

UD 5

Interacción con el usuario

DESARROLLO WEB EN ENTORNO CLIENTE

Técnico de Grado Superior Desarrollo de Aplicaciones Web

2024-25



J. Mario Rodríguez
jrodper183e@g.educaand.es

Contenidos

Interacción con el usuario. Eventos y formularios:

- Eventos.
- Modelo de gestión de eventos.
- Manejadores de eventos.
- Obtener información del evento (objeto event).
- Formularios.
- Utilización de formularios desde código.
- Modificación de apariencia y comportamiento.
- Validación y envío de formularios.
- Expresiones regulares.
- Utilización de cookies.

Eventos

Los eventos permiten que los usuarios interactúen con una página web a través de clics, teclas, movimientos de ratón, etc.

JavaScript es capaz de "escuchar" estos eventos y ejecutar código en respuesta.

```
<button onclick="alert('¡Hola, usuario!')">Haz clic aquí</button>
```

Los eventos pueden ser *lanzados* por los usuarios o por el navegador.

Un usuario puede hacer click en un botón.

El navegador puede avisar cuando una página ha terminado de cargarse.

etc.

Modelo de gestión de eventos

Los eventos en JS siguen un modelo de propagación que se divide en:

1. Fase de *captura*: El evento viaja desde el elemento raíz hacia el objetivo.
2. Fase de *burbuja*: El evento vuelve al elemento raíz, "burbujeando" hacia arriba.

Podemos utilizar `addEventListener`

nos permite controlar en qué fase queremos actuar

```
<button id="miBoton">Haz clic aquí</button>
```

```
const boton = document.getElementById("miBoton");  
boton.addEventListener("click", function() {  
    alert( "¡Hola, usuario!");  
});
```



Modelo de gestión de eventos

`addEventListener` es la forma de registrar un listener de eventos, como se especifica en W3C DOM. Sus beneficios son los siguientes:

- Permite agregar mas de un listener a un solo evento.
- Da un control mas detallado de la fase en la que el listener se
- Funciona en cualquier elemento del DOM, no únicamente con elementos de HTML.

¿De qué forma solía hacerse antes?

```
el.onclick = modifyText;  
  
// Using a function expression  
element.onclick = function () {  
    // ... function logic ...  
};
```

Manejadores de eventos

En JavaScript, existen varias formas de definir manejadores de eventos:

- Incrustado en HTML con `onclick`, `onchange`, etc.
- Directamente en JavaScript usando `addEventListener`
- Eliminación de eventos con `removeEventListener`

```
function mostrarMensaje() {  
    alert('¡Evento activado!');  
}  
  
let boton = document.getElementById("miBoton");  
boton.addEventListener("click", mostrarMensaje);  
  
// Para eliminar el manejador de evento  
boton.removeEventListener("click", mostrarMensaje);
```

Manejadores de eventos

Se producen eventos en muchos tipos de interacciones:

- El usuario selecciona, hace clic o pasa el ratón por encima de cierto elemento.
- El usuario presiona una tecla del teclado.
- El usuario redimensiona o cierra la ventana del navegador.
- Una página web terminó de cargarse.
- Un formulario fue enviado.
- Un vídeo se reproduce, se pausa o termina.
- Ocurrió un error.

...

Reto:

Crea dos botones; uno que agregue un evento y otro que lo quite.

Usa `addEventListener` y `removeEventListener`

Obtener información del evento

El objeto Event contiene detalles sobre el evento que ha ocurrido:

- `event.type`: Tipo de evento.
- `event.currentTarget`: Elemento que tiene registrado el evento.
- `event.target`: Elemento que desencadenó el evento.
- `event.timeStamp`: En qué momento (milisegundos) se produjo el evento.
- `event.preventDefault()`: Evita la acción predeterminada del evento.

La diferencia entre ambos: quién maneja el evento / dónde se disparó

Podemos hacer el `preventDefault` del click en el button o del submit del form

Reto:

Implementa un formulario que no se envíe si el usuario hace clic en "Enviar". En lugar de enviarse, muestra un mensaje en consola.

