5.4 VALIDACIÓN Y ENVÍO

A la hora de de enviar un formulario, no podemos garantizar que el usuario haya insertado cada dato del formulario como esperamos. Supongamos que en un campo de texto el usuario tiene que incluir un código postal, pero el usuario en ese campo ha incluido una cadena de texto, por ejemplo, Madrid. Si el formulario se envía con ese dato se producirá un error. Para controlar si los campos de un formulario han sido rellenados correctamente haremos uso de las validaciones. Existen varios tipos de validaciones. Podemos realizar validaciones en el servidor, a nivel de servidor y también en la base de datos. Realizar validaciones a nivel de servidor supone que se recibe una petición, el servidor tiene que procesarla y emitir una respuesta. En el caso de que el usuario incluya muchos datos que no son correctos el servidor se puede ver sobrecargado. Para solucionar esta sobrecarga del servidor, realizamos un primer filtrado de los datos en el cliente. Estas validaciones son las que vamos a ver a continuación. En ellas analizamos con *scripts* en el navegador los datos que ha incluido el usuario una vez que pulsa el botón de enviar formulario. La petición no es enviada al servidor y se analiza dentro del navegador si existe algún dato que no se haya introducido correctamente. En el caso de que exista algún dato que no es correcto el *script* asignado muestra un mensaje al usuario indicando que debe realizar alguna modificación en el dato introducido. Si, por el contrario, se validan como correctos los datos, se envían los datos del formulario al servidor.

Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones que se han impuesto como, por ejemplo, en el caso anterior, donde el dato tenía que cumplir con los caracteres que puede tener un código postal. A continuación vamos a ver algunos de los ejemplos de validación más habituales, los tipos de validación que pueden existir son de lo más variado y basta con definirlas dentro de cada una de las funciones que valida un dato.

5.4.1 ESTRUCTURA DEL FORM PARA VALIDAR DATOS

Para que el formulario detecte que tiene que realizar una validación de los datos antes de enviar la petición al servidor, debemos indicárselo en su estructura. Para ello utilizaremos el evento `onsubmit` que ejecuta una acción cuando el `action` del formulario es llamado. Dentro del evento `onsubmit`, debemos llamar a una función, dependiendo de la respuesta de la función, el formulario enviará los datos al servidor (si los datos son correctos) o mostrará los diferentes mensajes de error, para los controles del formulario que se hayan realizado validaciones. El código correspondiente sería el siguiente:

```
<form action="URL" method="post" name="formValidado"
  onsubmit="return validar()">
  ...
</form>
```

En el código anterior observamos como la función `validar()` es llamada dentro del evento `onsubmit` y que dependiendo de la respuesta de validar el servidor envía o no los datos del formulario al servidor. Al realizarse un `submit` desde algún botón del formulario (petición de envío del formulario al servidor), siempre se realiza la llamada a la función `validar()`. Para que los datos del formulario sean validados, debemos tener implementada la función `validar()`. En esta función indicaremos cuáles son los datos del formulario que vamos a validar. A continuación mostramos algunos ejemplos de cómo validar datos de un formulario.

- **Validar un campo como obligatorio**. En ocasiones puede que nos interese que el usuario ingrese un campo como obligatorio en la aplicación web. En esta situación si el usuario no introduce el dato y antes de que el formulario sea enviado debemos comprobar si el campo está vacío. Habiendo llamado en la estructura del `form` a una función, la implementación de la función será la siguiente:

```
<script type="text/javascript">
  function validacion() {
    valor = document.getElementById("campo").value;

    if( valor == null || valor.length == 0 ){
      alert("El campo no puede ser vacío");
      return false;
    }
    return true;
  }
</script>
```

Este código deber ir integrado en la cabecera de la página HTML (`<head>`...código JavaScript...`</head>`). En primer lugar, recogemos el valor a través del objeto `document`, el nombre del *input* es `campo`. En el capítulo posterior veremos con detalle el árbol de estructura de una página web. En la estructura `if` comprobamos que el valor no sea nulo y que su longitud no sea cero. En caso de que el valor del campo `input` sea nulo o igual a cero, mostramos un mensaje de alerta y, posteriormente, devolvemos falso para que sea recuperado por el evento `onsubmit`. En ese caso no se enviaría el formulario.

