DESARROLLO WEB EN ENTORNO CLIENTE TEMA5: GESTIÓN DE EVENTOS Y FORMULARIOS

CFGS - DAW 2

JESUITAS

Sagrado Corazón
Jesuitas · Logroño

Índice

1. Modelos de gestión de eventos

- Los eventos son los mecanismos que se accionan al suceder dentro del navegador (carga de la página) o como acción del usuario (hacer click, pulsar teclas, mover el ratón…)
- Eventos ocurren todo el tiempo en el navegador.
- El encargado de crear la jerarquía de objetos que compone una página web es el DOM (Document Object Model).
- Por tanto es el DOM el encargado de gestionar los eventos.

1. Modelos de gestión de eventos

- El DOM permite a JavaScript identificar y responder a eventos que suceden
- Para poder controlar un evento se necesita un **manejador** *(event handler)*.
- El manejador es la palabra reservada que indica la acción que va a manejar.
- En el caso del evento click, el manejador sería onClick. Ejemplo:

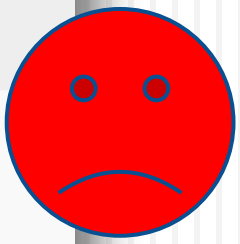
```
<body>
    <IMG SRC="mundo.jpg" onclick="alert('Click en imagen');">
</body>
```

1. Modelos de gestión de eventos

- El ejemplo anterior se puede realizar de otro modo llamando a una función:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script type="text/javascript">
        function func1() {
            alert("Click en imagen");
        }

    </script>
</head>
<body>
    <IMG SRC="mundo.jpg" onclick="func1();">
</body>
</html>
```



1. Modelos de gestión de eventos

- También podemos acceder a las propiedades de los elementos para manejar los eventos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Counter</title>
    <script>
        window.onload = initializer;
        var theCount = 0;
        function initializer(){
            document.getElementById("incrementButton").onclick =
            increaseCount;
        }
        function increaseCount(){
            theCount++;
            document.getElementById("currentCount").innerHTML =
            theCount;
        }
    </script>
</head>
<body>
    <h1>Click the button to count.</h1>
    <p>Current Number: <span id="currentCount">0</span></p>
    <button id="incrementButton">Increase Count</button>
</body>
</html>
```

A.5.1. Cambiarlo para usar *.addEventListener().

1. Modelos de gestión de eventos

- *addEventListener*() tiene 3 argumentos:
  - El primero es el tipo de evento. Nombre del evento sin el prefijo 'on'.
  - El segundo parámetro es el método a ejecutar cuando ocurra el evento.
  - El tercero indica el orden en el que se ejecutan los eventos. Si en una página ocurren más de un evento a la vez, lo normal es que se ejecuten de dentro hacia afuera (*bubbling up vs. capturing*), para eso dejarlo siempre con el valor por defecto *false*.

A.5.2. Pensar o buscar un ejemplo que ejemplifique porqué es importante controlar el orden en el que se ejecutan los eventos.

1. Modelos de gestión de eventos

- La especificación DOM define cuatro grupos de eventos dividiéndolos según su origen:
  - Eventos del ratón.
  - Eventos del teclado.
  - Eventos HTML.
  - Eventos DOM.

1. Modelos de gestión de eventos

1. **Eventos del ratón:**
   - *Click*: Este evento se produce cuando pulsamos sobre el botón izquierdo del ratón. El manejador de este evento es *onclick*.
   - *Dblclick*: Este evento se acciona cuando hacemos un doble click sobre el botón izquierdo del ratón. El manejador de este evento es *ondblclick*.
   - *Mousedown*: Este evento se produce cuando pulsamos un botón del ratón. El manejador de este evento es *onmousedown*.
   - *Mouseout*: Este evento se produce cuando el puntero del ratón esta dentro de un elemento y este puntero es desplazado fuera del elemento. El manejador de este evento es *onmouseout*.

1. Modelos de gestión de eventos

- *Mouseover*: Este evento al revés que el anterior se produce cuando el puntero del ratón se encuentra fuera de un elemento, y este se desplaza hacia el interior. El manejador de este evento es *onmouseover*.
- *Mouseup*: Este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El manejador de este evento es *onmouseup*.
- *Mousemove*: Se produce cuando el puntero del ratón se encuentra dentro de un elemento. Es importante señalar que este evento se producirá continuamente una vez tras otra mientras el puntero del ratón permanezca dentro del elemento. El manejador de este evento es *onmousemove*.