Características de los eventos

Reto:

Crea una web con varios elementos anidados. Por ejemplo, un div con un div dentro con otro div dentro; o un body, con un div, con un button...

Intenta aplicar `addEventListener` tanto a los contenedores como a cada uno de los elementos, para observar los elementos implicados y la fase.

```
function manejarEvento(event) {  
  console.log(`Evento en ${event.currentTarget.id} - Fase: ${event.eventPhase}`);  
}
```



Existe el método `event.stopPropagation()` para evitar que se transmita

¿Qué significan los valores 1 2 3?

Características de los eventos

Los eventos pueden propagarse en fase de **captura** o **burbuja**.

Por defecto, se transmiten hacia los elementos contenedores, **burbuja**.

¿Cómo cambiar la transmisión del evento para que sea en fase **captura**?

Reto:

Cambia la web anterior, modificando el **addEventListener** para que se propague en captura.

element.addEventListener

✓ Baseline Widely available




Resumen

`addEventListener()` Registra un evento a un objeto en específico. El [Objeto específico](#) ^(inglés) puede ser un simple [elemento](#) en un archivo, el mismo [documento](#), una [ventana](#) o un [XMLHttpRequest](#) ^(inglés).

Para registrar más de un `eventListener`, puedes llamar `addEventListener()` para el mismo elemento pero con diferentes tipos de eventos o parámetros de captura.

Sintaxis

```
target.addEventListener(tipo, listener[, useCapture]);  
target.addEventListener(tipo, listener[, useCapture, wantsUntrusted 
```

tipo

Una cadena representando el [tipo de evento](#) a escuchar.

listener

El objeto que recibe una notificación cuando un evento de el tipo especificado ocurre. Debe ser un objeto implementando la interfaz [EventListener](#)  o solo una [function](#) ^(inglés) en JavaScript.

useCapture Opcional

Si es `true`, `useCapture` indica que el usuario desea iniciar la captura. Después de iniciar la captura, todos los eventos del tipo especificado serán lanzados al `listener` registrado antes de comenzar a ser controlados por algún `EventTarget` que esté por debajo en el árbol DOM del documento.

Características de los eventos

Reto:

Crea una página con:

Una lista sin orden de elementos

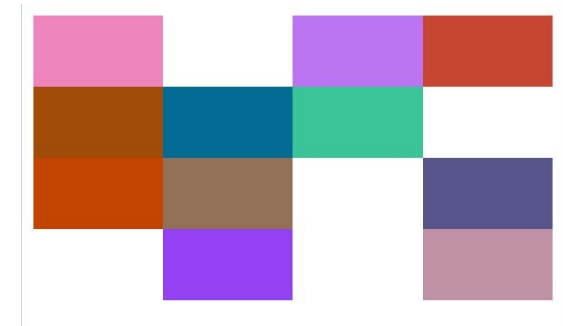
Un input de texto

Cuando se introduce texto, al pulsar *Intro* crea un nuevo elemento en la lista

Cuando se hace click en alguno de los elementos de la lista, su texto se tacha.

Se trata de un ejemplo de **delegación de eventos**: queremos que la interacción del usuario abarque a unos elementos hijos de otro elemento. Estrategia: ubicar el listener en el elemento padre y que los eventos en los hijos “hagan burbuja” al manejador del padre.

Los eventos con listeners son relativamente pesados para el procesamiento de la página.
¿Cómo optimizarías su uso?



Formularios

recolectando
información del
usuario

```
<form id="miFormulario">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre" required>

  <label for="email">Correo electrónico:</label>
  <input type="email" id="email" name="email" required>

  <label for="mensaje">Mensaje:</label>
  <textarea id="mensaje" name="mensaje"></textarea>

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

Atributos importantes:

- **action**: URL donde se envían los datos.
- **method**: Método HTTP (GET o POST).
- **name** e **id**: Identificadores del formulario o sus elementos.
- **Validaciones** HTML5 como *required*, *minlength*, *maxlength*, y *pattern*.

Formularios: desde código

con JavaScript podemos acceder a los formularios y a sus elementos para interactuar dinámicamente con ellos

