WebForms

Ruldin Efrain Ayala Ramos

104

Formularios Web

- La Web 2.0 está completamente enfocada en el usuario.
- Y cuando el usuario es el centro de atención, todo está relacionado con interfaces, en cómo hacerlas más intuitivas, más naturales, más prácticas, y por supuesto más atractivas.
- Los formularios web son la interface más importante de todas, permiten a los usuarios insertar datos, tomar decisiones, comunicar información y cambiar el comportamiento de una aplicación.

El elemento <form> ejemplo 1

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Formularios</title>
</head>
<body>
    <section>
    <form name="miformulario" id="miformulario" method="get">
    <input type="text" name="nombre" id="nombre">
    <input type="submit" value="Enviar">
    </form>
    </section>
</body>
</html>
```

Atributos nuevos:

- **autocomplete** Este es un viejo atributo que se ha vuelto estándar en esta especificación.
- Puede tomar dos valores: **on** y **off**.
- El valor por defecto es **on**.
- Cuando es configurado como **off** los elementos **<input>** pertenecientes a ese formulario tendrán la función de autocompletar desactivada, sin mostrar entradas previas como posibles valores.
- Puede ser implementado en el elemento **<form>** o en cualquier elemento **<input>** independientemente.
- **novalidate** Una de las características de formularios en HTML5 es la capacidad propia de validación.
- Los formularios son automáticamente validados.
- Para evitar este comportamiento, podemos usar el atributo **novalidate**.
- Para lograr lo mismo para elementos **<input>** específicos, existe otro atributo llamado **formnovalidate**.
- Ambos atributos son booleanos, ningún valor tiene que ser especificado (su presencia es suficiente para activar su función).

El elemento <input>

- El elemento más importante en un formulario es **<input>**.
- Este elemento puede cambiar sus características gracias al atributo **type** (tipo).
- Este atributo determina qué clase de entrada es esperada desde el usuario.
- En HTML5 estos nuevos tipos no solo están especificando qué clase de entrada es esperada sino también diciéndole al navegador qué debe hacer con la información recibida.
- El navegador procesará los datos ingresados de acuerdo al valor del atributo **type** y validará la entrada o no.

Tipo email

- `<input type="email" name="miemail" id="miemail">`
- El texto insertado en este campo será controlado por el navegador y validado como un email.
- Si la validación falla, el formulario no será enviado.
- Cómo cada navegador responderá a una entrada inválida no está determinado en la especificación de HTML5.
- Por ejemplo, algunos navegadores mostrarán un borde rojo alrededor del elemento **<input>** que produjo el error y otros lo mostrarán en azul.
- Existen formas de personalizar esta respuesta, pero las veremos más adelante.

Tipo search

- El tipo **search** (búsqueda) no controla la entrada, es solo una indicación para los navegadores.
- Al detectar este tipo de campo algunos navegadores cambiarán el diseño del elemento para ofrecer al usuario un indicio de su propósito.

Tipo url

- Este tipo de campo trabaja exactamente igual que el tipo **email** pero es específico para direcciones web.
- Está destinado a recibir solo URLs absolutas y retornará un error si el valor es inválido.
- `<input type="url" name="miurl" id="miurl">`

Tipo tel

- Este tipo de campo es para números telefónicos.
- A diferencia de los tipos **email** y **url**, el tipo **tel** no requiere ninguna sintaxis en particular.
- Es solo una indicación para el navegador en caso de que necesite hacer ajustes de acuerdo al dispositivo en el que la aplicación es ejecutada.
- `<input type="tel" name="telefono" id="telefono">`

Tipo number

- Como su nombre lo indica, el tipo **number** es sólo válido cuando recibe una entrada numérica.
- Existen algunos atributos nuevos que pueden ser útiles para este campo:
- **min** El valor de este atributo determina el mínimo valor aceptado para el campo.
