

Formulario Web interactivo: validaciones en tiempo real, expresiones regulares y retroalimentación visual



■ Desarrollo Web Entorno Cliente

Fecha: Apr 22, 2025

Realizado por: Lidia García Muñoz



<https://github.com/Lidiagarmu/registro-cursos-app.git>

*se puede acceder al código fuente de la aplicación en el enlace



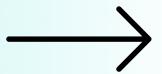
www.linkedin.com/in/lidia-garcia-munoz



CONTENIDO

1. Contexto	3
2. Tecnologías utilizadas	4
3. Formulario interactivo	5
1. Introducción a la app “Formulario de registro app”	
2. Estructura general del HTML	
3. Estilo y diseño con CSS	
4. Validaciones en JavaScript	7
1. Expresiones regulares	
2. Validación en tiempo real	
3. Retroalimentación visual al usuario	
5. Pruebas	14
1. Manuales: probando funcionalidades de la app	
2. Automáticas con Jest	
3. De compatibilidad con Google DevTools	
4. Análisis SEO con Lighthouse	
6. Documentación	25
1. Archivo README.md	
2. Comentarios en línea	
3. JSDoc	
7. Conclusión	28
8. Fuentes externas y recursos utilizados	29

1 - Contexto



En esta presentación voy a mostrar cómo he desarrollado un formulario web interactivo, centrado en la validación eficiente de datos, la experiencia de usuario y el control de calidad a través de pruebas.

En primer lugar, hablaremos de cómo se han utilizado **expresiones regulares** para validar campos clave como el correo electrónico, el teléfono y la contraseña.

A lo largo del formulario, los usuarios reciben **feedback en tiempo real** a través de eventos como *blur*, *focus* o *submit*, que muestran mensajes positivos o de error dependiendo del estado del campo. Esta retroalimentación no solo mejora la usabilidad, sino que guía al usuario hacia un llenado correcto y seguro.

Además, he añadido una **barra de progreso para contraseñas** que ayuda visualmente a comprender si se están cumpliendo los requisitos mínimos de seguridad, reforzando la confianza del usuario.

En cuanto a la validación técnica, se han desarrollado **pruebas manuales** para asegurar el correcto funcionamiento del formulario desde el punto de vista del usuario. También se ha incorporado el uso de **pruebas automatizadas con Jest**, validando funciones como *esEmailValido*, *esTelefonoValido* y *esPasswordValido* con distintos casos de uso.

Finalmente, he utilizado herramientas como **DevTools** para comprobar la compatibilidad del formulario en diferentes navegadores y dispositivos, y **Lighthouse** para medir el SEO y las partes más técnicas de la app.

Formulario de registro App



2 - Tecnologías utilizadas



Esta aplicación ha sido desarrollada utilizando **tecnologías del lado del cliente**, lo que significa que todo el funcionamiento ocurre directamente en el navegador del usuario sin necesidad de un servidor.



HTML

Lenguaje de marcado estándar utilizado para **estructurar y presentar contenido en la web** a través de un conjunto de etiquetas y atributos.



Bootstrap

Framework de CSS que **facilita la creación de sitios web responsivos y modernos** con un sistema de grid, componentes pre-diseñados y plugins de JavaScript.



Javascript

Lenguaje de programación que **añade interactividad**, permitiendo que los usuarios interactúen dinámicamente con los elementos de la página.



CSS

Lenguaje utilizado para describir la **apariencia y el diseño de una página web**. Se utiliza junto con HTML para controlar cómo se ven los elementos en la pantalla.

3 - Formulario interactivo



1 - Introducción a la app “Formulario de registro app”

Esta aplicación web presenta un **formulario de registro interactivo diseñado para gestionar el alta de estudiantes en distintos cursos**. Está construido con tecnologías web modernas como HTML, CSS y JavaScript, y apoyado en Bootstrap para una interfaz accesible y responsive.

2 - Estructura general del HTML

El archivo HTML contiene el esqueleto del formulario. Aquí **se encuentran definidos los distintos campos de entrada (inputs)**, junto a sus etiquetas, validaciones y retroalimentación visual.

Estos son los principales campos del formulario:

Nombre completo (required)	campo obligatorio con validación directa
Municipio y Provincia (opcional)	campos opcionales que el usuario puede dejar en blanco
Correo electrónico (required)	exige un formato válido del tipo ejemplo@mail.com o ejemplo@mail.es
Teléfono (required)	debe cumplir con el formato internacional 000-000-000 y permite elegir el prefijo del país
Contraseña (required)	con validación en tiempo real de los requisitos mínimos: <ul style="list-style-type: none">● Al menos 8 caracteres● Una mayúscula● Una minúscula● Un número● Un carácter especial Además, se muestra una barra de progreso para indicar la seguridad de la contraseña.
Confirmar contraseña (required)	debe coincidir con la contraseña anterior
Curso (required)	selección obligatoria entre varios cursos ofrecidos

En la siguiente imagen, un **ejemplo** en HTML del *input* Teléfono:

```
<!--INPUT TELÉFONO-->

<div class="mb-3">
  <label for="telefono" class="form-label">Teléfono <span class="text-danger">*</span></label>
  <div class="input-group">
    <span class="input-group-text"><i class="bi bi-telephone-fill"></i></span>
    <select id="prefijo" class="form-select w-auto">
      <option value="+34">🇪🇸 +34</option>
      <option value="+33">🇫🇷 +33</option>
      <option value="+1">🇺🇸 +1</option>
    </select>
    <input type="text" class="form-control" id="telefono" placeholder="600-123-456" required>
  </div>
  <div class="invalid-feedback">Introduce un número válido con formato (600-123-456).</div>
</div>
```

También se incluye un botón de envío que, al presionarse, valida todos los campos y muestra un **modal de confirmación** indicando si el registro ha sido exitoso.

3 - Estilo y diseño con CSS

El archivo CSS personaliza el aspecto visual del formulario, destacando su diseño en **modo oscuro**. Algunos elementos clave son:

- Fondo oscuro con texto claro para facilitar la lectura.
- Colores interactivos al enfocar o pasar el ratón por los campos (*hover, focus*).
- Iconos en los *inputs* gracias a Bootstrap Icons.
- Estilos especiales para *inputs* válidos/incorrectos con bordes verdes o rojos.
- Barra de progreso dinámica que indica el nivel de seguridad de la contraseña.
- Botón de registro en color verde con efecto *hover*.

En la siguiente imagen, un **ejemplo** en CSS del estilo de los campos cuando están enfocados:

```
/* Aplica a todos los inputs, selects y textareas cuando están enfocados */
input:focus,
select:focus,
textarea:focus,
button:focus {
    border-color: #0d6efd;
    box-shadow: 0 0 0 0.2rem rgba(13, 110, 253, 0.25);
    outline: none;
}
```



4 - Validaciones en JavaScript



Uno de los pilares clave de este formulario interactivo es la validación de los datos ingresados por el usuario.

Estas validaciones aseguran que la información proporcionada sea coherente, segura y esté correctamente estructurada antes de su envío.

