UNIDAD FORMATIVA 2. PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN DE PÁGINAS WEB

En esta unidad formativa se abordará la importancia de **probar** el sitio web para garantizar su correcto funcionamiento y optimizar la experiencia del usuario. Entre los aspectos esenciales se incluyen:

- **Pruebas de usabilidad**: Verificar que la navegación y la interfaz sean claras.
- **Pruebas de rendimiento**: Medir tiempos de carga y uso de recursos.
- **Pruebas de compatibilidad**: Asegurar que la página funciona en distintos navegadores y dispositivos.
- **Optimización de recursos**: Reducir tamaño de imágenes, minificar archivos CSS/JS, emplear *caching*, etc.

UNIDAD DIDÁCTICA 1. VALIDACIONES DE DATOS EN PÁGINAS WEB

Dentro de la optimización y las pruebas de funcionalidades, uno de los aspectos clave es la **validación de datos**. Se persigue asegurar que la información suministrada por el usuario cumpla ciertos criterios, evitando errores y posibles vulnerabilidades de seguridad.

1. FUNCIONES DE VALIDACIÓN

Explicación

Las **funciones de validación** son fragmentos de código (scripts) diseñados para comprobar que los datos ingresados por el usuario cumplen con ciertas reglas (por ejemplo, un correo electrónico con el formato correcto, un campo numérico que no exceda un valor máximo, etc.). Estas validaciones pueden realizarse:

- **En el cliente (Front-end)**, usando JavaScript para una respuesta inmediata al usuario.
- **En el servidor (Back-end)**, para asegurar la consistencia de los datos antes de guardarlos en la base de datos o procesarlos.

Tipos de validación en JavaScript

1. **Validación de tipo**: Asegurar que los datos sean de un tipo esperado (número, texto, etc.).
2. **Validación de formato**: Uso de **expresiones regulares** para verificar patrones (correos, teléfonos, contraseñas, etc.).
3. **Validación de longitud**: Comprobar que un texto cumpla la longitud mínima y/o máxima.
4. **Validación personalizada**: Reglas más complejas (por ejemplo, comparar dos campos de contraseña).

Ejemplo

// Función simple para validar un email usando expresión regular

```
function validarEmail(email) {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}

// Ejemplo de uso:
const correo = "usuario@dominio.com";
if (validarEmail(correo)) {
  console.log("Correo válido");
} else {
  console.log("Correo inválido");
}
```

Ejercicio Propuesto

1. **Múltiples reglas**: Implementar una función validarFormulario() que compruebe:
    - Que el nombre no esté vacío.
    - Que la edad sea un número entre 1 y 120.
    - Que el email tenga el formato correcto.
    - Mostrar mensajes de error concretos para cada campo que no cumpla la regla.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Formulario de Ejemplo</title>
    <style>
        .error { color: red; }
    </style>
</head>
<body>
    <h1>Formulario de Ejemplo</h1>
    <form id="miFormulario">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre"><br><br>

        <label for="edad">Edad:</label>
        <input type="text" id="edad" name="edad"><br><br>

        <label for="email">Email:</label>
        <input type="text" id="email" name="email"><br><br>

        <input type="submit" value="Enviar">
    </form>
    <div id="errores" class="error"></div>

    <script>

document.getElementById('miFormulario').addEventListener('submit',
function(event) {
            event.preventDefault();
            const nombre = document.getElementById('nombre').value;
```

```
            const edad = document.getElementById('edad').value;
            const email = document.getElementById('email').value;

            const errores = validarFormulario(nombre, edad, email);
            const erroresDiv = document.getElementById('errores');
            erroresDiv.innerHTML = '';

            if (errores.length > 0) {
                errores.forEach(error => {
                    const p = document.createElement('p');
                    p.textContent = error;
                    erroresDiv.appendChild(p);
                });
            } else {
                alert('Formulario enviado correctamente.');
            }
        });

        function validarFormulario(nombre, edad, email) {
            let errores = [];

            if (!nombre.trim()) {
                errores.push("El nombre no puede estar vacío.");
            }

            const edadNum = Number(edad);
            if (isNaN(edadNum) || edadNum < 1 || edadNum > 120) {
                errores.push("La edad debe ser un número entre 1 y
120.");
            }

            const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
            if (!emailPattern.test(email)) {
                errores.push("El email no tiene el formato
correcto.");
            }

            return errores;
        }
    </script>
```

```
</body>
</html>
```

Explicación:

1. **HTML**: Se crea un formulario con tres campos: nombre, edad y email. También hay un div para mostrar los mensajes de error.
2. **CSS**: Una simple clase .error para estilizar los mensajes de error en rojo.
3. **JavaScript**:
    - Se escucha el evento submit del formulario para prevenir el comportamiento por defecto (recargar la página).
    - Se obtienen los valores de los campos del formulario.
    - Se llama a la función validarFormulario() para obtener los errores.
    - Si hay errores, se muestran en el div de errores. Si no hay errores, se muestra un mensaje de éxito.

2. Expresión regular personalizada: Crear una función validarTelefono() para verificar un formato de teléfono local (por ejemplo, 9 dígitos en un país específico), retornando true o false según corresponda

