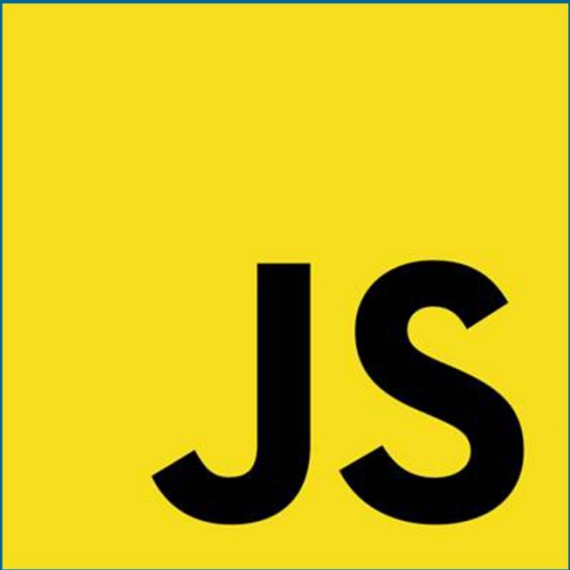


Javascript

01. Nivel inicial

02. Formularios

A large yellow square is centered on the page. Inside the square, the letters 'JS' are written in a bold, black, sans-serif font.

Índice

1.	Procesando formularios	4
2.	Validación	8
	Errores personalizados	8
	El evento invalid	11
	El objeto ValidityState	12
3.	Seudoclases	15
4.	Validación de formulario con JavaScript forma básica	18

JavaScript

formularios



Capítulo 4.2

API Formularios

1. Procesando formularios

La API Formularios es un grupo de propiedades, métodos y eventos que podemos usar para procesar formularios y crear nuestro propio sistema de validación. La API está integrada en los formularios y elementos de formularios, por lo que podemos responder a eventos o llamar a los métodos desde los mismos elementos. Los siguientes son algunos de los métodos disponibles para el elemento `<form>`.

submit()—Este método envía el formulario.

reset()—Este método reinicializa el formulario (asigna los valores por defecto a los elementos).

checkValidity()—Este método devuelve un valor booleano que indica si el formulario es válido o no.

La API también ofrece el siguiente evento para anunciar cada vez que se inserta un carácter o se selecciona un valor en un elemento de formulario.

input—Este evento se desencadena en el formulario o sus elementos cuando el usuario inserta o elimina un carácter en un campo de entrada, y también cuando se selecciona un nuevo valor.

change—Este evento se desencadena en el formulario o sus elementos cuando se introduce o selecciona un nuevo valor.

Usando estos métodos y eventos, podemos controlar el proceso de envío del formulario desde JavaScript. El siguiente ejemplo envía el formulario cuando se pulsa un botón.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    function iniciar() {
      var boton = document.getElementById("enviar");
      boton.addEventListener("click", enviarformulario);
    }
    function enviarformulario() {
      var formulario = document.querySelector("form[name='informacion']");
      formulario.submit();
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
      <p><button type="button" id="enviar">Registrarse</button></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-1: *Enviando un formulario desde JavaScript*

El código del Listado 4-2-1 responde al evento `click` desde el elemento `<button>` para ejecutar la función `enviarformulario()` cada vez que se pulsa el botón. En esta función, obtenemos una referencia al elemento `<form>` y luego enviamos el formulario con el método `submit()`.

En este ejemplo, hemos decidido obtener la referencia al elemento `<form>` con el método `querySelector()` y un selector que busca el elemento por medio de su atributo `name`, pero podríamos haber agregado un atributo `id` al elemento para obtener la referencia con el método `getElementById()`, como hemos hecho con el botón. Otra alternativa es obtener la referencia desde la propiedad `forms` del objeto `Document`. Esta propiedad devuelve un array con referencias a todos los elementos `<form>` del documento. En nuestro caso, solo tenemos un elemento `<form>` y, por lo tanto, la referencia se encuentra en el índice 0.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    function iniciar() {
      var boton = document.getElementById("enviar");
      boton.addEventListener("click", enviarformulario); }
    function enviarformulario() {
      var lista = document.forms;
      var formulario = lista[0];
      formulario.submit(); }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
      <p><button type="button" id="enviar">Registrarse</button></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-2: *Obteniendo una referencia al elemento `<form>` desde la propiedad `forms`*

