CLASE 10 · 10/06/2026

DOM, validación de formularios e integración final

Una clase enfocada en validar interfaces reales: errores por campo, feedback visual, funciones auxiliares y código integrado.

VIRTUAL JAVASCRIPT FORMULARIOS

PROBLEMA GUÍA

Un formulario puede verse bien y funcionar mal

- Puede aceptar campos vacíos o cargados solo con espacios.
- Puede aceptar un email mal escrito.
- Puede mostrar un error demasiado general.
- Puede dejar errores viejos aunque el usuario corrija los datos.
- Puede borrar lo cargado aunque todavía haya errores.

OBJETIVO

Qué vamos a resolver hoy

- Construir un formulario con campos obligatorios.
- Detectar errores antes de aceptar los datos.
- Mostrar mensajes específicos cerca de cada campo.
- Marcar visualmente los campos con problemas.
- Separar la lógica en funciones para que el código sea mantenible.

CASO GUÍA

Formulario de solicitud de contacto

Escenario

Una página tiene un formulario para que una persona deje sus datos y una consulta.

1. Nombre completo.
2. Email.
3. Tipo de consulta.
4. Mensaje.
5. Aceptación de condiciones.

PROBLEMA

Validar no es solamente preguntar si algo está vacío

Dato faltante

El usuario deja un campo obligatorio sin completar.

Dato débil

El usuario escribe muy poco o solo espacios.

Dato inválido

El formato no tiene sentido, por ejemplo un email sin @.

Una buena validación debe orientar, no solo bloquear.

HTML BASE

Primero: una estructura simple y clara

HTML EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head><meta charset="UTF-8"><title>Formulario base</title></head>
<body>
  <form>
    <h1>Contacto</h1>
    <label>Nombre</label>
    <input type="text" placeholder="Ej: Ana Pérez">
    <label>Email</label>
    <input type="email" placeholder="Ej: ana@mail.com">
    <label>Consulta</label>
    <textarea placeholder="Escribí tu consulta"></textarea>
    <button type="submit">Enviar</button>
  </form>
</body>
</html>
```

- El formulario debe tener campos identificables.
- Los labels ayudan a entender cada dato.
- El botón de envío debe ser **type="submit"**.
- Después agregaremos mensajes de error.

CSS BASE

El diseño también prepara la validación

HTML + CSS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Formulario con estilo</title>
<style>
body { font-family: Arial, sans-serif; background:#f3f5f7; padding:30px; }
form { max-width:480px; margin:auto; background:white; padding:24px; border-radius:14px; }
label { display:block; margin-top:12px; font-weight:bold; }
input, textarea, button { width:100%; padding:10px; margin-top:6px; box-sizing:border-box; }
button { margin-top:16px; cursor:pointer; }
</style>
</head>
<body>
<form>
  <h1>Contacto</h1>
  <label>Nombre</label><input type="text">
  <label>Email</label><input type="email">
  <label>Consulta</label><textarea></textarea>
```

- Un formulario legible reduce errores.
- Los campos deben tener separación suficiente.
- Los mensajes de error deben tener un lugar claro.
- El diseño no reemplaza la validación, la acompaña.

REGLAS

Qué vamos a validar

Nombre	Obligatorio y con al menos 3 caracteres reales.
Email	Obligatorio y con una validación básica de formato.
Tipo	Debe seleccionarse una opción válida del select.
Mensaje	Debe tener una longitud mínima para evitar consultas vacías.
Condiciones	Debe estar marcado si el formulario lo requiere.

FEEDBACK

Error general vs error por campo

Error general

“Hay errores en el formulario”.
Es rápido, pero no indica exactamente dónde corregir.

Error por campo

“El email es obligatorio”.
Es más claro porque aparece cerca del campo afectado.

En esta clase vamos a usar **errores por campo**.

MENSAJES

Reservar un lugar para cada error

HTML EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head><meta charset="UTF-8"><title>Errores por campo</title></head>
<body>
<form>
  <label for="nombre">Nombre</label>
  <input id="nombre" type="text">
  <small id="errorNombre"></small>

  <label for="email">Email</label>
  <input id="email" type="email">
  <small id="errorEmail"></small>
</form>
</body>
</html>
```

- Cada campo tiene su propio **small**.
- JavaScript escribirá ahí el mensaje de error.
- El usuario entiende qué dato debe corregir.

ESTADOS VISUALES

El error también debe verse

HTML + CSS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Estados visuales</title>
<style>
body { font-family: Arial, sans-serif; padding:30px; }
input { display:block; width:280px; padding:10px; margin:6px 0; }
small { color:#b00020; font-weight:bold; }
.campo-error { border:2px solid #b00020; }
.campo-ok { border:2px solid #006b3c; }
</style>
</head>
<body>
<label>Email</label>
<input class="campo-error" value="email sin arroba">
<small>Ingresá un email válido.</small>
</body>
</html>
```

- El texto explica el error.
- El borde marca el campo afectado.
- La clase CSS define el estado visual.
- JavaScript solo agrega o quita clases.