```
function validarTelefono(telefono) {
    // Eliminar espacios en blanco y otros caracteres no numéricos
    const telefonoLimpio = telefono.replace(/\D/g, '');

    // Verificar que la longitud sea 9 y que todos los caracteres sean dígitos
    const esValido = telefonoLimpio.length === 9 && /^\d{9}$/.test(telefonoLimpio);

    return esValido;
}

// Ejemplos de uso
console.log(validarTelefono("123456789")); // true
console.log(validarTelefono("123 456 789")); // true
console.log(validarTelefono("12345678")); // false
console.log(validarTelefono("1234567890")); // false
console.log(validarTelefono("123a56789")); // false
```

Explicación:

1. **Limpieza del Input**: Utiliza una expresión regular \D para eliminar cualquier carácter no numérico del input. Esto es útil si el usuario ingresa espacios o guiones.
2. **Validación**: Verifica que la longitud del número limpio sea exactamente 9 dígitos y que todos los caracteres sean numéricos usando la expresión regular ^\d{9}$.

3. **Retorno**: La función retorna true si el número cumple con las condiciones y false en caso contrario.

---

2. VERIFICAR FORMULARIOS

Explicación

"Verificar un formulario" consiste en:

1. **Detener el envío** de datos si se detectan errores (usando eventos como submit y event.preventDefault() en JavaScript).
2. **Proporcionar retroalimentación** al usuario, resaltando campos incorrectos o mostrando mensajes explicativos.
3. **Mantener los valores** en caso de error, para evitar que el usuario tenga que rellenar todo de nuevo.

Métodos de verificación

- **Verificación en tiempo real (on-the-fly)**: Mediante el evento input o change, se van mostrando al usuario los errores conforme escribe.
- **Verificación al enviar (submit)**: Más tradicional, se ejecuta toda la validación al pulsar el botón "Enviar".

Ejemplo

```
<!DOCTYPE html>
<html>
<body>
  <form id="formulario">
    <label>Nombre:
      <input type="text" id="campoNombre" required>
    </label>
    <label>Email:
      <input type="email" id="campoEmail" required>
    </label>
    <button type="submit">Enviar</button>
  </form>

  <script>
    const form = document.getElementById("formulario");
    form.addEventListener("submit", (e) => {
      // Ejemplo simple de verificación:
      let nombre = document.getElementById("campoNombre").value;
      let email = document.getElementById("campoEmail").value;
```

```
    if (nombre.trim() === "" || email.trim() === "") {
      e.preventDefault(); // Detiene el envío
      alert("Por favor, rellena todos los campos.");
    }
    // Podrían añadirse más validaciones avanzadas aquí
  });
 </script>
</body>
</html>
```

Ejercicio Propuesto

1. **Retroalimentación visual**: Modificar el ejemplo anterior para que, en vez de mostrar un alert, resalte en rojo los campos que estén vacíos o incorrectos, usando element.style.borderColor = 'red'.

```
<!DOCTYPE html>
<html>
<body>
 <form id="formulario">
  <label>Nombre:
   <input type="text" id="campoNombre" required>
  </label>
  <br>
  <label>Email:
   <input type="email" id="campoEmail" required>
  </label>
  <br>
  <button type="submit">Enviar</button>
 </form>

 <script>
  const form = document.getElementById("formulario");
  form.addEventListener("submit", (e) => {
   // Obtener valores de los campos
   let nombre = document.getElementById("campoNombre").value;
   let email = document.getElementById("campoEmail").value;

   // Inicializar bandera para verificar si hay errores
   let hayErrores = false;
```

```
      // Validar campo nombre
      if (nombre.trim() === "") {
        document.getElementById("campoNombre").style.borderColor =
'red';
        hayErrores = true;
      } else {
        document.getElementById("campoNombre").style.borderColor =
'';
      }

      // Validar campo email
      if (email.trim() === "") {
        document.getElementById("campoEmail").style.borderColor =
'red';
        hayErrores = true;
      } else {
        document.getElementById("campoEmail").style.borderColor =
'';
      }

      // Si hay errores, prevenir el envío del formulario
      if (hayErrores) {
        e.preventDefault();
      }
    });
  </script>
</body>
</html>
```

Cambios Realizados:

1. **Eliminación del** alert(): Se eliminó el uso de alert() para mostrar mensajes de error, ya que ahora se resaltarán los campos erróneos.

2. **Estilo de Borde Rojo**: Se utiliza element.style.borderColor = 'red' para resaltar los bordes de los campos que están vacíos. Si un campo está vacío, se cambia su borde a rojo.

3. **Restablecimiento del Estilo**: Si un campo no está vacío, se restablece su borde a su estado original (vacío), eliminando el color rojo de la validación anterior.

4. **Bandera de Errores**: Se introdujo una bandera hayErrores para determinar si debe prevenirse el envío del formulario (e.preventDefault()) si hay algún campo vacío.

5. **Validación Individual**: Se valida cada campo por separado para identificar cuáles están vacíos y cuáles no, y se ajusta el estilo de borde en consecuencia.

2. **Validaciones múltiples**: Agregar un tercer campo (por ejemplo, teléfono), validar su formato y mostrar un mensaje debajo de cada campo que no cumpla la regla, empleando pequeños con texto de error.

