Validación de formularios de datos

Antes de enviar datos al servidor, es importante asegurarse de que se completan todos los controles de formulario requeridos, y en el formato correcto. Esto se denomina validación de formulario en el lado del cliente y ayuda a garantizar que los datos que se envían coinciden con los requisitos establecidos en los diversos controles de formulario. Este artículo te guiará por los conceptos básicos y ejemplos de validación de formularios en el lado del cliente.

Prerrequisitos:	Conocimientos básicos de informática, y entender cómo funcionan el HTML, el CSS y el JavaScript.
Objetivo:	Entender qué es la validación de formularios en el lado del cliente, porqué es importante y cómo aplicar diversas técnicas para implementarla.

La validación en el lado del cliente es una verificación inicial y una característica importante para garantizar una buena experiencia de usuario; mediante la detección de datos no válidos en el lado del cliente, el usuario puede corregirlos de inmediato. Si el servidor lo recibe y, a continuación, lo rechaza; se produce un retraso considerable en la comunicación entre el servidor y el cliente que insta al usuario a corregir sus datos.

Sin embargo, ¡la validación en el lado del cliente *no debe considerarse* una medida de seguridad exhaustiva! Tus aplicaciones siempre deben realizar comprobaciones de seguridad de los datos enviados por el formulario *en el lado del servidor*, así como también en el lado del cliente, porque la validación en el lado del cliente es demasiado fácil de evitar, por lo que los usuarios malintencionados pueden enviar fácilmente datos incorrectos a tu servidor. Lee Seguridad en los sitios web para ver qué *podría* suceder. Cómo implementar la validación en el lado del servidor está fuera del alcance de este módulo, pero debes tenerlo en cuenta.

¿Qué es la validación de formularios?

Ve a cualquier sitio web popular que incluya un formulario de registro y observa que proporcionan comentarios cuando no introduces tus datos en el formato que se espera. Recibirás mensajes como:

- «Este campo es obligatorio» (No se puede dejar este campo en blanco).
- «Introduzca su número de teléfono en el formato xxx-xxxx» (Se requiere un formato de datos específico para que se considere válido).
- «Introduzca una dirección de correo electrónico válida» (los datos que introdujiste no están en el formato correcto).
- «Su contraseña debe tener entre 8 y 30 caracteres y contener una letra mayúscula, un símbolo y un número». (Se requiere un formato de datos muy específico para tus datos).

Esto se llama validación de formulario. Cuando introduces los datos, el navegador y/o el servidor web verifican que estén en el formato correcto y dentro de las restricciones establecidas por la aplicación. La validación realizada en el navegador se denomina validación en el lado del cliente, mientras que la validación realizada en el servidor se denomina validación en el lado del servidor. En este capítulo nos centraremos en la validación en el lado del cliente.

Si la información está en el formato correcto, la aplicación permite que los datos se envíen al servidor y (en general) que se guarden en una base de datos; si la información no está en el formato correcto, da al usuario un mensaje de error que explica lo que debe corregir y le permite volver a intentarlo.

Queremos que completar formularios web sea lo más fácil posible. Entonces, ¿por qué insistimos en validar nuestros formularios? Hay tres razones principales:

- Queremos obtener los datos correctos en el formato correcto. Nuestras aplicaciones no funcionarán correctamente si los datos de nuestros usuarios se almacenan en el formato incorrecto, son incorrectos o se omiten por completo.
- Queremos proteger los datos de nuestros usuarios. Obligar a nuestros usuarios a introducir contraseñas seguras facilita proteger la información de su cuenta.
- Queremos protegernos a nosotros mismo. Hay muchas formas en que los usuarios maliciosos puedan usar mal los formularios desprotegidos y dañar la aplicación (consulta Seguridad del sitio web).

> Advertencia: No confíes nunca en los datos que se pasan al servidor desde el cliente. Incluso si tu formulario se valida correctamente y evita la introducción de datos con formato incorrecto en el lado del cliente, un usuario malintencionado puede alterar la petición de red.

Diferentes tipos de validación en el lado del cliente

Hay dos tipos diferentes de validación por parte del cliente que encontrarás en la web:

- La validación de formularios incorporada utiliza características de validación de formularios HTML5, que hemos visto en muchos lugares a lo largo de este módulo. Esta validación generalmente no requiere mucho JavaScript. La validación de formularios incorporada tiene un mejor rendimiento que JavaScript, pero no es tan personalizable como la validación con JavaScript.
- La validación con JavaScript se codifica utilizando JavaScript. Esta validación es completamente personalizable, pero debes crearlo todo (o usar una biblioteca).

