JavaScript: autocomplete con validazione

In questo articolo vedremo come implementare la funzionalità di autocomplete con validazione dei dati inseriti in JavaScript.

1. La struttura HTML
2. Gli stili CSS
3. Il codice JavaScript
4. Demo

La struttura HTML

Partiamo dalla seguente struttura HTML.

```
<form action="" method="get" class="autosuggest-
form">
        <p>Pick a value between those suggested.</p>
        <div>
            <input type="text" class="autosuggest-
input" placeholder="Search...">
            <button type="button" class="autosuggest-
btn">Search</button>
        </div>
        <div class="suggestions">
            <ul></ul>
        </div>
    </form>
```

Gli stili CSS

Iniziamo col definire gli stili CSS globali.

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

html {
    font-size: 16px;
}

body  {
    margin: 2.5rem auto;
    max-width: 40%;
    padding: 0 1rem;
    font: 1rem/1.5 sans-serif;
    background-color: #fff;
    color: #000;
}

::placeholder {
    color: #555;
}

:focus,
:active {
    outline-style: none;
    box-shadow: none;
}

button {
    appearance: none;
    cursor: pointer;
```

```
    border: none;
    background-color: #000;
    color: #fff;
    padding: 0.85rem 1.25rem;
    letter-spacing: .1em;
    text-transform: uppercase;
}

input {
    padding: 0.85rem;
    background-color: #fff;
    border: 1px solid #000;
}
```

Ora definiamo le classi CSS che assegneranno gli stili al campo di testo principale, compresa l'evidenziazione dell'errore e la sua descrizione testuale.

```
input.autosuggest-input {
    min-width: 250px;
}

.error {
    color: #d00;
    margin: 1rem 0;
}

input.input-error {
    border: 1px solid #d00;
    background-color: #ffebee;
}
```

Il contrasto del colore rosso sullo sfondo bianco è ancora accettabile dal punto di vista della leggibilità, così come lo sfondo rosato del campo di

input quando viene segnalato un errore. È sempre una buona pratica verificare la leggibilità anche per i messaggi di validazione.

Il contenitore dei suggerimenti verrà posizionato in modo assoluto in relazione al form principale con questi stili.

```
form {
    position: relative;
}

.suggestions {
    position: absolute;
    top: auto;
    left: 0;
    background-color: #fff;
    border-width: 0 1px 1px 1px;
    border-color: #000;
    border-style: solid;
    min-width: 250px;
    display: none;
}

.suggestions ul {
    list-style: none;
}
.suggestions ul li {
    display: block;
    padding: 0.5rem;
    cursor: pointer;
}
.suggestions ul li:hover {
    background-color: #f5f5f5;
}
```

La proprietà `top` impostata sul valore `auto` posizionerà l'elemento sempre dopo l'elemento che lo precede nella struttura HTML, ma solo se quest'ultimo elemento non è posizionato. Nel nostro caso il box dei suggerimenti si posizionerà dopo l'elemento `div` che contiene il campo di input.

Il codice JavaScript

Definiremo varie funzioni che, eseguite nell'ordine stabilito, ci permetteranno di reperire i suggerimenti via AJAX, filtrarli in base ai caratteri inseriti dall'utente e infine validarli, ossia segnalare all'utente che il testo inserito non è presente nei suggerimenti indicati.

Partiamo dal reperimento dei dati via AJAX.

```
const getData = async () => {
        try {
            const data = await
fetch('products.json');
            const res = await data.json();
            return res.products.map(product => {
return { title: product.title }; });
        } catch(err) {
            return [];
        }
    };
```

Tramite il modello async/await e le Fetch API, reperiamo l'array di dati richiesto e lo trasformiamo in un nuovo array di oggetti aventi ciascuno la sola proprietà `title`. È su questa proprietà infatti che andremo ad implementare il filtro e la validazione sui risultati.

Creiamo ora tre funzioni di utility per permetterci di gestire la visibilità e resettare i contenuti del blocco dei suggerimenti.

```
const showSuggeestions = () => {

document.querySelector('.suggestions').style.display
= 'block';
    };

    const hideSuggestions = () => {

document.querySelector('.suggestions').style.display
= 'none';
    };

    const clearSuggestions = () => {
        document.querySelector('.suggestions
ul').innerHTML = '';
    };
```

Quindi definiamo una funzione che, data una stringa testuale di input, crei un nuovo elemento DOM `li` da aggiungere alla lista dei suggerimenti con il testo specificato come parametro.