- **Validar un campo de texto como numérico**. A menudo en los formularios nos encontramos con campos de texto que solo admiten números. Los teléfonos, código postal, DNI (sin letra) y otros muchos datos que obligatoriamente tienen una nomenclatura numérica. Para advertir al usuario de que el dato que ha introducido no es correcto utilizaremos el siguiente código implementado en una función:

```
<script type="text/javascript">
  function validaNum() {
    valor = document.getElementById("telefono").value;

    if( isNaN(valor) ) {
      alert("El campo tiene que ser numérico");
      return false;
    }
    return true;
  }
</script>
```

En el código anterior, tenemos una variable que se llama `valor`. Esta variable tomará el valor del `teléfono` o en su defecto el dato que haya introducido el usuario en la aplicación web. Con la función `isNaN()` comprobamos si el valor es numérico. En caso de que no lo sea devolverá un mensaje de alerta indicando que el dato tiene que ser numérico. Si por el contrario el dato es numérico la función devolverá verdadero.

- **Validar si una fecha es correcta**. Las fechas a menudo suelen ser datos en los formularios difíciles de comprobar, la razón es que hay infinidad de formas de expresarlas. En el ejemplo siguiente mostramos cómo validar de una forma sencilla una fecha, en la que se han ingresado el año, mes y día de forma independiente cada uno a través de tres campos diferentes:

```
<script type="text/javascript">
  function validaFecha() {
    var dia = document.getElementById("dia").value;
    var mes = document.getElementById("mes").value;
    var ano = document.getElementById("ano").value;

    fecha = new Date(ano, mes, dia);

    if( !isNaN(fecha) ) {
      return false;
    }
    return true;
  }
</script>
```

En el código anterior recogemos tres valores uno por el día otro por la fecha y el último por el año. Con la función `Date()` generamos una fecha introduciendo los valores. Posteriormente, comprobamos si la fecha que hemos creado es correcta, en caso de que el resultado sea que la fecha no es válida la función devuelve `false`.

- **Validar un checkbox**. Otra de las validaciones que podemos necesitar habitualmente, es comprobar si un *checkbox* ha sido seleccionado. A menudo en la aceptación de ciertas condiciones para acceder a servicios nos encontramos con esta comprobación, sin la que el formulario no tiene mucho sentido que continúe. En el siguiente código vamos a ver cómo se implementa la comprobación de si se ha activado un *checkbox*:

```
<script type="text/javascript">
  function validaCheck() {
    elemento = document.getElementById("campoCondiciones");

    if( !elemento.checked ) {
      return false;
    }
    return true;
  }
</script>
```

En el código anterior recuperamos el valor que ha tomado el elemento *checkbox* del formulario. En la siguiente condición comprobamos si éste ha sido chequeado, en caso de que no se haya pulsado la función devolverá falso. Si se ha chequeado la función devuelve verdadero.

Conviene tener en cuenta que si somos demasiado estrictos a la hora de validar o damos poca información, podemos bloquear al usuario de la aplicación web. Es posible que el usuario no sepa la manera de continuar en el proceso de manejo debido a un error en el tipo y validación de los datos.

5.5 EXPRESIONES REGULARES

Las expresiones regulares describen un conjunto de elementos que siguen un patrón. Dicho de otra forma, es una regla que identifica a una serie de elementos que tiene algo en común. Un ejemplo de expresión regular podrían ser todas las palabras que comienzan por la letra "a" minúscula. La expresión regular para este patrón debe asegurarse de que la palabra comienza por "a" minúscula, el resto de palabras que precedan a la cadena podrán ser cualquier carácter.

La mayoría de los lenguajes de programación incluyen la implementación de expresiones regulares. En el caso que nos atañe, JavaScript implementa una configuración para implementar expresiones regulares y así facilitar las comprobaciones de ciertos datos que deben seguir una estructura concreta. Este tipo de expresiones es muy útil a la hora de evaluar algunos tipos de datos que normalmente utilizamos en los formularios.