Enviar el formulario con el método `submit()` es lo mismo que hacerlo con un elemento `<input>` de tipo `submit` (ver Capítulo 2), la diferencia es que este método evita el proceso de validación del navegador. Si queremos que el formulario sea validado, tenemos que hacerlo nosotros con el método `checkValidity()`. Este método controla el formulario y devuelve `true` o `false` para indicar si es válido o no.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    function iniciar() {
      var boton = document.getElementById("enviar");
      boton.addEventListener("click", enviarformulario);
    }
    function enviarformulario() {
      var formulario = document.querySelector("form[name='informacion']");
      var valido = formulario.checkValidity();
      if (valido) {
        formulario.submit();
      } else {
        alert("El formulario no puede ser enviado");
      }
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
      <p><button type="button" id="enviar">Registrarse</button></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-3: Controlando la validez del formulario

El código del Listado 4-2-3 controla los valores en el formulario para determinar su validez. Si el formulario es válido, se envía con el método `submit()`. En caso contrario, se muestra un mensaje en pantalla para advertir al usuario.



Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-2-3. Abra el documento en su navegador e intente enviar el formulario. Debería ver una ventana emergente advirtiéndole de que el formulario no se puede enviar. Inserte una cuenta de correo válida en el campo. Ahora, el formulario sí se debería enviar.

2. Validación

Como hemos visto en el Capítulo 2, existen diferentes maneras de validar formularios en HTML. Podemos usar campos de entrada del tipo que requieren validación por defecto, como **email**, convertir un campo regular de tipo **text** en un campo requerido con el atributo **required**, o incluso usar tipos especiales como **pattern** para personalizar los requisitos de validación. Sin embargo, cuando tenemos que implementar mecanismos de validación más complejos, como comparar dos o más campos o controlar el resultado de una operación, nuestra única opción es la de personalizar el proceso de validación usando la API Formularios.

Errores personalizados

Los navegadores muestran un mensaje de error cuando el usuario intenta enviar un formulario que contiene un campo no válido. Estos son mensajes predefinidos que describen errores conocidos, pero podemos definir mensajes personalizados para establecer nuestros propios requisitos de validación. Con este fin, los objetos **Element** que representan elementos de formulario incluyen el siguiente método.

setCustomValidity(mensaje)—Este método declara un error personalizado y el mensaje a mostrar si se envía el formulario. Si no se especifica ningún mensaje, el error se anula.