```
// Acceso al formulario
const formulario = document.getElementById("miFormulario");

// Acceso a los campos
const nombre = formulario.elements["nombre"];
const email = formulario.elements["email"];

// Modificación de valores
nombre.value = "Juan Pérez";
console.log(`Email inicial: ${email.value}`);
```

Formularios: modificación de apariencia y comportamiento

```
nombre.addEventListener("focus", function() {  
    nombre.style.backgroundColor = "grey";  
});  
  
nombre.addEventListener("blur", function() {  
    nombre.style.backgroundColor = "white";  
});
```

```
formulario.addEventListener("submit", function(event) {  
    event.preventDefault(); // Evita el envío  
    alert(`Formulario con: ${nombre.value}, ${email.value}`);  
});
```

Validación de formularios

JavaScript permite validar formularios **antes de enviarlos**
esto es una parte fundamental para garantizar datos correctos

```
formulario.addEventListener("submit", function(event) {  
    if (!email.value.includes("@")) {  
        alert("Por favor, ingrese un correo válido.");  
        event.preventDefault(); // Evita el envío si hay errores  
    }  
});
```

HTML5 también permite algunas validaciones

```
<input type="text" id="telefono" name="telefono" pattern="^\d{9}$" required>
```

Envío de formularios

los formularios se envían automáticamente cuando el usuario interactúa:
con el botón de tipo submit o al presionar la tecla Enter en un campo de texto

```
<form action="/submit" method="POST">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre">

  <label for="email">Correo:</label>
  <input type="email" id="email" name="email">

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

Atributos clave <form>

- **action**: especifica la URL donde se enviarán los datos.
- **method**: define el método HTTP para enviar los datos:
 - **GET**: Los datos se envían como parte de la URL.
No recomendado para datos *sensibles*.
 - **POST**: Los datos se envían en el body de la solicitud.
Es más seguro y permite enviar datos más grandes.

1. El navegador recoge los datos de los campos con el atributo name.
2. Envía los datos al destino especificado en action usando el método definido.

Envío de formularios

1. El navegador recoge los datos de los campos con el atributo name.
2. Envía los datos al destino especificado en action usando el método definido.

Aunque es fácil de usar, el envío directo desde HTML tiene desventajas:

- Falta de control dinámico: es difícil realizar validaciones personalizadas antes de enviar
- Recarga de página: cada envío recarga la página
Esto ~~puede~~ afectar a la experiencia del usuario.
- Limitada seguridad: Si se usa GET, los datos aparecen en la URL... peligrosillo!

Es más común manejar el envío mediante JavaScript.

Con JavaScript podemos interceptar el envío, realizar validaciones y enviar datos de forma asincrónica sin recargar la página (usando AJAX o fetch)

Envío de formularios

Es más común manejar el envío mediante JavaScript.

Con JavaScript podemos interceptar el envío, realizar validaciones y enviar datos de forma asincrónica sin recargar la página (usando AJAX o fetch)

```
formulario.addEventListener("submit", async function(event) {  
    event.preventDefault();  
  
    const datos = new FormData(formulario);  
  
    const respuesta = await fetch("/submit", {  
        method: "POST",  
        body: datos  
    });  
  
    if (respuesta.ok) {  
        alert("Formulario enviado correctamente.");  
    } else {  
        alert("Hubo un error al enviar el formulario.");  
    }  
});
```

Envío de formularios

Los datos de un formulario pueden enviarse a:

```
<form action="/procesar.php" method="POST">
```

Un endpoint web
PHP, python, nodejs, java...

Un API REST

```
fetch("https://api.ejemplo.com/endpoint", { method: "POST", body: datos });
```

¿En qué **formato** se envían los datos?

- `application/x-www-form-urlencoded` (por defecto)
datos codificados como pares clave=valor en el cuerpo de la solicitud
- `multipart/form-data`
necesario para archivos, es el formato usado automáticamente con FormData

Envío de formularios

¿Se pueden enviar archivos?