Se implementan con JavaScript de manera dinámica, ofreciendo una experiencia más fluida gracias a la **validación en tiempo real**, el uso de **expresiones regulares** y la **retroalimentación visual inmediata**.

1 - Expresiones regulares

Las expresiones regulares (RegEx) permiten comprobar que ciertos campos siguen un formato específico. **Permiten definir patrones complejos de búsqueda que pueden ser utilizados para validar y procesar datos de formularios**, como verificar si un correo electrónico tiene el formato correcto, asegurar que una contraseña contenga caracteres específicos, o encontrar coincidencias dentro de una cadena de texto.

En este formulario se utilizan para validar:

- **Correo electrónico:** solo se aceptan los dominios @mail.com o @mail.es

```
/**  
 * Expresión regular para validar correos con dominio @mail.com o @mail.es  
 * @type {RegExp}  
 */  
const emailRegex = /^[^@]+\@[mail\.com|es]\$/;
```

- **Teléfono:** debe seguir el formato 000-000-000

```
/**  
 * Expresión regular para validar números de teléfono con formato 000-000-000  
 * @type {RegExp}  
 */  
const telefonoRegex = /^\d{3}-\d{3}-\d{3}\$/;
```

- **Contraseña:** mínimo 8 caracteres, al menos una mayúscula, una minúscula, un número y un carácter especial.

```
/**  
 * Expresión regular para validar contraseñas seguras:  
 * mínimo 8 caracteres, al menos una mayúscula, una minúscula, un número y un símbolo.  
 * @type {RegExp}  
 */  
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&*()_+=\{\}\};':"\\".,<>\?]).{8,}\$/;
```

A continuación, a modo de ejemplo, una breve explicación de la expresión regular que acompaña al campo “Correo electrónico”:

/^	<p><u>Inicio de la cadena</u></p> <p>Asegura que la validación empiece desde el principio del texto. No permite que haya caracteres antes.</p>
[^@]+	<p><u>Uno o más caracteres que NO sean @</u></p> <p>[^@] es una negación: significa “cualquier carácter excepto @”.</p> <ul style="list-style-type: none">• + indica que debe haber al menos uno o más de estos caracteres.• Ejemplo válido: usuario, nombre.apellido, juan123 <p>Esto cubre la parte antes del @, que sería el nombre del usuario del correo.</p>
@mail\.	<p><u>El texto literal @mail.</u></p> <ul style="list-style-type: none">• Se exige que después del nombre de usuario venga exactamente @mail..• El punto se escapa con \. porque los puntos en regex significan “cualquier carácter”, así que al poner \. decimos literalmente un punto. <p>Esto restringe el dominio a solo aquellos que sean de tipo @mail.</p>
(com es)	<p><u>Coincide con "com" o "es":</u></p> <p>Los paréntesis con crean un grupo alternativo:</p> <ul style="list-style-type: none">• (com es) significa “puede ser com o puede ser es”.
\$/	<p><u>Fin de la cadena:</u></p> <p>Asegura que no haya más texto después del dominio, como .net o .org.</p>

Estas expresiones se usan dentro de la función `validarCampoIndividual`, que detecta errores y muestra mensajes personalizados si el valor no cumple las reglas.

2 - Validación en tiempo real

La validación de formularios en JavaScript permite garantizar que los datos introducidos por el usuario cumplan ciertos requisitos antes de ser procesados.

En esta app, se utilizan validaciones tanto **automáticas (en tiempo real)** como al **enviar el formulario**, combinando expresiones regulares con técnicas de interacción visual para mejorar la experiencia del usuario.

La **validación en tiempo real** se refiere a revisar los datos del usuario **a medida que los escribe o al moverse entre campos**, sin esperar a que pulse el botón “Registrarse”. Esto se implementa usando principalmente eventos como ***input* y *blur***.

En nuestra app:

- Se usa el **evento *blur*** en cada campo (*input* y *select*) para validar cuando el usuario **abandona el campo** (pierde el foco):

```
* Añade eventos de validación positiva al salir de cada input/select del formulario (blur)
*/
document.querySelectorAll('#registroForm input, #registroForm select').forEach(el => {
  el.addEventListener('blur', () => {
    validarCampoIndividual(el, true);
  });
});
```

- En el caso específico de la contraseña, se usa ***input*** para detectar cada pulsación del usuario y mostrarle en directo qué requisitos está cumpliendo:

```
passwordInput.addEventListener('input', () => {
  const pass = passwordInput.value;
```

Esto ayuda al usuario a corregir errores sin esperar al final, reduciendo la frustración y aumentando la tasa de formularios completados correctamente.

Ejemplo, campo contraseña con barra visual en la app, donde el usuario puede ver claramente cuáles son los requisitos que faltan para que la contraseña sea válida:

Contraseña *

Mínimo 8 caracteres

Una letra mayúscula

Una letra minúscula

Un número

Un carácter especial

3 - Retroalimentación visual al usuario

Este apartado abarca cómo se **comunica al usuario si los datos están bien o mal**, mediante colores, mensajes y animaciones. Todo esto lo conseguimos gracias a los **eventos focus, submit, blur**, y de **estilos visuales como .is-valid e .is-invalid**.

- Al salir de un campo (evento **blur**), si el valor está mal, se le muestra un **mensaje personalizado** debido al elemento **feedback**. Además de una retroalimentación positiva o negativa al usuario gracias a las clases **.is-valid e .is-invalid** :

```
el.classList.add('is-valid');
el.classList.remove('is-invalid');
```

```
if (feedback) {
  if (invalido) {
    feedback.textContent = mensaje;
    feedback.style.display = 'block';
  } else {
    feedback.textContent = "";
    feedback.style.display = 'none';
  }
}
```

- Si el valor es correcto, se aplica la clase **.is-valid** para **resaltar positivamente** ese campo (con borde verde y check):

Nombre completo *

Lidia

- Si el valor es incorrecto, se aplica la clase **.is-invalid** para **resaltar negativamente** ese campo (con borde rojo y mensaje informativo del error)

Teléfono *

 +34  1 ⓘ

- El evento **submit** del formulario se dispara cuando, se validan todos los campos y el usuario presiona el botón “Registrarse”:

```
/*
 * Maneja el evento de envío del formulario de registro
 */
document.getElementById('registroForm').addEventListener('submit', async (e) => {
  e.preventDefault();

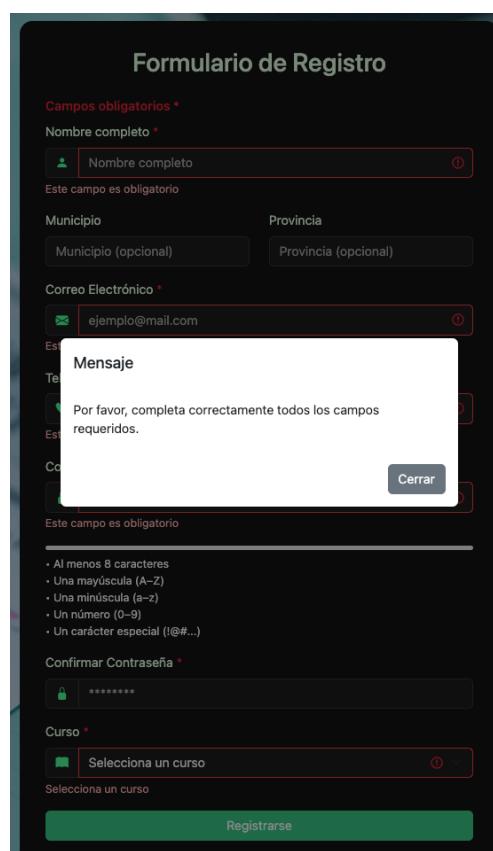
  validarCampos();

  const hayErrores = [...document.querySelectorAll('#registroForm input[required], #registroForm select[required]')].some(el => el.classList.contains('is-invalid'));

  if (hayErrores) {
    mostrarModal("Por favor, completa correctamente todos los campos requeridos.", false);
    return;
  }

  // Simulación de registro exitoso
  setTimeout(() => {
    mostrarModal("¡Registro completado con éxito!", true);
  }, 500);
});
```