ESTRATEGIA

Antes de validar, limpiar

Una validación se puede ejecutar muchas veces. Si el usuario corrige un campo, no deben quedar errores anteriores.

- Limpiar textos de error.
- Quitar clases de error.
- Quitar mensajes de éxito anteriores.
- Volver a evaluar desde cero.

FUNCIÓN

limpiarErrores()

HTML + CSS + JS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8"><title>Limpiar errores</title>
<style>.campo-error{border:2px solid #b00020} small{color:#b00020}</style>
</head>
<body>
<label>Nombre</label>
<input id="nombre" class="campo-error" value="">
<small id="errorNombre">El nombre es obligatorio.</small>
<button id="btn">Limpiar errores</button>
<script>
const nombre = document.querySelector("#nombre");
const errorNombre = document.querySelector("#errorNombre");
const btn = document.querySelector("#btn");

function limpiarErrores() {
  errorNombre.textContent = "";
  nombre.classList.remove("campo-error");
```

Responsabilidad

Esta función deja la interfaz lista para una nueva validación.

1. Borra mensajes.
2. Quita clases visuales.
3. Evita errores viejos.

FUNCIÓN

mostrarError()

HTML + CSS + JS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8"><title>Mostrar error</title>
<style>.campo-error{border:2px solid #b00020} small{color:#b00020;font-weight:bold}</style>
</head>
<body>
<label>Email</label>
<input id="email" type="email">
<small id="errorEmail"></small>
<button id="btn">Mostrar error</button>
<script>
const email = document.querySelector("#email");
const errorEmail = document.querySelector("#errorEmail");
const btn = document.querySelector("#btn");

function mostrarError(campo, contenedor, mensaje) {
  campo.classList.add("campo-error");
  contenedor.textContent = mensaje;
```

- Recibe el campo afectado.
- Recibe dónde mostrar el mensaje.
- Recibe el texto del error.
- Evita repetir código para cada campo.

PATRÓN

La variable valido

HTML + JS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head><meta charset="UTF-8"><title>Variable valido</title></head>
<body>
<button id="btn">Validar ejemplo</button>
<p id="resultado"></p>
<script>
const btn = document.querySelector("#btn");
const resultado = document.querySelector("#resultado");

btn.addEventListener("click", function() {
  let valido = true;
  const nombre = "Ana";
  const email = "ana@mail.com";

  if (nombre.length < 3) valido = false;
  if (!email.includes("@")) valido = false;

  resultado.textContent = valido ? "Formulario válido" : "Formulario con errores";
```

Idea

La variable empieza en **true**. Si aparece cualquier error, pasa a **false**.

Al final, decide si se muestra éxito o no.

VALIDACIÓN

Nombre: no alcanza con que no esté vacío

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Validar nombre</title></head><body>
<input id="nombre" placeholder="Nombre"><button id="btn">Validar</button><p id="msg"></p>
<script>
const nombre=document.querySelector("#nombre"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  if(nombre.value.trim().length < 3){ msg.textContent="El nombre debe tener al menos 3 caracteres."; return; }
  msg.textContent="Nombre válido.";
});
</script></body></html>
```

- Se usa **trim()** para eliminar espacios.
- Se revisa una longitud mínima.
- Se corta con **return** si hay error.

VALIDACIÓN

Email: primero obligatorio

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Email obligatorio</title></head><body>
<input id="email" placeholder="Email"><button id="btn">Validar</button><p id="msg"></p>
<script>
const email=document.querySelector("#email"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  if(email.value.trim() === ""){ msg.textContent="El email es obligatorio."; return; }
  msg.textContent="Email cargado.";
});
</script></body></html>
```

Conviene validar por etapas:

- Primero, si está vacío.
- Después, si el formato parece correcto.
- Así los mensajes son más precisos.

VALIDACIÓN

Email: validación básica de formato

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Email formato</title></head><body>
<input id="email" placeholder="Email"><button id="btn">Validar</button><p id="msg"></p>
<script>
const email=document.querySelector("#email"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  const valor = email.value.trim();
  if(valor === ""){ msg.textContent="El email es obligatorio."; return; }
  if(!valor.includes("@")){ msg.textContent="Ingresá un email válido."; return; }
  msg.textContent="Email válido.";
});
</script></body></html>
```

Aclaración

Esta no es una validación perfecta de email. Es una validación introductoria para practicar lógica.