El siguiente ejemplo presenta una situación de validación compleja. Se crean dos campos para recibir el nombre y el apellido del usuario. Sin embargo, el formulario solo es no válido cuando ambos campos están vacíos. El usuario puede introducir su nombre o su apellido para validarlo. En un caso como este, es imposible usar el atributo **required** porque no sabemos qué campo va a elegir completar el usuario. Solo usando errores personalizados podemos crear un mecanismo de validación efectivo para este escenario.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    var nombre1, nombre2;
    function iniciar() {
      nombre1 = document.getElementById("nombre");
      nombre2 = document.getElementById("apellido");
      nombre1.addEventListener("input", validacion);
      nombre2.addEventListener("input", validacion);
      validacion();
    }
    function validacion() {
      if (nombre1.value == "" && nombre2.value == "") {
        nombre1.setCustomValidity("Inserte su nombre o su apellido");
        nombre1.style.background = "#FFDDDD";
        nombre2.style.background = "#FFDDDD";
      } else {
        nombre1.setCustomValidity("");
        nombre1.style.background = "#FFFFFF";
        nombre2.style.background = "#FFFFFF";
      }
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="registracion" method="get" action="procesar.php">
      <p><label>Nombre: <input type="text" name="nombre"
id="nombre"></label></p>
      <p><label>Apellido: <input type="text" name="apellido"
id="apellido"></label></p>
      <p><input type="submit" value="Registrarse"></p>
    </form>
  </section>
</body>
</html>

```

Listado 4-2-4: Declarando mensajes de error personalizados

El código del Listado 4-2-4 comienza creando referencias a dos elementos `<input>` y agregando listeners al evento `input` para cada uno de ellos. Este evento se desencadena cada vez que el usuario inserta o elimina un carácter, lo que nos permite detectar cada valor insertado en los campos, y validar o invalidar el formulario desde la función `validacion()`.

Debido a que los elementos `<input>` se encuentran vacíos cuando se carga el documento, tenemos que declarar una condición no válida para no permitir al usuario enviar el formulario antes de insertar al menos uno de los valores en los campos. Por esta razón, la función `validacion()` también se llama al final de la función `iniciar()` para comprobar esta condición.

La función `validacion()` controla si el formulario es válido o no, y declara o elimina el error con el método `setCustomValidity()`. Si ambos campos están vacíos, se declara un error personalizado y el color de fondo de ambos elementos se cambia a rojo para indicar al usuario el error. Sin embargo, si la condición cambia debido a que se ha introducido al menos uno de los valores, el error se elimina llamando al método con una cadena de caracteres vacía y el color blanco se asigna nuevamente al fondo de ambos campos.

Es importante recordar que el único cambio producido durante el proceso es la modificación del color de fondo de los campos. El mensaje de error que declara el método `setCustomValidity()` solo se mostrará al usuario cuando intente enviar el formulario.



Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-2-4 y abra el documento en su navegador. Intente enviar el formulario. Debería ver un error con el mensaje "Inserte su nombre o apellido". Inserte un valor. El error se debería eliminar.



Lo básico: se pueden declarar varias variables separadas por comas en la misma línea. En el Listado 4-2-4 hemos declarado dos variables globales llamadas `nombre1` y `nombre2`. Esta instrucción no es necesaria porque las variables declaradas dentro de funciones sin el operador `var` se asignan al ámbito global, pero declarar las variables que vamos a utilizar al comienzo del código simplifica su mantenimiento porque nos ayuda a identificar sin demasiado esfuerzo las variables que nuestro código necesita para trabajar.

El evento invalid

Cada vez que el usuario envía un formulario, se desencadena un evento si se detecta un campo no válido. El evento se llama **invalid** y se desencadena en el elemento que ha producido el error. Para personalizar una respuesta, podemos responder a este evento desde el elemento **<form>**, como lo hacemos en el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    var formulario;
    function iniciar() {
      var boton = document.getElementById("enviar");
      boton.addEventListener("click", enviarformulario);
      formulario = document.querySelector("form[name='informacion']");
      formulario.addEventListener("invalid", validacion, true);
    }
    function validacion(evento) {
      var elemento = evento.target;
      elemento.style.background = "#FFDDDD";
    }
    function enviarformulario() {
      var valido = formulario.checkValidity();
      if (valido) {
        formulario.submit();
      }
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Apodo: <input pattern="[A-Za-z]{3,}" name="apodo"
id="apodo" maxlength="10" required></label></p>
      <p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
      <p><button type="button" id="enviar">Registrarse</button></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-5: Creando un sistema de validación personalizado

En el Listado 4-2-5 hemos creado un nuevo formulario con dos campos de entrada para introducir un apodo y una cuenta de correo. El campo **correo** tiene sus limitaciones naturales debido a su tipo y un atributo **required** que lo declara como campo obligatorio, pero el campo **apodo** contiene tres atributos de validación: el atributo **pattern** que solo admite un mínimo de tres caracteres de la A a la Z (mayúsculas y minúsculas), el atributo **maxlength** que limita el campo a un máximo de diez caracteres, y el atributo **required** que invalida el campo si está vacío.

