REACT

Sommario

[1.CONCETTI CHIAVE 4](#_Toc116061666)

[1.1. Introduzione a JSX 4](#_Toc116061667)

[1.1.1. Separazione delle responsabilità anziché separazione delle tecnologie 4](#_Toc116061668)

[1.1.2.Incorporare espressioni in JSX 5](#_Toc116061669)

[1.1.3.JSX è un espressione (si possono utilizzare costrutti come if e for) 5](#_Toc116061670)

[1.1.4.Specificare gli attributi con JSX(virgolette per le stringhe e parentesi graffe per le espressioni) 5](#_Toc116061671)

[1.1.5.Specificare figli in JSX 5](#_Toc116061672)

[1.1.6.JSX previene attacchi XSS 5](#_Toc116061673)

[1.1.7.JSX rappresenta oggetti 5](#_Toc116061674)

[1.2.Renderizzare elementi 6](#_Toc116061675)

[1.2.1.Renderizzare un elemento nel DOM(nodo DOM root) 6](#_Toc116061676)

[1.2.2.Aggiornare un elemento renderizzato 6](#_Toc116061677)

[1.3.Componenti e Props 7](#_Toc116061678)

[1.3.1.Funzioni e classi componente 7](#_Toc116061679)

[1.3.2.Renderizzare un componente 8](#_Toc116061680)

[1.3.3.Comporre componenti(React fragment) 8](#_Toc116061681)

[1.3.4. Estrarre componenti 9](#_Toc116061682)

[1.3.5.Le Props sono in sola lettura 10](#_Toc116061683)

[1.3.6.Esempio su composizione di componenti 10](#_Toc116061684)

[1.4.State e lifecycle 14](#_Toc116061685)

[1.4.1.Localstate di una classe 14](#_Toc116061686)

[1.4.2.Aggiungere metodi lifecycle ad una classe 15](#_Toc116061687)

[1.5.Gestione degli eventi 16](#_Toc116061688)

[1.5.1.Prevent default on submit 16](#_Toc116061689)

[1.5.2.Interruttore che cambia il suo stato in risposta all’event click 16](#_Toc116061690)

[1.6.Renderizzazione condizionale 17](#_Toc116061691)

[1.6.1.Variabili elemento 17](#_Toc116061692)

[1.6.2.Condizione if inline e operatore logico && 17](#_Toc116061693)

[1.6.3.Condizione if else con operatore condizionale(operatore ternario) 17](#_Toc116061694)

[1.7.Liste e Chiavi 18](#_Toc116061695)

[1.8.Forms 19](#_Toc116061696)

[1.9.Spostare lo stato 20](#_Toc116061697)

[1.9.1.Spostare lo stato “in alto” 20](#_Toc116061698)

[1.10.Composizione vs ereditarietà 21](#_Toc116061699)

[1.10.1.Contenimento 21](#_Toc116061700)

[1.10.2.Specializzazioni 21](#_Toc116061701)

[2.GUIDE AVANZATE 22](#_Toc116061702)

[2.1.Context 22](#_Toc116061703)

[2.1.1.Quando usare Context 22](#_Toc116061704)

[2.1.2.Il Provider Pattern 22](#_Toc116061705)

[3.API DI RIFERIMENTO 23](#_Toc116061706)

[3.1.React 23](#_Toc116061707)

[3.2.ReactDOM 23](#_Toc116061708)

[3.3.ReactDOMClient 23](#_Toc116061709)

[3.4.ReactDOMServer 23](#_Toc116061710)

[3.5.Elementi DOM 23](#_Toc116061711)

[3.6.SyntheticEvent 23](#_Toc116061712)

[3.7.Test Utilities 23](#_Toc116061713)

[3.8.Test Renderer 23](#_Toc116061714)

[4.HOOKS 24](#_Toc116061715)

[4.1.Introduzione agli hooks 24](#_Toc116061716)

[4.2.Panoramica sugli Hooks 24](#_Toc116061717)

[4.3.Usare l’Hook State 24](#_Toc116061718)

[4.4.Usare l’Hook Effect 24](#_Toc116061719)

[4.5.Regole degli Hooks 24](#_Toc116061720)

[4.6.Hooks Personalizzati 24](#_Toc116061721)

[4.7.API di Riferimento degli Hooks 25](#_Toc116061722)

[4.7.1.useState 25](#_Toc116061723)

[4.7.2.useEffect 26](#_Toc116061724)

[4.7.3.useContext 26](#_Toc116061725)

[4.7.4.useReducer 26](#_Toc116061726)