VALIDACIÓN

Consulta: longitud mínima

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Consulta mínima</title></head><body>
<textarea id="consulta" placeholder="Escribí tu consulta"></textarea><br><button id="btn">Validar</button><p id="msg">
</p>
<script>
const consulta=document.querySelector("#consulta"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  if(consulta.value.trim().length < 10){ msg.textContent="La consulta debe tener al menos 10 caracteres."; return; }
  msg.textContent="Consulta válida.";
});
</script></body></html>
```

- Evita mensajes como “hola”.
- Obliga a ingresar un mínimo de información.
- La longitud mínima depende del caso.

VALIDACIÓN

Select: evitar la opción vacía

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Select obligatorio</title></head><body>
<select id="tipo"><option value="">Seleccionar tipo</option><option value="consulta">Consulta</option><option
value="turno">Turno</option></select>
<button id="btn">Validar</button><p id="msg"></p>
<script>
const tipo=document.querySelector("#tipo"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  if(tipo.value === ""){ msg.textContent="Seleccioná un tipo de consulta."; return; }
  msg.textContent="Tipo seleccionado: " + tipo.value;
});
</script></body></html>
```

- La primera opción suele tener **value=""**.
- Si el valor sigue vacío, no se seleccionó nada real.
- Este patrón es muy usado en formularios.

VALIDACIÓN

Checkbox: usar checked

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Checkbox obligatorio</title></head><body>
<label><input id="acepta" type="checkbox"> Acepto las condiciones</label><br>
<button id="btn">Validar</button><p id="msg"></p>
<script>
const acepta=document.querySelector("#acepta"); const msg=document.querySelector("#msg");
document.querySelector("#btn").addEventListener("click",()=>{
  if(!acepta.checked){ msg.textContent="Debés aceptar las condiciones."; return; }
  msg.textContent="Condiciones aceptadas.";
});
</script></body></html>
```

- En un checkbox no alcanza con leer **value**.
- La propiedad importante es **checked**.
- Devuelve verdadero o falso.

ÉXITO

Qué pasa cuando todo está bien

Cuando el formulario es válido, la interfaz debería:

- Mostrar un mensaje de confirmación claro.
- Aplicar una clase visual de éxito.
- Opcionalmente limpiar los campos.
- No borrar datos si todavía hay errores.

Regla práctica

El formulario solo se limpia cuando el envío fue válido.

CUIDADO

No limpiar el formulario si hay errores

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Reset correcto</title></head><body>
<form id="form"><input id="nombre" placeholder="Nombre"><button>Enviar</button></form><p id="msg"></p>
<script>
const form=document.querySelector("#form"); const nombre=document.querySelector("#nombre"); const
msg=document.querySelector("#msg");
form.addEventListener("submit", function(e){
  e.preventDefault();
  if(nombre.value.trim() === "") { msg.textContent="Corregí el nombre antes de enviar."; return; }
  msg.textContent="Enviado correctamente. Ahora sí se limpia.";
  form.reset();
});
</script></body></html>
```

- Si se borra con errores, el usuario pierde lo que escribió.
- Eso genera una mala experiencia.
- Primero validar, después resetear.