- Si hay errores, se muestra un **modal de error**; si todo está bien, uno de éxito:



- Tras un registro exitoso se limpian los campos y se reinician los estilos:;

```
/*
 * Maneja el cierre del modal para limpiar y resetear el formulario si el registro fue exitoso
 */
const modal = document.getElementById('mensajeModal');

modal.addEventListener('hidden.bs.modal', () => {
  if (!fueRegistroExitoso) return; // Si no fue éxito, no reseteas

  const form = document.getElementById('registroForm');
  form.reset();

  // Ocultar feedbacks de error
  form.querySelectorAll('.invalid-feedback').forEach(el => {
    el.style.display = 'none';
    el.textContent = ''; // Opcional, por limpieza
  });

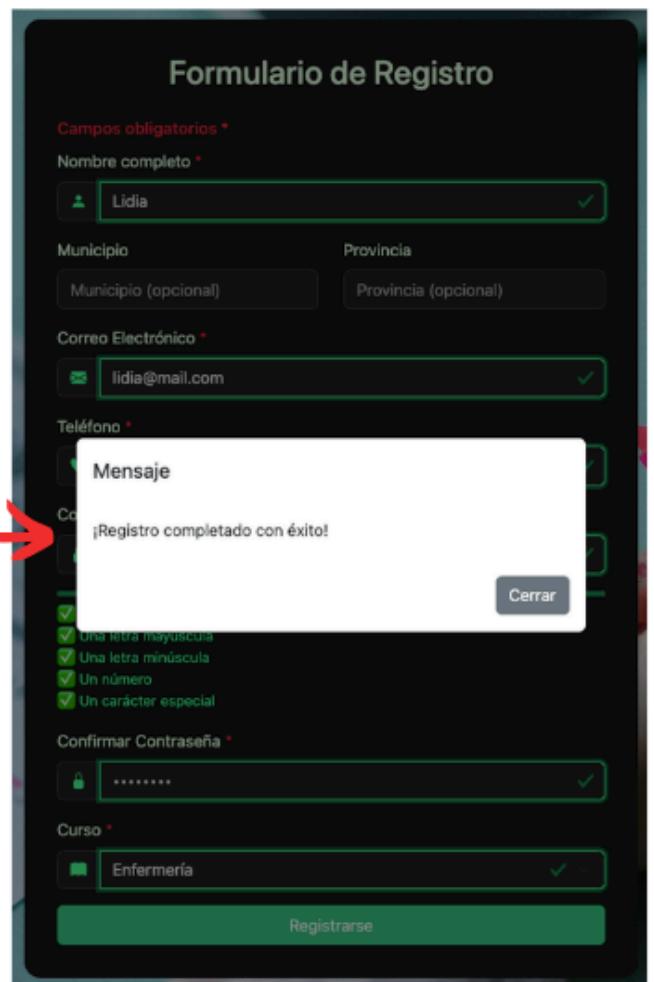
  // Resetear estilos de validación visual (.is-valid / .is-invalid)
  form.querySelectorAll('input, select').forEach(el => {
    el.classList.remove('is-valid', 'is-invalid');
  });

  // Resetear requisitos contraseña
  const requisitos = {
    length: '• Al menos 8 caracteres',
    uppercase: '• Una mayúscula (A-Z)',
    lowercase: '• Una minúscula (a-z)',
    number: '• Un número (0-9)',
    symbol: '• Un carácter especial (@#...)'
  };

  for (let clave in requisitos) {
    const el = document.getElementById(`req-${clave}`);
    el.textContent = requisitos[clave];
    el.classList.remove('text-success');
    el.classList.add('text-light');
  }

  // Reset barra de progreso
  passwordStrengthBar.style.width = '0%';
  passwordStrengthBar.className = 'progress-bar';
});

});
```



- Elemento **focus**: se usa para **resaltar un campo cuando el usuario empieza a escribir**, ayudando a enfocar la atención.

En esta app, gracias a la magia de Bootstrap hemos conseguido este efecto de manera automática.

Bootstrap aplica estilos automáticamente cuando un campo recibe el foco. Es decir, cuando haces clic o tabulas hacia un campo, el navegador y Bootstrap resaltan el campo con un **borde azul claro**.

```
<input type="text" class="form-control" id="nombre" required placeholder="Nombre completo">
```



```
/* Aplica a todos los inputs, selects y textareas cuando están enfocados */  
input:focus,  
select:focus,  
textarea:focus,  
button:focus {  
    border-color: #0d6efd;  
    box-shadow: 0 0 0.2rem rgba(13, 110, 253, 0.25);  
    outline: none;  
}
```