1. Modelos de gestión de eventos

2. **Eventos del teclado:**
   - *Keydown*: Este evento se produce cuando pulsamos una tecla del teclado. Si mantenemos pulsada una tecla de forma continua, el evento se produce una y otra vez hasta que soltemos la misma. El manejador de este evento es *onkeydown*.
   - *Keypress*: Este evento se produce si pulsamos una tecla de un carácter alfanumérico (El evento no se produce si pulsamos enter, la barra espaciadora, etc.). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El manejador de este evento es *onkeypress*.
   - *Keyup*: Este evento se produce cuando soltamos una tecla. El manejador de este evento es *onkeyup*.

1. Modelos de gestión de eventos

3. **Eventos del HTML:**
   - *Load*: El evento load hace referencia a la carga de distintas partes de la página. Este se produce en el objeto Window cuando la página se ha cargado por completo. En el elemento <img> actúa cuando la imagen se ha cargado. En el elemento <object> se acciona al cargar el objeto completo. El manejador es *onload*.
   - *Unload*: El evento unload actúa sobre el objeto Window cuando la pagina ha desaparecido por completo (por ejemplo, si pulsamos el aspa cerrando la ventana del navegador). También se acciona en el elemento <object> cuando desaparece el objeto. El manejador es *onunload*.
   - *Abort*: Este evento se produce cuando el usuario detiene la descarga de un elemento antes de que haya terminado, actúa sobre un elemento <object>. El manejador es *onabort*.

1. Modelos de gestión de eventos

- *Error*: El evento error se produce en el objeto Window cuando se ha producido un error en JavaScript. En el elemento <img> cuando la imagen no se ha podido cargar por completo y en el elemento <object> en el caso de que un elemento no se haya cargado correctamente. El manejador es *onerror*.
- *Select*: Se acciona cuando seleccionamos texto de los cuadros de textos <input> y <textarea>. El manejador es *onselect*.
- *Change*: Este evento se produce cuando los cuadros de texto <input> y <textarea> pierden el foco y el contenido que tenían ha variado. También se producen cuando un elemento <select> cambia de valor. El manejador es *onchange*.
- *Submit*: Este evento se produce cuando pulsamos sobre un botón de tipo submit. El manejador es *onsubmit*.

1. Modelos de gestión de eventos

- *Reset*: Este evento se produce cuando pulsamos sobre un botón de tipo reset. El manejador es *onreset*.
- *Resize*: Este evento se produce cuando redimensionamos el navegador, actúa sobre el objeto Window. El manejador es *onresize*.
- *Scroll*: Se produce cuando varía la posición de la barra de scroll en cualquier elemento que la tenga. El manejador es *onscroll*.
- *Focus*: Este evento se produce cuando un elemento obtiene el foco. El manejador es *onfocus*.
- *Blur*: Este evento se produce cuando un elemento pierde el foco. El manejador es *onblur*.

1. Modelos de gestión de eventos

4. **Eventos del DOM** (*legacy y está por especificar*):
   - *DOMSubtreeModified*: Este evento se produce cuando añadimos o eliminamos nodos en el subárbol de un elemento o documento.
   - *DOMNodeInserted*: Este evento se produce cuando añadimos un nodo hijo a un nodo padre.
   - *DOMNodeRemoved*: Este evento se produce cuando eliminamos un nodo que tiene nodo padre.
   - *DOMNodeRemovedFromDocument*: Este evento se produce cuando eliminamos un nodo del documento.
   - *DOMNodeInsertedIntoDocument*: Este evento se produce cuando añadimos un nodo al documento.

2. Modificación del comportamiento de un formulario

- Los formularios tienen unas acciones predeterminadas por defecto (submit/reset).
- Sin embargo es posible darle otro comportamiento.

2. Modificación del comportamiento de un formulario

- Para tener acceso al formulario:

```
//Acceso a través del atributo ID
var miForm=document.getElementById('formDatos');
var miForm2=document.getElementsByTagName('form')

//Acceso a la colección formularios del objeto document, filtrando por nombre
var miForm3=document.forms['infoUsuario']
```

2. Modificación del comportamiento de un formulario

- Por ejemplo podríamos querer que los datos se envíen a URLs diferentes en base a un dato que introduzca el usuario:

```
function enviar(){
    console.log("Numero: ");
    if (document.formValidado.publ.checked == true){
        document.formValidado.action = "paginas/alta.html";
    }

    if (document.formValidado.publ.checked == false){
        document.formValidado.action = "paginas/baja.html";
    }
    document.formValidado.submit();
}
```

A.5.3. Prueba la función… aplicado a un formulario que definas. ¿Funciona¿?Qué ocurre?¿Cómo podríamos controlarlo?

