# Immagine che contiene testo, Carattere, Biglietto Post-it, design Descrizione generata automaticamente04\_08 REACT LIFE-CYCLE

Ci sono diversi Hook che possono agire durante queste fasi:, in particolare ci concentriamo su:

* Immagine che contiene testo, schermata, diagramma, linea

  Descrizione generata automaticamenteuseState
* useContext

Nella fase di rendering, react agisce sul virtual DOM e non su DOM effettivo, nella fase di Commit ci sono degli Hook ad hoc:

* useEffect()
* useLayoutEffect()

Nota: nella **fase di commit**, a differebza della fase di rendering, ci possono essere side-effect (calcolo che non ha come obiettivo l’output finale del componente ma influisce qualcos’altro al di fuori del componente funzionale che viene eseguito… es: Console.log)

* data fetching
* log recording
* subscription
* cambiare manualmente il DOM
* Gestire time-out e intervalli di tempo

Immagine che contiene testo, schermata

Descrizione generata automaticamente

## Immagine che contiene testo, Carattere, schermata, design Descrizione generata automaticamenteCome usare useEffect

**Callback**: indica cosa eseguire

* Funzione contenente tutte le logiche di side-effect
* useEffect esegue la callback dopo che react ha fatto il committ dei cambiamenti sullo schermo (dopoc he il DOM del browser è stato aggiornato).

**Array di dipendenze**: indicano quando eseguire

* Array di dipendenze opzionale
* Serve per fare in modo che la callback venga eseguita se almeno una delle depencies dell’arrat è cambiata tra due rendering
* Può avere tre stati:
  + Immagine che contiene testo, Carattere, schermata, linea

    Descrizione generata automaticamente**Not provided** → array non c’è
    - Side-effect eseguito ad ogni rendering, comodo se c’è operazione da fare
  + Immagine che contiene testo, Carattere, schermata

    Descrizione generata automaticamente**Empty** array []
    - C’è rendering solo quando il componente viene creato la prima volta o viene fatto il refresh di quel componente
  + Immagine che contiene testo, schermata, Carattere

    Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata

    Descrizione generata automaticamente**Con props o stati**: side-effect eseguito una volta dopo rendering inziiali e ogni volta ch euno di questi valori di dipendenza cambia

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Quando una variabile di stato cambia, effetto eseguito:

* Se stato aggiornato e valore non cambia → effetto non eseguito.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteDentro uno useEfffect si può **schedulare un aggiornamento di stato** → stato viene aggiornato in maniera asincrona:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

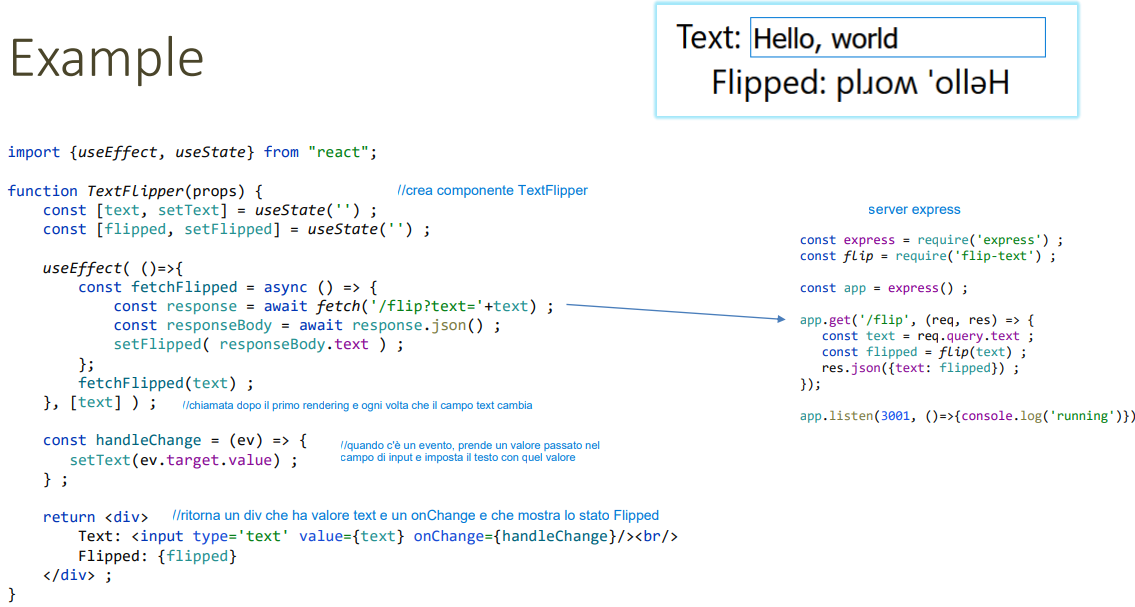
Nota: array dipendenze useEffect:

* Deve includere tutti i valori (props stati) del componente che possono cambiare nel tempo e che sono usati dall’effetto stesso
* Se l’array include variabili che cambiano sempre, rischio loop infinito

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteSe si considera anche la fetch:

→ useEffect non prevede che la callback possa avere una async davanti ma va inserita all’interno della callback.



Per **gestire eventuali risposte lente**, si può aggiungere uno stato aggiuntivo che segnala waiting

nota: di default, mostra orologio. Quando la risposta viene ricevuta e lo stato viene aggiornato, bisogna aggiornare anche lo stato del waiting a false.

Alcuni side-effect necessitano di **clean-up**, ad esempio per chiudere un socket o pulire timer.

* Se la side-effect ritorna una funzione, il side-effect considera quella funzione come una **funzione di pulizia/clean-up**:
  + Dopo il rendering iniziale, viene chiamato useEffect normale, senza clean-up
  + Ad un rendering successivo, prima di invocare l’use-effect, viene chiamata la funzione di clean-up

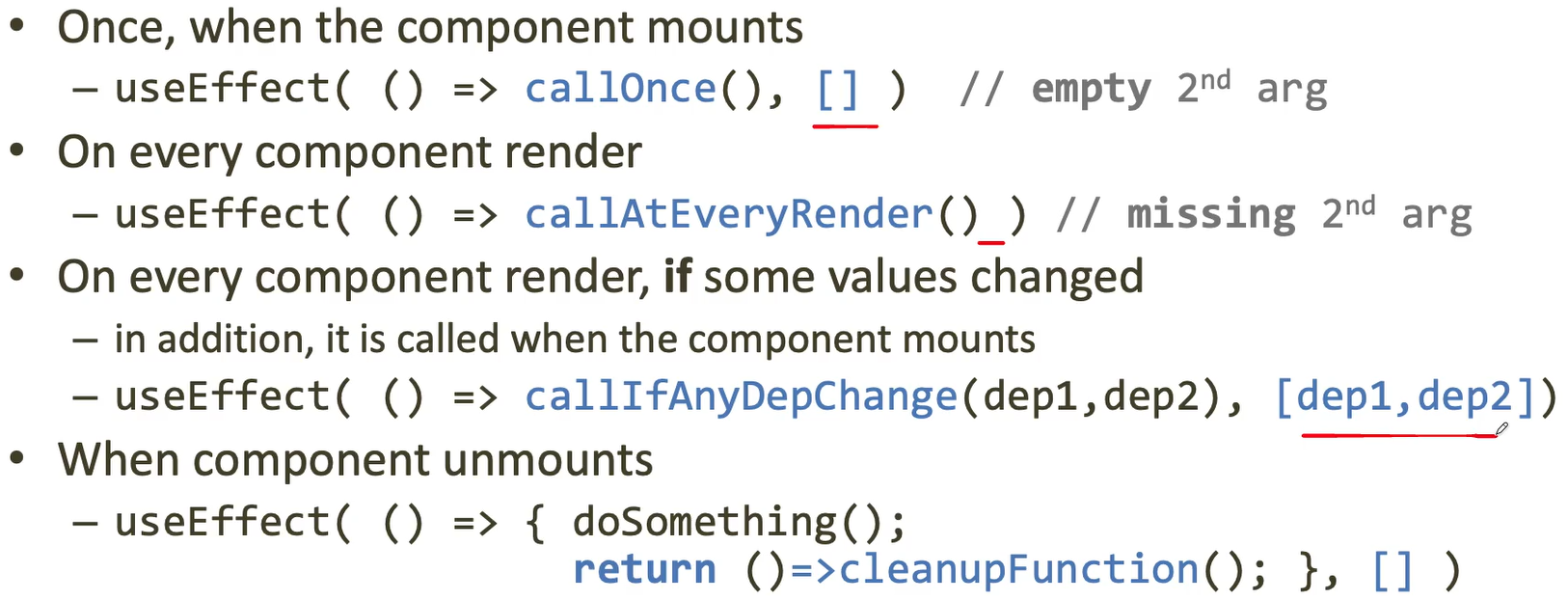
**Nota:**  non c’è biosgno di use-effect quando si vogliono trasformare i dati per il rendering