El código es muy similar a los ejemplos anteriores. Respondemos al evento `load` con la función `iniciar()` cuando el documento termina de cargarse y al evento `click` del elemento `<button>`, como siempre, pero luego agregamos un listener para el evento `invalid` al elemento `<form>` en lugar de los campos `<input>`. Esto se debe a que queremos establecer un sistema de validación para todo el formulario, no solo elementos individuales. Para este propósito, tenemos que incluir el valor `true` como tercer atributo del método `addEventListener()`. Este atributo le indica al navegador que tiene que propagar el evento al resto de los elementos de la jerarquía. Como resultado, a pesar de que el *listener* se ha agregado al elemento `<form>`, este responde a eventos desencadenados por los elementos dentro del formulario. Para determinar cuál es el elemento no válido que ha llamado a la función `validacion()`, leemos el valor de la propiedad `target`. Como hemos visto en capítulos anteriores, esta propiedad devuelve una referencia al elemento que desencadenó el evento. Usando esta referencia, la última instrucción en esta función cambia el color de fondo del elemento a rojo.

El objeto `ValidityState`

El documento del Listado 4-2-5 no realiza una validación en tiempo real. Los campos se validan solo cuando se envía el formulario. Considerando la necesidad de un sistema de validación más dinámico, la API Formularios incluye el objeto `ValidityState`. Este objeto ofrece una serie de propiedades para indicar el estado de validez de un elemento del formulario.

`valid`—Esta propiedad devuelve `true` si el valor del elemento es válido.

La propiedad `valid` devuelve el estado de validez de un elemento considerando todos los demás estados de validez. Si todas las condiciones son válidas, la propiedad `valid` devuelve `true`. Si queremos controlar una condición en particular, podemos leer el resto de las propiedades que ofrece el objeto `ValidityState`.

`valueMissing`—Esta propiedad devuelve `true` cuando se ha declarado el atributo `required` y el campo está vacío.

`typeMismatch`—Esta propiedad devuelve `true` cuando la sintaxis del texto introducido no coincide con el tipo de campo. Por ejemplo, cuando el texto introducido en un campo de tipo `email` no es una cuenta de correo.

`patternMismatch`—Esta propiedad devuelve `true` cuando el texto introducido no respeta el formato establecido por el atributo `pattern`.

`tooLong`—Esta propiedad devuelve `true` cuando se ha declarado el atributo `maxlength` y el texto introducido es más largo que el valor especificado por este atributo.

`rangeUnderflow`—Esta propiedad devuelve `true` cuando se ha declarado el atributo `min` y el valor introducido es menor que el especificado por este atributo.

`rangeOverflow`—Esta propiedad devuelve `true` cuando se ha declarado el atributo `max` y el valor introducido es mayor que el especificado por este atributo.

`stepMismatch`—Esta propiedad devuelve `true` cuando se ha declarado el atributo `step` y el valor introducido no corresponde con el valor de los atributos `min`, `max` y `value`.

`customError`—Esta propiedad devuelve `true` cuando declaramos un error personalizado con el método `setCustomValidity()`.

El objeto **ValidityState** se asigna a una propiedad llamada **validity**, disponible en cada elemento de formulario. El siguiente ejemplo lee el valor de esta propiedad para determinar la validez de los elementos en el formulario dinámicamente.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    var formulario;
    function iniciar() {
      var boton = document.getElementById("enviar");
      boton.addEventListener("click", enviarformulario);
      formulario = document.querySelector("form[name='informacion']");
      formulario.addEventListener("invalid", validacion, true);
      formulario.addEventListener("input", comprobar);
    }
    function validacion(evento) {
      var elemento = evento.target;
      elemento.style.background = "#FFDDDD";
    }
    function enviarformulario() {
      var valido = formulario.checkValidity();
      if (valido) {
        formulario.submit();
      }
    }
    function comprobar(evento) {
      var elemento = evento.target;
      if (elemento.validity.valid) {
        elemento.style.background = "#FFFFFF";
      } else {
        elemento.style.background = "#FFDDDD";
      }
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Apodo: <input pattern="[A-Za-z]{3,}" name="apodo"
id="apodo" maxlength="10" required></label></p>
      <p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
      <p><button type="button" id="enviar">Registrarse</button></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-6: Validación en tiempo real

En el código del Listado 4-2-6, agregamos un listener para el evento `input` al formulario. Cada vez que el usuario modifica un campo, insertando o eliminando un carácter, se ejecuta la función `comprobar()` para responder al evento.

La función `comprobar()` también aprovecha la propiedad `target` para obtener una referencia al elemento que desencadenó el evento y controlar su validez leyendo el valor de la propiedad `valid` dentro de la propiedad `validity` del objeto `Element` (`elemento.validity.valid`). Con esta información, cambiamos el color de fondo del elemento que desencadenó el evento `input` en tiempo real. El color será rojo hasta que el texto introducido por el usuario sea válido.



Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-2-6. Abra el documento en su navegador e inserte valores en los campos de entrada. Debería ver el color de fondo de los campos cambiar según su validez (válido blanco, no válido rojo).

Podemos usar el resto de las propiedades que ofrece el objeto `ValidityState` para saber exactamente qué ha producido el error, tal como muestra el siguiente ejemplo.