[4.7.5.useCallback 26](#_Toc116061727)

[4.7.6.useMemo 26](#_Toc116061728)

[4.7.7.useRef 26](#_Toc116061729)

[4.7.8.useImperativeHandle 26](#_Toc116061730)

[4.7.9.useTransition 26](#_Toc116061731)

[4.7.10.useId 26](#_Toc116061732)

[4.7.11.useSyncExternalStore 26](#_Toc116061733)

[4.7.12.useInsertionEffect 26](#_Toc116061734)

[5.TESTING 27](#_Toc116061735)

[5.1.Testing Overview 27](#_Toc116061736)

[5.2.Testing Recipes 27](#_Toc116061737)

[5.3.Testing Environments 27](#_Toc116061738)

[6.PRINCIPI DI DESIGN 28](#_Toc116061739)

[RIFERIMENTI 29](#_Toc116061740)

# 1.CONCETTI CHIAVE

**CREACT A REACT APP**



## 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)

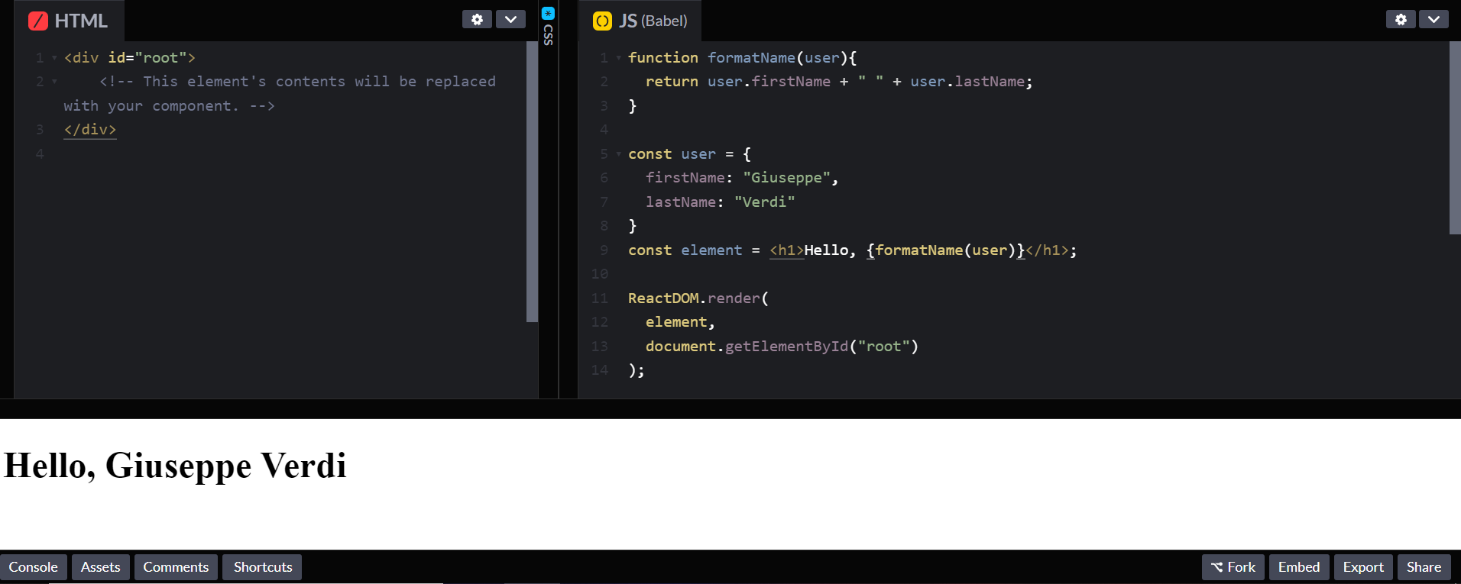
#### Components are reusable

#### 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)

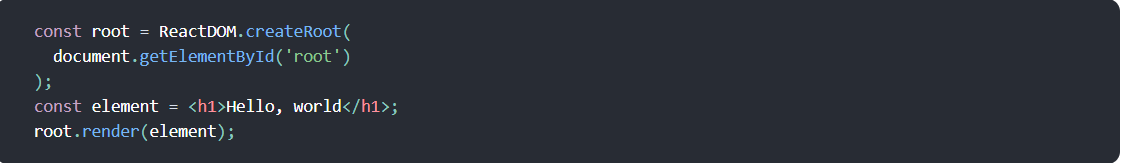
### 1.1.5.Specificare figli in JSX

### 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)



### 1.2.2.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().**

****

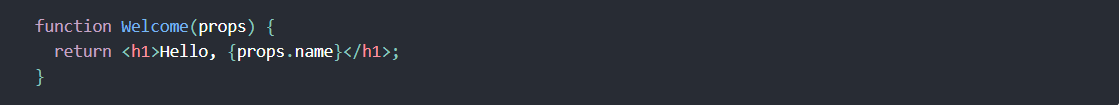
## 1.3.Componenti e Props

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

The simplest way to define a component is to write a JavaScript function:



You can also use an **ES6 class** to define a component:



Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

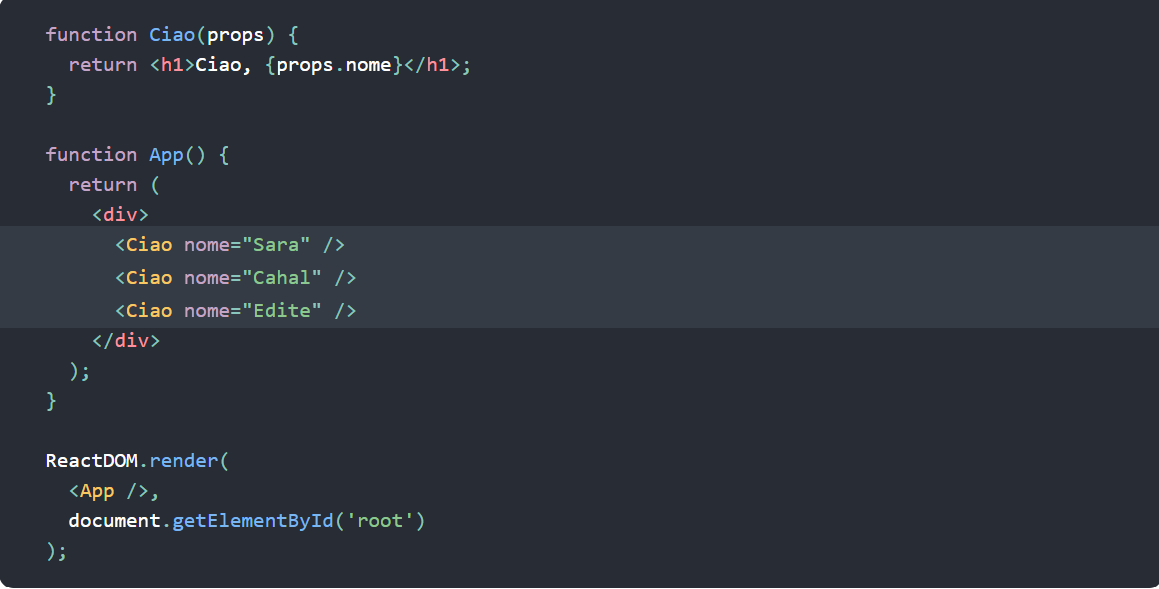
Descrizione generata automaticamente

### 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

Immagine che contiene testo

Descrizione generata automaticamente

**Modularizzando si ottiene**

**Avatar**

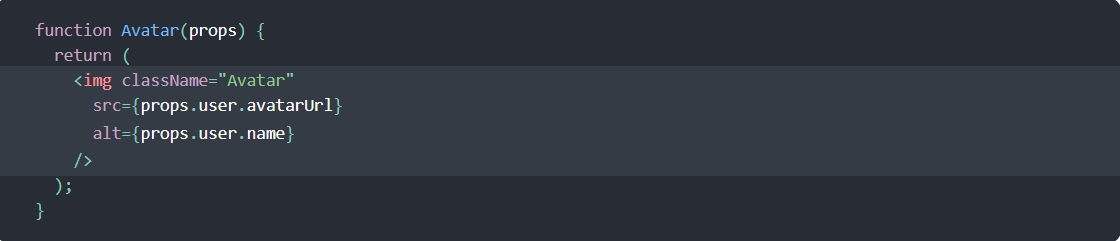