* **Esempio:** perfiltrare una lista prima di mostrarla, possiamo trasformare uttti i dati a livello alto del componenete senza usare use-effect

**Nota:** non c’è bisogno di use-effect pergestire eventi che dipendono dagli utenti → vanno gestiti nell’event handler

* **Esempio:** submit su form, click sul bottone

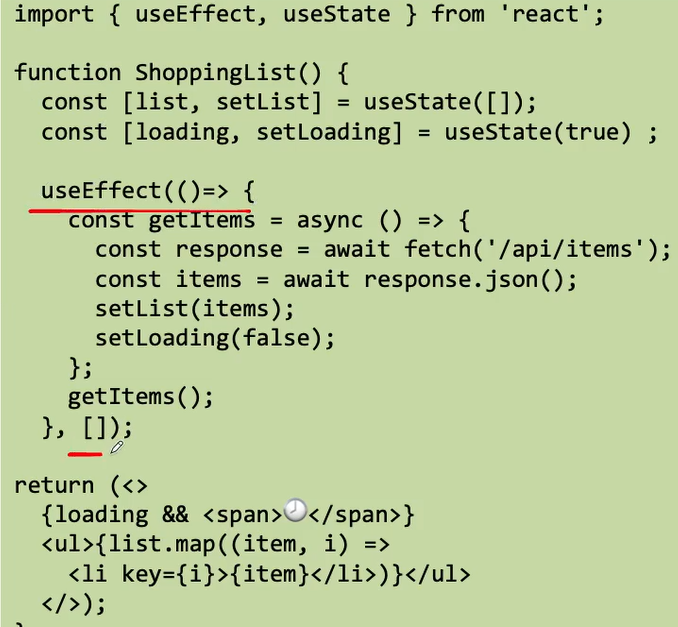
**Dunque, sommario useEffect:**

****

**Nota**: i *metodi di fetch non devono essere mantenuti all’interno dell’use effect ma in un modulo javascript separato (API.js).*

In un’applicazione ci possono essere 2 tipi di stato:

* **Application state**: quello che recuperiamo dal backend e in caso di aggiornamento, dovrebbe aggiornare il backend
  + Dovrebbe, periodicamente , controllare se ci sono degli aggiornamenti
* **Presentation state**: memorizzato solo in react, non c’è bisogno che persista, vive e muore nel componente che lo controlla

**Reidratazione di un application stat**e: prenderlo dalle API

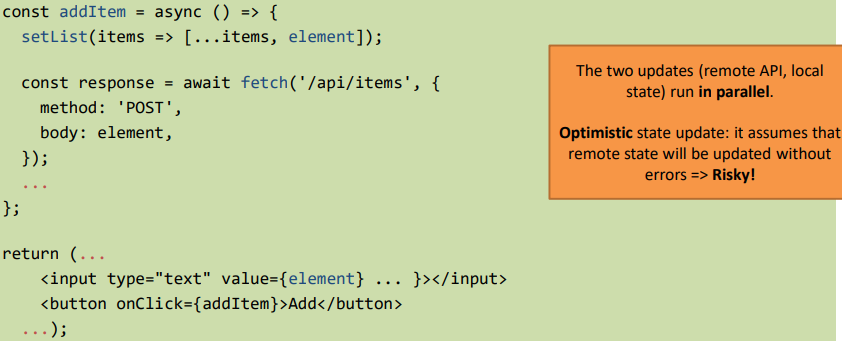
* Deve avvenire al primo rendering (attraverso use effect con array delle dipendenze presente, vuoto)