- **max** El valor de este atributo determina el máximo valor aceptado para el campo.
- **step** El valor de este atributo determina el tamaño en el que el valor será incrementado o disminuido en cada paso.
- Por ejemplo, si declara un valor de 5 para **step** en un campo que tiene un valor mínimo de 0 y máximo de 10, el navegador no le permitirá especificar valor es entre 0 y 5 o entre 5 y 10.
- `<input type="number" name="numero" id="numero" min=”0” max=”10”`
- `step=”5”>`

Tipo range

- <input type="number" name="numero" id="numero" min=”0” max=”10” step=”5”>

Otros Tipos

- **Tipo date:** <input type="date" name="fecha" id="fecha">
- **Tipo week:** <input type="week" name="semana" id="semana">
- Normalmente el valor esperado tiene la sintaxis **2011-W50** donde **2011** es al año y **50** es el número de la semana
- **Tipo month:**<input type="month" name="mes" id="mes">
- **Tipo time:**<input type="time" name="hora" id="hora">
- **Tipo datetime:** <input type="datetime" name="fechahora" id="fechahora">
- **Tipo datetime-local:** <input type="datetime-local" name="tiempolocal" id="tiempolocal"> no tiene zona horaria
- **Tipo color:** <input type="color" name="micolor" id="micolor">

Otros atributos de input

- **Atributo placeholder:**
- <input type="search" name="busqueda" id="busqueda" **placeholder="escriba su búsqueda"**>
- **Atributo required:** <input type="email" name="miemail" id="miemail" **required**>
- **Atributo multiple:** <input type="email" name="miemail" id="miemail" **multiple**>
- **Atributo autofocus:**
- Esta es una función que muchos desarrolladores aplicaban anteriormente utilizando el método **focus()** de Javascript.
- Este método era efectivo pero forzaba el foco sobre el elemento seleccionado, incluso cuando el usuario ya se encontraba posicionado en otro diferente.
- El atributo **autofocus** enfocará la página web sobre el elemento seleccionado pero considerando la situación actual.
- <input type="search" name="busqueda" id="busqueda" **autofocus**>

Atributo pattern

- El atributo **pattern** es para propósitos de validación.
- Usa expresiones regulares para personalizar reglas de validación.
- Algunos de los tipos de campo ya estudiados validan cadenas de texto específicas, pero no permiten hacer validaciones personalizadas, como por ejemplo un código postal que consiste en 5 números.
- No existe ningún tipo de campo predeterminado para esta clase de entrada.
- El atributo **pattern** nos permite crear nuestro propio tipo de campo para controlar esta clase de valores no ordinarios.
- Puede incluso incluir un atributo **title** para personalizar mensajes de error.

```
<input pattern=“[0-9]{5}" name="codigopostal"
id="codigopostal”
title=”inserte los 5 números de su código postal”>
```

IMPORTANTE: Expresiones regulares son un tema complejo y no relacionado directamente con HTML5.
Para obtener
información adicional al respecto, visite google.com

Atributo form

- **<form>**. Hasta ahora, para construir un formulario teníamos que escribir las etiquetas **<form>** de apertura y cierre y luego declarar cada elemento del formulario entre ellas.
- En HTML5 podemos insertar los elementos en cualquier parte del código y luego hacer referencia al formulario que pertenecen usando su nombre y el atributo **form**:
- El input buscar está fuera de etiquitas FORM
- EJEMPLO3.HTML

Otros elementos para formularios

- **El elemento <datalist>**
- El elemento **<datalist>** es un elemento específico de formularios usado para construir una lista de ítems que luego, con la ayuda del atributo **list**, será usada como sugerencia en un campo del formulario.