5.5.1 CARACTERES ESPECIALES DE LAS EXPRESIONES REGULARES

A continuación vamos describir algunos de los elementos que utiliza JavaScript para crear expresiones regulares. En el siguiente apartado vamos a representar la simbología que nos presta la expresión regular y, posteriormente, significado o interpretación:

- ✓ **^ Principio de entrada o línea**. Este carácter indica que las cadenas deberán comenzar por el siguiente carácter. Si éste fuera una "a" minúscula, como indicamos en el punto anterior, la expresión regular seria sería, `^a`.
- ✓ **$ Fin de entrada o línea**. Indica que la cadena debe terminar por el elemento precedido al dólar.
- ✓ *** El carácter anterior 0 o más veces**. El asterisco indica que el carácter anterior se puede repetir en la cadena 0 o más veces.
- ✓ **+ El carácter anterior 1 o más veces**. El símbolo más indica que el carácter anterior se puede repetir en la cadena una o más veces.
- ✓ **? El carácter anterior una vez como máximo**. El símbolo interrogación indica que el carácter anterior se puede repetir en la cadena cero o una vez.
- ✓ **. Cualquier carácter individual**. El símbolo punto indica que puede haber cualquier carácter individual salvo el de salto de línea.
- ✓ **x|y x ó y**. La barra vertical indica que puede ser el carácter *x* o el *y*.
- ✓ **{n} n veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer exactamente *n* veces.
- ✓ **{n,m} Entre n y m veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer como mínimo *n* y como máximo *m* veces.
- ✓ **[abc] Cualquier carácter de los corchetes**. En la cadena puede aparecer cualquier carácter que esté incluido en los corchetes. Además, podemos especificar rangos de caracteres que sigan un orden. Si se especifica el rango [a-z] equivaldría a incluir todas las letras minúsculas del abecedario.

- **[^abc] Un carácter que no esté en los corchetes**. En la cadena pueden aparecer todos los caracteres que no estén incluidos en los corchetes. También podemos especificar rangos de caracteres como en el punto anterior.
- **\b Fin de palabra**. El símbolo `\b` indica que tiene que haber un fin de palabra o retorno de carro.
- **\B No fin de palabra**. El símbolo `\B` indica cualquiera que no sea un límite de palabra.
- **\d Cualquier carácter dígito**. El símbolo `\d` indica que puede haber cualquier carácter numérico, de 0 a 9.
- **\D Carácter que no es dígito**. El símbolo `\D` indica que puede haber cualquier carácter siempre que no sea numérico.
- **\f Salto de página**. El símbolo `\f` indica que tiene que haber un salto de página.
- **\n Salto de línea**. El símbolo `\n` indica que tiene que haber un salto de línea.
- **\r Retorno de carro**. El símbolo `\r` indica que tiene que haber un retorno de carro.
- **\s Cualquier espacio en blanco**. El símbolo `\s` indica que tiene que haber un carácter individual de espacio en blanco: espacios, tabulaciones, saltos de página o saltos de línea.
- **\S Carácter que no sea blanco**. El símbolo `\S` indica que tiene que haber cualquier carácter individual que no sea un espacio en blanco.
- **\t Tabulación**. El símbolo `\t` indica que tiene que haber cualquier tabulación.
- **\w Carácter alfanumérico**. El símbolo `\w` indica que puede haber cualquier carácter alfanumérico, incluido el de subrayado.
- **\W Carácter que no sea alfanumérico**. El símbolo `\W` indica que puede haber cualquier carácter que no sea alfanumérico.

5.5.2 VALIDAR UN FORMULARIO CON EXPRESIONES REGULARES

Con la combinación de estas expresiones podemos abordar infinidad de patrones para validar datos en la mayoría de los formularios. Combinando cada una de las condiciones obtenemos patrones como pueden ser, una dirección de correo electrónico, un teléfono, un código postal, un DNI, etc. A continuación, vamos a ver cómo realizar algunas de las configuraciones de expresiones regulares más utilizadas para validad datos de un formulario.

- **Validar una dirección de correo electrónico**. En el caso de que el usuario de la aplicación web introduzca una dirección de correo electrónico, es casi imprescindible que esta sea correcta. Utilizando expresiones regulares vamos a comprobar que la estructura de la dirección sea correcta. En primer lugar, comprobaremos que comienza por una cadena de texto, que sigue por una @ y que termina por otra cadena de texto un punto y otra cadena de texto. De esta forma aseguraremos que, al menos, la estructura de la dirección es correcta. Combinando las funciones con las expresiones regulares, obtenemos un código que valida una dirección de correo electrónico.