5 - Pruebas



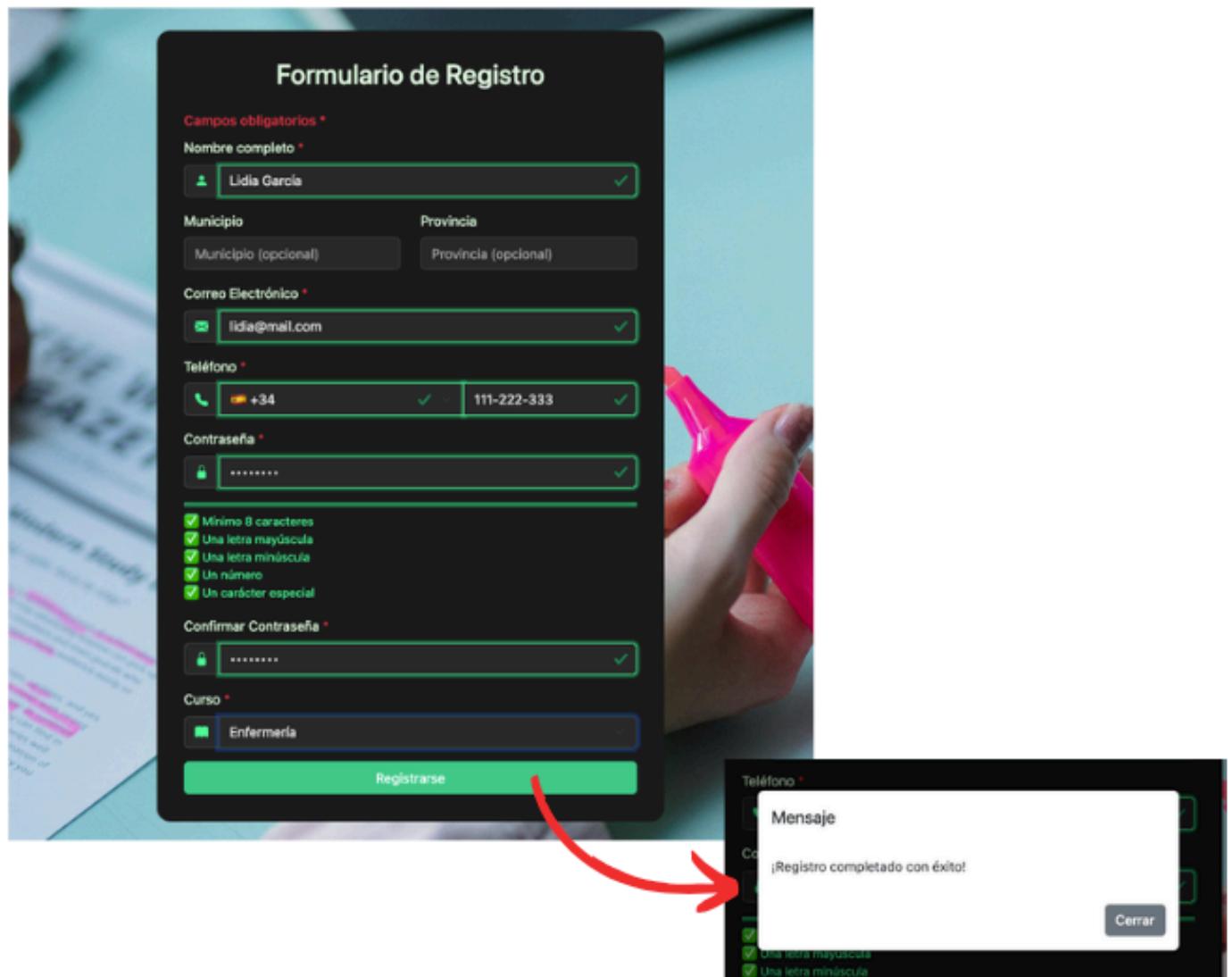
Para garantizar el correcto funcionamiento de la aplicación y una experiencia óptima para el usuario, se ha llevado a cabo un proceso de pruebas dividido en varias fases.

Estas pruebas permiten verificar que cada funcionalidad responda según lo esperado, asegurar la compatibilidad en distintos entornos y evaluar el rendimiento general de la aplicación.

1 - Manuales: probando funcionalidades de la app

Se han realizado **comprobaciones prácticas e interactivas de cada una de las funcionalidades clave de la aplicación**, enfocándose en la experiencia del usuario y la respuesta del formulario de registro ante distintos escenarios.

1) Registro exitoso por todos los campos OK - modal “¡Registro completado con éxito!”



2) Mensaje de ERROR por campos vacíos - “Este campo es obligatorio”

Formulario de Registro

Campos obligatorios *

Nombre completo *

Nombre completo !

Este campo es obligatorio

Municipio Provincia

Municipio (opcional) Provincia (opcional)

Correo Electrónico *

ejemplo@mail.com !

Este campo es obligatorio

Teléfono *

+34 ✓ 600-123-456 !

Este campo es obligatorio

Contraseña *

***** !

Este campo es obligatorio

- Al menos 8 caracteres
- Una mayúscula (A-Z)
- Una minúscula (a-z)
- Un número (0-9)
- Un carácter especial (!@#...)

Confirmar Contraseña *

Curso *

Selecciona un curso !

Selecciona un curso

Registrarse

3) Mensaje de ERROR en el campo Correo electrónico - “Introduce un correo válido (ejemplo@mail.com)”

Correo Electrónico *

lidia !

Introduce un correo válido (ejemplo@mail.com)

4) Mensaje de ERROR en el campo Teléfono - “Introduce un número válido con formato (600-123-456)”

Teléfono *

+34 ✓ 606 !

Introduce un número válido con formato (600-123-456)

5) Mensaje de ERROR en campo Contraseña - “La contraseña no cumple con los requisitos”

Contraseña *

The screenshot shows a dark-themed password input field. The placeholder text "Contraseña *" is at the top. Below it is a lock icon, a three-dot menu icon, and an exclamation mark icon in a red circle. The main message "La contraseña no cumple con los requisitos" is displayed in pink. Below this, a horizontal bar shows progress from yellow to white. A list of requirements follows:

- ◆ Mínimo 8 caracteres
- Una letra mayúscula
- Una letra minúscula
- ◆ Un número
- Un carácter especial

6) Mensaje de ERROR en campo Confirmar contraseña - “Las contraseñas deben coincidir”

Confirmar Contraseña *

The screenshot shows a dark-themed password confirmation input field. The placeholder text "Confirmar Contraseña *" is at the top. Below it is a lock icon, a five-star icon, and an exclamation mark icon in a red circle. The main message "Las contraseñas deben coincidir" is displayed in pink.

7) Mensaje de ERROR en campo Curso - “Selecciona un curso”

Curso *

The screenshot shows a dark-themed course selection input field. The placeholder text "Curso *" is at the top. Below it is a book icon, the text "Selecciona un curso", and an exclamation mark icon in a red circle. The message "Selecciona un curso" is displayed in pink below the input field.

2 - Automáticas con Jest



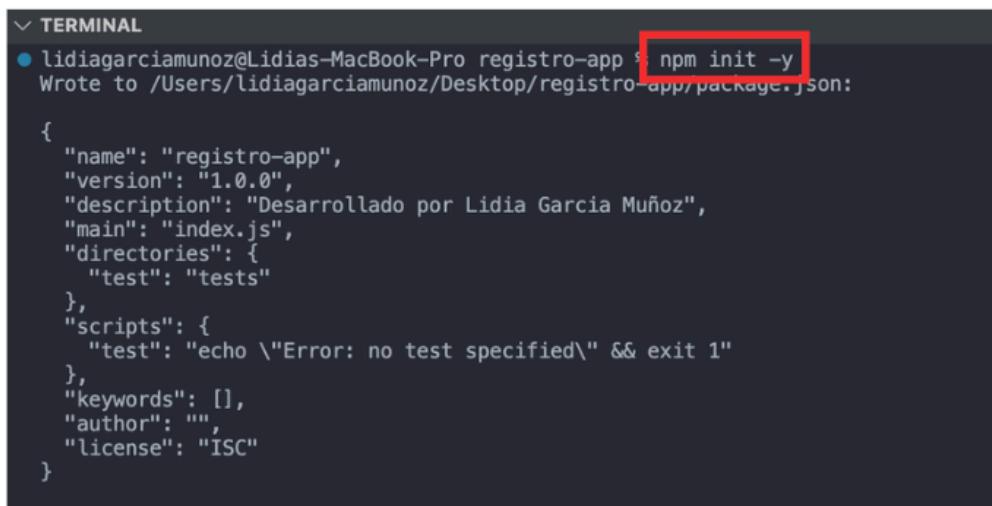
Se han desarrollado pruebas automatizadas utilizando el *framework* de pruebas **Jest**, un entorno de pruebas en JavaScript que permite verificar la funcionalidad del código de forma precisa y repetible.

