REACT

Sommario

[1. CONCETTI CHIAVE 2](#_Toc107928668)

[1.1. Introduzione a JSX 2](#_Toc107928669)

[1.1.1. Separazione delle responsabilità anziché separazione delle tecnologie 2](#_Toc107928670)

[1.1.2.Incorporare espressioni in JSX 3](#_Toc107928671)

[1.1.3.JSX è un espressione (si possono utilizzare costrutti come if e for) 3](#_Toc107928672)

[1.1.4.Specificare gli attributi con JSX(virgolette per le stringhe e parentesi graffe per le espressioni) 3](#_Toc107928673)

[1.1.5.Specificare figli in JSX 3](#_Toc107928674)

[1.1.6.JSX previene attacchi XSS 3](#_Toc107928675)

[1.1.7.JSX rappresenta oggetti 3](#_Toc107928676)

[1.2.Renderizzare elementi 3](#_Toc107928677)

[1.2.1.Renderizzare un elemento nel DOM(nodo DOM root) 3](#_Toc107928678)

[1.2.3.Aggiornare un elemento renderizzato 3](#_Toc107928679)

[1.3.Componenti e Props 4](#_Toc107928680)

[1.3.1.Funzioni e classi componente 4](#_Toc107928681)

[1.3.2.Renderizzare un componente 4](#_Toc107928682)

[1.3.3.Comporre componenti(react fragment) 4](#_Toc107928683)

[1.3.4. Estrarre componenti 4](#_Toc107928684)

[1.3.5.Le Props sono in sola lettura 4](#_Toc107928685)

[1.4.State e lifecycle 5](#_Toc107928686)

[1.4.1.Localstate di una classe 5](#_Toc107928687)

[1.4.2.Aggiungere metodi lifecycle ad una classe 6](#_Toc107928688)

[1.4.3.Set state 7](#_Toc107928689)

[1.4.4.Update state objects 7](#_Toc107928690)

[1.5.Gestione degli eventi 8](#_Toc107928691)

[1.5.1.Prevent default on submit 8](#_Toc107928692)

[1.5.2.Interruttore che cambia il suo stato in risposta all’event click 8](#_Toc107928693)

[1.6.Renderizzazione condizionale 9](#_Toc107928694)

[1.6.1.Variabili elemento 9](#_Toc107928695)

[1.6.2.Condizione if inline e operatore logico && 9](#_Toc107928696)

[1.6.3.Condizione if else con operatore condizionale(operatore ternario) 9](#_Toc107928697)

[1.7.Liste e Chiavi 10](#_Toc107928698)

[1.8.Forms 11](#_Toc107928699)

[1.9.Spostare lo stato 12](#_Toc107928700)

[1.9.1.Spostare lo stato “in alto” 12](#_Toc107928701)

[1.10.Composizione vs ereditarietà 13](#_Toc107928702)

[1.10.1.Contenimento 13](#_Toc107928703)

[1.10.2.Specializzazioni 13](#_Toc107928704)

[1.11.Pensare in react 13](#_Toc107928705)

[1.11.1.Passo 1: scomporre la UI in una gerarchia di componenti 13](#_Toc107928706)

[1.11.2.Passo 2: sviluppa una versione statica di react 13](#_Toc107928707)

[1.11.3.Passo 3: identifica la minima ma completa rappresentazione dello stato della UI 13](#_Toc107928708)

[1.11.4.Passo 4: identifica dove posizionare il tuo stato 13](#_Toc107928709)

[1.11.5.Passo 5: invertire il flusso dati 13](#_Toc107928710)

[2.GUIDE AVANZATE 13](#_Toc107928711)

[3.API DI RIFERIMENTO 13](#_Toc107928712)

[4.HOOKS 13](#_Toc107928713)

[5.TESTING 13](#_Toc107928714)

[6.PRINCIPI DI DESIGN 13](#_Toc107928715)

