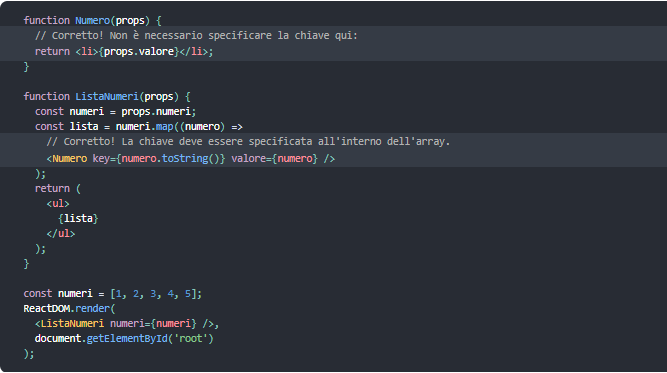


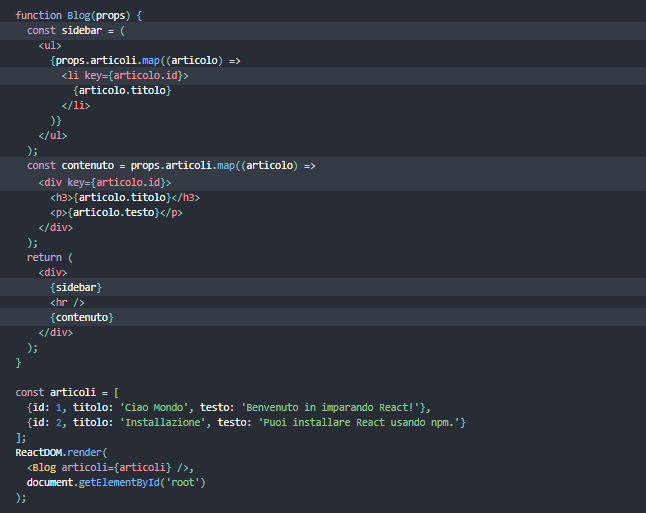
Funziona perché in **javascript , true && espressione** si risolve sempre in **espressione**, mentre **false && espressione** si risolve sempre in **false**.

Se la condizione è **true**, l’elemento dopo && verrà renderizzato, se invece è **false** react lo ignorerà.

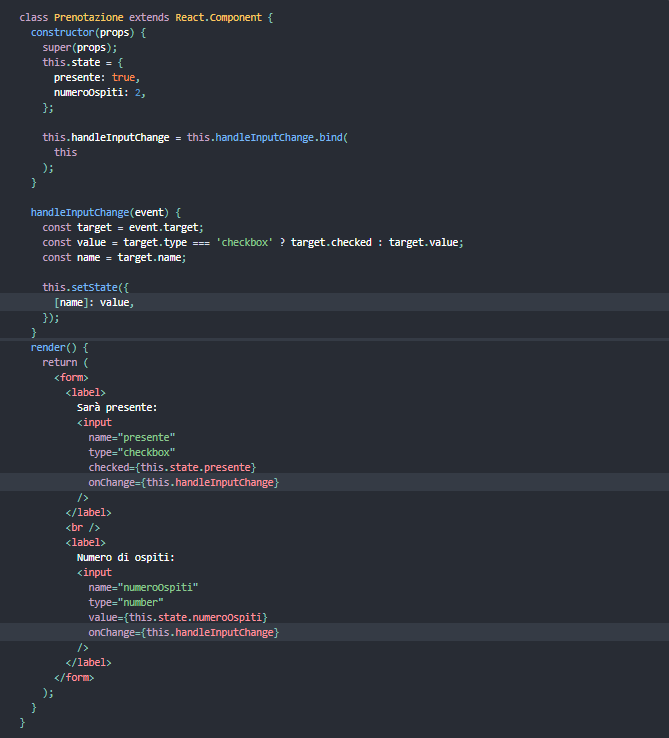
### 1.6.3.Condizione if else con operatore condizionale(operatore ternario)