```
<datalist id="informacion">
    <option value=“123123123” label=”Teléfono 1”>
    <option value=“456456456” label=”Teléfono 2”>
</datalist>
```

- Este elemento utiliza el elemento **<option>** en su interior para crear la lista de datos a sugerir. Con la lista ya declarada,lo único que resta es referenciarla desde un elemento **<input>** usando el atributo **list**:
- <input type="tel" name="telefono" id="telefono" **list="informacion"**>

API Forms

- Seguramente no le sorprenderá saber que, al igual que cada uno de los aspectos de HTML5, los formularios HTML cuentan con su propia API para personalizar todos los aspectos de procesamiento y validación.
- Existen diferentes formas de aprovechar el proceso de validación en HTML5. Podemos usar los tipos de campo para activar el proceso de validación por defecto (por ejemplo, **email**) o volver un tipo de campo regular como **text** (o cualquier otro) en un campo requerido usando el atributo **required**.
- También podemos crear tipos de campo especiales usando **pattern** para personalizar requisitos de validación.
- Sin embargo, cuando se trata de aplicar mecanismos complejos de validación (por ejemplo, combinando campos o comprobando los resultados de un cálculo) deberemos recurrir a nuevos recursos provistos por esta API.

setCustomValidity()

- Los navegadores que soportan HTML5 muestran un mensaje de error cuando el usuario intenta enviar un formulario que contiene un campo inválido.
- Podemos crear mensajes para nuestros propios requisitos de validación usando el método
- **setCustomValidity(mensaje)**.
- Con este método especificamos un error personalizado que mostrará un mensaje cuando el formulario es enviado.
- Cuando un mensaje vacío es declarado, el error es anulado.
- Ejemplo4.html

- El código presenta una situación de validación compleja.
- Dos campos fueron creados para recibir el nombre y apellido del usuario.
- Sin embargo, el formulario solo será inválido cuando ambos campos se encuentran vacíos.
- El usuario necesita ingresar solo uno de los campos, su nombre o su apellido, para validar la entrada.
- En casos como éste no es posible usar el atributo **required** debido a que no sabemos cuál campo el usuario decidirá
- utilizar.
- Solo con código Javascript y errores personalizados podremos crear un efectivo mecanismo de validación para este
- escenario.

- Nuestro código comienza a funcionar cuando el evento **load** es disparado.
- La función **iniciar()** es llamada para responder al evento.
- Esta función crea referencias para los dos elementos **<input>** y agrega una escucha para el evento **input** en ambos.
- Estas escuchas llamarán a la función **validacion()** cada vez que el usuario escribe dentro de los campos.
- Debido a que los elementos **<input>** se encuentran vacíos cuando el documento es cargado, debemos declarar una condición inválida para no permitir que el usuario envíe el formulario antes de ingresar al menos uno de los valores.
- Por esta razón la función **validacion()** es llamada al comienzo.
- Si ambos campos están vacíos el error es generado y el color de fondo del campo **nombre** es cambiado a rojo.
- Sin embargo, si esta condición ya no es verdad porque al menos uno de los campos fue completado, el error es anulado y el color del fondo de **nombre** es nuevamente establecido como blanco.
- Es importante tener presente que el único cambio producido durante el procesamiento es la modificación del color defondo del campo.
- El mensaje declarado para el error con **setCustomValidity()** será visible sólo cuando el usuario intente enviar el formulario.

El evento invalid

- Cada vez que el usuario envía el formulario, un evento es disparado si un campo inválido es detectado.
- El evento es llamado **invalid** y es disparado por el elemento que produce el error.
- Podemos agregar una escucha para este evento y así ofrecer una respuesta personalizada, como en el siguiente ejemplo:
  - EJEMPLO5.HTML

- En el ejemplo 5, creamos un nuevo formulario con tres campos para ingresar el nombre de usuario, un email y un rango de 20 años de edad.
- El campo **usuario** tiene tres atributos para validación: el atributo **pattern** solo admite el ingreso de un texto de tres caracteres mínimo, desde la A a la Z (mayúsculas o minúsculas), el atributo **maxlength** limita la entrada a 10 caracteres máximo, y el atributo **required** invalida el campo si está vacío.