# 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)

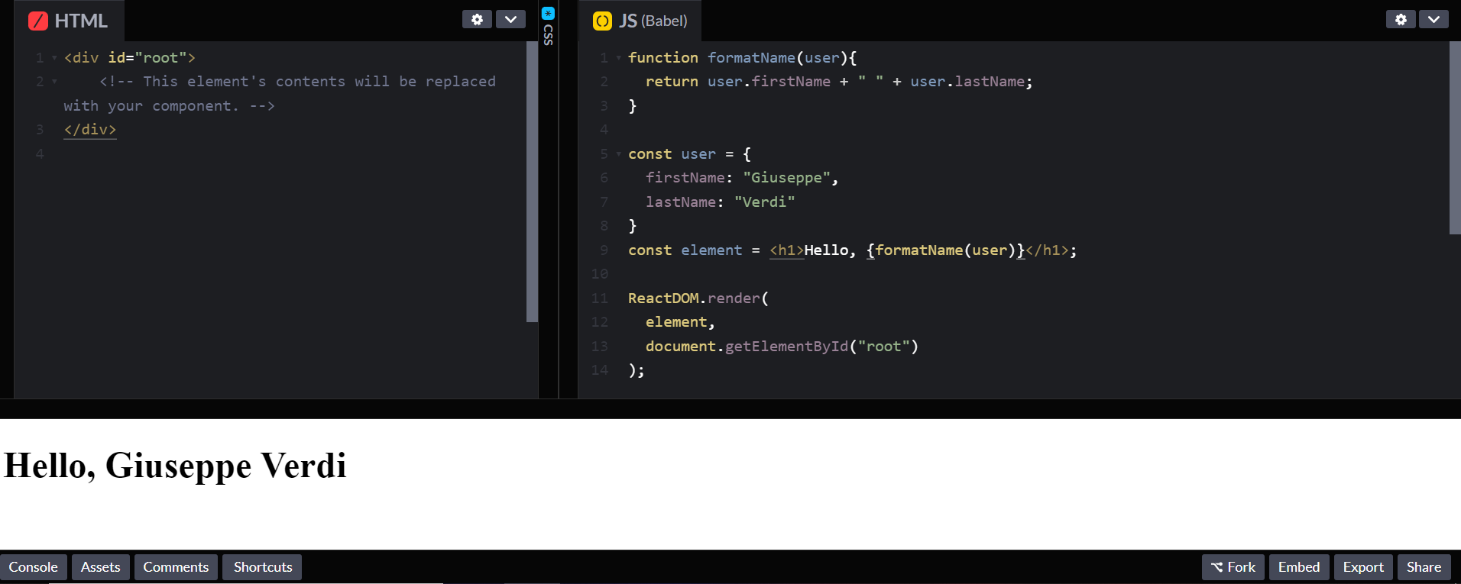
#### Components are reusable

#### 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.

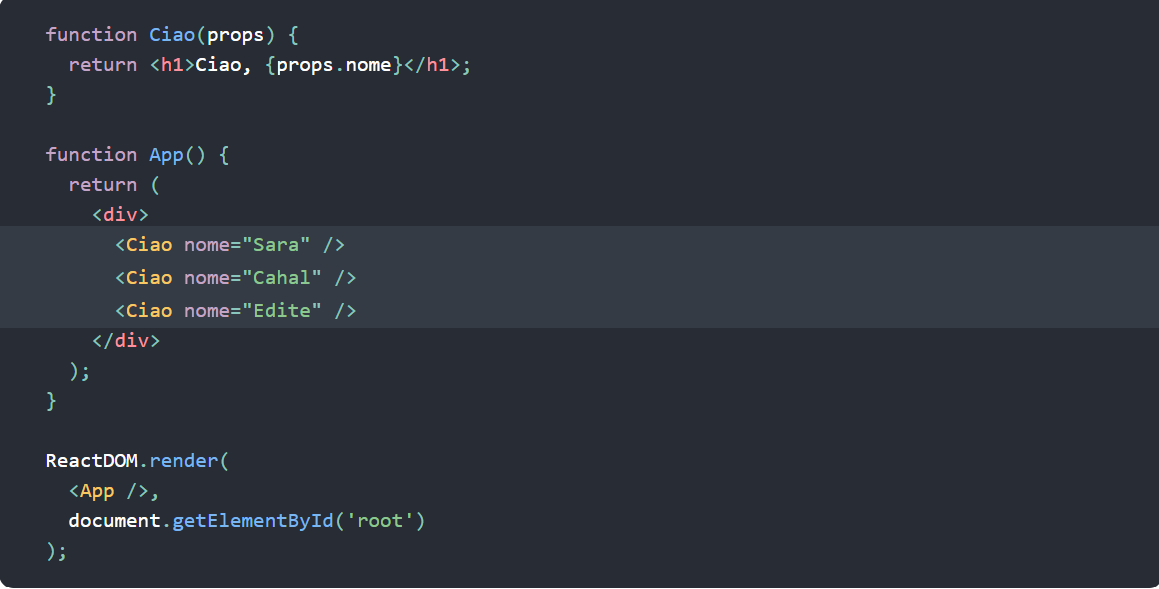
