UD3 - Formularios

DAW2 - DEWC

Formularios

Permiten a los usuarios acceder y manipular datos.

El documento dispone de una colección con todos los formularios hijo “**document.forms**”.

- Podemos acceder a través del índice o mediante el valor de “name”.

Cada formulario dispone de una colección con sus entradas “**formulario.elements**”.

- Sólo elementos de entrada: “<input>”, “<textarea>”, “<select>”, o “<button>”.

```
<form name="miFormulario">
    <input type="text" name="nombre" value="Juan">
    <input type="email" name="correo" placeholder="Ingresa tu correo"
value="juan@example.com">
    <button type="submit">Enviar</button>
</form>
```

```
const formulario = document.forms['miFormulario'];

// Acceder a los campos por su índice
console.log(formulario.elements[0].name); // "nombre"

// Acceder a los campos por su nombre
const campoCorreo = formulario.elements['correo'];
```

Propiedades

action: URL envío de datos.

method: método de envío (GET o POST)

enctype: formato de codificación de los datos (valores frecuentes)

- application/x-www-form-urlencoded (por defecto)
- multipart/form-data (envío de archivos)

target: modo de apertura (valores frecuentes)

- _self (por defecto)
- _blank (nueva ventana)

Métodos

Podemos lanzar algunos eventos por código.

formulario**.submit()**

formulario**.reset()**

Validaciones - atributos

required: (De tipo booleano) Indica que el campo es obligatorio.

min y **max**: Definen el rango permitido de valores para campos numéricos o fechas.

maxlength y **minlength**: Establecen el número máximo y mínimo de caracteres permitidos.

pattern: Define una expresión regular que el campo debe cumplir.

step: Define el incremento para valores numéricos.

placeholder: Proporciona un texto indicativo dentro del campo.

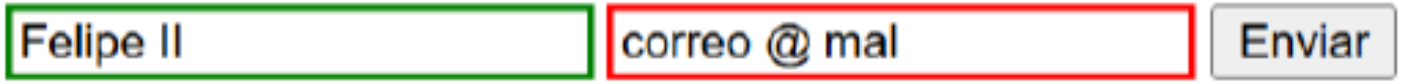
Ejemplo atributos

```
<form name="miFormulario">
    <label for="nombre">Nombre (requerido, mínimo 3
caracteres):</label><br>
    <input type="text" id="nombre" name="nombre" required
minlength="3"><br><br>

    <label for="edad">Edad (entre 18 y 99):</label><br>
    <input type="number" id="edad" name="edad" min="18" max="99"><br><br>

    <label for="correo">Correo (formato válido):</label><br>
    <input type="email" id="correo" name="correo" required><br><br>

    <label for="codigo">Código Postal (5 dígitos):</label><br>
    <input type="text" id="codigo" name="codigoPostal" pattern="\d{5}"
required><br><br>

    <button type="submit">Enviar</button>
</form>
```

Validaciones - Pseudoclases

Podemos aplicar estilos cuando el campo es valido "**:valid**" o invalido "**:invalid**"

```
<style>
    input:valid {
        border: 2px solid green;
    }

    input:invalid {
        border: 2px solid red;
    }
</style>
```



Validaciones – Métodos y propiedades

formulario**.checkValidity()**: devuelve true cuando el formulario es valido.

campo**.checkValidity()**: devuelve true cuando el campo es valido.

campo.**validity**: objeto con el estado de todas las validaciones del campo.

campo.**validationMessage**: mensaje de error para el usuario (sólo lectura).

campo.**setCustomValidity('Mensaje error')**: establece un error personalizado.

- RECUERDA: Al poner un mensaje vacío se elimina el error.

Técnicas de validación

Automática con atributos de validación => muestra popups con los errores.