/upload es un ejemplo del endpoint que “espera” o “acepta” recibir archivo

enctype

```
<form action="/upload" method="POST" enctype="multipart/form-data">
  <label for="archivo">Subir archivo:</label>
  <input type="file" id="archivo" name="archivo">
  <button type="submit">Enviar</button>
</form>
```

```
const archivo = document.getElementById("archivo").files[0];
const formData = new FormData();
formData.append("archivo", archivo);

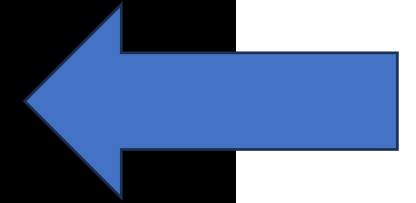
fetch("/upload", {
  method: "POST",
  body: formData
}).then(response => console.log("Archivo subido"));
```

POST nos permite enviar
ficheros

Envío de formularios

Validación antes del envío

```
formulario.addEventListener("submit", function(event) {  
    const nombre = formulario.elements["nombre"].value;  
    if (nombre.length < 3) {  
        alert("El nombre debe tener al menos 3 caracteres.");  
        event.preventDefault(); // Evita el envío  
    }  
});
```




Expresiones regulares

a menudo llamadas RegExp o RegEx, son un sistema para buscar, capturar o reemplazar texto utilizando patrones. Estos patrones permiten realizar una búsqueda de texto de forma abstracta, que abarca una gran cantidad de posibilidades que de otra forma sería imposible o muy extensa y compleja.

Estos patrones se representan mediante una cadena de texto, donde ciertos símbolos tienen un significado especial.

```
// Notación literal
const regexp = /.a.o/i;

// Notación de objeto
const regexp = new RegExp(".a.o", "i");
const regexp = new RegExp(/.a.o/, "i");
```



Forma preferida

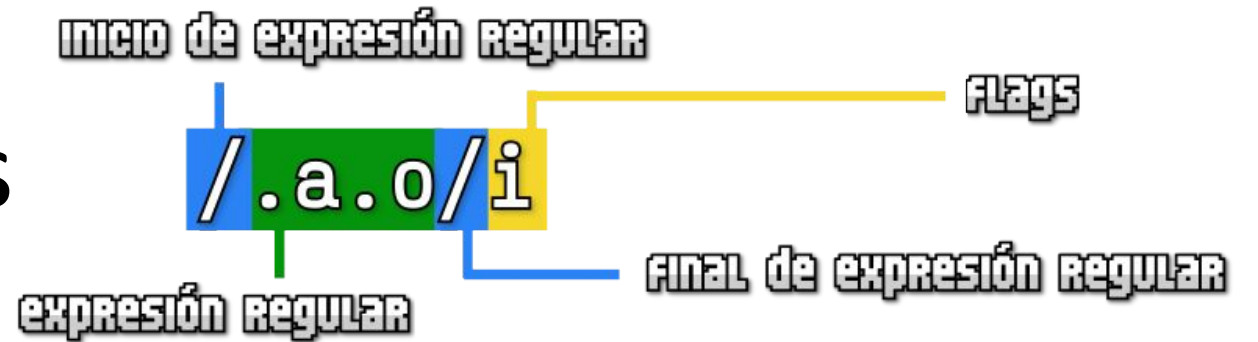
Expresiones regulares

permiten verificar patrones complejos en cadenas de texto, como correos electrónicos, números de teléfono, etc.

```
const regexEmail = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;

formulario.addEventListener("submit", function(event) {
  if (!regexEmail.test(email.value)) {
    alert("El correo no tiene un formato válido.");
    event.preventDefault();
  }
});
```