### 1.3.1.Funzioni e classi componente

### 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.



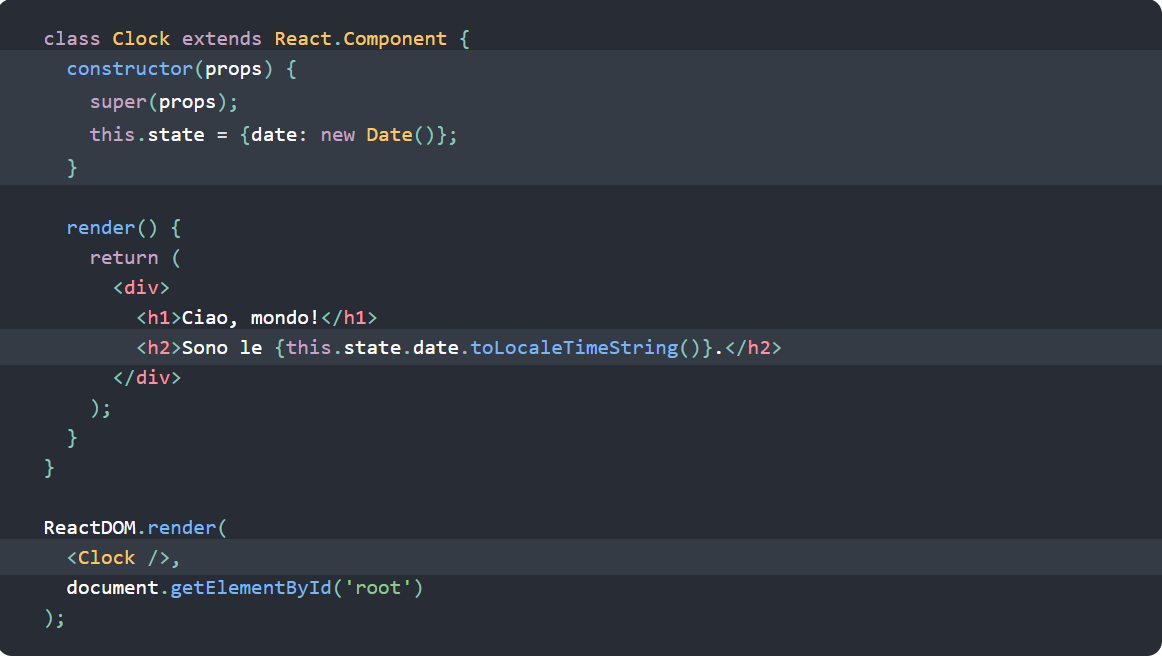
### 1.3.4. Estrarre componenti

### 1.3.5.Le Props sono in