MATTEO VAVUSOTTO

MATTEOVAVUSOTTO.GITHUB.IO/COUNTERJS/

COUNTERJS

È stato un piacere lavorare su questo progetto, poiché, dopo aver assimilato tanta teoria, si ha l'opportunità di confrontarsi concretamente con le proprie capacità. Sarei poco sincero se non ammettessi di aver dovuto rivedere le slide per completare con successo il progetto.

Sono soddisfatto del lavoro fatto, credo sia essenziale ma completo.

GitHub

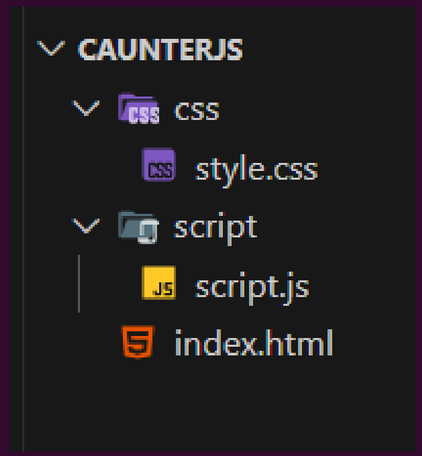
CARTELLA/FILE

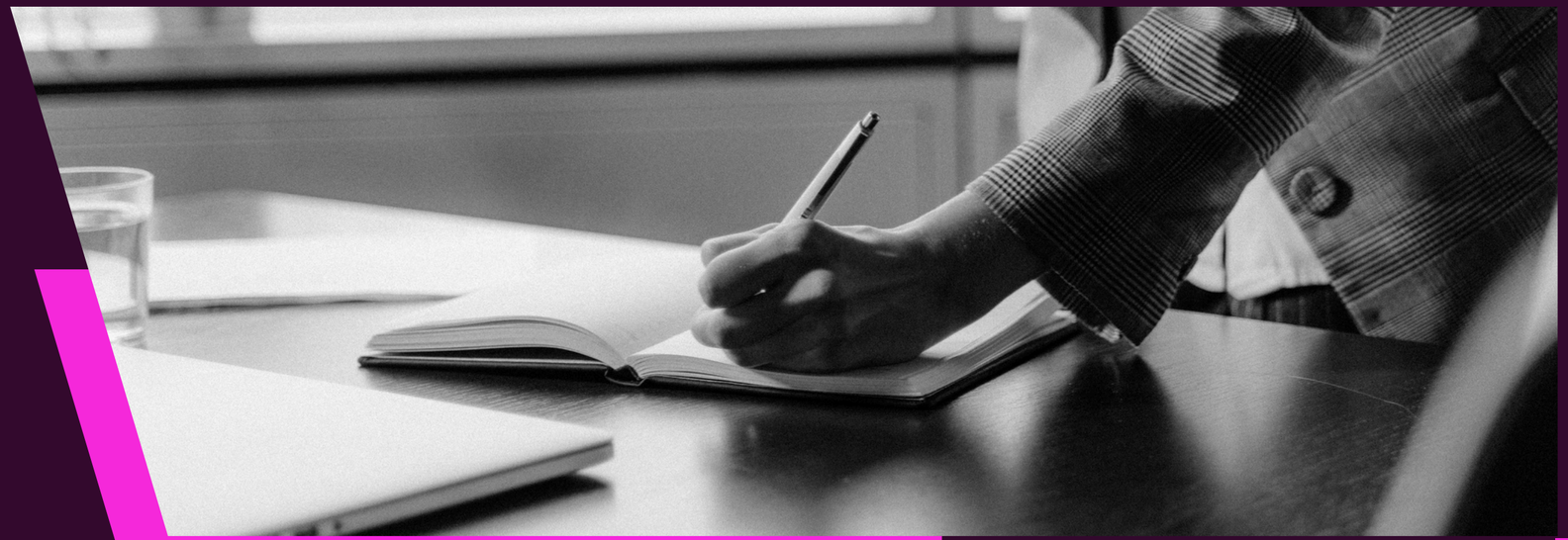
01

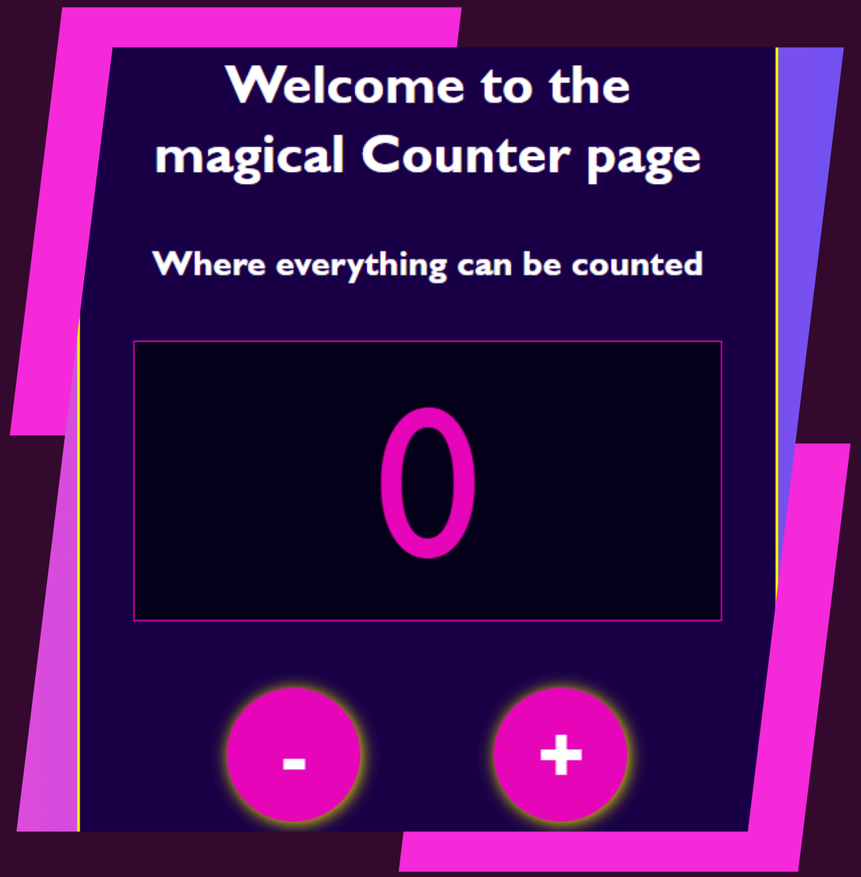
Il file "style.css" è stato utilizzato per definire le classi che conferiscono stile alla mia pagina. Inizialmente, ho creato il contenitore; successivamente, una volta riempito, ho sviluppato le classi necessarie per gestire il contenuto in modo efficace.