Estas pruebas se ejecutaron a través de Visual Studio Code para **comprobar que la lógica del formulario interactivo funciona correctamente**.

A continuación, se describen los pasos realizados:

1) Inicialización del proyecto con `package.json`:

En la terminal de Visual Studio Code, se ejecutó el siguiente comando para crear el archivo `package.json` que permite gestionar las dependencias del proyecto:



```
npm init -y
Wrote to /Users/lidiagarciamunoz/Desktop/registro-app/package.json:

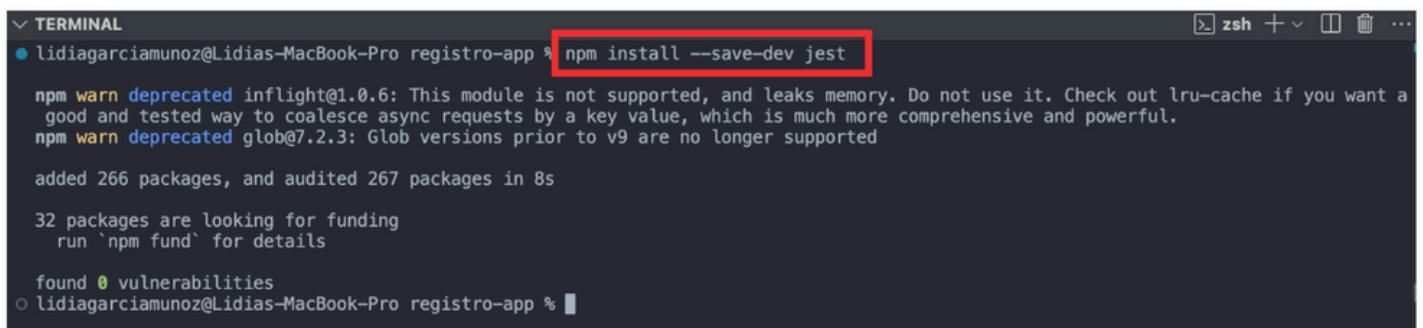
{
  "name": "registro-app",
  "version": "1.0.0",
  "description": "Desarrollado por Lidia Garcia Muñoz",
  "main": "index.js",
  "directories": {
    "test": "tests"
  },
  "scripts": {
    "test": "echo \\"$Error: no test specified\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



```
package.json > ...
1  {
2    "name": "registro-app",
3    "version": "1.0.0",
4    "description": "Desarrollado por Lidia Garcia Muñoz",
5    "main": "index.js",
6    "directories": {
7      "test": "tests"
8    },
9    "scripts": {
10      "test": "jest"
11    },
12    "keywords": [],
13    "author": "",
14    "license": "ISC",
15    "devDependencies": {
16      "jest": "^29.7.0"
17    }
18 }
```

2) Instalación de Jest:

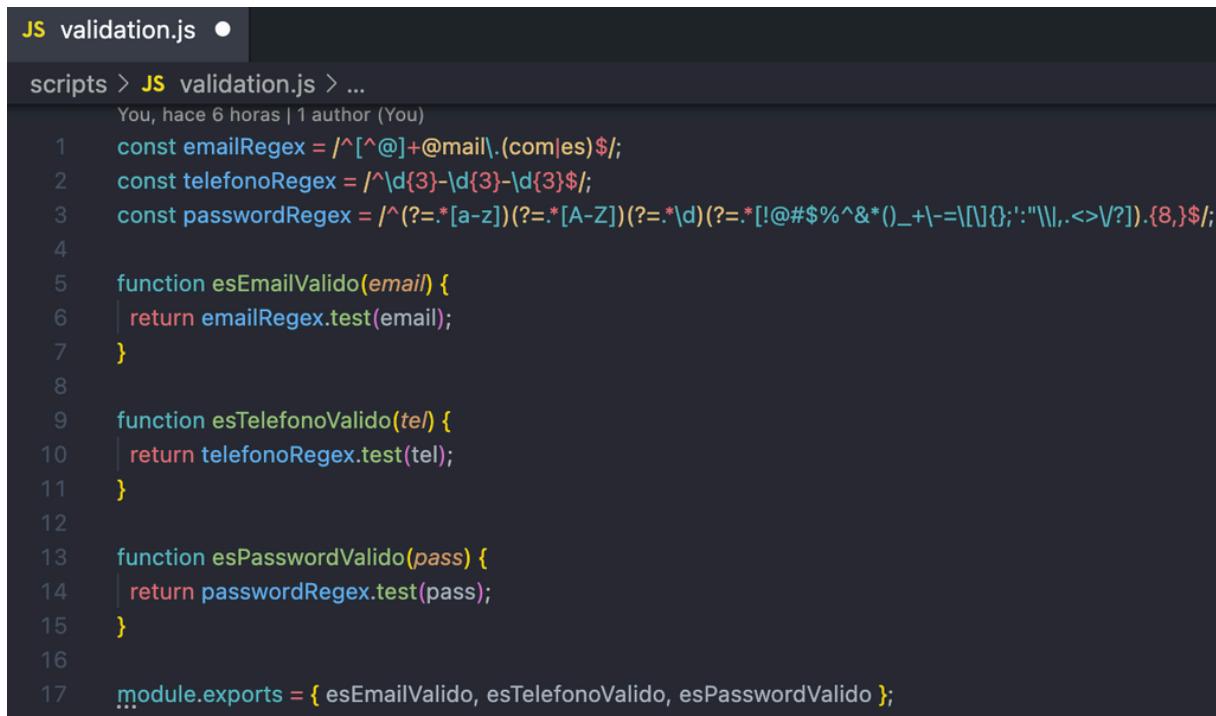
Luego se instaló Jest como dependencia de desarrollo mediante:



```
TERMINAL
● lidiagarciamunoz@Lidias-MacBook-Pro registro-app % npm install --save-dev jest
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
added 266 packages, and audited 267 packages in 8s
32 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
○ lidiagarciamunoz@Lidias-MacBook-Pro registro-app %
```

3) Creación del archivo *validation.js*:

Se creó dentro de la carpeta *scripts*, el archivo *validation.js*, donde se definió la lógica de validación del formulario (por ejemplo, verificar campos vacíos, formato de correo, etc.).



```
validation.js ●

scripts > validation.js > ...
You, hace 6 horas | 1 author (You)
1 const emailRegex = /^[^@]+@[mail\.com|es]\$/;
2 const telefonoRegex = /^\d{3}-\d{3}-\d{3}\$/;
3 const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&*()_+=\[\]\{\};':"\\".,<>\?]).{8,}\$/;
4
5 function esEmailValido(email) {
6   return emailRegex.test(email);
7 }
8
9 function esTelefonoValido(tel) {
10  return telefonoRegex.test(tel);
11 }
12
13 function esPasswordValido(pass) {
14  return passwordRegex.test(pass);
15 }
16
17 module.exports = { esEmailValido, esTelefonoValido, esPasswordValido };
```

4) Creación del archivo de pruebas *validation.test.js*:

En la carpeta *test*, se creó el archivo *validation.test.js*, donde se escribieron los tests usando Jest para comprobar que las funciones del archivo *validation.js* se comportan como se espera.