sola lettura

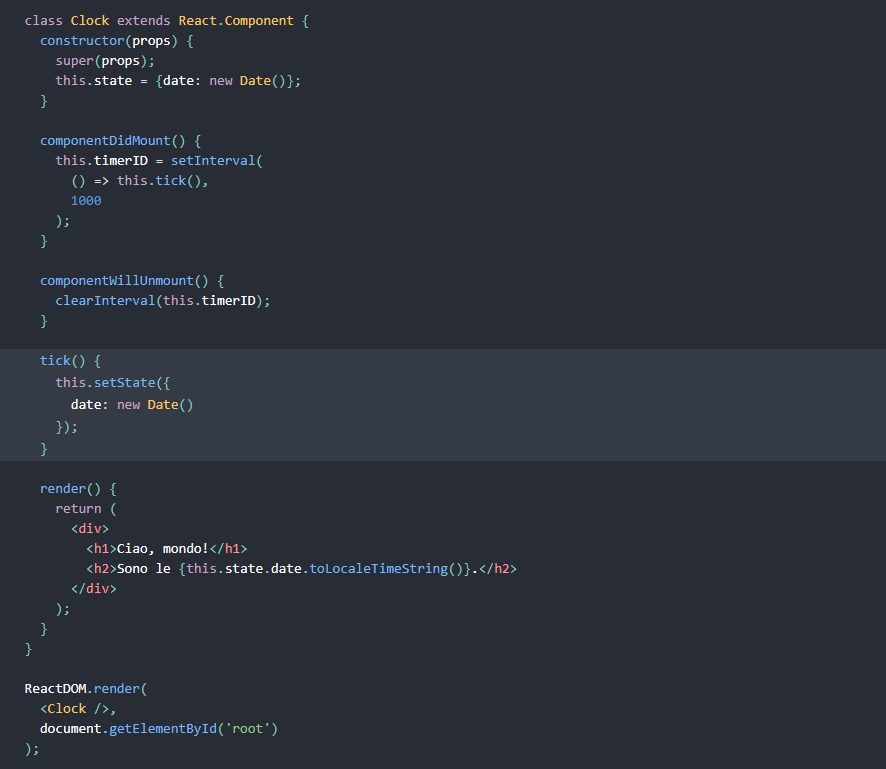
## 1.4.State e lifecycle



### 1.4.1.Localstate di una classe



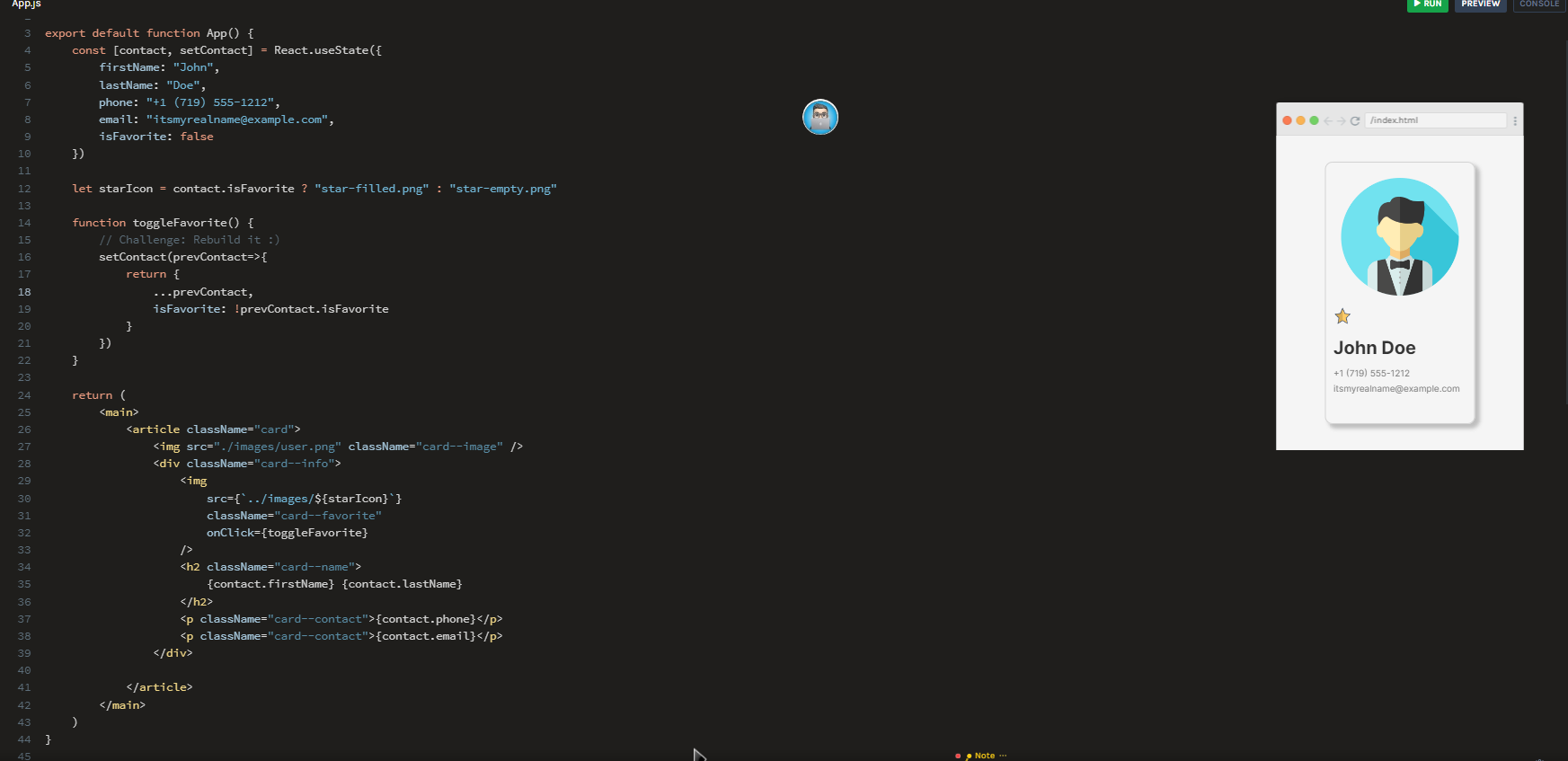
### 1.4.2.Aggiungere metodi lifecycle ad una classe