## 1.7.Liste e Chiavi





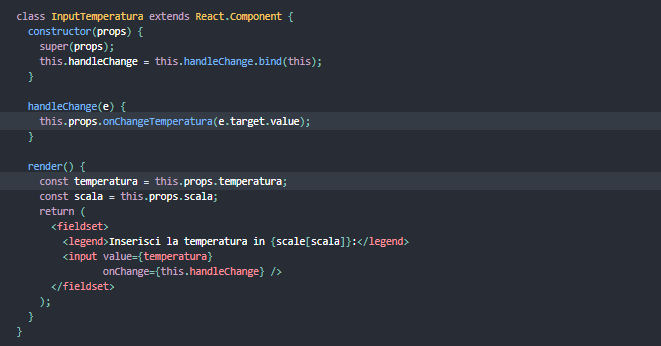
## 1.8.Forms

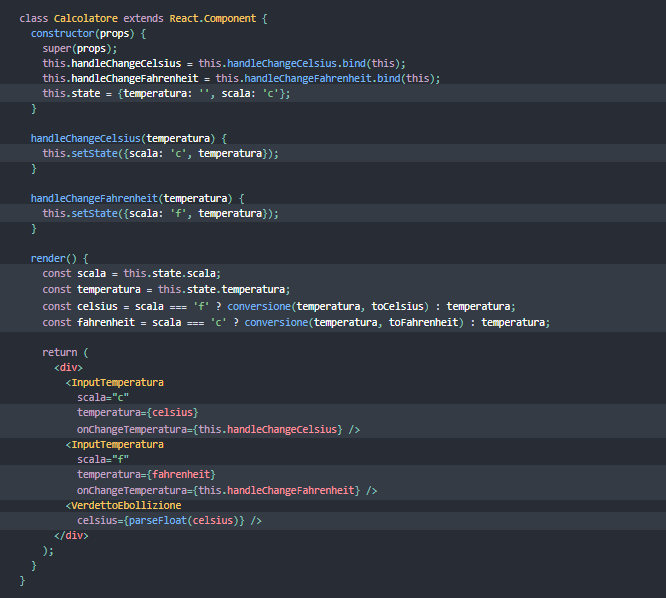


## 1.9.Spostare lo stato

### 1.9.1.Spostare lo stato “in alto”

In React, la condivisione dello stato si ottiene spostandolo verso il più vicino antenato comune dei componenti che ne hanno bisogno. Questo processo viene detto “spostare lo stato verso l’alto” (lifting state up).





## 1.10.Composizione vs ereditarietà

React ha un potente modello di composizione**, raccomandiamo che lo si** **usi in alternativa all’ereditarietà** per riutilizzare codice tra componenti.

### 1.10.1.Contenimento

Esistono componenti che si comportano da contenitori per altri componenti, non possono quindi sapere a priori quali componenti avranno come figli.

### 1.10.2.Specializzazioni

A volte pensiamo ai componenti come **specializzazioni** di altri componenti, ad esempio potremmo dire che FinestraBenvenuto è una specializzazione di Finestra.

# 2.GUIDE AVANZATE

## 2.1.Accessibilità

## 2.2.Code splitting

## 2.3.Context

**Context** provvede un modo per passare dati attraverso il componente albero senza dover passare le props sotto manualmente ad ogni livello.

In una tipica applicazione React, i dati sono passati dal padre al figlio attriverso le props, ma un simile uso può essere scomodo per certi tipi di proprietà (e.g. preferenza locale, tema UI) che sono richiesti da molti componenti dentro un applicazione. Il **context** provvede un modo **per condividere valori come questi tra componenti** senza dover esplicitamente passare una prop attraverso ogni livello dell’albero.

### 2.3.1.Quando usare Context

**Context** è designato per condividere dati che possono essere considerati **“globali”** per un albero dei componenti di React, come l’utente correntemente autenticato, il tema o il linguaggio preferito.

**Context** è usato principalmente quando alcuni dati necessitano di essere **accessibili da molti componenti a differenti livelli**.

### 2.3.2.Il Provider Pattern

**Context** fa uso del pattern **Provider**.

Con il pattern Provider possiamo rendere i dati disponibili a molti componenti, piuttosto che passare i dati ad ogni strato tramite le props, noi possiamo incapsulare tutti i componenti in un **Provider**. Un **Provider** è un componente di ordine superiore che in React è a noi provvisto dall’oggetto Context. Possiamo creare un oggetto Context usando il metodo di React **createContext.**

**Il Provider** riceve un **valore prop,** che contiene i dati che vogliamo passare. Tutti i componenti che sono incapsulati dentro questo provider hanno accesso al **valore prop**(quindi a tutti questi dati).



# 3.API DI RIFERIMENTO

# 4.HOOKS

## 4.1.Introduzione agli hooks

Gli **hooks** ti permettono di utilizzare **state** ed altre funzioni React senza dover scrivere una classe.

Gli **hooks** sono funzioni che ti permettono di **“ancorarti”** all’interno delle funzioni React state e lifecycle da componenti funzione. Gli **hooks** non funzionano all’interno delle classi, , ti permettono di usare React senza classi s

# 5.TESTING

# 6.PRINCIPI DI DESIGN

# RIFERIMENTI

[1] [Build a react info site](https://scrimba.com/playlist/pKNqYAZ)

[2] [Build an AirBnb Experience Clone](https://scrimba.com/playlist/pqQgrcN)

[3] [Build a meme generator](https://scrimba.com/playlist/prXJpCQ)

[4] [Build a notes app and Tenzies game](https://scrimba.com/playlist/pYkgpAP)

[5] [Documentazione ufficiale di react](https://it.reactjs.org/docs/hello-world.html)