Usar la validación de formulario incorporada

Una de las características más importantes de los controles de formulario de HTML5 es la capacidad de validar la mayoría de los datos de usuario sin depender de JavaScript. Esto se realiza mediante el uso de atributos de validación en los elementos del formulario. Los hemos visto anteriormente en el curso, pero recapitulamos aquí:

- required (en-US): Especifica si un campo de formulario debe completarse antes de que se pueda enviar el formulario.
- minlength y maxlength (en-US): Especifican la longitud mínima y máxima de los datos de texto (cadenas).
- min y max (en-US): Especifican los valores mínimo y máximo de los tipos de entrada numéricos.
- type : Especifica si los datos deben ser un número, una dirección de correo electrónico o algún otro tipo de preajuste específico.
- pattern (en-US): Especifica una expresión regular que define un patrón que los datos que se introduzcan deben seguir.

Si los datos que se introducen en un campo de formulario siguen todas las reglas que especifican los atributos anteriores, se consideran válidos. Si no, se consideran no válidos.

Cuando un elemento es válido, se cumplen los aspectos siguientes:

- El elemento coincide con la pseudoclase :valid de CSS, lo que te permite aplicar un estilo específico a los elementos válidos.
- Si el usuario intenta enviar los datos, el navegador envía el formulario siempre que no haya nada más que lo impida (por ejemplo, JavaScript).

Cuando un elemento no es válido, se cumplen los aspectos siguientes:

- El elemento coincide con la pseudoclase :invalid de CSS, y a veces con otras pseudoclases de interfaz de usuario (UI) –por ejemplo, :out-of-range – dependiendo del error, que te permite aplicar un estilo específico a elementos no válidos.
- Si el usuario intenta enviar los datos, el navegador bloquea el formulario y muestra un mensaje de error.

> Nota: Hay varios errores que evitan que el formulario se envíe, incluidos badInput (en-US), patternMismatch (en-US), rangeOverflow (en-US) o rangeUnderflow (en-US), stepMismatch (en-US), tooLong (en-US) o tooShort (en-US), typeMismatch (en-US), valueMissing (en-US) o customError .

Ejemplos de validación de formularios incorporados

En esta sección probaremos algunos de los atributos que hemos comentado antes.

Archivo de inicio sencillo

Vamos a empezar con un ejemplo sencillo: una entrada que te permite elegir si prefieres un plátano o una cereza. Este ejemplo implica una simple entrada (<input>) de texto con una etiqueta (<label>) asociada y un botón de envío (<button>). Puedes encontrar el código fuente en GitHub en fruit-start.html y el ejemplo en vivo a continuación.

```
<form>
  <label for="choose">¿Prefieres un plátano o una cereza?</label>
  <input id="choose" name="i_like">
  <button>Enviar</button>
</form>
```

```
input:invalid {
  border: 2px dashed red;
}

input:valid {
  border: 2px solid black;
}
```

Para empezar, haz una copia de `fruit-start.html` en un nuevo directorio de tu disco duro.

El atributo `required`

La característica de validación de HTML5 más simple es el atributo `required` (en-US). Añade este atributo al elemento para que una entrada sea obligatoria. Cuando se establece este atributo, el elemento coincide con la pseudoclase de la interfaz de usuario `:required` y el formulario no se envía; muestra un mensaje de error al enviarlo si la entrada está vacía. Si está vacía, la entrada también se considera inválida, coincidiendo con la pseudoclase de interfaz de usuario `:invalid` .

Añade un atributo `required` a tu entrada, como se muestra a continuación.

```
<form>
  <label for="choose">¿Prefieres un plátano o una cereza? (requerido) </label>
  <input id="choose" name="i_like" required>
  <button>Enviar</button>
</form>
```

Ten en cuenta el CSS que en el archivo de ejemplo se ha incluido:

```
input:invalid {
  border: 2px dashed red;
}

input:invalid:required {
  background-image: linear-gradient(to right, pink, lightgreen);
}

input:valid {
  border: 2px solid black;
}
```

Este CSS da un borde discontinuo rojo cuando la entrada no es válida, y un borde negro sólido más sutil cuando es válida. También añadimos un gradiente de fondo cuando la entrada es obligatoria *y* no válida. Prueba el nuevo comportamiento en el ejemplo siguiente:

> Nota: Puedes encontrar este ejemplo en vivo en GitHub como fruit-validation.html (consulta también el código fuente).

Intenta enviar el formulario sin introducir ningún valor. Observa que la entrada no válida recibe el cursor, aparece un mensaje de error predeterminado («Complete este campo») y el formulario no se puede enviar.

La presencia del atributo `required` en cualquier elemento que admite este atributo significa que el elemento coincide con la pseudoclase `:required`, tenga o no un valor. Si en el elemento `<input>` no se ha introducido ningún valor, `input` coincidirá con la pseudoclase `:invalid`.

> Nota: Para una buena experiencia de usuario, indica al usuario que campos de formulario se requieren. No solo es una buena experiencia de usuario, sino que lo exigen las pautas de accesibilidad de WCAG. Además, solo requiere que los usuarios introduzcan los datos que realmente necesitas: Por ejemplo, ¿por qué realmente necesitas saber el género o el tratamiento de alguien?