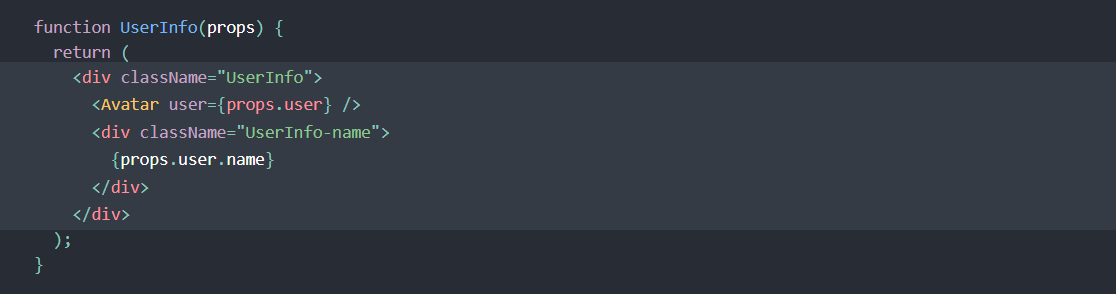
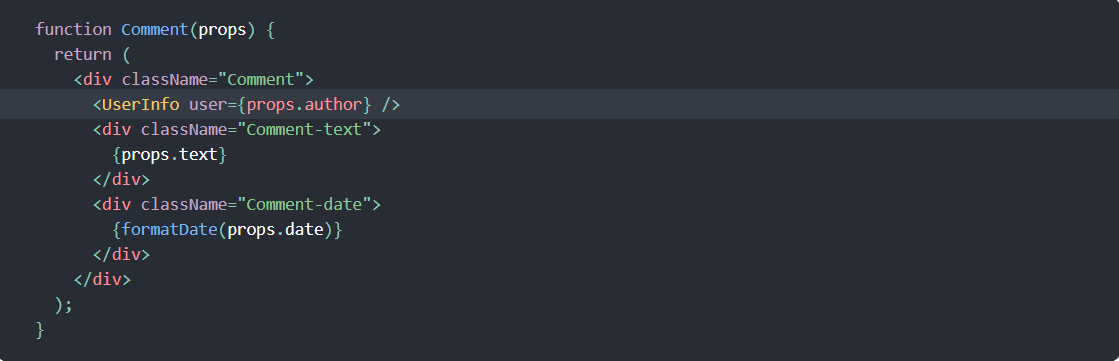


Immagine che contiene testo

Descrizione generata automaticamente

**UserInfo**





### 1.3.5.Le Props sono in sola lettura

### 1.3.6.Esempio su composizione di componenti

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

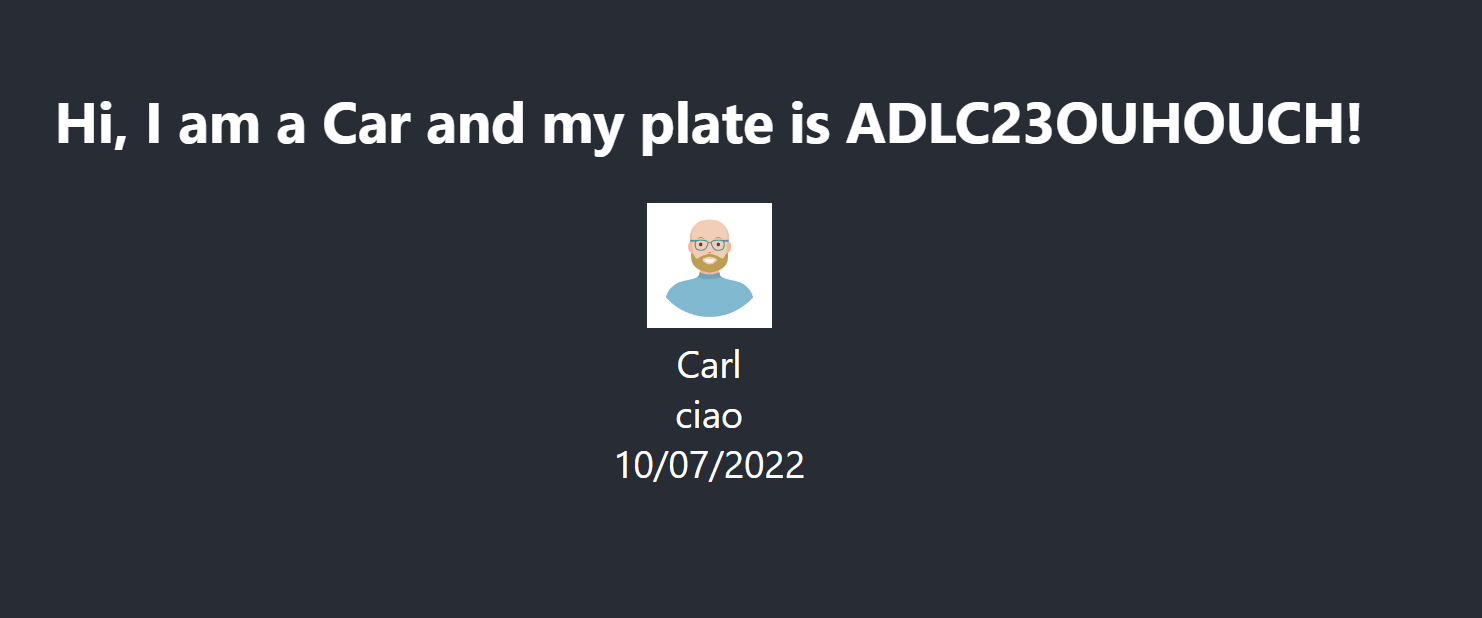
Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