Expresiones regulares



Propiedades	Flag	Descripción
BOOLEAN <code>.global</code>	<code>g</code>	Búsqueda global. Permite seguir buscando coincidencias en lugar de pararse al encontrar una.
BOOLEAN <code>.ignoreCase</code>	<code>i</code>	Le da igual mayúsculas y minúsculas. Se suele denominar insensible a mayús/minús .
BOOLEAN <code>.multiline</code>	<code>m</code>	Multilínea. Permite a <code>^</code> y <code>\$</code> tratar los finales de línea <code>\r</code> o <code>\n</code> .
BOOLEAN <code>.unicode</code>	<code>u</code>	Unicode. Interpreta el patrón como un código de una secuencia Unicode.
BOOLEAN <code>.sticky</code>	<code>y</code>	Sticky. Busca sólo desde la posición indicada por <code>lastIndex</code> .
BOOLEAN <code>.dotAll</code>	<code>s</code>	Establece si <code>\n</code> , <code>\r</code> , separación de párrafo o separación de línea deberían considerarse en los <code>.</code>
BOOLEAN <code>.hasIndices</code>	<code>d</code>	Establece si al ejecutar un <code>.exec()</code> el resultado deberá tener propiedad <code>.indices</code> .

Caracter especial	Descripción
.	Comodín, significa cualquier carácter (letra, número, símbolo...), pero que ocupe sólo 1 carácter .
\	Precedido de un carácter especial, lo invalida (se llama «escapar»).

Caracter especial	Descripción
[]	Rango de caracteres. Cualquiera de los caracteres del interior de los corchetes.
[^]	Que no exista cualquiera de los caracteres del interior de los corchetes.
	Establece una alternativa: lo que está a la izquierda o lo que está a la derecha.

Caracter especial	Alternativa	Descripción
[0-9]	\d	Un dígito del 0 al 9.
[^0-9]	\D	No exista un dígito del 0 al 9.
[A-Z]		Letra mayúscula de la A a la Z. Excluye ñ o letras acentuadas.
[a-z]		Letra minúscula de la a a la z. Excluye ñ o letras acentuadas.
[A-Za-z0-9]	\w	Carácter alfanumérico (letra mayúscula, minúscula o dígito).
[^A-Za-z0-9]	\W	No exista carácter alfanumérico (letra mayúscula, minúscula o dígito).
[\t\r\n\f]	\s	Carácter de espacio en blanco (espacio, TAB, CR, LF o FF).
[^ \t\r\n\f]	\S	No exista carácter de espacio en blanco (espacio, TAB, CR, LF o FF).
	\xN	Carácter hexadecimal número N.
	\uN	Carácter Unicode número N.

Caracter especial	Descripción
<code>^</code>	Ancla. Delimita el inicio del patrón. Significa empieza por .
<code>\$</code>	Ancla. Delimita el final del patrón. Significa acaba en .
<code>\b</code>	Límite de una palabra separada por espacios, puntuación o inicio/final.
<code>\B</code>	Opuesta al anterior. Posición entre 2 caracteres alfanuméricos o no alfanuméricos.

Caracter especial	Descripción
<code>*</code>	El carácter anterior puede aparecer 0 o más veces.
<code>+</code>	El carácter anterior puede aparecer 1 o más veces.
<code>?</code>	El carácter anterior puede aparecer o no (es opcional).
<code>{n}</code>	El carácter anterior aparece n veces.
<code>{n,}</code>	El carácter anterior aparece n o más veces.
<code>{n,m}</code>	El carácter anterior aparece de n a m veces.

Expresiones regulares

¿Cómo usarlas con JavaScript?

Método	Descripción
BOOLEAN <code>test(text)</code>	Comprueba si la expresión regular «casa» con el texto <code>text</code> pasado por parámetro.
ARRAY <code>exec(text)</code>	Ejecuta una búsqueda en el texto <code>text</code> . Devuelve ARRAY con capturas de lo que coincide.


```
const regexp = /.a.o/i;

regexp.test("gato");    // true
regexp.test("pato");    // true
regexp.test("perro");   // false
regexp.test("DATO");    // true  (el flag i ignora mayúsculas/minúsculas)
```

Expresiones regulares

¿Cómo usarlas con JavaScript?