```
function enviarformulario() {
    var elemento = document.getElementById("apodo");
    var valido = formulario.checkValidity();
    if (valido) {
        formulario.submit();
    } else if (elemento.validity.patternMismatch ||
elemento.validity.valueMissing) {
        alert('El apodo debe tener un mínimo de 3 caracteres');
    }
}
```

Listado 4-2-7: Leyendo los estados de validez para mostrar un mensaje de error específico

En el Listado 4-2-7 se modifica la función `enviarformulario()` para detectar errores específicos. El formulario lo valida el método `checkValidity()` y, si es válido, se envía con el método `submit()`. En caso contrario, se leen los valores de las propiedades `patternMismatch` y `valueMissing` del campo `apodo` y se muestra un mensaje de error cuando una o ambas devuelven `true`.



Hágalo usted mismo: reemplace la función `enviarformulario()` en el documento del Listado 4-2-6 con la nueva función del Listado 4-2-7 y abra el documento en su navegador. Escriba un solo carácter en el campo `apodo` y envíe el formulario. Debería ver una ventana emergente pidiéndole que inserte un mínimo de tres caracteres.

3. Seudoclases

Además de todas las propiedades y métodos provistos por la API Formularios, CSS incluye algunasseudoclases para modificar los estilos de un elemento dependiendo de su estado, incluidos inválido, válido, requerido, opcional, e incluso cuando un valor se encuentra fuera del rango permitido.

Valid e Invalid

Estasseudoclases afectan a cualquier elemento `<input>` con un valor válido o inválido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <style>
    input:valid{
      background: #EEEEFF;
    }
    input:invalid{
      background: #FFEEEE;
    }
  </style>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <input type="email" name="correo" required>
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Listado 4-2-8: Usando lasseudoclases `:valid` e `:invalid`

El formulario del Listado 4-2-8 incluye un elemento `<input>` para cuentas de correo. Cuando el contenido del elemento no es válido, laseudoclase `:valid` asigna un color de fondo azul al campo, pero tan pronto como el contenido se vuelve no válido, laseudoclase `:invalid` cambia el color de fondo a rojo.

Optional y Required

Estas pseudoclases afectan a todos los elementos de formulario declarados como obligatorios u opcionales.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <style>
    input:optional{
      border: 2px solid #009999;
    }
    input:required{
      border: 2px solid #000099;
    }
  </style>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="nombre"></p>
      <p><input type="text" name="apellido" required></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 4-2-9: Usando las pseudoclases `:required` y `:optional`

El ejemplo del Listado 4-2-9 incluye dos campos de entrada: **nombre** y **apellido**. El primero es opcional, pero **apellido** es obligatorio. Las pseudoclases asignan un color de borde diferente a estos campos de acuerdo con su condición (el campo obligatorio se muestra en azul y el campo opcional en verde).

In-range y Out-of-range

Estas pseudoclases afectan a todos los elementos con un valor dentro o fuera de un rango específico.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <style>
    input:in-range{
      background: #EEEEFF;
    }
    input:out-of-range{
      background: #FFEEEE;
    }
  </style>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <input type="number" name="numero" min="0" max="10">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Listado 4-2-10: Usando las pseudoclases `:in-range` y `:out-of-range`

En este ejemplo se ha incluido un campo de entrada de tipo **number** para probar estas pseudoclases. Cuando el valor introducido en el elemento es menor que 0 o mayor que 10, el color de fondo es rojo, pero tan pronto como introducimos un valor dentro del rango especificado, el color de fondo cambia a azul.

4. Validación de formulario con JavaScript forma básica

En el pasado, una vez que alguien había introducido toda su información y pulsado el botón de enviar, la validación de los formularios tenía lugar en el servidor.

Si faltaba algún dato o era inexacto, el servidor tenía que enviar toda la solicitud hacia atrás junto con un mensaje en el que se indicaba al destinatario que modificara primero el formulario antes de volver a enviarlo.

Era un proceso interminable que suponía una gran carga para el servidor.

Hoy en día, JavaScript ofrece diversos métodos para validar los datos de los formularios en el navegador antes de enviarlos al servidor la mejor manera es utilizar expresiones regulares, pero en este artículo veras una validación Básica de Formularios, para que puedas tener idea de como lo aplicarías en un proyecto simple.

Estructura de nuestro formulario con HTML

Para crear una validación de formulario con JavaScript, primero creamos la estructura de nuestro proyecto en el archivo index.html con un formulario sencillo y también haciendo referencia a nuestro archivo de JS y CSS

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <title>Validaciones en JavaScript</title>
</head>

<body>

  <div class="container">
    <h2>BIENVENIDO</h2>
    <form id="formulario">
      <input type="text" id="usuario" placeholder="Nombre de Usuario">
      <input type="password" id="password" placeholder="Contraseña">
      <button type="submit" class="boton">Iniciar Sesión</button>
    </form>
  </div>

  <script src="app.js"></script>
</body>
</html>
```