### 1.4.3.Set state

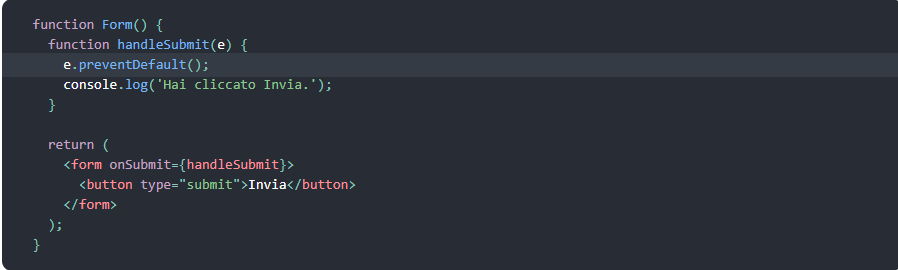


### 1.4.4.Update state objects

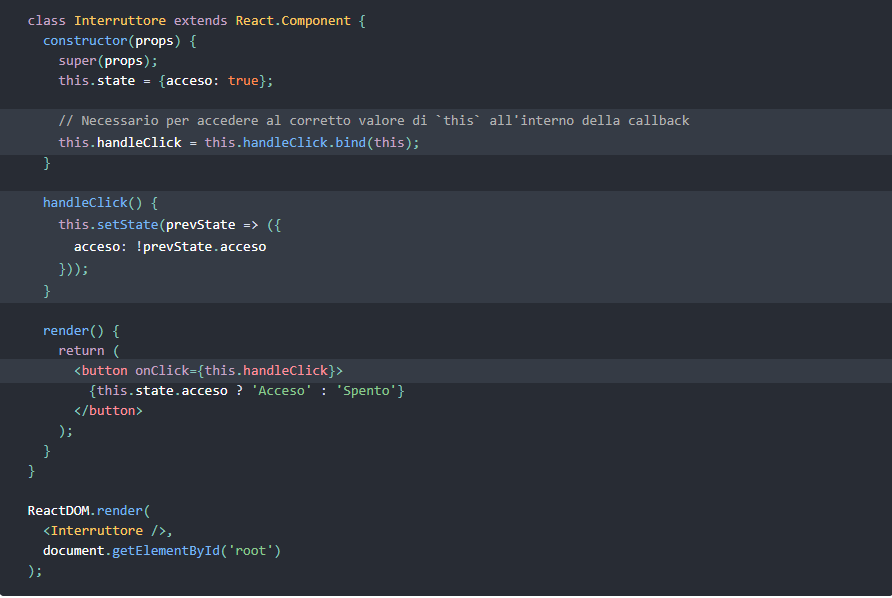


## 1.5.Gestione degli eventi

### 1.5.1.Prevent default on submit

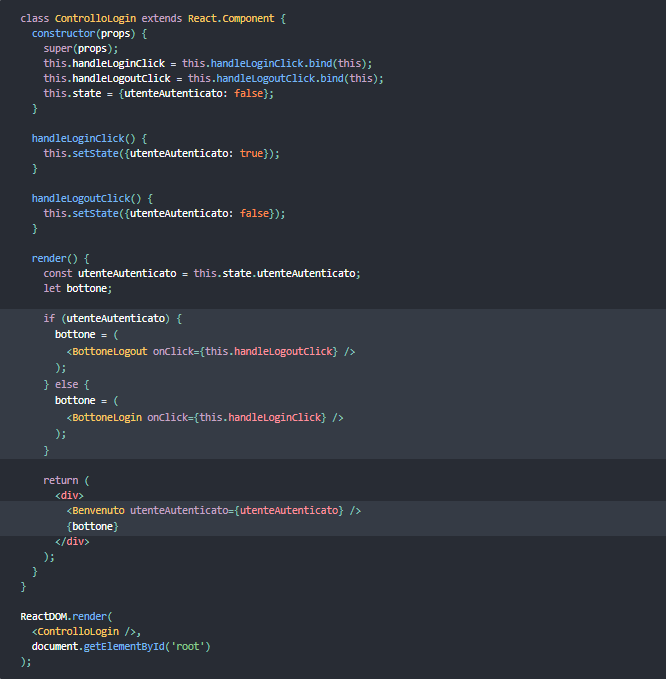


### 1.5.2.Interruttore che cambia il suo stato in risposta all’event click

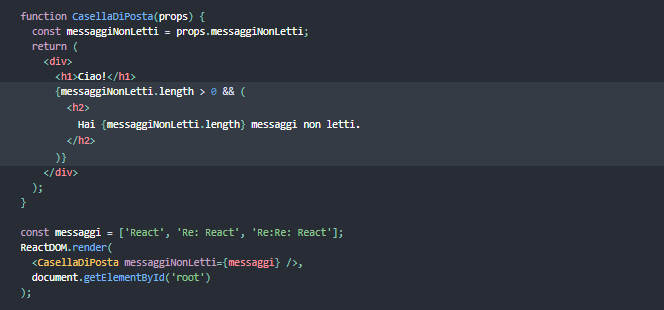


## 1.6.Renderizzazione condizionale

### 1.6.1.Variabili elemento



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