Método	Descripción
BOOLEAN <code>test(text)</code>	Comprueba si la expresión regular «casa» con el texto <code>text</code> pasado por parámetro.
ARRAY <code>exec(text)</code>	Ejecuta una búsqueda en el texto <code>text</code> . Devuelve ARRAY con capturas de lo que coincide.



```
const text = `Hola Manz,  
  
Soy el otro Manz (el gato) y necesito Whiskas.  
El pato del patio sigue haciendo ruido. Te lo digo como dato.  
  
Gracias.`;  
  
const regexp = /(a.o)/g;  
  
regexp.exec(text); // ["gato)", "gato"]  
regexp.exec(text); // ["pato ", "pato"]  
regexp.exec(text); // ["dato.", "dato"]  
regexp.exec(text); // null
```

Expresiones regulares

Expresión regular para un teléfono

`^\d{9}$`

- `^` Indica que el patrón debe comenzar al inicio del texto
- `\d` Representa un dígito (0-9)
- `{9}` Especifica que debe haber exactamente 9 dígitos
- `$` Indica que el patrón debe terminar al final del texto

Ejemplo válido: 123456789

Ejemplo no válido: 12345abc (contiene letras) 12345678 (menos de 9 dígitos)

Reto adicional:
Haz que el campo de teléfono permita espacios opcionales entre los dígitos (por ejemplo, 123 456 789)

Expresiones regulares

Un *lookahead* en expresiones regulares es un tipo de aserción que permite verificar si, en una posición determinada, a una cadena le sigue un patrón.

Sintaxis:

- **Positivo:** (? = . . .)

Verifica que el patrón dentro del paréntesis sí aparece después de la posición actual.

- **Negativo:** (? ! . . .)

Verifica que el patrón dentro del paréntesis no aparece después de la posición actual.

- (? = . * [A - Z]) : Lookahead positivo que verifica que haya al menos una letra mayúscula.
- (? = . * \d) : Lookahead positivo que verifica que haya al menos un número.
- (? = . * [\W _]) : Lookahead positivo que verifica que haya al menos un carácter especial.
- . { 8 , } : Al menos 8 caracteres en total.

Expresiones regulares

Expresión regular para una contraseña

al menos una mayúscula, un número y un carácter especial

`^(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*])(?=.{8,})$`

`^`: Indica que el patrón debe comenzar al inicio del texto.

`(?=.*[A-Z])`: Un lookahead que exige al menos una letra mayúscula.

`[A-Z]`: Representa cualquier letra mayúscula (de la A a la Z).

`(?=.*\d)`: Un lookahead que exige al menos un dígito (número).

`\d`: Representa un dígito (0-9).

`(?=.*[!@#$%^&*])`: Un lookahead que exige al menos un carácter especial.

`[!@#$%^&*]`: Representa cualquier carácter que no sea una letra o número. Incluye símbolos como @, #, \$, &, _, etc.