EJEMPLO COMPLETO

Formulario validado: versión ejecutable

HTML + CSS + JS EJECUTABLE

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Formulario validado</title>
<style>
body { font-family: Arial, sans-serif; background:#f3f5f7; padding:30px; }
.formulario { max-width:520px; margin:auto; background:white; padding:24px; border-radius:14px; }
label { display:block; margin-top:12px; font-weight:bold; }
input, select, textarea, button { width:100%; padding:10px; margin-top:6px; box-sizing:border-box; }
textarea { min-height:90px; }
button { margin-top:16px; cursor:pointer; }
small { color:#b00020; font-weight:bold; display:block; min-height:20px; }
.campo-error { border:2px solid #b00020; }
.campo-ok { border:2px solid #006b3c; }
.exito { color:#006b3c; font-weight:bold; }
</style>
</head>
<body>
```

- Este es el caso completo integrado.
- Incluye estructura, diseño y comportamiento.
- Se puede ejecutar desde el botón.
- Permite probar errores y envío correcto.

LECTURA DEL CÓDIGO

Parte HTML: campos y contenedores de error

input	Recibe datos simples como nombre o email.
select	Obliga a elegir una opción válida.
textarea	Permite escribir una consulta más larga.
small	Reserva un lugar para el mensaje de error de cada campo.
p final	Muestra el resultado general cuando todo está correcto.

LECTURA DEL CÓDIGO

Parte CSS: clases de estado

.campo-error

Marca visualmente el campo que debe corregirse.
El estilo puede cambiar borde, fondo o cualquier señal visual.

.exito

Marca el mensaje final cuando el formulario es válido.
No se aplica a los campos, sino al resultado general.

El CSS no decide cuándo hay error. Solo define cómo se ve cada estado.

LECTURA DEL CÓDIGO

Parte JS: selección de elementos

Antes de validar, JavaScript necesita referencias a los elementos:

- Campos del formulario.
- Contenedores de error.
- Mensaje final.
- Formulario completo.

Sin selección correcta, no hay manipulación posible.

LECTURA DEL CÓDIGO

Parte JS: funciones auxiliares

mostrarError()

Recibe campo, contenedor y mensaje. Marca el campo y escribe el error.

limpiarErrores()

Borra mensajes viejos y quita clases antes de volver a validar.

Separar estas funciones hace que la validación principal sea más legible.

LECTURA DEL CÓDIGO

Parte JS: submit

1
El usuario envía

2
preventDefault evita recarga

3
Se limpian errores

4
Se validan campos

Clave

El evento **submit** es el lugar correcto para validar el formulario completo.

MEJORA

Validación en tiempo real: cuándo usarla

Puede ayudar

- Búsquedas.
- Contadores de caracteres.
- Confirmación de contraseña.
- Corrección rápida de un email.

Puede molestar

- Si muestra error apenas se empieza a escribir.
- Si interrumpe demasiado.
- Si la interfaz parece agresiva.

MEJORA

Ejemplo útil: contador de caracteres

HTML + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Contador</title></head><body>
<textarea id="consulta" placeholder="Escribí al menos 10 caracteres"></textarea>
<p id="contador">0 caracteres</p>
<script>
const consulta = document.querySelector("#consulta");
const contador = document.querySelector("#contador");
consulta.addEventListener("input", function() {
  contador.textContent = consulta.value.length + " caracteres";
});
</script></body></html>
```

- No muestra error agresivo.
- Ayuda al usuario a entender cuánto escribió.
- Es una validación asistida.

MEJORA

Validar email mientras se escribe

HTML + CSS + JS EJECUTABLE

```
<!DOCTYPE html><html lang="es"><head><meta charset="UTF-8"><title>Email en tiempo real</title>
<style>.error{color:#b00020}.ok{color:#006b3c}</style></head><body>
<input id="email" placeholder="Email"><p id="msg"></p>
<script>
const email=document.querySelector("#email"); const msg=document.querySelector("#msg");
email.addEventListener("input", function(){
  msg.className = "";
  if(email.value.length === 0){ msg.textContent=""; return; }
  if(!email.value.includes("@")){ msg.textContent="Falta @"; msg.classList.add("error"); return; }
  msg.textContent="Formato básico correcto"; msg.classList.add("ok");
});
</script></body></html>
```

Criterio

Conviene no mostrar error cuando el campo todavía está vacío. Primero dejamos escribir.

ERRORES COMUNES

Qué revisar cuando no funciona

Selector mal escrito	El id o clase del HTML no coincide con el selector de JavaScript.
Script antes del HTML	Usar script al final del body o atributo defer.
value/textContent	Inputs usan value; textos comunes usan textContent.
Falta preventDefault	El formulario recarga la página y no permite ver la validación.
No limpiar errores	Quedan mensajes viejos después de corregir datos.

CHECKLIST

Formulario bien validado

- Los campos tienen **id** claro y coherente.
- Los mensajes de error aparecen cerca del campo.
- Los errores se limpian antes de validar de nuevo.
- Hay validaciones específicas por tipo de dato.
- El éxito solo aparece cuando todo está correcto.
- El código está separado en funciones simples.

INTEGRACIÓN

Cómo se reparte la responsabilidad

HTML

Define campos, labels, botones y contenedores de mensaje.

CSS

Define cómo se ven los estados normales, de error y de éxito.

JavaScript

Escucha el submit, valida datos y aplica cambios al DOM.

Regla de orden

Cuando cada parte cumple su rol, el código se entiende mejor y se rompe menos.

CIERRE

Qué tiene que quedar claro

Conceptos clave

- Validar es guiar al usuario.
- Los errores por campo son más claros que un mensaje general.
- Las clases CSS representan estados visuales.
- Las funciones auxiliares evitan código repetido.
- El submit es el punto central para validar formularios.

Una interfaz no solo debe verse bien: debe responder bien cuando el usuario se equivoca.