Validación de una expresión regular

Otra característica útil de validación es el atributo pattern (en-US), que espera una expresión regular como valor. Una expresión regular (*regex*) es un patrón que se puede usar para establecer combinaciones de caracteres en cadenas de texto, por lo que las expresiones regulares son ideales para la validación de formularios y sirven para una gran variedad de otros usos en JavaScript.

Las expresiones regulares son bastante complejas y no vamos a exponerlas exhaustivamente en este artículo. A continuación hay algunos ejemplos para que te hagas una idea de cómo funcionan.

- a : coincide con un carácter que es a (ni b , ni aa , etc.).
- abc : coincide con a , seguido de b , seguido de c .
- ab?c : coincide con a , seguida opcionalmente de una sola b , seguida de c (ac o abc).
- ab*c : coincide con a , seguido opcionalmente de cualquier número de b , seguido de c . (ac , abc , abbbbbc , etc.)
- a|b : coincide con un carácter que es a o b .
- abc|xyz : coincide exactamente con abc o xyz (pero no con abcxyz a o y , y así sucesivamente).

Hay muchas más posibilidades que no exponemos aquí. Para obtener una lista completa y muchos ejemplos, consulta nuestro documento de expresiones regulares.

Implementemos un ejemplo. Actualiza tu HTML para añadir un atributo pattern (en-US) como este:

```
<form>
  <label for="choose">¿Prefieres un plátano o una cereza?</label>
  <input id="choose" name="i_like" required pattern="[Pp]látano|[Cc]ereza ">
  <button>Enviar</button>
</form>
```

Esto nos da la siguiente actualización; pruébalo:

> Nota: Puedes encontrar este ejemplo en vivo en GitHub como fruit-pattern.html (consulta también su código fuente).

En este ejemplo, el elemento <input> acepta uno de los cuatro valores posibles: las cadenas «plátano», «Plátano», «cereza» o «Cereza». Las expresiones regulares distinguen entre mayúsculas y minúsculas, pero hemos hecho que admita versiones en mayúsculas y minúsculas utilizando un patrón «Aa» adicional anidado dentro de corchetes.

En este punto, intenta cambiar el valor dentro del atributo pattern (en-US) para que se vean iguales que algunos de los ejemplos vistos anteriormente, y observa que esto afecta a los valores que puedes añadir para que el valor de entrada sea válido. Intenta escribir algo por tu cuenta y mira cómo va. ¡Haz que estén relacionadas con la fruta siempre que sea posible para que tus ejemplos tengan sentido!

Si un valor no vacío de <input> no coincide con el patrón de la expresión regular, input coincidirá con la pseudoclase :invalid .

> Nota: Algunos tipos de elementos <input> no necesitan validar una expresión regular con el atributo pattern (en-US). Especificar el tipo de correo electrónico (email), por ejemplo, valida el valor de las entradas con un patrón de dirección de correo electrónico bien formado o un patrón que coincida con una lista de direcciones de correo electrónico separadas por comas si tiene el atributo multiple .

> Nota: El elemento <textarea> no admite el atributo pattern (en-US).

Restringir la longitud de tus entradas

Puedes restringir la longitud de los caracteres de todos los campos de texto creados por <input> o <textarea> utilizando los atributos minlength y maxlength (en-US). Un campo no es válido si tiene un valor y ese valor tiene menos caracteres que el valor de longitud mínima (minlength), o más que el valor de longitud máxima (maxlength (en-US)).

Los navegadores a menudo no permiten que el usuario escriba un valor más largo de lo esperado en los campos de texto. Lo que otorga una mejor experiencia de usuario que maxlength es proporcionar comentarios de recuento de caracteres de manera accesible y permitirles editar su contenido a un tamaño más reducido. Un ejemplo de esto es el límite de caracteres de Twitter. JavaScript, incluidas las soluciones que utilizan maxlength , se puede utilizar para proporcionar esta funcionalidad.

Restringir los valores de tus entradas

Los atributos min y max (en-US) se pueden usar para proporcionar a los campos numéricos (es decir, <input type="number">) un rango de valores válidos. El campo no será válido si contiene un valor fuera de este rango.

Veamos otro ejemplo. Crea una nueva copia del archivo fruit-start.html.

Ahora elimina el contenido del elemento <body> y reemplázalo con lo siguiente:

```
<form>
  <div>
    <label for="choose">¿Prefieres un plátano o una cereza?</label>
    <input type="text" id="choose" name="i_like" required minlength="6" maxlength="6">
  </div>
  <div>
    <label for="number">¿Cuántos te gustaría comer?</label>
    <input type="number" id="number" name="amount" value="1" min="1" max="10">
  </div>
  <div>
    <button>Enviar</button>
  </div>
</form>
```

- Aquí verás que le hemos dado al campo de text unos valores minlength y maxlength de seis, que es la misma longitud que tienen el plátano y la cereza.
- También le hemos dado al campo number un min de uno y un max de diez. Los números introducidos que queden fuera de este rango se mostrarán como no válidos; los usuarios no podrán usar las flechas de incremento/decremento para mover el valor fuera de este rango. Si el usuario introduce un número desde el teclado fuera de este rango, los datos no serán válidos. El número no es obligatorio, por lo que eliminar el valor aún dará como resultado un valor válido.