*Si può anche effettuare una reidratazione per rinfrescare lo stato → aggiornarsi quando avviene un aggiornamento.*

*Nel caso in cui ci siano tanti browser e un solo server, nel caso in cui il browser 3 aggiorna qualcosa e vogliamo fare in modo che le altre lo sappiano → server deve comunicare i cambiamenti ai vari browser che ha a disposizione.*

* *Noi usiamo un modo semplificato che minimizza il problema usando http standard:*
  + *Applicazione web chiede aggiornamenti il più frequentemente possibile*

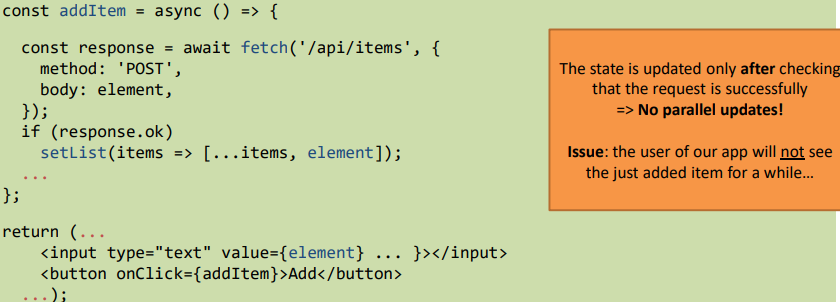
**Deidratazione**: estrare application state da react e inviarlo al back-end

* Può avvenire diverse volte

→ due aggiornamenti ch efunzionano in parallelo:

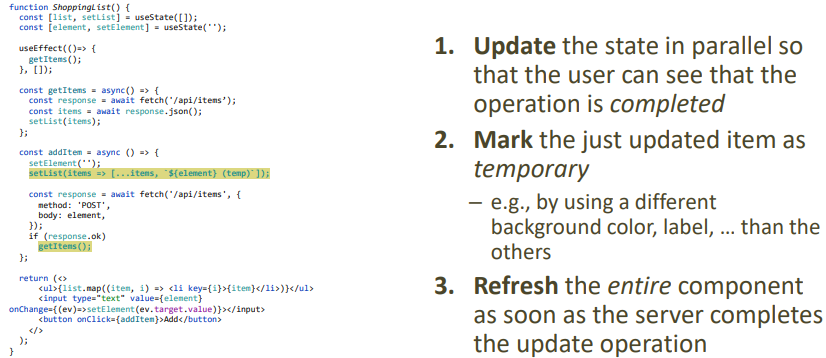
* Lista della pagina
* Server in parallelo

**Nota**: bisogna sperare che tutto vada bene, potrebbero esserci degli errori



Faccio richiesta al server, se la risposta è positiva aggiorno pagina, altrimenti no

* *Stato aggiornato solo dopo aver controllato che la richiesta si server sia andata a buon fine. Problema: l’utente della nostra applicazione deve attendere che il tempa risponda, altrimenti non vede subito la risposta*

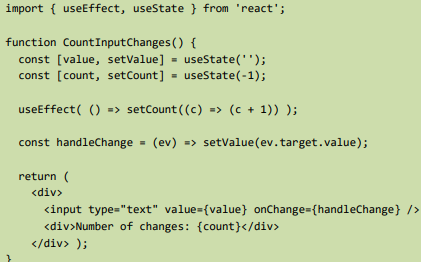
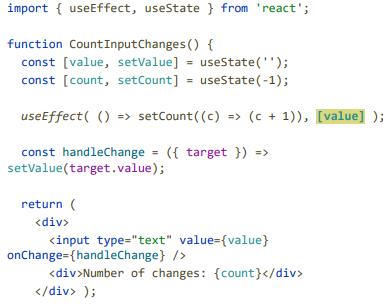