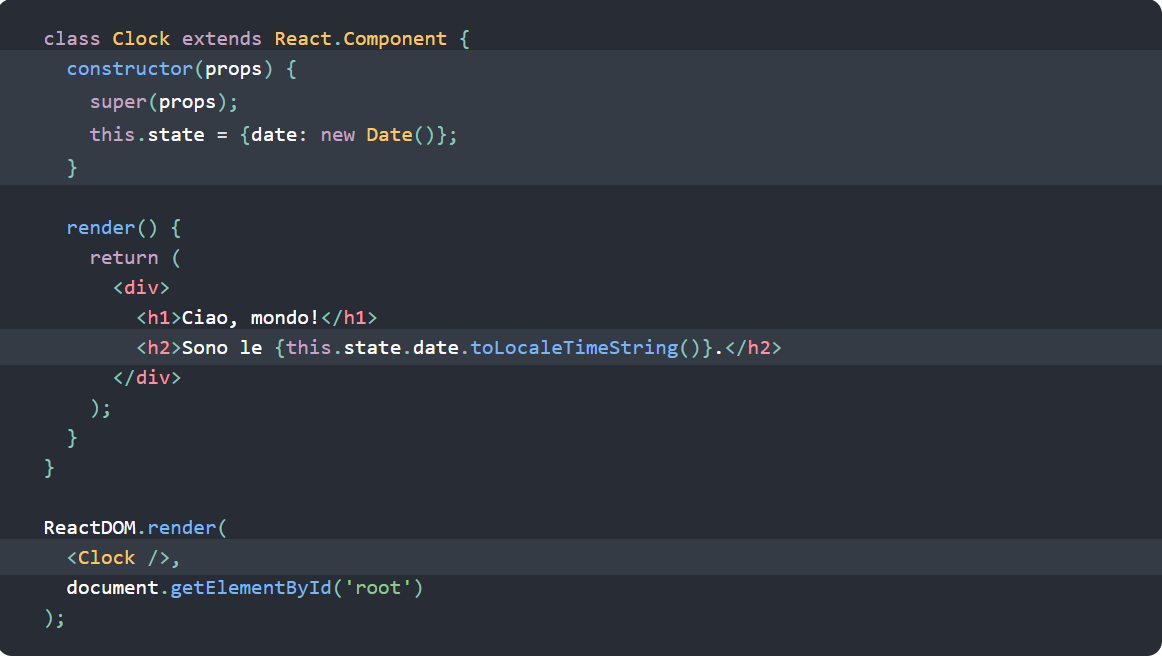
Descrizione generata automaticamente



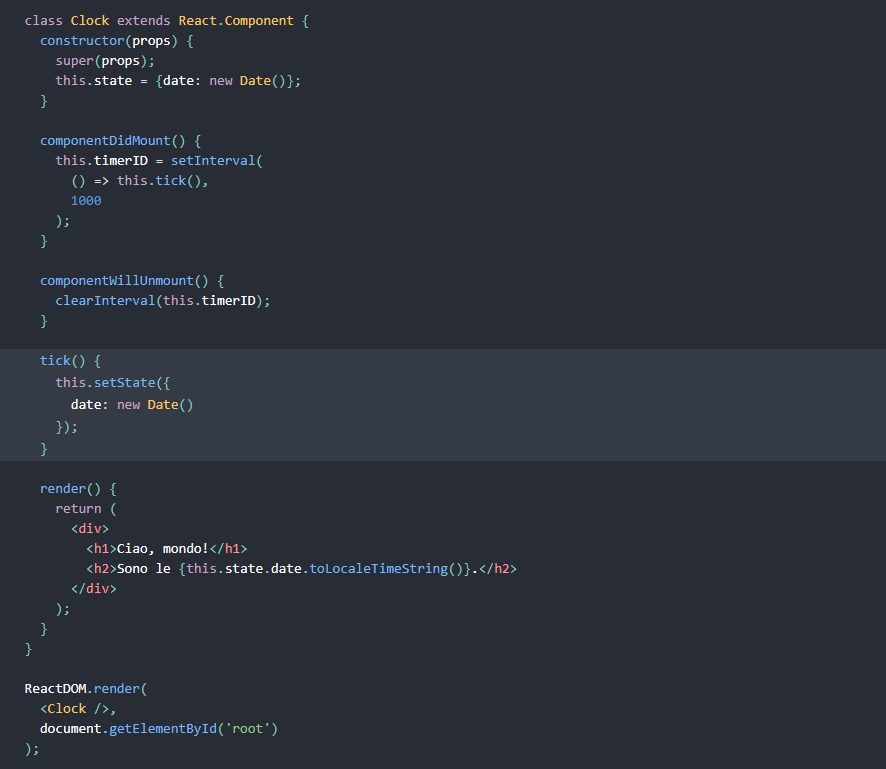
## 1.4.State e lifecycle



### 1.4.1.Localstate di una classe

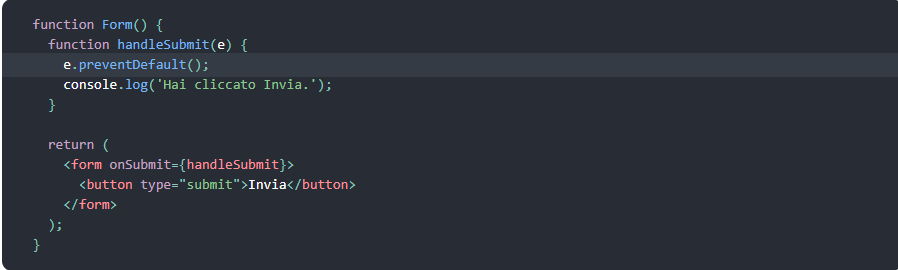


### 1.4.2.Aggiungere metodi lifecycle ad una classe

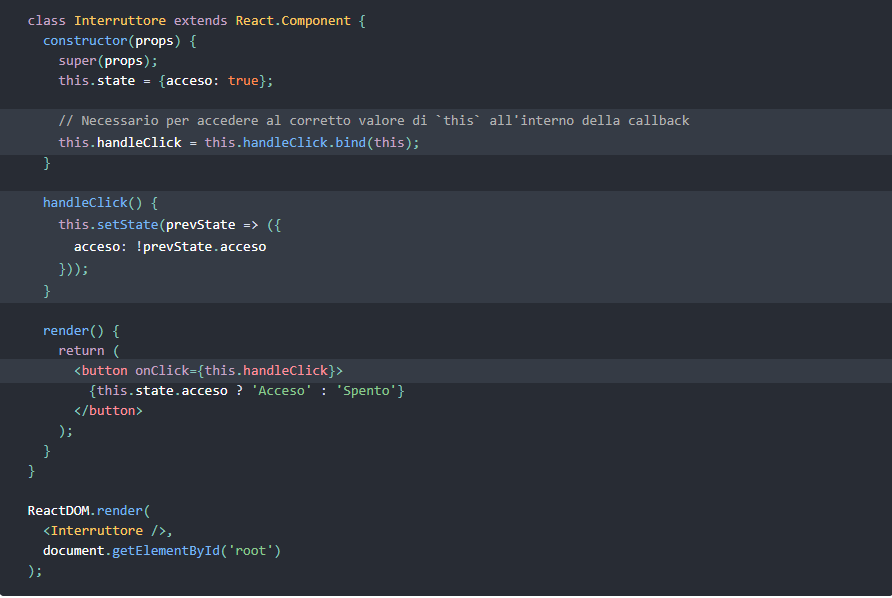