Manual, al modificar un campo => evento “blur” o ~~“input”~~ y método setCustomValidity()

=> blur o input, depende lo que quieras construir

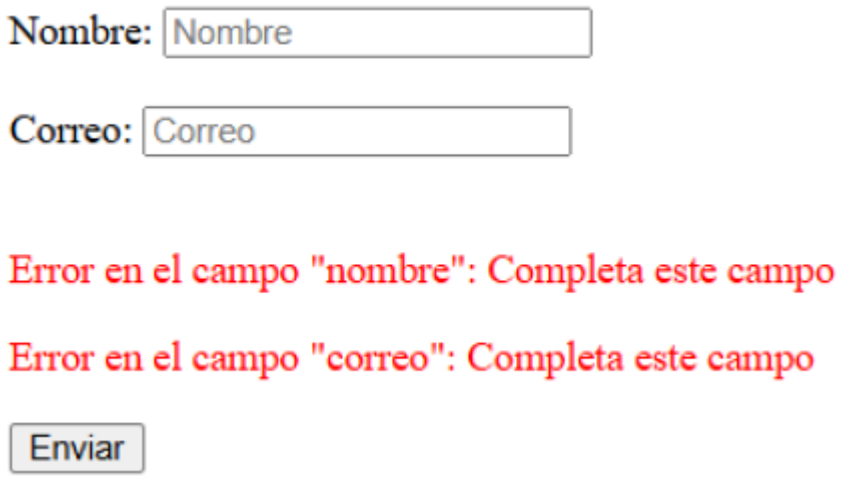
Manual, al enviar el formulario => evento “submit” y método preventDefault()

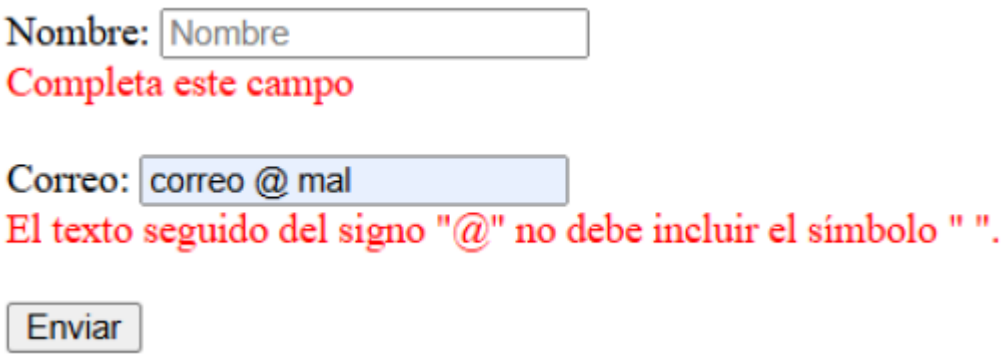
Cómo notificar errores de validación

Popup con el error (por defecto)

Recuadro resumen => empleamos un contenedor externo para mostrar los errores de manera agrupada.

Resumen en campo => cada campo tiene asociado un span tras el input en el que se muestran los errores del campo.





Formulario con detalles (1:N)

Podemos construir formularios con múltiples entidades asociadas (por ej: una factura)

Problema => No podemos repetir atributos “id”

Solución => Asignar el atributo “name” con un nombre con formato array (Rompe la estructura del objeto agrupando por campo).

```
<div class="filaDetalle">
    <div>
        <label>Descripción:</label>
        <input type="text" name="descripcion[]" required>
    </div>
```

```
{
  "cliente": "Candela",
  "fecha": "2024-11-20",
  "descripcion[]": [
    "lego casa gaby",
    "amigos de gaby"
  ],
  "cantidad[]": [
    "1",
    "3"
  ],
```

Gestionar POST y GET

Ambas peticiones con colecciones de pares de clave y valor.

Clase **FormData** => permite componer el cuerpo de una petición POST.

- La información va en el header de la petición.
- Uso típico: const datos = new FormData(miFormulario);

Clase **URLSearchParams** => permite leer y componer una petición GET.

- La información va en la URL (longitud limitada).
- Uso típico: const parámetros = new URLSearchParams(window.location.search);

Conclusiones

Hay que tener clara la estructura del formulario, propiedades y “elements”

Hay que tener claro los atributos HTML para validaciones y las propiedades y métodos JS para validar.

Los formularios complejos se montan mediante JS.

No está de más saber como se gestionan las peticiones GET y POST.

Preguntas