```
<!DOCTYPE html>
<html>
<body>
  <form id="formulario">
    <label>Nombre:
      <input type="text" id="campoNombre" required>
    </label>
    <span id="errorNombre" class="error"></span>
    <br>
    <label>Email:
      <input type="email" id="campoEmail" required>
    </label>
    <span id="errorEmail" class="error"></span>
    <br>
    <label>Teléfono:
      <input type="text" id="campoTelefono" required>
    </label>
    <span id="errorTelefono" class="error"></span>
    <br>
    <button type="submit">Enviar</button>
  </form>

  <script>
    const form = document.getElementById("formulario");
    form.addEventListener("submit", (e) => {
      // Obtener valores de los campos
      let nombre = document.getElementById("campoNombre").value;
      let email = document.getElementById("campoEmail").value;
      let telefono =
document.getElementById("campoTelefono").value;

      // Inicializar bandera para verificar si hay errores
      let hayErrores = false;

      // Validar campo nombre
```

```
      if (nombre.trim() === "") {
        document.getElementById("campoNombre").style.borderColor = 'red';
        document.getElementById("errorNombre").textContent = "El nombre no puede estar vacío.";
        hayErrores = true;
      } else {
        document.getElementById("campoNombre").style.borderColor = '';
        document.getElementById("errorNombre").textContent = '';
      }

      // Validar campo email
      if (email.trim() === "") {
        document.getElementById("campoEmail").style.borderColor = 'red';
        document.getElementById("errorEmail").textContent = "El email no puede estar vacío.";
        hayErrores = true;
      } else if (!validarEmail(email)) {
        document.getElementById("campoEmail").style.borderColor = 'red';
        document.getElementById("errorEmail").textContent = "El email no tiene el formato correcto.";
        hayErrores = true;
      } else {
        document.getElementById("campoEmail").style.borderColor = '';
        document.getElementById("errorEmail").textContent = '';
      }

      // Validar campo teléfono
      if (!validarTelefono(telefono)) {
        document.getElementById("campoTelefono").style.borderColor = 'red';
        document.getElementById("errorTelefono").textContent = "El teléfono debe tener 9 dígitos.";
        hayErrores = true;
      } else {
        document.getElementById("campoTelefono").style.borderColor = '';
        document.getElementById("errorTelefono").textContent = '';
      }
```

```
      // Si hay errores, prevenir el envío del formulario
      if (hayErrores) {
        e.preventDefault();
      }
    });

    function validarEmail(email) {
      const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      return emailPattern.test(email);
    }

    function validarTelefono(telefono) {
      const telefonoLimpio = telefono.replace(/\D/g, '');
      return telefonoLimpio.length === 9 &&
/^\d{9}$/.test(telefonoLimpio);
    }
  </script>

  <style>
    .error {
      color: red;
      font-size: 0.9em;
    }
  </style>
</body>
</html>
```

Explicaciones de los Cambios:

1. **Campo de Teléfono**: Se añade un campo de entrada para el número de teléfono con el id="campoTelefono".

2. **Mensajes de Error**: Se incluyen elementos debajo de cada campo para mostrar mensajes de error específicos. Cada uno tiene un id único como errorNombre, errorEmail, y errorTelefono.

3. **Validación de Teléfono**: Se crea una función validarTelefono() que verifica que el teléfono tenga exactamente 9 dígitos después de eliminar cualquier carácter no numérico.

4. **Validación de Email**: Se crea una función validarEmail() que utiliza una expresión regular para verificar que el formato del correo electrónico sea correcto.

5. **Estilo de Error**: Se añade una clase .error en CSS para estilizar los mensajes de error en color rojo y con un tamaño de fuente ligeramente más pequeño.

6. **Mostrar Mensajes de Error**: Si un campo es inválido, su borde se cambia a rojo y se muestra el mensaje de error correspondiente en el . Si es válido, se restablece el borde y se limpia el mensaje de error.

7. **Prevención del Envío**: Si hay algún error, se evita el envío del formulario utilizando e.preventDefault().

Conclusión

En esta sección, hemos abordado:

- **La reutilización de scripts**: Encontrar, evaluar y adaptar librerías o fragmentos de código para agilizar el desarrollo y asegurar calidad.
- **Las pruebas de funcionalidades y optimización**: Una perspectiva global que permitirá, en las siguientes unidades, adentrarnos en métodos y herramientas para evaluar el rendimiento y la experiencia de usuario.
- **Validaciones de datos en páginas web**: Un componente imprescindible para garantizar la fiabilidad de la información y mejorar la usabilidad.

Los **ejercicios propuestos** sirven para afianzar habilidades en la búsqueda inteligente de recursos (scripts), la validación de formularios y la verificación de datos, sentando bases sólidas para un desarrollo web seguro, mantenible y con altos estándares de calidad.