## 1.5.Gestione degli eventi

### 1.5.1.Prevent default on submit

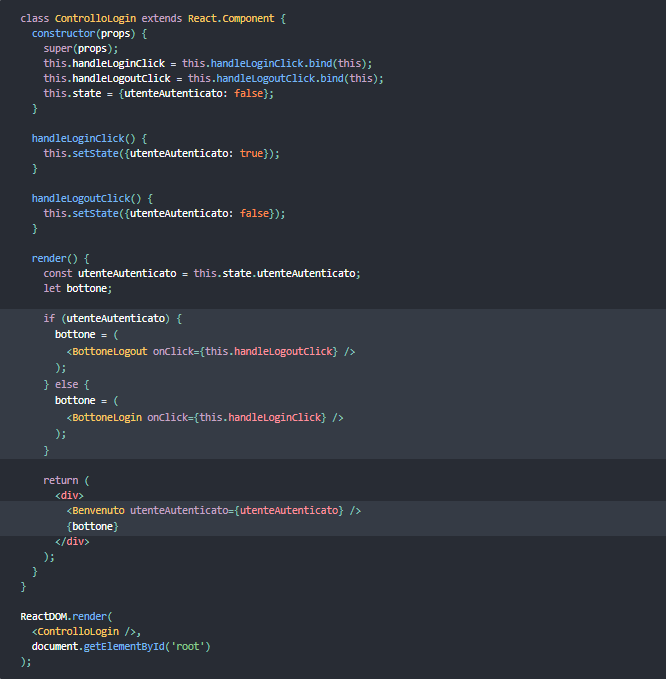


### 1.5.2.Interruttore che cambia il suo stato in risposta all’event click

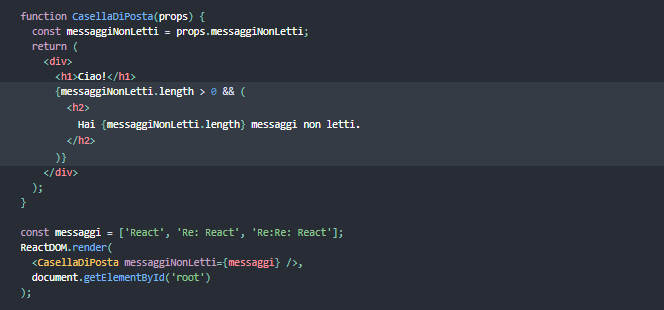


## 1.6.Renderizzazione condizionale

### 1.6.1.Variabili