```
const createSuggestion = text => {
        const target =
document.querySelector('.suggestions ul');
        const suggestion =
document.createElement('li');
        suggestion.className = 'suggestion';
        suggestion.innerText = text;
        target.appendChild(suggestion);
    };
```

Per velocizzare l'accesso ai suggerimenti e per gestire il reload della pagina, salviamo i dati nel web storage usando una funzione specifica.

```
const setData = async () => {
        let data = [];
        try {
            data = await getData();
        } catch(err) {

        }
        sessionStorage.setItem('suggestions',
JSON.stringify(data));
    };
```

L'array di suggerimenti, una volta reperito, viene salvato come stringa JSON nel web storage. A questo punto possiamo filtrare questo array con il termine digitato dall'utente:

```
const getSuggestions = term => {
        const data =
sessionStorage.getItem('suggestions');
        if(data === null) {
            return [];
        }
        const suggestions = JSON.parse(data);
        return suggestions.filter(sugg => {
            return
sugg.title.toLowerCase().includes(term.toLowerCase())
;
        });
    };
```

L'array ottenuto dal web storage viene filtrato confrontando la forma minuscola del termine inserito dall'utente con la forma minuscola della proprietà `title`. Questa conversione è necessaria perchè il metodo `includes()` delle stringhe è sensibile alle maiuscole e minuscole.

L'azione principale ha luogo sul campo di input, quindi dobbiamo intercettare l'inserimento del testo su questo elemento.

```
const handleInput = () => {
        const input =
document.querySelector('.autosuggest-input');
        input.addEventListener('keyup', function() {
            const value = this.value;
            if(value && value.length >= 3 &&
!/^\s+$/.test(value)) {
                hideSuggestions();
                clearSuggestions();
                const suggestions =
getSuggestions(value);
                if(suggestions.length > 0) {
                    for(const sugg of suggestions) {
                        createSuggestion(sugg.title);
                    }
                    showSuggeestions();
                }
            }  else {
                hideSuggestions();
                clearSuggestions();
            }
        }, false);
    };
```

Se l'utente ha inserito almeno tre caratteri validi (non spazi), resettiamo il box dei suggerimenti e filtriamo l'array dei suggerimenti. Qualora siano presenti dei suggerimenti, creiamo e inseriamo gli elementi `li` corrispondenti e mostriamo il box dei suggerimenti.

Avendo popolato la lista dei suggerimenti, dobbiamo ora fare in modo che cliccando su una voce, il suo testo venga inserito come valore del campo di

input.

```
const handleSuggestion = () => {
        const suggestionsContainer =
document.querySelector('.suggestions');

suggestionsContainer.addEventListener('click', e => {
            const el = e.target;
            if(el.matches('.suggestion')) {
                document.querySelector('.autosuggest-
input').value = el.innerText;
            }
        }, false);
    };
```

Poichè le voci della lista sono create dinamicamente, dobbiamo usare la event delegation sfruttando la proprietà `target` dell'oggetto `event` che ci offre un riferimento all'elemento `li` corrente, da cui possiamo estrarre il testo e inserirlo come valore del campo di input.

Ora possiamo implementare la validazione dei dati legandola al click sul pulsante del form.

```
const validateSuggestions = () => {
        const data =
sessionStorage.getItem('suggestions');
        if(data === null) {
            return [];
        }
        const err = document.querySelector('.error');
        if(err !== null) {
            err.remove();
        }
        const suggestions = JSON.parse(data);
```

```
        const input =
document.querySelector('.autosuggest-input');
        input.classList.remove('input-error');
        hideSuggestions();
        clearSuggestions();
        const term = input.value;
        const found = suggestions.find(sgg =>
sgg.title === term);
        if(!found) {
            const error =
document.createElement('div');
            error.className = 'error';
            error.innerText = 'You must choose one of
the suggested values.';
            document.querySelector('.autosuggest-
form').appendChild(error);
            input.classList.add('input-error');
            return false;
        }
        return true;
    };

    const handleSubmit = () => {
        const btn =
document.querySelector('.autosuggest-btn');
        btn.addEventListener('click', () => {
            validateSuggestions();
        }, false);
    };
```

La validazione avviene resettando prima il messaggio di errore e gli stili aggiuntivi del campo di errore e quindi confrontando il valore inserito nel campo con l'array di suggerimenti. Se il metodo `find()` non trova una

corrispondenza (ossia restituisce `undefined`), inseriamo il messaggio di errore e aggiungiamo gli stili dell'errore al campo di input.

Infine, eseguiamo il nostro codice quando il DOM è stato completamente caricato.

```
document.addEventListener('DOMContentLoaded', () => {
        (async() => {
            handleSuggestion();
            await setData();
            handleInput();
            handleSubmit();
        })();
    }, false);
```

Demo

JavaScript: autocomplete with validation