```
validation.test.js ● | tests > validation.test.js > ... | You, hace 6 horas | 1 author (You) | 1 const { esEmailValido, esTelefonoValido, esPasswordValido } = require('../scripts/validation'); | 2 | 3 test('Email válido', () => { | 4 | | expect(esEmailValido('lidia@mail.com')).toBe(true); | 5 |}); | 6 | 7 test('Email inválido', () => { | 8 | | expect(esEmailValido('lidia@gmail.com')).toBe(false); | 9 |}); | 10 | 11 test('Teléfono válido', () => { | 12 | | expect(esTelefonoValido('123-456-789')).toBe(true); | 13 |}); | 14 | 15 test('Teléfono inválido', () => { | 16 | | expect(esTelefonoValido('123456789')).toBe(false); | 17 |}); | 18 | 19 test('Password válida', () => { | 20 | | expect(esPasswordValido('Hola1234!')).toBe(true); | 21 |}); | 22 | 23 test('Password inválida', () => { | 24 | | expect(esPasswordValido('hola')).toBe(false); | 25 |}); | 26
```

5) Ejecución de las pruebas con Jest:

Finalmente, en la terminal se ejecutó Jest con el comando:

```
✓ TERMINAL
● lidiagarciamunoz@Lidias-MacBook-Pro registro-app % npm test
  ↗ registro-app@1.0.0 test
  ↗ jest
    PASS tests/validation.test.js
      ✓ Email válido (2 ms)
      ✓ Email inválido (1 ms)
      ✓ Teléfono válido (1 ms)
      ✓ Teléfono inválido
      ✓ Password válida
      ✓ Password inválida (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:  0 total
Time:        0.289 s
Ran all test suites.
```

Este flujo permitió verificar de forma automatizada el correcto funcionamiento de la lógica del formulario.

3 - De compatibilidad con Google DevTools

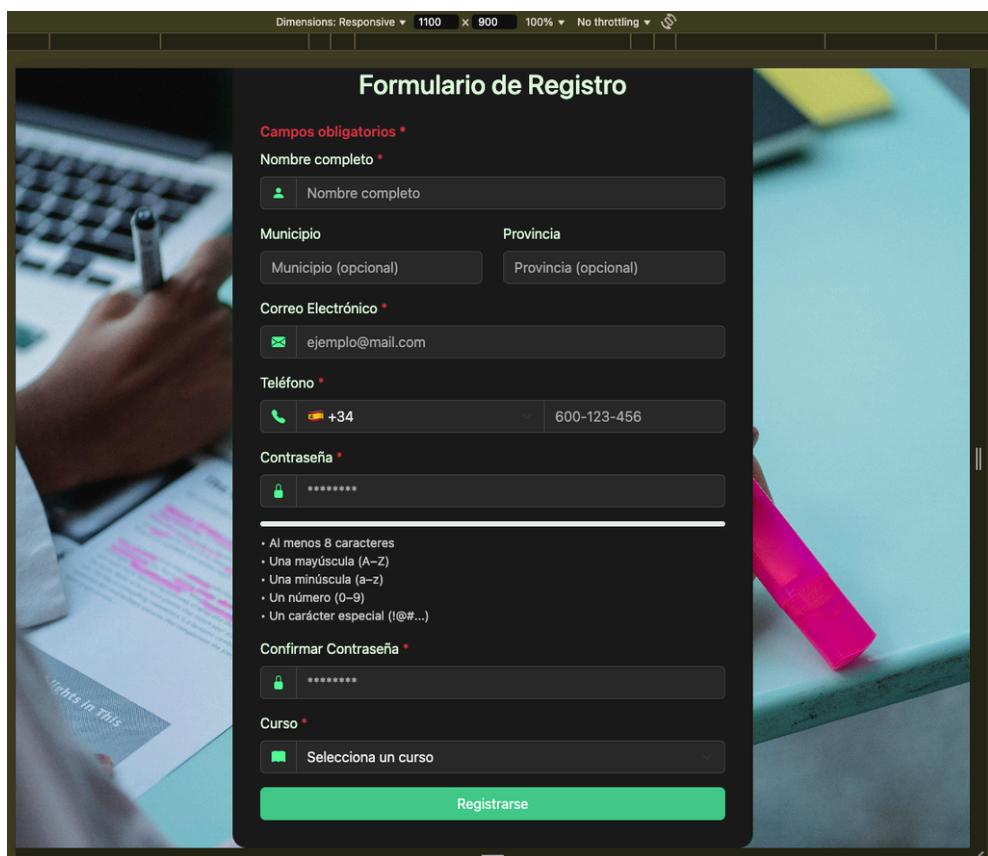


Para comprobar la correcta visualización y funcionamiento de la aplicación en distintos dispositivos y tamaños de pantalla, se utilizó **Google DevTools**, conjunto de herramientas integradas en el navegador **Google Chrome** que te permite inspeccionar, depurar y optimizar sitios web directamente desde el navegador.

Exactamente, hemos utilizado el Modo Responsive de *DevTools*. Esto nos ha permitido simular la visualización en móviles, tablets y pantallas personalizadas, ajustando estilos y corrigiendo posibles errores.

A continuación, se muestran capturas de la app en diferentes tamaños de pantalla y dispositivos móviles y *tablets*:

- Pantallas tamaño 1100 x 900



- Iphone 12 PRO modo vertical

Dimensions: iPhone 12 Pro ▾ 390 × 844 100% ▾ No throttling ▾

Formulario de Registro

Campos obligatorios *

Nombre completo *

Nombre completo

Municipio

Municipio (opcional)

Provincia

Provincia (opcional)

Correo Electrónico *

ejemplo@mail.com

Teléfono *

+34 600-123-4

Contraseña *

• Al menos 8 caracteres
• Una mayúscula (A–Z)
• Una minúscula (a–z)
• Un número (0–9)
• Un carácter especial (!@#...)

Confirmar Contraseña *

- Iphone 12 PRO modo horizontal

Dimensions: iPhone 12 Pro ▾ 844 × 390 100% ▾ No throttling ▾

Formulario de Registro

Campos obligatorios *

Nombre completo *

Nombre completo

Municipio

Municipio (opcional)

Provincia

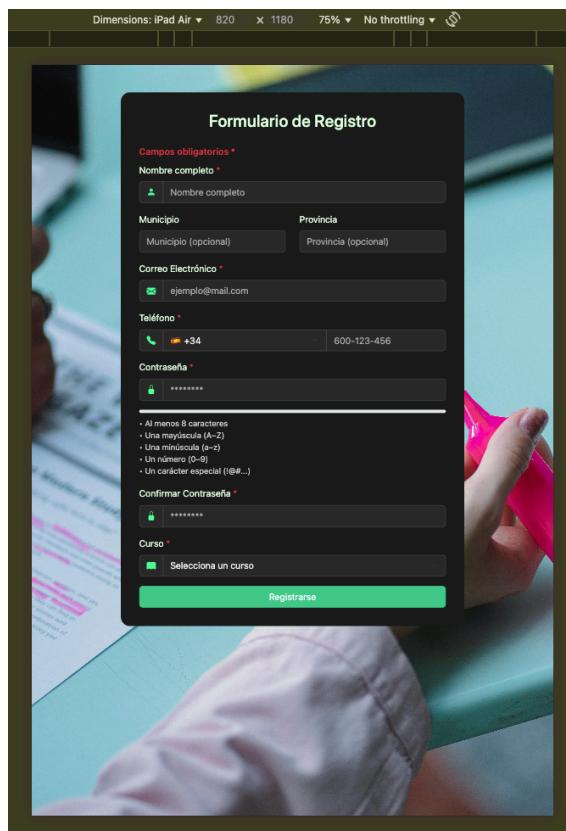
Provincia (opcional)