Aquí está el ejemplo que se ejecuta en vivo:

> Nota: Puedes encontrar este ejemplo en vivo en GitHub como fruit-length.html (consulta también su código fuente).

> Nota: <input type="number"> (y otros tipos, como range y date) también pueden tomar un atributo step (en-US), que especifica en qué incremento aumenta o disminuye el valor cuando se utilizan los controles de entrada (como el botones numéricos arriba y abajo). En el ejemplo anterior no hemos incluido un atributo step , por lo que el valor predeterminado es 1 . Esto significa que los valores de coma flotante, como 3.2, también se mostrarán como no válidos.

Ejemplo completo

Aquí hay un ejemplo completo que muestra el uso de las funciones de validación integradas en HTML. En primer lugar, un poco de HTML:

```
<form>
  <p>
    <fieldset>
      <legend>¿Tienes carné de conducir?<abbr title="Este campo es obligatorio" aria-
label="required">*</abbr></legend>
      <!-- Solo se puede seleccionar un botón de opción en un grupo con el mismo nombre,
           y, por lo tanto, solo un botón de opción en un grupo con el mismo nombre que tiene marcado
el atributo «requerido»
           basta para hacer de una selección un requisito -->
      <input type="radio" required name="driver" id="r1" value="yes"><label for="r1">Sí</label>
      <input type="radio" required name="driver" id="r2" value="no"><label for="r2">No</label>
    </fieldset>
  </p>
  <p>
    <label for="n1">¿Qué edad tienes?</label>
    <!-- El atributo pattern puede actuar como una alternativa para los navegadores que
         no implementan el tipo de entrada de número, pero admiten el atributo pattern.
         Ten en cuenta que los navegadores que admiten el atributo pattern lo harán
         fallar silenciosamente cuando se use con un campo numérico.
         Su uso aquí solo actúa como una alternativa -->
     <input type="number" min="12" max="120" step="1" id="n1" name="age"
           pattern="\d+">
  </p>
  <p>
    <label for="t1">¿Cuál es tu fruta favorita?<abbr title="Este campo es obligatorio" aria-
label="required">*</abbr></label>
    <input type="text" id="t1" name="fruit" list="l1" required
           pattern="[Bb]anana|[Cc]herry|[Aa]pple|[Ss]trawberry|[Ll]emon|[Oo]range ">
    <datalist id="l1">
      <option>Plátano</option>
      <option>Cereza</option>
      <option>Manzana</option>
      <option>Fresa</option>
      <option>Limón</option>
      <option>Naranja</option>
     </datalist>
  </p>
  <p>
    <label for="t2">¿Cuál es tu dirección de correo electrónico? </label>
    <input type="email" id="t2" name="email">
  </p>
  <p>
    <label for="t3">Deja un mensaje</label>
```

```
    <textarea id="t3" name="msg" maxlength="140" rows="5"></textarea>
  </p>
  <p>
    <button>Enviar</button>
  </p>
</form>
```

Y ahora, algo de CSS para añadir estilo al HTML:

```
form {
  font: 1em sans-serif;
  max-width: 320px;
}

p > label {
  display: block;
}

input[type="text"],
input[type="email"],
input[type="number"],
textarea,
fieldset {
  width : 100%;
  border: 1px solid #333;
  box-sizing: border-box;
}

input:invalid {
  box-shadow: 0 0 5px 1px red;
}

input:focus:invalid {
  box-shadow: none;
}
```

Esto se traduce de la siguiente manera:

Consulta Atributos relacionados con la validación para obtener una lista completa de los atributos que se pueden usar para restringir los valores de entrada y los tipos de entrada que los admiten.

Nota: Puedes encontrar este ejemplo en vivo en GitHub como full-example.html (consulta

también su código fuente).

Validar formularios utilizando JavaScript

Debes usar JavaScript si quieres controlar la apariencia de los mensajes de error nativos o tratar con navegadores heredados que no admiten la validación de formularios incorporados en HTML. En esta sección veremos las diferentes formas de hacer esto.

La API de validación de restricciones

La mayoría de los navegadores admiten la API de validación de restricciones (en-US), que consta de un conjunto de métodos y propiedades disponibles en las interfaces DOM de elementos de formulario siguientes:

- HTMLButtonElement (en-US) (representa un elemento <button>)
- HTMLFieldSetElement (en-US) (representa un elemento <fieldset>)
- HTMLInputElement (representa un elemento <input>)
- HTMLOutputElement (en-US) (representa un elemento <output> (en-US))
- HTMLSelectElement (representa un elemento <select>)
- HTMLTextAreaElement (en-US) (representa un elemento <textarea>)

La API de validación de restricciones hace que las propiedades siguientes estén disponibles en los elementos anteriores.