02

In "Script.js", ho inizialmente implementato l'evento “DOMContentLoaded” per interagire con il DOM una volta che è completamente carico. Successivamente, ho definito le mie variabili e ho progettato un costruttore per gestire dinamicamente il contenuto.







COUNTER VIEW

L'HTML del counter è estremamente semplificato. Ho creato un "container" che funge da contenitore, permettendo di aggiungere successivamente altri elementi in modo dinamico.
Sempre a partire dal codice HTML, ho creato un div in cui ho inserito il testo visualizzato utilizzando i tag h1 e h3.

Il contenitore è posizionato al centro della pagina tramite l'uso di Flexbox. Nel frattempo, il contenuto al suo interno è stato suddiviso e distribuito in una grid strutturata a due colonne, permettendo così una disposizione ordinata e bilanciata degli elementi, facilitando la lettura e migliorando l'estetica complessiva della pagina.

STYLE.CSS

01

Ho sviluppato la prima parte del codice dedicata alla struttura del contatore, concentrandomi sul ".container" e sul ".view-text".
Quest' ultimo è stato utilizzato come contenitore principale, al quale ho successivamente collegato il codice in modo dinamico.

02

La seconda parte del codice, distinta dall'altra tramite l'uso di ".js-*", consiste interamente in codice CSS, il quale è progettato per fornire stile agli elementi aggiunti in modo dinamico tramite JavaScript.

```css
.js-display{
  grid-column: span 2;
  font-size: 8rem;
  width: auto;
  color: #ea07b9;
  margin: 1rem 1.5rem;
  background-color: rgb(9, 0, 27);
  border: 1px solid rgb(234, 7, 185);
}

.js-view{
  grid-column: span 2;
  display: flex;
  justify-content: center;
  gap: 5rem;
  margin: 1.5rem;

}

.js-button{
  background-color: #ea07b9;
  color:white;
  width: 5rem;
  height: 5rem;
  display:flex;
  justify-content: center;
  align-items: center;
  font-weight: bold;
  font-size: 3rem;
  border-radius: 50%;
  border: none;
  cursor: pointer;
  box-shadow: 1px 1px 10px rgb(255, 247, 0);
  transition: all 0.1s ease;
}

.js-button:active {
  transform: scale(0.95);
  box-shadow: none;
  transition: all 0.1s ease-in-out;
}
```

```css
.container{
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
}

.view-text{
  display: grid;
  grid-template-columns: 1fr 1fr;
  justify-content: center;
  align-items: center;
  text-align: center;
  width: 60vh;
  height: auto;
  max-width: 700px;
  background-color: rgb(26, 2, 73);
  color: rgb(255, 255, 255);
  font-size:2rem;
  padding:.5rem;
  border: 2px solid rgb(255, 247, 0);
}
```

SCRIPT.JS

```
script.js > ...
document.addEventListener("DOMContentLoaded", () =>{
  class updateView{
    constructor(counter){
      this.counter = counter;

    }
    increment(){
      this.counter++;
      this.updateDisplay();
    }
    decrement(){
      this.counter--;
      this.updateDisplay();
    }
    updateDisplay(){
      return display.innerText = this.counter;
    }

  }
```

01

Ho inizializzato il contenuto utilizzando l'evento “DOMContentLoaded” per garantire che tutti gli elementi siano completamente caricati. Inoltre, ho implementato un costruttore che accetta il valore della variabile “counter”. A seconda del metodo invocato, eseguo operazioni di somma o sottrazione, visualizzando successivamente il risultato.

02

Ho impiegato il metodo “createElement” per generare dinamicamente i pulsanti delle operazioni, il display di output e un contenitore per gestire la posizione dei bottoni. A ciascun elemento è stata assegnata una classe specifica, consentendo poi l’utilizzo di “appendChild” per aggiungere gli elementi all'HTML.

```
const container = document.querySelector('.view-text');
const minButton = document.createElement('button');
const plusButton = document.createElement('button');
const display = document.createElement('span');
const buttonView = document.createElement('div');

buttonView.classList.add('js-view');
plusButton.textContent = '+';
plusButton.classList.add('js-button');
minButton.textContent = '-';
minButton.classList.add('js-button');
display.textContent = '0';
display.classList.add('js-display');

container.appendChild(display);
container.appendChild(buttonView);
buttonView.appendChild(minButton);
buttonView.appendChild(plusButton);


let counter = 0;
const view = new updateView(counter);

plusButton.addEventListener('click', () => {
  view.increment(counter);
  view.updateDisplay();
});

minButton.addEventListener('click', () => {
  view.decrement(counter);
  view.updateDisplay();
});

})
```

THANK YOU

LINK AL REPOSITORI GITHUB

LINK AL PAGES PER LA DEMO