**Create filter.js:**

This script adds a live search filter to a table (#infoTable). When the user types in the input field (#search), it:

Waits until the user stops typing for 300ms (debounceTime)

Checks if the typed value is different from the last one (distinctUntilChanged)

Then filters the table rows to only show matching rows.

**1. Import from RxJS**

const { fromEvent } = rxjs;

const { debounceTime, map, distinctUntilChanged } = rxjs.operators;

**Explanation:**

fromEvent: Converts DOM events (like key presses) into a stream.

map: Transforms the event object into a value (e.g., the input’s value).

debounceTime(300): Waits 300ms after the user stops typing before proceeding. Prevents excessive filtering.

distinctUntilChanged(): Ignores repeated values (e.g., user types and deletes to the same value again).

**2. Grab DOM Elements**

const input = document.getElementById('search');

const rows = document.querySelectorAll('#infoTable tbody tr');

**Explanation:**

input: The search input field

rows: All the rows in the table that you want to show/hide based on the search

**3. Filtering Logic**

function filterRows(value) {

rows.forEach(row => {

const text = row.textContent.toLowerCase();

row.style.display = text.includes(value) ? '' : 'none';

});

}

**Explanation:**

Checks if the row’s text includes the search string.

If it does, shows the row. Otherwise, hides it.

**4. Creating the Event Stream**

fromEvent(input, 'input').pipe(

map(e => e.target.value.toLowerCase()),

debounceTime(300),

distinctUntilChanged()

).subscribe(filterRows);

**Explanation:**

What this RxJS chain does:

fromEvent(input, 'input'): Listens to input events (every keystroke).

.pipe(...):

map: Turns the event into the input’s text, in lowercase.

debounceTime(300): Waits 300ms of no typing before moving on.

distinctUntilChanged: Skips filtering if the value hasn’t changed.

.subscribe(filterRows): Runs the filtering function with the processed search value.

RxJS is perfect when:

You want to compose multiple asynchronous operations cleanly

You need fine control over timing, cancellation, or conditional logic

You’re already using a reactive UI framework (like Angular)

Plain JavaScript Version (without RxJS):

const input = document.getElementById('search');

const rows = document.querySelectorAll('#infoTable tbody tr');

let timeout = null;

input.addEventListener('input', function () {

const value = input.value.toLowerCase();

clearTimeout(timeout);

timeout = setTimeout(() => {

rows.forEach(row => {

const text = row.textContent.toLowerCase();

row.style.display = text.includes(value) ? '' : 'none';

});

}, 300);

});