- El campo **miemail** cuenta con sus limitaciones naturales debido a su tipo y además no podrá enviarse vacío gracias al atributo **required.**
- El campo **miedad** usa los atributos **min**, **max**, **step** y **value** para configurar las condiciones del rango.
- También declaramos un elemento **<output>** para mostrar en pantalla una referencia del rango seleccionado.
- Lo que el código Javascript hace con este formulario es simple: cuando el usuario hace clic en el botón “ingresar”, un evento **invalid** será disparado desde cada campo inválido y el color de fondo de esos campos será cambiado a rojo por la función **validacion()**.

- Veamos este procedimiento con un poco más de detalle.
- El código comienza a funcionar cuando el típico evento **load** es disparado luego que el documento fue completamente cargado.
- La función **iniciar()** es ejecutada y tres escuchas son agregadas para los eventos **change**, **invalid** y **click**.
- Cada vez que el contenido de los elementos del formulario cambia por alguna razón, el evento **change** es disparado desde ese elemento en particular.
- Lo que hicimos fue escuchar a este evento desde el campo **range** y llamar a la función **cambiarrango()** cada vez que el evento ocurre.
- Por este motivo, cuando el usuario desplaza el control del rango o cambia los valores dentro de este campo para seleccionar un rango de edad diferente, los nuevos valores son calculados por la función **cambiarrango()**.

- Los valores admitidos para este campo son períodos de 20 años (por ejemplo, 0 a 20 o 20 a 40).
- Sin embargo, el campo solo retorna un valor, como 20, 40, 60 u 80. Para calcular el valor de comienzo del rango, restamos 20 al valor actual del campo con la fórmula **edad.value - 20**, y grabamos el resultado en la variable **calc**.
- El período mínimo admitido es 0 a 20, por lo tanto con un condicional **if** controlamos esta condición y no permitimos un período menor (estudie la función **cambiarrango()** para entender cómo funciona).
- La segunda escucha agregada en la función **iniciar()** es para el evento **invalid**.
- La función **validacion()** es llamada cuando este evento es disparado para cambiar el color de fondo de los campos inválidos.
- Recuerde que este evento será disparado desde un campo inválido cuando el botón “ingresar” sea presionado.
- El evento no contiene una referencia del formulario o del botón “ingresar”, sino del campo que generó el error.
- En la función **validacion()** esta referencia es capturada y grabada en la variable **elemento** usando la variable **e** y la propiedad **target**. La construcción **e.target** retorna una referencia al elemento **<input>** inválido. Usando esta referencia, en la siguiente línea cambiamos el color de fondo del elemento.

- Volviendo a la función **iniciar()**, encontraremos una escucha más que necesitamos analizar.
- Para tener control absoluto sobre el envío del formulario y el momento de validación, creamos un botón regular en lugar del típico botón **submit**.
- Cuando este botón es presionado, el formulario es enviado, pero solo si todos sus elementos son válidos.
- La escucha agregada para el evento **click** en la función **iniciar()** ejecutará la función **enviar()** cuando el usuario haga clic sobre el botón.
- Usando el método **checkValidity()** solicitamos al navegador que realice el proceso de validación y solo enviamos el formulario usando el tradicional método **submit()** cuando ya no hay más condiciones inválidas.
- Lo que hicimos con el código Javascript fue tomar control sobre todo el proceso de validación, personalizando cada aspecto y modificando el comportamiento del navegador.

Validación en tiempo real

- Cuando abrimos el archivo con la plantilla del Listado 6-24 en el navegador, podremos notar que no existe una validación en tiempo real.
- Los campos son sólo validados cuando el botón “ingresar” es presionado.
- Para hacer más práctico nuestro sistema personalizado de validación, tenemos que aprovechar los atributos provistos por el objeto **ValidityState**.
- Copiar ejemplo 5 y convertirlo en ejempo6.html

- En el Ejemplo6, una nueva escucha fue agregada para el evento **input** sobre el formulario.