Correo Electrónico *

ejemplo@mail.com

Teléfono *

- IPAD AIR



4 - Análisis SEO con *Lighthouse*



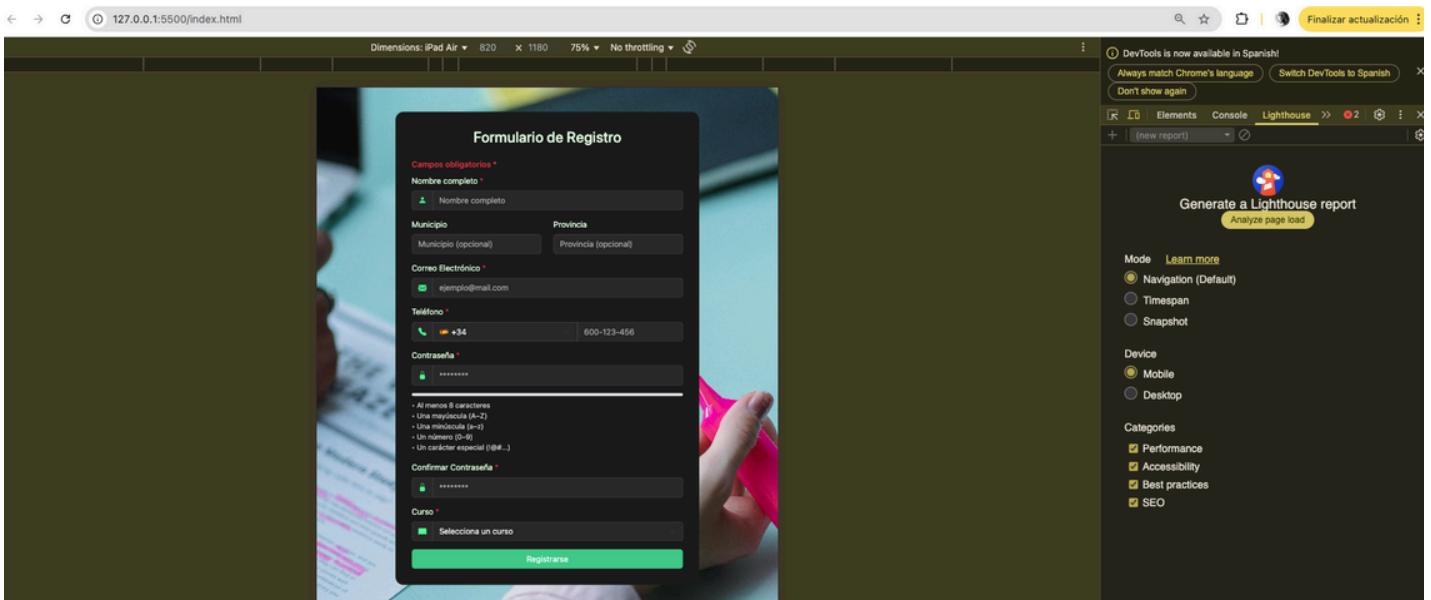
Google Lighthouse es una herramienta automatizada de código abierto desarrollada por Google para **auditar la calidad de las páginas web**. Permite evaluar aspectos clave como el rendimiento, la accesibilidad, las mejores prácticas de desarrollo, el SEO y la experiencia del usuario (UX). Está integrada directamente en el navegador Google Chrome, lo que la hace accesible y fácil de usar sin necesidad de instalar software adicional.

En este trabajo, Lighthouse se ha utilizado para **realizar un análisis técnico de la app “Formulario de registro app” teniendo en cuenta el rendimiento general, la accesibilidad del sitio, las buenas prácticas de programación y la optimización SEO**.

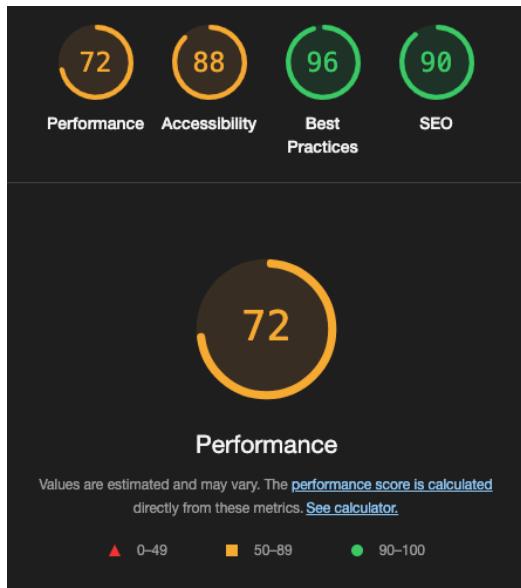
Pasos seguidos para ejecutar el análisis:

1. Desplegamos la app con “Live Server” de Visual Studio Code
2. Hacemos clic derecho en cualquier parte de la página y seleccionamos “**Inspeccionar**”.
3. Vamos a la pestaña superior llamada “**Lighthouse**”.

4. Seleccionamos los dispositivos (Mobile o Desktop) y las categorías a analizar (Performance, Accessibility, Best Practices, SEO).
5. Pulsamos en el botón “**Analyze page load**” para comenzar el análisis.
6. Tras unos segundos, Lighthouse genera un informe con puntuaciones y recomendaciones detalladas.



Resultados obtenidos:



- **🚀 Performance (72/100):**

La puntuación de rendimiento es muy buena, lo que indica que la app carga rápido y fluido en su mayoría.

- **♿ Accessibility (88/100):**

La accesibilidad también es muy buena, aunque siempre mejorable. Es **muy accesible**.

para usuarios con discapacidades.

- **Best Practices (96/100):**
Código y estructura bastante sólidos, aunque podría haber uno o dos detalles a revisar.
- **SEO (90/100):**
Muy bien optimizado para buscadores. Solo detalles menores podrían mejorarse para llegar al 100.

El informe de Lighthouse ha sido clave para detectar debilidades técnicas que afectan directamente a la **experiencia de usuario**, como el rendimiento y la accesibilidad.

Estos resultados proporcionan una base objetiva para aplicar mejoras concretas y priorizar cambios en la estructura, el contenido y el código fuente de la app.

6 - Documentación



1 - Archivo README.md

El archivo README.md es esencial en proyectos de cualquier tamaño. Es, en esencia, la primera impresión y guía para cualquier persona interesada en el proyecto.

Entre sus **ventajas** destacan:

- **Documenta el propósito**, es decir, explica la funcionalidad y objetivo del proyecto
- **Facilita la colaboración**, especialmente para nuevos desarrolladores
- **Aumenta la accesibilidad** ofreciendo un contexto rápido de dependencias y tecnologías empleadas, estructura del proyecto, entre otras.
- **Mejora la profesionalidad**

En la imagen se puede observar el archivo README.md de nuestra pequeña aplicación Formulario de registro app.