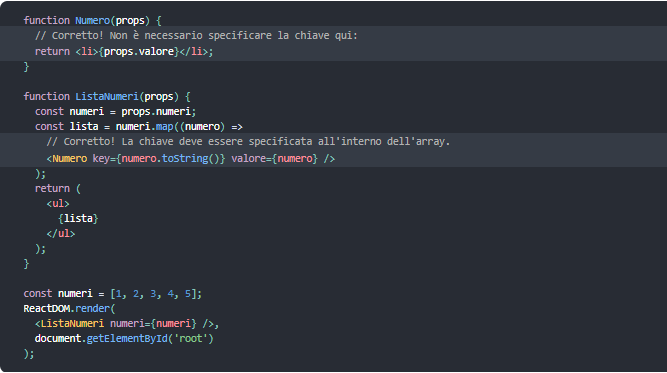


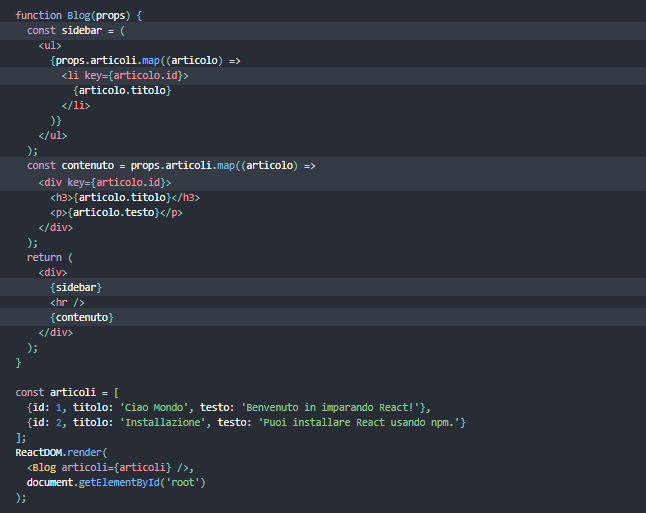
Funziona perché in **javascript , true && espressione** si risolve sempre in **espressione**, mentre **false && espressione** si risolve sempre in **false**.

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

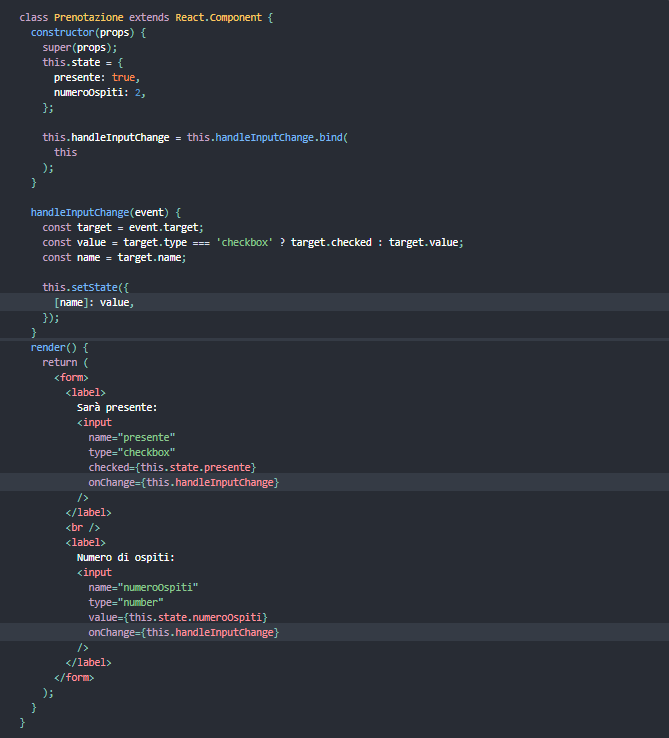
### 1.6.3.Condizione if else con operatore condizionale(operatore ternario)