- validationMessage : Devuelve un mensaje localizado que describe las restricciones de validación que el control no satisface (si corresponde). Si el control no es candidato para la validación de restricciones (willValidate es false) o el valor del elemento satisface sus restricciones (es válido), esto devolverá una cadena vacía.
- validity : Devuelve un objeto ValidityState que contiene varias propiedades que describen el estado de validez del elemento. Puedes encontrar todos los detalles de todas las propiedades disponibles en la página de referencia ValidityState (en-US); a continuación se enumeran algunos de los más comunes:
  - patternMismatch (en-US): Devuelve true si el valor no coincide con el pattern especificado, y false si coincide. Si es verdadero, el elemento coincide con la pseudoclase :invalid de CSS.
  - tooLong (en-US): Devuelve true si el valor es mayor que la longitud máxima especificada por el atributo maxlength , o false si es menor o igual al máximo. Si es verdadero, el elemento coincide con la pseudoclase :invalid de CSS.

- tooShort (en-US): Devuelve true si el valor es menor que la longitud mínima especificada por el atributo minlength, o false si es mayor o igual al mínmo. Si es verdadero, el elemento coincide con la pseudoclase :invalid de CSS.
- rangeOverflow (en-US): Devuelve true si el valor es mayor que el máximo especificado por el atributo max, o false si es menor o igual que el máximo. Si es verdadero, el elemento coincide con las pseudoclases :invalid y :out-of-range de CSS.
- rangeUnderflow (en-US): Devuelve true si el valor es menor que el mínimo especificado por el atributo min, o false si es mayor o igual que el mínimo. Si es verdadero, el elemento coincide con las pseudoclases :invalid y :out-of-range de CSS.
- typeMismatch (en-US): Devuelve true si el valor no está en la sintaxis requerida (cuando type es email o url), o false si la sintaxis es correcta. Si es verdadero, el elemento coincide con la pseudoclase :invalid de CSS.
- valid : Devuelve true si el elemento cumple con todas sus restricciones de validación y por lo tanto se considera válido, o false si falla alguna restricción. Si es verdadero, el elemento coincide con la pseudoclase :valid de CSS; o con la pseudoclase :invalid de CSS de lo contrario.
- valueMissing : Devuelve true si el elemento tiene un atributo required pero no tiene valor, o false de lo contrario. Si es verdadero, el elemento coincide con la pseudoclase :invalid de CSS.

- willValidate : Devuelve true si el elemento se valida cuando se envía el formulario; false de lo contrario.

La API de validación de restricciones también pone a disposición los siguientes métodos en los elementos anteriores.

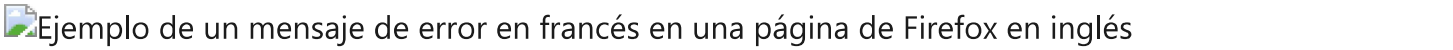
- checkValidity() : Devuelve true si el valor del elemento no tiene problemas de validez; false en caso contrario. Si el elemento no es válido, este método también activa un evento invalid en el elemento.
- setCustomValidity(message) : Añade un mensaje de error personalizado al elemento; si configuras un mensaje de error personalizado, el elemento se considera no válido y se muestra el error especificado. Esto te permite utilizar el código JavaScript para establecer un fallo de validación distinto de los ofrecidos por las restricciones estándar de validación de HTML5. El mensaje se muestra al usuario cuando se informa del problema.

Implementar un mensaje de error personalizado

Como has visto en los ejemplos de restricciones de validación de HTML5 anteriores, cada vez que un usuario intenta enviar un formulario no válido, el navegador muestra un mensaje de error. La forma en que se muestra este mensaje depende del navegador.

Estos mensajes automatizados tienen dos inconvenientes:

- No hay una forma estándar de cambiar su aspecto con CSS.
- Dependen de la configuración regional del navegador, lo que significa que puedes tener una página en un idioma pero un mensaje de error en otro idioma, como se ve en la siguiente captura de pantalla de Firefox.


Ejemplo de un mensaje de error en francés en una página de Firefox en inglés

La personalización de estos mensajes de error es uno de los casos de uso más comunes de la API de validación de restricciones (en-US). Veamos un ejemplo simple de cómo hacer esto.

Comenzaremos con un HTML simple (siéntete libre de poner esto en un archivo HTML en blanco; usa una copia nueva de fruit-start.html como base, si lo deseas):

```
<form>
  <label for="mail">Me gustaría que me proporcionara una dirección de correo electrónico:<label>
  <input type="email" id="mail" name="mail">
  <button>Enviar</button>
</form>
```

Y añade a la página el JavaScript siguiente:

```
const email = document.getElementById("mail");

email.addEventListener("input", function (event) {
  if (email.validity.typeMismatch) {
    email.setCustomValidity("¡Se esperaba una dirección de correo electrónico!");
  } else {
    email.setCustomValidity("");
  }
});
```

Aquí guardamos una referencia para la entrada de la dirección de correo electrónico, luego le añadimos un detector de eventos que ejecuta el código de `content` cada vez que el valor de la entrada cambia.