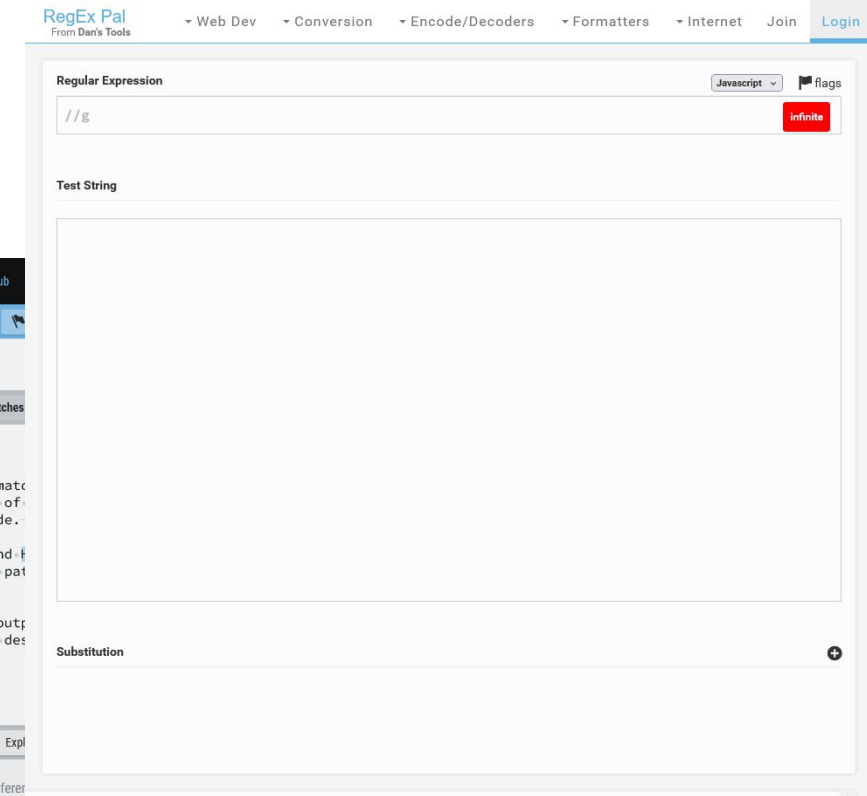
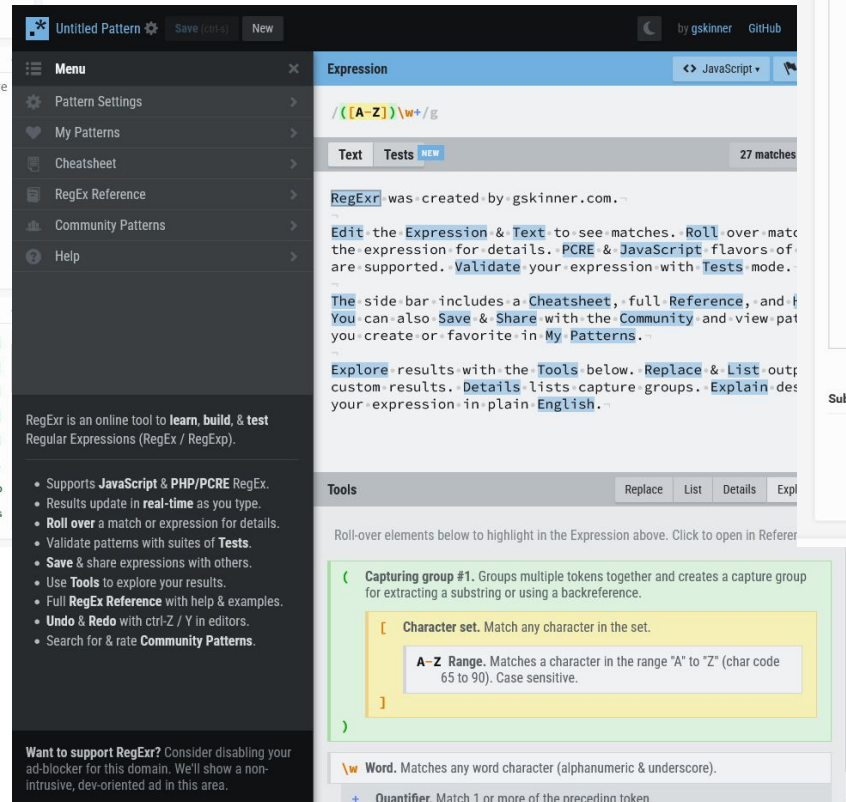
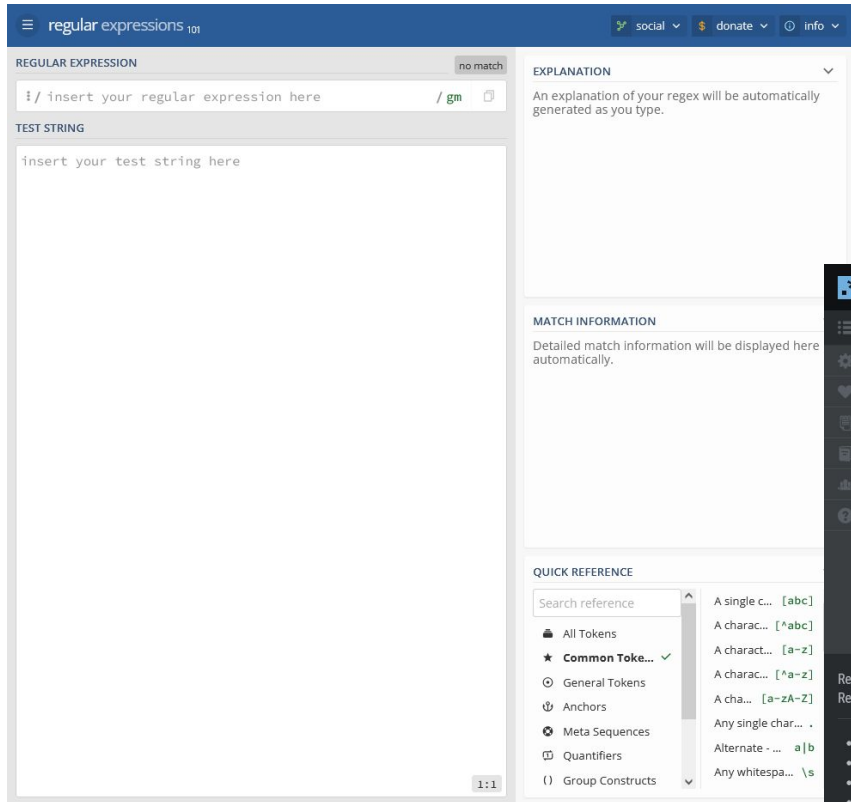
`(?=.{8,})`: Indica que la contraseña debe tener al menos 8 caracteres.