elemento



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

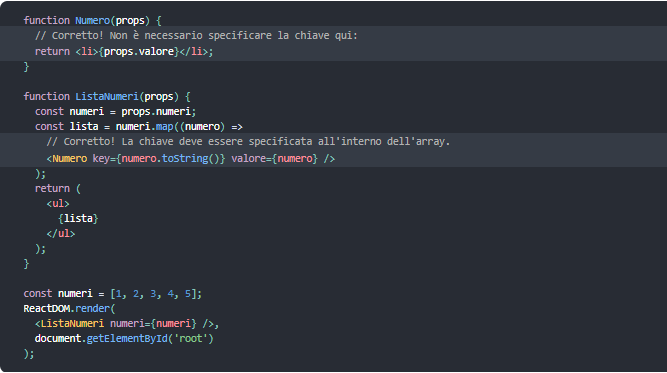


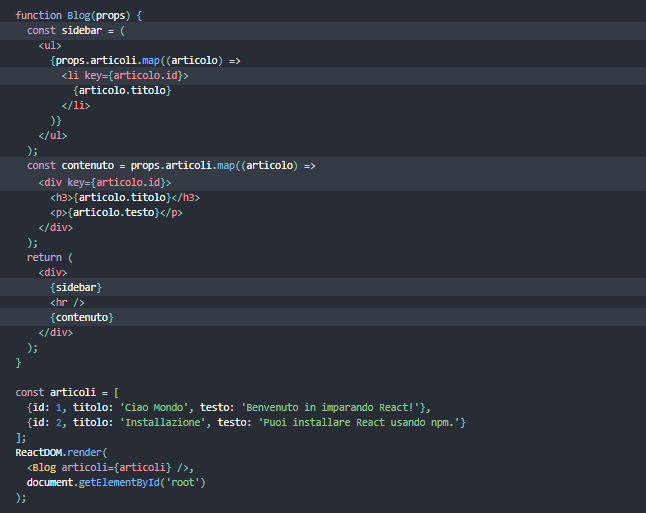
Funziona perché in **javascript , true && espressione** si risolve sempre in **espressione**, mentre **false && espressione** si risolve sempre in **false**.