Dentro del código que contiene, verificamos si la propiedad `validity.typeMismatch` de la entrada de la dirección de correo electrónico devuelve `true`, lo que significa que el valor que contiene no coincide con el patrón para una dirección de correo electrónico bien formada. Si es así, llamamos al método `setCustomValidity()` con un mensaje personalizado. Esto hace que la entrada no sea válida, de modo que cuando intentas enviar el formulario, el envío falla y se muestra el mensaje de error personalizado.

Si la propiedad validity.typeMismatch devuelve false , llamamos al método setCustomValidity() con una cadena vacía. Esto hace que la entrada sea válida, y el formulario se envía.

Puedes probarlo a continuación:

> Nota: Puede encontrar este ejemplo vivo en GitHub como custom-error-message.html (véase también su código fuente).

Un ejemplo más detallado

Ahora que hemos visto un ejemplo realmente sencillo, veamos cómo podemos usar esta API para construir una validación personalizada un poco más compleja.

En primer lugar, el código HTML. Una vez más, siéntete libre de construir esto junto con nosotros:

```
<form novalidate>
  <p>
    <label for="mail">
      <span>Por favor, introduzca una dirección de correo electrónico: </span>
      <input type="email" id="mail" name="mail" required minlength="8">
      <span class="error" aria-live="polite"></span>
    </label>
  </p>
  <button>Enviar</button>
</form>
```

Este sencillo formulario usa el atributo novalidate para desactivar la validación automática del navegador; esto permite que nuestra secuencia de comandos tome control sobre la validación. Sin embargo, esto no deshabilita la compatibilidad para la API de validación de restricciones ni la aplicación de pseudoclases de CSS como :valid , etc. Eso significa que, aunque el navegador no verifica automáticamente la validez del formulario antes de enviar los datos, puedes hacerlo tú mismo y diseñar el formulario en consecuencia.

Nuestra entrada para validar es <input type="email"> , que es obligatoria y tiene una longitud mínima (minlength) de 8 caracteres. Vamos a verificar esto con nuestro propio código para que muestre un mensaje de error personalizado para cada elemento.

Nuestro objetivo es mostrar los mensajes de error dentro de un elemento . El atributo aria-live (en-US) se establece en ese para asegurar que todo el mundo podrá ver nuestro mensaje

de error personalizado, incluidos los usuarios de lectores de pantalla.

> Nota: Un punto clave a tener en cuenta es que establecer el atributo `novalidate` en el formulario impide que el formulario muestre sus propios cuadros de diálogo de error, y nos permite mostrar los mensajes de error personalizados en el DOM de la manera que nosotros elijamos.

Ahora aplicaremos algo de CSS básico para mejorar ligeramente el aspecto del formulario y proporcionar algunos comentarios visuales cuando los datos de entrada no sean válidos:

```
body {
  font: 1em sans-serif;
  width: 200px;
  padding: 0;
  margin : 0 auto;
}

p * {
  display: block;
}

input[type=email]{
  -webkit-appearance: none;
  appearance: none;

  width: 100%;
  border: 1px solid #333;
  margin: 0;

  font-family: inherit;
  font-size: 90%;

  box-sizing: border-box;
}

/* Este es nuestro diseño para los campos no válidos */
input:invalid{
  border-color: #900;
  background-color: #FDD;
}

input:focus:invalid {
```

```
  outline: none;
}

/* Este es el diseño para nuestros mensajes de error */
.error {
  width : 100%;
  padding: 0;

  font-size: 80%;
  color: white;
  background-color: #900;
  border-radius: 0 0 5px 5px;

  box-sizing: border-box;
}

.error.active {
  padding: 0.3em;
}
```