```

README.md X
 README.md > # Registro de Cursos App > ## ✅ Validaciones del formulario
You, hace 7 horas | 1 author (You)
# Autor

Desarrollado por Lidia García Muñoz

# Registro de Cursos App

Una aplicación web sencilla y atractiva para registrar estudiantes en distintos cursos formativos. Ofrece validaciones en tiempo real, feedback visual y un diseño moderno enfocado en la experiencia del usuario.

## 🚧 Tecnologías Utilizadas

- **HTML5**
- **CSS3** (Diseño personalizado con enfoque en Dark UI)
- **JavaScript (Vanilla)** para validación de formularios y comportamiento dinámico
- **Bootstrap 5.3** para estilos y componentes
- **Bootstrap Icons** para mejorar la accesibilidad visual

## 🌐 Características

- Validación de campos al momento y al enviar el formulario.
- Feedback visual instantáneo (errores, éxito, requisitos de contraseña).
- Evaluación dinámica de la fortaleza de la contraseña con barra.
- Modal de confirmación al finalizar el registro.
- Formulario responsive con diseño oscuro moderno.
- Opciones dinámicas para cursos y selección de país para prefijos telefónicos.

## 🔗 Requisitos Previos

- Tener instalado **Visual Studio Code**
- Extensión recomendada: **Live Server**

## 🛡️ Cómo Desplegar la App Localmente

1. **Clona el repositorio:**

```bash
git clone https://github.com/Lidiagarmu/registro-cursos-app.git
```
...

```

2 -Comentarios en línea

Los comentarios en línea son anotaciones dentro del código fuente que **ayudan a explicar lo que hace una instrucción o bloque específico**. Aunque no afectan la ejecución del programa, cumplen un rol esencial en la claridad, mantenibilidad y colaboración dentro de un proyecto.

Utilizar comentarios de forma adecuada permite que otros desarrolladores (o incluso uno mismo en el futuro) comprendan rápidamente la lógica detrás del código, evitando malentendidos y facilitando tareas como la depuración, revisión o ampliación del mismo. Un buen comentario en línea no repite lo obvio, sino que aporta contexto, justifica decisiones técnicas o señala posibles mejoras.

En definitiva, comentar con criterio es una práctica clave en el desarrollo profesional de software.

En las siguientes imágenes, un par de ejemplos para ver cómo sería un comentario en línea:

```
// Actualizar textos de los requisitos
document.getElementById('req-length').textContent = (checks.length ? '✅ : '⚠') + ' Mínimo 8 caracteres';
document.getElementById('req-uppercase').textContent = (checks.uppercase ? '✅ : '⚠') + ' Una letra mayúscula';
document.getElementById('req-lowercase').textContent = (checks.lowercase ? '✅ : '⚠') + ' Una letra minúscula';
document.getElementById('req-number').textContent = (checks.number ? '✅ : '⚠') + ' Un número';
document.getElementById('req-symbol').textContent = (checks.symbol ? '✅ : '⚠') + ' Un carácter especial';
```

```
// Actualizar barra de progreso
const cumplidos = Object.values(checks).filter(v => v).length;
const porcentaje = cumplidos * 20;

passwordStrengthBar.style.width = `${porcentaje}%`;

passwordStrengthBar.classList.remove('bg-danger', 'bg-warning', 'bg-info', 'bg-success');
if (porcentaje <= 40) {
  passwordStrengthBar.classList.add('bg-danger');
} else if (porcentaje <= 60) {
  passwordStrengthBar.classList.add('bg-warning');
} else if (porcentaje <= 80) {
  passwordStrengthBar.classList.add('bg-info');
} else {
  passwordStrengthBar.classList.add('bg-success');
}
});
```

3 - JSDoc

JSDoc es una herramienta súper útil para dejar comentarios en el código JavaScript que explican qué hace cada parte. Con JSDoc, podemos describir funciones, sus parámetros, lo que devuelven, y otros detalles importantes.

Usarlo hace que el código sea más fácil de entender, especialmente cuando trabajamos en equipo o volvemos a mirar el código después de un tiempo. Además, genera documentación automáticamente, lo que facilita saber cómo usar y modificar el código sin tener que adivinar qué hace cada función.

A continuación, un ejemplo acerca de cómo emplearlo en una de las funciones creadas en nuestro archivo .js.

Para crear un JSDoc basta con escribir `/` delante de la función o variable que queramos documentar.**

```
/*
 * Muestra un modal con mensaje de éxito o error
 * @param {string} mensaje - Mensaje a mostrar
 * @param {boolean} [success=false] - Define si es un mensaje de éxito
 */
function mostrarModal(mensaje, success = false) {
    fueRegistroExitoso = success; // <-- ACTUALIZA ESTADO

    document.getElementById('modalMensaje').textContent = mensaje;
    const modalContent = document.getElementById('modalContenido');
    modalContent.classList.remove('success', 'error');
    modalContent.classList.add(success ? 'success' : 'error');

    const modal = new bootstrap.Modal(document.getElementById('mensajeModal'));
    modal.show();
}
```

7 - Conclusión



Este trabajo demuestra la **importancia de combinar el diseño visual con técnicas de validación robustas para mejorar la experiencia del usuario** y asegurar la integridad de los datos.

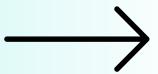
Las expresiones regulares permiten validar con precisión los campos más sensibles del formulario, mientras que los eventos como blur, focus y submit se encargan de activar validaciones contextuales en tiempo real.

A nivel de testing, **implementar pruebas automatizadas en paralelo con pruebas manuales garantiza un desarrollo más confiable y escalable**. Y gracias a herramientas de compatibilidad y rendimiento, hemos asegurado que el formulario funcione de forma óptima en múltiples entornos.



En resumen, este proyecto no solo refleja la parte técnica de validar formularios, sino también el esfuerzo por ofrecer una experiencia fluida, accesible y profesional para el usuario final.

8 - Fuentes externas y recursos utilizados



- **Cesur, Desarrollo web entorno cliente** - “*Interacción con el usuario: eventos y formularios*”.
- **Lidia García Muñoz** – “*Creación de una página web interactiva: HTML, CSS. y JavaScript en acción*”
- **Lidia García Muñoz** – “*Usabilidad y accesibilidad en la web*”.
- **Lidia García Muñoz** – “*Desarrollando una aplicación web interactiva* ”
- **Lidia García Muñoz** – “*Creando una app de Gestión de Stock en Aplicaciones Cliente: creación, uso y manejo de arrays*”.



Visual Studio



Google
Lighthouse