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

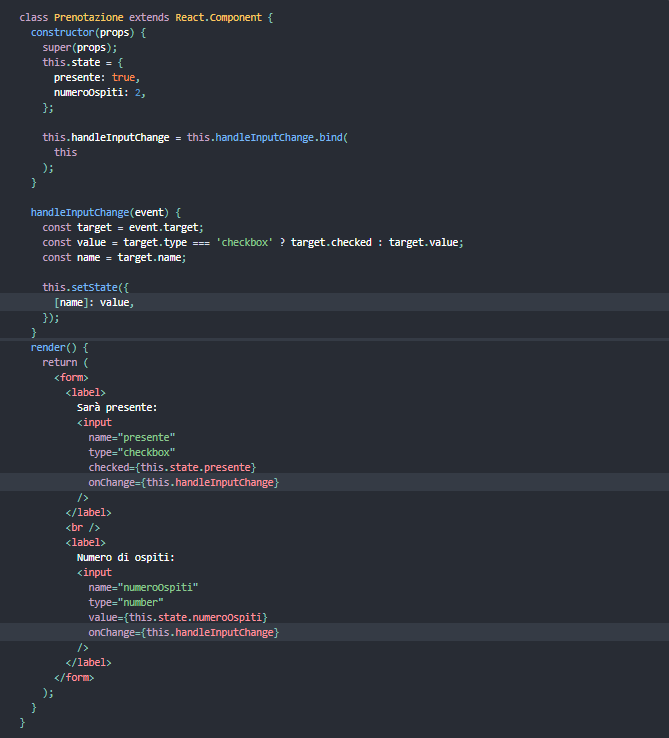
### 1.6.3.Condizione if else con operatore condizionale(operatore ternario)

## 1.7.Liste e Chiavi





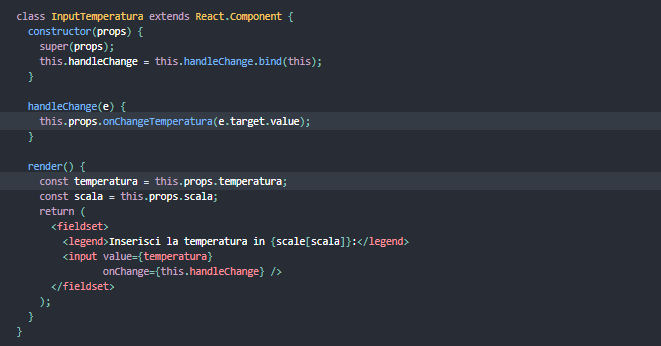
## 1.8.Forms

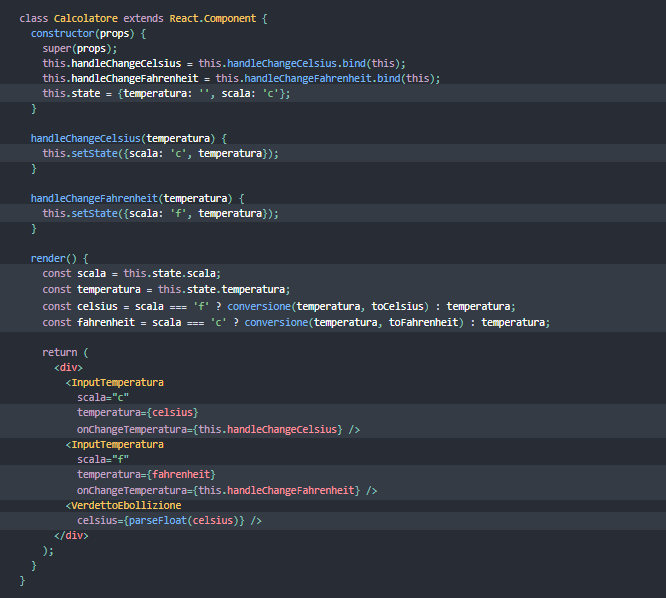


## 1.9.Spostare lo stato

### 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.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.

### 2.1.1.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**.

### 2.1.2.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

## 3.1.React

## 3.2.ReactDOM

## 3.3.ReactDOMClient

## 3.4.ReactDOMServer

## 3.5.Elementi DOM

## 3.6.SyntheticEvent

## 3.7.Test Utilities

## 3.8.Test Renderer

# 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 classis, ti permettono di usare React senza classi s

## 4.2.Panoramica sugli Hooks

## 4.3.Usare l’Hook State

## 4.4.Usare l’Hook Effect

## 4.5.Regole degli Hooks

## 4.6.Hooks Personalizzati

## 4.7.API di Riferimento degli Hooks

### 4.7.1.useState



Immagine che contiene testo

Descrizione generata automaticamente

### 4.7.2.useEffect

### 4.7.3.useContext

### 4.7.4.useReducer

### 4.7.5.useCallback

### 4.7.6.useMemo

### 4.7.7.useRef

### 4.7.8.useImperativeHandle

### 4.7.9.useTransition

### 4.7.10.useId

### 4.7.11.useSyncExternalStore

### 4.7.12.useInsertionEffect

# 5.TESTING

## 5.1.Testing Overview

## 5.2.Testing Recipes

## 5.3.Testing Environments

# 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)