Vamos a ver el JavaScript que implementa la validación de error personalizada.

```
// Hay muchas formas de elegir un nodo DOM; aquí obtenemos el formulario y, a continuación, el
campo de entrada
// del correo electrónico, así como el elemento span en el que colocaremos el mensaje de error.
const form  = document.getElementsByTagName('form')[0];

const email = document.getElementById('mail');
const emailError = document.querySelector('#mail + span.error');

email.addEventListener('input', function (event) {
  // Cada vez que el usuario escribe algo, verificamos si
  // los campos del formulario son válidos.

  if (email.validity.valid) {
    // En caso de que haya un mensaje de error visible, si el campo
    // es válido, eliminamos el mensaje de error.
    emailError.innerHTML = ''; // Restablece el contenido del mensaje
    emailError.className = 'error'; // Restablece el estado visual del mensaje
  } else {
    // Si todavía hay un error, muestra el error exacto
    showError();
  }
});

form.addEventListener('submit', function (event) {
```

```
  // si el campo de correo electrónico es válido, dejamos que el formulario se envíe

  if(!email.validity.valid) {
    // Si no es así, mostramos un mensaje de error apropiado
    showError();
    // Luego evitamos que se envíe el formulario cancelando el evento
    event.preventDefault();
  }
});

function showError() {
  if(email.validity.valueMissing) {
    // Si el campo está vacío
    // muestra el mensaje de error siguiente.
    emailError.textContent = 'Debe introducir una dirección de correo electrónico.';
  } else if(email.validity.typeMismatch) {
    // Si el campo no contiene una dirección de correo electrónico
    // muestra el mensaje de error siguiente.
    emailError.textContent = 'El valor introducido debe ser una dirección de correo electrónico.';
  } else if(email.validity.tooShort) {
    // Si los datos son demasiado cortos
    // muestra el mensaje de error siguiente.
    emailError.textContent = 'El correo electrónico debe tener al menos ${ email.minLength }
caracteres; ha introducido ${ email.value.length }.';
  }

  // Establece el estilo apropiado
  emailError.className = 'error activo';
}
```

Los comentarios explican las cosas bastante bien, pero de una manera muy breve:

- Cada vez que cambiamos el valor de la entrada, verificamos si contiene datos válidos. Si es así, eliminamos cualquier mensaje de error que se muestre. Si los datos no son válidos, ejecutamos `showError()` para mostrar el error correspondiente.
- Cada vez que intentamos enviar el formulario, verificamos nuevamente si los datos son válidos. Si es así, dejamos que envíe el formulario. Si no, ejecutamos `showError()` para mostrar el error correspondiente y detenemos el envío del formulario con `preventDefault()`.
- La función `showError()` usa varias propiedades de la entrada `validity` para determinar cuál es el error y luego muestra un mensaje de error según corresponda.

Este es el resultado:

> Nota: Puedes encontrar este ejemplo en vivo en GitHub como detailed-custom-validation.html (consulta también su código fuente).

La API de validación de restricciones te proporciona una herramienta poderosa para manejar la validación de formularios, y te permite tener un control enorme sobre la interfaz de usuario más allá de lo que puedas hacer solo con HTML y CSS.

> Nota: Para obtener más información, consulta nuestra guía de validación de restricciones y la referencia de API de validación de restricciones (en-US).

Validar formularios sin una API incorporada

En algunos casos, como la compatibilidad heredada del navegador o los controles personalizados, no podrás o no querrás usar la API de validación de restricciones. Todavía puedes usar JavaScript para validar tu formulario, pero vas a tener que escribirlo.

Antes de validar el formulario, hazte estas preguntas:

¿Qué tipo de validación debería realizar?

Debes determinar cómo validar los datos: operaciones de cadena, conversión de tipos, expresiones regulares, etc. Tú decides.

¿Qué debo hacer si el formulario no se valida?

Esto es claramente un problema de la interfaz de usuario. Tienes que decidir cómo se comportará el formulario. ¿El formulario va a enviar los datos de todos modos? ¿Deberías resaltar los campos que dan error? ¿Deberías mostrar mensajes de error?

¿Cómo puedo ayudar al usuario a corregir datos no válidos?

Para reducir la frustración del usuario, es muy importante proporcionar tanta información útil como sea posible para guiarlo a fin de que corrija sus entradas de datos. Debes ofrecer sugerencias por adelantado para que sepan lo que se espera de ellos, así como mensajes de error claros. Si deseas profundizar en los requisitos de interfaz de usuario para la validación de formularios, aquí hay algunos artículos útiles que debes leer:

- SmashingMagazine: Form-Field Validation: The Errors-Only Approach [Validación de campo de formulario: El enfoque de solo errores]

- SmashingMagazine: Web Form Validation: Best Practices and Tutorials [Validación de formularios web: Buenas prácticas y tutoriales]
- Six Revision: Best Practices for Hints and Validation in Web Forms [Buenas prácticas para sugerencias y validación de formularios web]
- A List Apart: Inline Validation in Web Forms [Validación en línea de formularios web]

Un ejemplo que no usa la API de validación de restricciones

Para ilustrar esto, mostramos una versión simplificada del ejemplo anterior que funciona con navegadores con compatibilidad heredada.

