**Create contact.js**

This is an RxJS-powered contact form handler. It:

Waits for the form to be submitted

Prevents the default behavior (page reload)

Validates phone, email, and zip code inputs

Shows an error message if validation fails

Otherwise, sends the data via fetch to a server

Shows a success message upon response

All this is done reactively using RxJS, rather than imperative addEventListener logic.

**Step 1: Import RxJS operators**

const { fromEvent } = rxjs;

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

**Explanation:**

These are RxJS tools:

fromEvent: turns the form submit into a stream

tap: side effects (like preventing the default)

debounceTime: delays processing after submission

map: transforms the data

**Step 2: DOM ready handler**

document.addEventListener('DOMContentLoaded', () => {

**Explanation:**

Waits until the DOM is fully loaded before touching any elements.

**Step 3: Grabbing elements**

const form = document.getElementById('myForm');

const responseDiv = document.getElementById('response');

const phoneInput = document.getElementById('phone');

const emailInput = document.getElementById('email');

const zipInput = document.getElementById('zip');

**Explanation:**

Gets form and input elements from the page.

**Step 4: Failsafe check**

if (!form) {

console.error('Form not found!');

return;

}

**Explanation:**

Stops the script if the form wasn’t found.

RxJS Event Stream

fromEvent(form, 'submit').pipe(

Starts listening for form submit events.

**Step 5: Add tap()**

tap(e => {

e.preventDefault();

console.log('Form submitted via RxJS');

}),

**Explanation:**

Stops the form from reloading the page and logs to the console. tap is used because this is a side effect, not a transformation.

**Step 6: Add debounceTime**

debounceTime(300),

**Explanation:**

Waits 300ms after the last event before proceeding. Prevents users from submitting the form twice by double-clicking.

**Step 7: Add map()**

map(() => {

const phone = phoneInput.value.trim();

const email = emailInput.value.trim();

const zip = zipInput.value.trim();

const errors = [];

if (!/^\d{10}$/.test(phone)) {

errors.push('Phone must be 10 digits.');

}

if (!/^[\w.-]+@[\w.-]+\.[a-zA-Z]{2,}$/.test(email)) {

errors.push('Email is invalid.');

}

if (!/^\d{5}$/.test(zip)) {

errors.push('Zip must be 5 digits.');

}

return { phone, email, zip, errors };

})

**Explanation:**

Transforms the event into a cleaned and validated payload:

Validates:

Phone = exactly 10 digits

Email = standard pattern

Zip = 5 digits

Returns the data + validation errors

**Step 8: Subscription (async handler)**

).subscribe(async ({ phone, email, zip, errors }) => {

**Explanation:**

Waits for the transformed data and then executes the next steps.

**Step 9: Show validation errors**

if (errors.length > 0) {

responseDiv.innerHTML = `<div class="alert alert-danger">${errors.join('<br>')}</div>`;

return;

}

**Explanation:**

If there are any errors, show them to the user and exit early.

**Step 10: Send data via fetch**

const res = await fetch('/submit-contact', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ phone, email, zip })

});

**Explanation:**

Sends the valid data to the server using fetch.

**Step 11: Handle server response**

const msg = await res.text();

responseDiv.innerHTML = `<div class="alert alert-success">${msg}</div>`;

form.reset();

**Explanation:**

Displays the server's success message

Resets the form

**Vanilla JavaScript Version (Without RxJS)**

document.addEventListener('DOMContentLoaded', () => {

const form = document.getElementById('myForm');

const responseDiv = document.getElementById('response');

form.addEventListener('submit', async (e) => {

e.preventDefault();

const phone = form.phone.value.trim();

const email = form.email.value.trim();

const zip = form.zip.value.trim();

const errors = [];

if (!/^\d{10}$/.test(phone)) errors.push('Phone must be 10 digits.');

if (!/^[\w.-]+@[\w.-]+\.[a-zA-Z]{2,}$/.test(email)) errors.push('Email is invalid.');

if (!/^\d{5}$/.test(zip)) errors.push('Zip must be 5 digits.');

if (errors.length > 0) {

responseDiv.innerHTML = `<div class="alert alert-danger">${errors.join('<br>')}</div>`;

return;

}

const res = await fetch('/submit-contact', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ phone, email, zip })

});

const msg = await res.text();

responseDiv.innerHTML = `<div class="alert alert-success">${msg}</div>`;

form.reset();

});

});