  La llamada en la estructura principal del `form` sigue siendo la misma que en los casos anteriores y recogemos en el evento `onsubmit` la respuesta de la llamada a la función que valida, en nuestro caso la dirección de correo electrónico. El código correspondiente a la función JavaScript que valida un correo electrónico sería el siguiente:

```
<script type="text/javascript">
  function validaEmail() {
    valor = document.getElementById("campo").value;

    if(!(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
      .]\w+)/.test(valor)) ) {
      return false;
    }
    return true;
  }
</script>
```

- **Validar un DNI**. Un dato muy habitual en los formularios es el documento nacional de identidad. A la hora de que el usuario interactúe con la aplicación web, es necesario comprobar que el DNI que introduce es correcto. No podemos asegurar que el DNI sea el de la persona que se está identificando, pero sí podemos asegurar que el DNI que introduce el usuario es correcto. Para validad un DNI introducido podemos utilizar el siguiente código.

```
<script type="text/javascript">
  function validaDNI(){
    valor = document.getElementById("dni").value;

    var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F',
      'P',  'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V',
      'H', 'L', 'C', 'K', 'E', 'T'];

    if( !(/^\d{8}[A-Z]$/.test(valor)) ){
      return false;
    }

    if(valor.charAt(8) != letras[(valor.substring(0, 8))%23])
    {
      return false;
    }
    return true;
}
</script>
```

En el código anterior validamos un DNI en varios pasos. Primero recogemos en la variable `valor` el campo del DNI que ha introducido el usuario. Generamos un *array* con las letras validas para el DNI. En la primera condición comprobamos a través de expresiones regulares que el patrón del campo introducido tiene una longitud de 8 números y una letra. Si no es así devuelve falso la función.

En la última condición realizamos la división entre 23 y tomamos el resto. Comprobamos que la posición del *array* `letras`, perteneciente al resto de la operación, se corresponde con el valor de la letra del DNI introducido. En caso de que sea igual devuelve verdadero (al final de la función). Si no es igual la letra del valor introducido a la calculada, entra en la condición y devuelve falso.

- **Validar un número de teléfono**. Otro dato que es importante que sea correcto para interactuar en una aplicación web con el usuario son los números de teléfono. Al igual que en casos anteriores, no podemos comprobar que el numero pertenezca al usuarios que lo introduce, pero si podemos asegurar que el número de teléfono tiene un formato correcto. Las expresiones regulares son un recurso muy útil para validar un teléfono. A continuación vamos a ver cómo validaremos un número de teléfono:

```
function validaTelefono(){
  valor = document.getElementById("telefono").value;

  if( !(/^\d{9}$/.test(valor)) ) {
    return false;
  }
  return true;
}
```

En el código anterior estamos validando que el número que introdujo el usuario este formado por dígitos y tenga una longitud de 9. A la hora de validar un teléfono podemos ser más estrictos. Si queremos que el campo solo corresponda a un número de teléfono móvil, podemos especificar que el número solo pueda comenzar por 6, puesto que en España los teléfonos móviles comienzan todos por 6. La expresión regular integrada en la condición anterior para comprobar esto sería la siguiente:

```
if( !(/^[6]\d{8}$/.test(valor)) ) {
```

Esta expresión indica que el dato debe comenzar por el número 6 y que los siguientes 8 caracteres deben ser numéricos. En el caso de que queramos validar un número de teléfono fijo, el código sería el que mostramos a continuación:

```
if( !(/^[89]\d{8}$/.test(valor)) ) {
```

En el código los números solo pueden comenzar por 8 o por 9, que son los números actuales válidos en España.

En el caso de que la nomenclatura de teléfonos cambiara por alguna razón tecnológica, solamente hay que cambiar la función que valida el dato, para que ésta permita devolver verdadero en los casos de los teléfonos que cumplan con las expresiones regulares.

Existen otros muchos datos que podemos validar con expresiones regulares, como pueden ser números de cuentas bancarias, CIF de empresas y cualquier dato que siga un patrón determinado. Validar los datos para que el usuario no envíe el formulario si estos no son correctos es algo que aumenta el rendimiento del servidor, al disminuir las peticiones. También es importante mostrar una información amplia de como el usuario tiene que introducir los datos que se validen, de lo contrario, si el usuario no conoce cómo debe ingresar el dato puede quedarse bloqueado ante la aplicación. Ser demasiado estrictos en las validaciones a veces es contraproducente puesto que la curva de aprendizaje de la aplicación web puede aumentar considerablemente.

En los casos de validaciones estrictas tenemos que tener en cuenta que el servicio de atención al usuario debe tener medios para facilitar el manejo de la aplicación web al usuario, bien sea telefónico, presencial o por correo electrónico, dependiendo de la infraestructura en la que se enmarque nuestra aplicación.