El HTML es casi el mismo; solo hemos eliminado las funciones de validación de HTML.

```
<form>
  <p>
    <label for="mail">
        <span>Por favor, introduzca una dirección de correo electrónico: </span>
        <input type="text" class="mail" id="mail" name="mail">
        <span class="error" aria-live="polite"></span>
    </label>
  </p>
  <!-- Algunos navegadores con compatibilidad heredada deben tener el atributo «type»
       establecido explícitamente en el elemento «button» de «submit»-->
  <button type="submit">Enviar</button>
</form>
```

Del mismo modo, no es necesario cambiar mucho el CSS; acabamos de convertir la pseudoclase `:invalid` de CSS en una clase real y evitamos usar el selector de atributos que no funciona en Internet Explorer 6.

```
body {
  font: 1em sans-serif;
  width: 200px;
  padding: 0;
  margin : 0 auto;
}

form {
  max-width: 200px;
}

p * {
```

```
  display: block;
}

input.mail {
  -webkit-appearance: none;

  width: 100%;
  border: 1px solid #333;
  margin: 0;

  font-family: inherit;
  font-size: 90%;

  box-sizing: border-box;
}

/* Este es nuestro diseño para los campos no válidos */
input.invalid{
  border-color: #900;
  background-color: #FDD;
}

input:focus.invalid {
  outline: none;
}

/* Este es el diseño para nuestros mensajes de error */
.error {
  width : 100%;
  padding: 0;

  font-size: 80%;
  color: white;
  background-color: #900;
  border-radius: 0 0 5px 5px;
  box-sizing: border-box;
}

.error.active {
  padding: 0.3em;
}
```

Los grandes cambios están en el código JavaScript, que necesita hacer mucho más trabajo pesado.

```
// Hay menos formas de elegir un nodo DOM con navegadores antiguos
const form  = document.getElementsByTagName('form')[0];
```

```
const email = document.getElementById('mail');

// Lo siguiente es un truco para llegar al siguiente nodo de elementos hermanos en el DOM
// Esto es peligroso porque puedes construir fácilmente un bucle infinito.
// En los navegadores modernos es mejor usar element.nextElementSibling
let error = email;
while ((error = error.nextSibling).nodeType != 1);

// según la especificación HTML5
const emailRegExp = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;

// Muchos navegadores antiguos no son compatibles con el método addEventListener.
// Aquí hay una manera simple de manejar esto; está lejos de ser la única.
function addEvent(element, event, callback) {
  let previousEventCallBack = element["on"+event];
  element["on"+event] = function (e) {
    const output = callback(e);

    // Una devolución de llamada que devuelve «false» detiene la cadena de devolución de llamada
    // e interrumpe la ejecución de la devolución de llamada del evento.
    if (output === false) return false;

    if (typeof previousEventCallBack === 'function') {
      output = previousEventCallBack(e);
      if(output === false) return false;
    }
  }
};

// Ahora podemos reconstruir nuestra restricción de validación
// Debido a que no confiamos en la pseudoclase de CSS, tenemos que
// establecer explícitamente la clase valid/invalid en nuestro campo de correo electrónico
addEvent(window, "load", function () {
  // Aquí probamos si el campo está vacío (recuerda, el campo no es obligatorio)
  // Si no es así, verificamos si su contenido es una dirección de correo electrónico con el formato
correcto.
  const test = email.value.length === 0 || emailRegExp.test(email.value);

  email.className = test ? "valid" : "invalid";
});

// Esto define lo que sucede cuando el usuario escribe en el campo
addEvent(email, "input", function () {
  const test = email.value.length === 0 || emailRegExp.test(email.value);
  if (test) {
    email.className = "valid";
```

```
    error.innerHTML = "";
    error.className = "error";
  } else {
    email.className = "invalid";
  }
});

// Esto define lo que sucede cuando el usuario intenta enviar los datos.
addEvent(form, "submit", function () {
  const test = email.value.length === 0 || emailRegExp.test(email.value);

  if (!test) {
    email.className = "invalid";
    error.innerHTML = "I expect an e-mail, darling!";
    error.className = "error active";

    // Algunos navegadores antiguos no son compatibles con el método event.preventDefault ()
    return false;
  } else {
    email.className = "valid";
    error.innerHTML = "";
    error.className = "error";
  }
});
```

El resultado es el siguiente:

Como puedes ver, no es tan difícil construir un sistema de validación por tu cuenta. La parte difícil es hacer que sea lo suficientemente genérico para que se pueda usar en diferentes plataformas y en cualquier forma. Hay muchas bibliotecas de archivos disponibles para realizar la validación de formularios, como por ejemplo Validate.js

.

Prueba tus habilidades!

Has llegado al final de este artículo pero ¿puedes recordar la información más importante?Puedes encontrar pruebas adicionales para comprovar que has comprendido la información antes de que continue — visita Prueba tus habilidades: Validación de formularios (en-US).

Resumen

La validación de formularios en el lado del cliente a veces requiere JavaScript si deseas personalizar el estilo y los mensajes de error, pero *siempre* requiere que pienses cuidadosamente en el usuario.

Recuerda que siempre debes ayudar a tus usuarios a corregir los datos que proporcionan. Para ese fin, asegúrate de:

- Mostrar mensajes de error explícitos.
- Ser permisivo con el formato de entrada.
- Señalar exactamente dónde se produce el error, especialmente en formularios extensos.

Una vez que hayas verificado que el formulario se ha completado correctamente, puedes proceder a enviarlo. Vamos a exponer el envío de los datos del formulario en el próximo artículo.