

Il Contatore

<https://github.com/Carlaisabelle/theCounter>
<https://papaya-beignet-c564d3.netlify.app/>

oooo —

+ ooo

Indice

01 - Obiettivi

Gli obiettivi
dell'applicazione

03 - Preparazione

Come ho progettato
l'applicazione

07 - Struttura

La struttura
dell'applicazione

09 - Il DOM

Come ho creato il
DOM

11 - Classe Counter

Gli elementi della
classe Counter

18 - I contatori

standardCounter e
randomCounter

20 - Cambio contatore

La logica del
cambio contatore

○ ○ ○ ○ —



01

Obiettivi

Il codice qui discusso rappresenta un'applicazione "contatore" completa delle funzioni di:

- incremento e decremento;
- reset del numero di partenza;
- cambio di visualizzazione tra un contatore che parte da 0 e un altro contatore che parte da un numero compreso tra 0 e 100 (generato casualmente).

○ ○ ○ ○ —



02

L'interfaccia utente (che, secondo le direttive, doveva essere generata e gestita interamente utilizzando JavaScript) prevede la presenza di quattro pulsanti (due per il decremento/ l'incremento del valore, uno per far ripartire il conteggio e uno per il cambio contatore) e un “display” in cui viene mostrato il valore corrente.

L'HTML è mantenuto volutamente molto semplice e pulito; il CSS, invece, gestisce non solo lo stile dell'applicazione, ma anche la visibilità dei contatori stessi.



Preparazione

Le direttive chiedevano che l'applicazione:

- mostrasse il valore iniziale del counter (0) quando l'utente vi accede;
- ci fossero due pulsanti (per incrementare e decrementare il valore);
- fosse sviluppata esclusivamente in JavaScript (compresi gli elementi del DOM);
- fosse scritta senza l'utilizzo di jQuery o di framework;
- fosse completata con funzionalità aggiuntive ritenute utili o interessanti.