`.`: Representa cualquier carácter.

`{8,}`: Especifica que debe haber un mínimo de 8 caracteres.

`$`: Indica que el patrón debe terminar al final del texto.

Expresiones regulares



Expresiones regulares

Reto:

Implementar un medidor de seguridad para la contraseña que indique si es "débil", "media" o "fuerte" en base a los caracteres introducidos.

Cookies



Cookies

son pequeños fragmentos de información almacenados por el navegador en el dispositivo del usuario

¿Por qué son útiles en formularios?

- **Persistencia de datos**
Guardan temporalmente información para evitar que los usuarios pierdan datos en formularios largos.
- **Preferencias del usuario**
Recuerdan opciones seleccionadas por el usuario.
- **Seguimiento de sesión**
Permiten identificar usuarios mientras navegan en una web.

Cookies

Crear una cookie

```
document.cookie = "nombre=Juan; expires=Fri, 31 Dec 2024 23:59:59 UTC; path="/;
```

Leer una cookie

```
console.log(document.cookie);
```

Eliminar una cookie

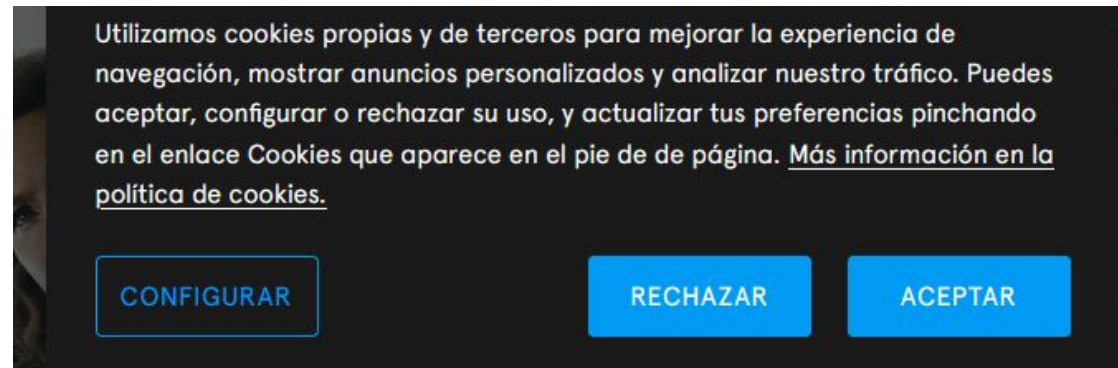
```
document.cookie = "nombre=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path="/;
```

Reto:

Al enviar el formulario, guarda el nombre en una cookie y muestra un mensaje de bienvenida en la siguiente visita.

Cookies

¿Son viables las cookies con navegadores actuales?



Sí, aunque su uso es un poco “dudoso”

Por políticas de privacidad y GDPR, requieren consentimiento del usuario!!!

...y hay modos de almacenamiento alternativo