3. Validación y envío

- El usuario puede cometer errores al rellenar un formulario.
- Si por ejemplo se espera un código postal y se introduce el nombre de una ciudad, se producirá un error.
- Para controlar estas situaciones se pueden usar las validaciones.
- Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones establecidas.

3. Validación y envío

- Validar campos obligatorios:

```
<script type="text/javascript">
 function validar(user,pwd)
  {
   user=document.getElementById(user);
   pwd=document.getElementById(pwd);
   if(user.value=="")
    {
     alert("El campo Nombre esta vacio");
     user.focus();
     return false;
    }
   else
    {
     if(pwd.value=="")
      {
       alert("El campo Password esta vacio");
       pwd.focus();
       return false;
      }
     else
      {
       return true;
      }
    }
  }
</script>
```

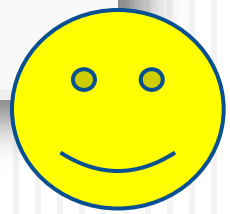
3. Validación y envío


```
<body>
 <form name="formulario" method="post" action="" onSubmit="return validar('nombre','passwd')">
  <table>
   <tr><td>Nombre: </td><td><input type="text" name="nombre" id="nombre" /></td></tr>
   <tr><td>Password: </td><td><input type="password" name="passwd" id="passwd" /></td></tr>
   <tr><td colspan="2"><input type="submit" name="Conectate" value="Conectate" /></td></tr>
  </table>
 </form>
</body>
```

3. Validación y envío

- Validar campo como numérico:

```
<script type="text/javascript">
 function validar()
 {
   num=document.getElementById("telefono");
   if (isNaN(num.value)) {
     alert("El campo Telefono debe ser un número valido");
     num.focus();
     return false;
   }else{
    return true;
   }
 }
</script>
```



3. Validación y envío

- Validar si una fecha es correcta:

```
<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>
```

3. Validación y envío

- Validar un checkbox:

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

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

3. Validación y envío

- Añadir, que la invocación a la validación se puede enfocar de distintas maneras.
- Hasta ahora hemos visto cómo capturar el evento en el código HTML para que devuelva true (en cuyo caso se enviará el formulario) o false (en cuyo caso no se enviará).

3. Validación y envío

- La forma quizás preferible es usar *addEventListener* y *preventDefault*:

```
document.nombreDelFormulario.addEventListener('submit', validarFormulario);

function validarFormulario(evObject) {
    evObject.preventDefault(); //Evita el envío del formulario hasta comprobar

}
```

3. Validación y envío

- Ejemplo:

```
<script type="text/javascript">
    window.onload = function () {
        document.formulario.nombre.focus();
        document.formulario.addEventListener('submit', validarFormulario);
    }

    function validarFormulario(evento) {
        evento.preventDefault();
        var todoCorrecto = true;
        var formulario = document.formulario;
        for (var i=0; i<formulario.length; i++) {
            if(formulario[i].type =='text') {
                if (formulario[i].value == null || formulario[i].value.length == 0 || /^\s*$/.test(
                    formulario[i].value)){
                    alert (formulario[i].name+ ' no puede estar vacío o contener sólo espacios en
                        blanco');
                    todoCorrecto=false;
                }
            }
        }
        if (todoCorrecto ==true) {formulario.submit();}
    }

</script>
```

3. Validación y envío


```
<form name ="formulario" class="formulario1" method="POST" action="">
    <p>Envía el siguiente formulario relleno:</p>
    <label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label
    >
    <label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="
    apellidos" /></label>
    <label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email"
    /></label>
    <label><input type="submit" value="Enviar"><input type="reset" value="Cancelar"></label>
</form>
```

3. Validación y envío

Recordar expresiones regulares… Final del TEMA3

A.5.4. Crear un formulario, donde ante el evento de submit, se valida una dirección de correo electrónico. Define el correspondiente método. Usa *addEventListener* y *preventDefault*.

A.5.5. Al igual que el anterior, añade un método que valide un DNI y otro para validar números de teléfono.