Estilos con CSS para el formulario

Luego te dejo aquí unos estilos sencillos para que tenga un diseño agradable nuestro formulario para eso creamos un archivo con el nombre de style.css

```
body {
  background-image: linear-gradient(to right, rgb(20, 32, 48), rgb(13, 24, 38));
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}

h1 {
  color: #6c5ce7;
  text-align: center;
  text-transform: uppercase;
  font-size: 60px;
}

.container {
  width: 400px;
  margin: 50px auto 0;
  padding: 15px;
  border-radius: 8px;
  background-color: #fff;
}

.container h2 {
  text-align: center;
  text-transform: uppercase;
  color: rgb(24, 32, 47);
}

form {
  display: flex;
  flex-direction: column;
  padding: 20px;
  gap: 15px;
}

input {
  padding: 10px;
  border: 1px solid rgba(102, 131, 140, 0.139);
  border-radius: 5px;
}

input:focus {
  outline: none;
  border-bottom: 2px solid #6c5ce7;
}

.boton {
  border: none;
  background-color: #6c5ce7;
  padding: 10px;
  color: #fff;
  cursor: pointer;
}
```

```

border-radius: 5px;
text-transform: uppercase;
}

.alerta {
padding: 5px;
border-radius: 5px;
text-align: center;
text-transform: uppercase;
color: #fff;
}

.error {
background-color: rgb(255, 0, 93);
}

.exito {
background-color: rgb(27, 122, 46);
}

```

Validacion en JavaScript

Y por último creamos la lógica de nuestro proyecto y para eso utilizamos JavaScript, entonces procedemos a crear nuestro archivo con el nombre de app.js

```

const formulario = document.querySelector('#formulario')

//evento
formulario.addEventListener("submit", validarFormulario);

//funcion validar el formulario al dar submit
function validarFormulario(e) {
  e.preventDefault();

  //ver los valores de los inputs
  const usuario = document.querySelector("#usuario").value
  const password = document.querySelector("#password").value

  // Lógica para validacion en JavaScript
  if (usuario == '' || password == '') {
    mostrarAlerta('Complete los campos', 'error');
  } else {
    if (password == '12345' && usuario == 'digitalnest') {
      mostrarAlerta('cargando...', 'exito');
      setTimeout(() => {
        window.location.href = "/paginalnicio.html"
      }, 2000);
      return
    }
    mostrarAlerta('Datos incorrectos', 'error');
  }
}

```

```
//Mostrar la alerta cuando agregamos datos incorrectos o vacíos
function mostrarAlerta(mensaje, tipo) {
  const alerta = document.createElement('div');
  alerta.className = `alerta ${tipo}`
  alerta.textContent = mensaje

  formulario.appendChild(alerta);
  setTimeout(() => {
    alerta.remove();
  }, 2000)
}
```

5. Resumen

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.

Durante los últimos años, códigos personalizados y librerías fueron creados para procesar formularios en el ordenador del usuario. HTML5 vuelve a estas funciones estándar agregando nuevos atributos, elementos y una API completa. Ahora la capacidad de procesamiento de información insertada en formularios en tiempo real ha sido incorporada en los navegadores y completamente estandarizada.

El elemento <form>

Los formularios en HTML no han cambiado mucho. La estructura sigue siendo la misma, pero HTML5 ha agregado nuevos elementos, tipos de campo y atributos para expandirlos tanto como sea necesario y proveer así las funciones actualmente implementadas en aplicaciones web.

```
<!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>
```