- Cada vez que el usuario modifica un campo, escribiendo o cambiando su contenido, la función **controlar()** es ejecutada para responder a este evento.
- La función **controlar()** también aprovecha la propiedad **target** para crear una referencia hacia el elemento que disparó el evento **input**.
- La validez del campo es controlada por medio del estado **valid** provisto por el atributo **validity** en la construcción **elemento.validity.valid**.
- El estado **valid** será **true** (verdadero) si el elemento es válido y **false** (falso) si no lo es.
- Usando esta información cambiamos el color del fondo del elemento.
- Este color será, por lo tanto, blanco para un campo válido y rojo para uno inválido.
- Con esta simple incorporación logramos que cada vez que el usuario modifica el valor de un campo del formulario, este campo será controlado y validado, y su condición será mostrada en pantalla en tiempo real.

Estados de validacion

- En el ejemplo6 controlamos el estado **valid**.
- Este estado particular es un atributo del objeto **ValidityState** que retornará el estado de un elemento considerando cada uno de los posibles estados de validación.
- Si cada condición es válida entonces el valor del atributo **valid** será **true** (verdadero).
- Existen ocho posibles estados de validez para las diferentes condiciones:

- **valueMissing** Este estado es **true** (verdadero) cuando el atributo **required** fue declarado y el campo está vacío.
- **typeMismatch** Este estado es **true** (verdadero) cuando la sintaxis de la entrada no corresponde con el tipo especificado (por ejemplo, si el texto insertado en un tipo de campo **email** no es una dirección de email válida).
- **patternMismatch** Este estado es **true** (verdadero) cuando la entrada no corresponde con el patrón provisto por el atributo **pattern**.
- **tooLong** Este estado es **true** (verdadero) cuando el atributo **maxlength** fue declarado y la entrada es más extensa que el valor especificado para este atributo.
- **rangeUnderflow** Este estado es **true** (verdadero) cuando el atributo **min** fue declarado y la entrada es menor que el valor especificado para este atributo.
- **rangeOverflow** Esta estado es **true** (verdadero) cuando el atributo **max** fue declarado y la entrada es más grande que el valor especificado para este atributo.
- **stepMismatch** Este estado es **true** (verdadero) cuando el atributo **step** fue declarado y su valor no corresponde con los valores de atributos como **min**, **max** y **value**.
- **customError** Este estado es **true** (verdadero) cuando declaramos un error personalizado usando el método **setCustomValidity()** estudiado anteriormente.

- Para controlar estos estados de validación, debemos utilizar la sintaxis **elemento.validity.estado** (donde **estado** es cualquiera de los valores listados arriba).
- Podemos aprovechar estos atributos para saber exactamente que originó el error en un formulario, como en el siguiente ejemplo:
- **sacar copia a ejemplo 6 y crear ejemplo 7.**
- La función **enviar()** fue modificada para incorporar esta clase de control.
- El formulario es validado por el método **checkValidity()** y si es válido es enviado con **submit()**.
- En caso contrario, los estados de validación **patternMismatch** y **valueMissing** para el campo **usuario** son controlados y un mensaje de error es mostrado cuando el valor de alguno de ellos es **true** (verdadero).

Tarea

- Usar bootstrap en todos los ejemplos vistos, queda a su propio critero utilizer cada uno de los estilos en los controles.
- Hacer 5 formularios, utilizando bootstrap, y validaciones vistas en clase, valide por tipo de dato y por información esperado del usuario.

1. Inscripción y registro médico.
2. Solicitud de empleo.
3. Reservación de una habitación de un hotel.
4. Inscripción del curso de paginas web, y cobro en linea (solo validar la tarjeta de crédito).
5. Solicitud de cotizacion de un vehículo.

API Web Storage

- Investigar y hacer ejemplos de:
- **sessionStorage** y **localStorage**