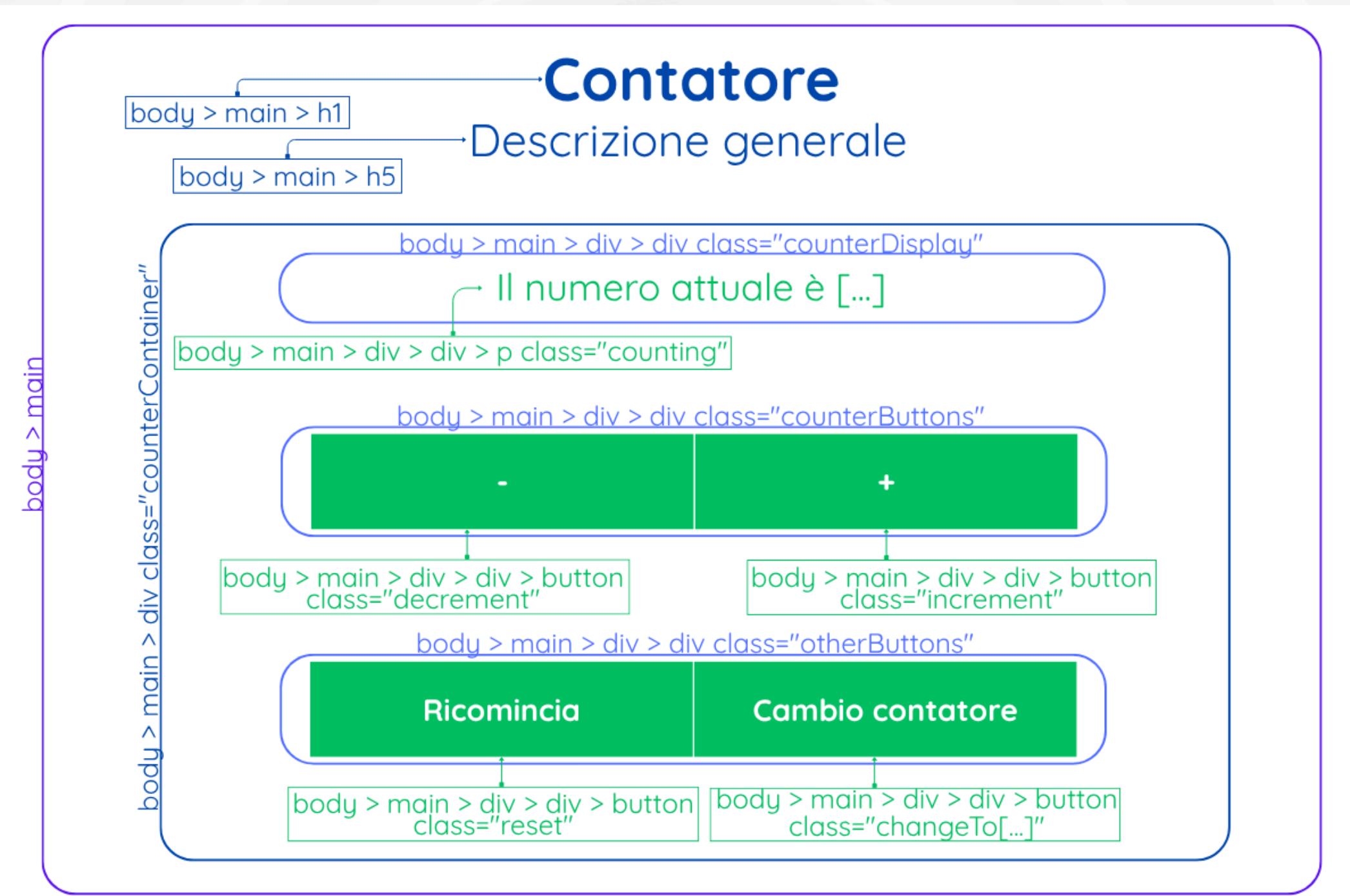
04

Per prima cosa, ho riflettuto su come avrei voluto strutturare l'applicazione e su quali funzionalità aggiuntive avrei voluto implementare. Ho deciso che mi sarebbe piaciuto creare una versione dello stesso contatore che, invece di partire da 0, consentisse di partire da un numero generato casualmente. Ho, quindi, immaginato la struttura dell'interfaccia, suddividendola in settori per poterli visualizzare correttamente e facilitarmi la successiva creazione del DOM con JavaScript.



05

La struttura della pagina web:



```
body
└ main: mainElement
    └ h1: mainTitle
    └ h5: description
    └ div: standardCounterContainer
        └ div: displayStandardCounter
            └ p: countingStandard
        └ div: buttonsStandard
            └ button: decrementStandard
            └ button: incrementStandard
        └ div: otherButtonsStandard
            └ button: resetStandard
            └ button: changeToRandom
    └ div: randomCounterContainer
        └ div: displayRandomContainer
            └ p: countingRandom
        └ div: buttonsRandom
            └ button: decrementRandom
            └ button: incrementRandom
        └ div: otherButtonsRandom
            └ button: resetRandom
            └ button: changeToStandard
```

Una volta progettata la pagina web, ho stilato la struttura "ad albero" del DOM per usarla come guida durante la scrittura del JavaScript.

07

Struttura

L'applicazione è strutturata per creare due contatori:

- uno "standard" che inizia il conteggio da 0 e si resetta a 0;
- uno "random" che parte da un numero compreso tra 0 e 100 (generato casualmente) e si resetta a un nuovo numero casuale.



Le logiche che gestiscono i due contatori sono identiche, ad eccezione di ciò che riguarda il valore di partenza e il reset.

○ ○ ○ ○ —



08

L'applicazione è composta da 4 file:

- un file HTML, chiamato *index.html*, con un *body* volutamente “spoglio” per consentire la creazione dinamica del DOM;
- un file CSS, chiamato *style.css*, che gestisce lo stile del contatore e “decide” quale contatore viene visualizzato;
- un primo file JavaScript, chiamato *documentCreation.js*, che si occupa di tutto ciò che ha a che fare con i contatori e la loro creazione;
- un secondo file JavaScript, chiamato *changeCounter.js*, che si occupa di contenere la logica per lo switch tra i contatori; per assicurarmi che tutto il DOM sia stato creato prima che venga caricato questo file, *changeCounter* ha la proprietà *async*.

Step 1: il DOM

Il primo step è stato la generazione dinamica del DOM con JavaScript. Usando come guida la struttura mostrata [qui](#), ho creato una funzione chiamata `createDocument()` composta da:

- una serie di costanti (`const`), “appese” in successione una all’altra a seconda della posizione che avrebbero avuto nell’HTML e che rappresentano gli elementi che compongono il DOM;
- un `return` che restituisce gli elementi che serviranno per le logiche dei contatori.



Ho poi chiamato la funzione `createDocument()` all'interno della costante `documentElements` creando, così, un oggetto le cui proprietà corrispondono agli elementi generati dalla funzione stessa. Ciò mi ha permesso di circoscrivere gli elementi del DOM allo scope della funzione `createDocument()` e, allo stesso tempo, di potervi accedere anche da blocchi di codice diversi senza dover ricorrere alla generazione di variabili globali.

Step 2: classe Counter

Siccome i contatori che ho previsto hanno una struttura pressoché identica, ho deciso di crearli ricorrendo alle classi. Ho usato una classe *Counter* il cui costruttore è formato da:

- il “contenitore” del contatore (*counterContainer*);
- il valore di partenza (*startingValue*);
- il “display” del contatore (*counterDisplay*);
- il pulsante di decremento (*decrementButton*);
- il pulsante di incremento (*incrementButton*);
- il pulsante di reset (*resetButton*).

Nota n.1: Siccome avevo bisogno di un valore di partenza, ma non volevo che questo fosse ciò che veniva decrementato/incrementato, quando ho dichiarato `this.startingValue` ho fatto in modo che questo valore rimanesse fisso dichiarando subito dopo `this.currentValue = startingValue`.