## 1.7.Liste e Chiavi





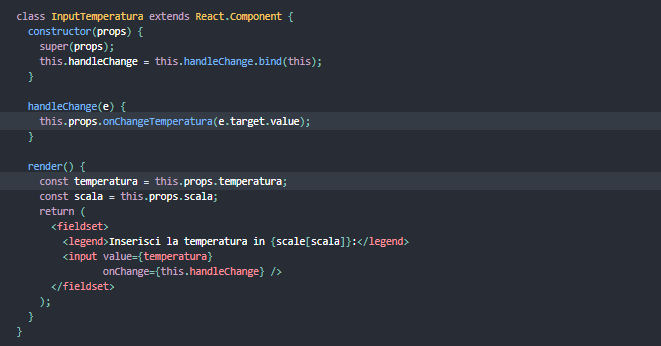
## 1.8.Forms

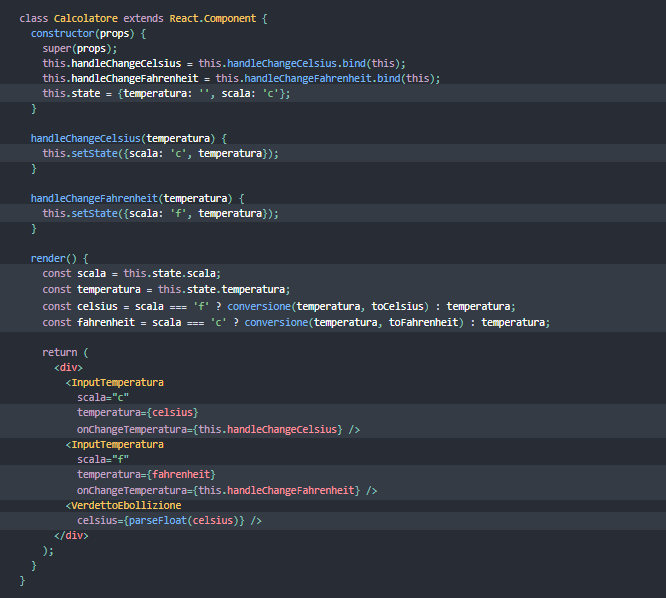


## 1.9.Spostare lo stato

### 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.

## 1.11.Pensare in react

### 1.11.1.Passo 1: scomporre la UI in una gerarchia di componenti

Ma come fare a sapere cosa deve essere un componente a se stante? Usa le stesse tecniche che usi per decidere se devi creare una nuova funzione od un oggetto. Una di queste tecniche è il [principio di singola responsabilità](https://it.wikipedia.org/wiki/Principio_di_singola_responsabilit%C3%A0), che dice che un componente dovrebbe idealmente occuparsi di una cosa soltanto. Nel caso in cui un componente diventi troppo grande e complesso, dovrebbe essere decomposto in sotto-componenti più piccoli e semplici.

### 1.11.2.Passo 2: sviluppa una versione statica di react

La chiave qui è [DRY: Don’t Repeat Yourself](https://it.wikipedia.org/wiki/Don%27t_repeat_yourself). Individua la minima rappresentazione di stato richiesta dall’applicazione e calcola tutto il resto al bisogno.

### 1.11.3.Passo 3: identifica la minima ma completa rappresentazione dello stato della UI

### 1.11.4.Passo 4: identifica dove posizionare il tuo stato

### 1.11.5.Passo 5: invertire il flusso dati

# 2.GUIDE AVANZATE

# 3.API DI RIFERIMENTO

# 4.HOOKS

# 5.TESTING

# 6.PRINCIPI DI DESIGN