Per risolvere, si fa reidratazione+deidratazione: aggiorno stato in parallelo così utente può vedere che l’oggetto è stato aggiunto ma lo informo che è provvisorio

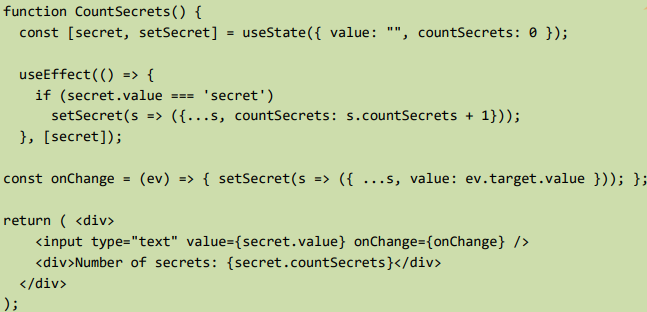
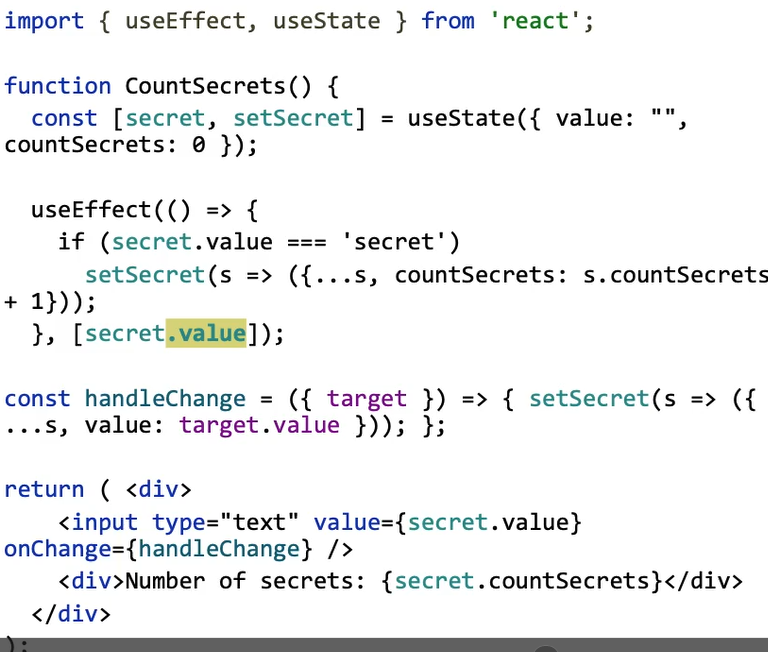
### Loop infiniti con useEffect

Usando use effect si possono creare dei loop infiniti a causa di:

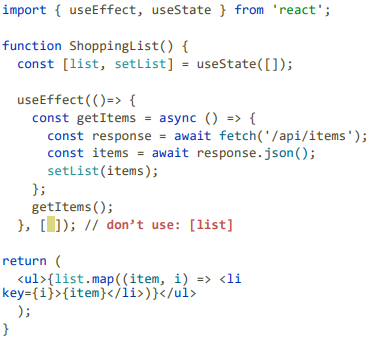
* Array delle dipendenze mancante
* Nell’array delle dipendenze si usa un oggetto o un array

→ **dipendenza mancante** e incremento lo stato count che causa re-rendering infinito.

Modo corretto 🡨



→ questo useState, **crea un nuovo oggetto** ogni volta che viene chiamato (Anche se contenuto non chiamato → quindi useEffect chiamato ogni volta)



→ nell’array di dipendenze, non usare un array di dipendenze, al massimo si può usare la lunghezza di un array o cose simili ma non l’array stesso.