Così facendo, il valore di `currentValue` sarà uguale a quello di `startingValue` quando viene inizializzata la classe, ma sarà poi ciò che verrà modificato dalle logiche di decremento/incremento lasciando `startingValue` invariato.



Nota n.2: Come si nota, nella classe non è presente nulla che faccia riferimento al cambio di contatore. Questo perché il passaggio dal contatore “standard” a quello “random” (e viceversa) è già gestito dal file *changeCounter.js* e, per non aggiungere un ulteriore livello di complessità al codice, non ho incluso alcuna istanza che facesse riferimento allo switch all'interno della classe *Counter*.

Al costruttore seguono i metodi per:

- inizializzare il display del contatore (`this.counterSetup()`);
- definire la logica del decremento (`this.numberDecrement()`);
- definire la logica dell'incremento (`this.numberIncrement()`);
- definire il reset del contatore (`resetCounter()`).

counterSetup()

Questo metodo è formato da un “`textContent`” che fa sì che nel display del contatore venga mostrato il numero da cui parte il conteggio (`Il numero attuale è ${this.currentValue}`); al momento dell'inizializzazione, siccome nella classe Counter viene dichiarato che `this.currentValue = startingValue`, viene mostrato il valore di partenza.

this.numberDecrement() e this.numberIncrement()

Questi sono i metodi che gestiscono il decremento e l'incremento del valore corrente: vengono innescati da un "event listener" (che "sente" il click sul pulsante corrispondente) e mandano il loro risultato al display grazie al metodo "this.counterDisplay.textContent = `Il numero attuale è \${this.currentValue}`".

Il decremento e l'incremento sono ottenuti usando gli operatori di decremento e di incremento prefissi alla variabile (`--this.currentValue` e `++this.currentValue`). Ho deciso di metterli prefissi perché volevo che l'operatore modificasse il valore della variabile prima di aggiornarla.



this.resetCounter()

Quest'ultimo metodo serve per "resettare" il contatore e far ripartire il conteggio. Viene innescato da un "event listener" che "ascolta" se avviene un click sul pulsante di reset e usa una logica *if/else if* per eseguire un'azione diversa a seconda che ci si trovi nel contatore "standard" o nel contatore "random" (il programma lo capisce controllando se il pulsante contiene la classe "standardReset" oppure "randomReset").

Come funziona `this.resetCounter()`:

- se il pulsante di reset ha la classe `standardReset`, il valore corrente (`this.currentValue`) viene riportato al valore di partenza (`this.startingValue`, ovvero 0) e viene aggiornato il testo all'interno del display;
- se il pulsante di reset ha la classe `randomReset`, viene inizializzata una variabile `newRandomValue` in cui viene generato casualmente un nuovo valore compreso tra 1 e 100; `newRandomValue` viene, poi, passato a `this.currentValue` e viene aggiornato di conseguenza il testo all'interno del display.

Step 3: i contatori

Una volta creata la classe `Counter`, ho finalmente generato i due contatori dichiarando le due costanti (`standardCounter` per il contatore che parte da 0 e `randomCounter` per il contatore che parte da un numero casuale) come due istanze della classe `Counter`; ho poi assegnato ai parametri dei costruttori di `standardCounter` e di `randomCounter` le proprietà dell'oggetto `documentElements` (che, come visto in precedenza, racchiude in sé gli elementi del DOM creati con la funzione `createDocument()`).

Lo *startingValue* dei due contatori, invece, viene impostato in modo diverso a seconda del contatore:

- in *standardCounter*, viene semplicemente indicato come 0;
- in *randomCounter*, lo *startingValue* viene impostato con "Math.random" (`Math.floor(Math.random() * 100)`).

Step 4: il cambio contatore

La logica per il passaggio da un contatore all'altro è contenuta nel file `changeCounter.js`. Questa è molto semplice e sfrutta la "collaborazione" tra CSS e JavaScript:

- durante la creazione del DOM, ho assegnato una classe specifica ai "contenitori" dei due contatori ("counterStandard" e "counterRandom");
- ho anche assegnato la classe "active" a uno dei due;
- nel CSS, poi, ho fatto in modo che sia l'elemento con classe `counterStandard` che quello con classe `counterRandom` fossero nascosti e che venisse mostrato solamente quello con la classe `active`.

Per passare dal contatore “standard” a quello “random”, ho aggiunto questa logica al file `changeCounter.js`:

- ho aggiunto un “event listener” al pulsante `changeToRandom`;
- la funzione innescata dall’“event listener” controlla se il contenitore del contatore standard (`standardCounterContainer`) ha la classe `active`;
- se ce l’ha, allora la classe `active` viene rimossa dallo `standardCounterContainer` e viene aggiunta al contenitore del contatore random (`randomCounterContainer`).

Ho usato la stessa logica (ma a contatori invertiti) per passare dal contatore “random” a quello “standard”.





**Grazie
per l'attenzione**

oooo —

+ oooo