T6: Romancero Gitano I

Javier Ribal del Río

2025-12-19

Table of contents

1 Objetivo de la tarea

El objetivo de esta tarea es aplicar los conceptos básicos de HTML, CSS y JavaScript (DOM) para crear una página web interactiva que muestre poemas del *Romancero gitano* de Federico García Lorca y modifique su apariencia visual en función de las diferentes metáforas.

Resultado esperado

2 Estructura HTML

El documento HTML debe:

- Estar correctamente estructurado (`<!DOCTYPE>`, `<html>`, `<head>`, `<body>`).
- Contener un **contenedor principal** centrado horizontalmente.
- Incluir, en este orden:
  1. Un elemento para mostrar **información métrica** del poema (número de estrofas y versos).

2. Un elemento para el **título del poema**.
3. Un elemento para el **texto del poema**, respetando los saltos de verso.
4. Un **botón** que permita cambiar de poema.

No debe utilizarse ningún *framework* ni librería externa.

3 JavaScript

3.1 Datos

En JavaScript se debe definir un **array de objetos**, donde cada objeto represente un poema y tenga al menos:

- Un título.
- El texto del poema como una cadena de texto, usando saltos de línea para los versos.

El cambio de poema debe hacerse recorriendo este array de forma cíclica (cola circular) mediante el botón.

Descargar poemas (click derecho guardar enlace como)

3.2 Manipulación del DOM

El comportamiento de la página debe implementarse **exclusivamente con DOM nativo**, cumpliendo lo siguiente:

- Todas las funciones deben declararse como **arrow functions**.
- Al pulsar el botón:
  - Se actualiza el título.
  - Se actualiza el texto del poema.
  - Se modifica el formato del fondo.
  - Se recalculan y muestran:
    - Número de versos.
    - Número de estrofas (bloques separados por líneas en blanco).

El cálculo debe realizarse a partir del texto del poema, no con valores predefinidos.

4 CSS

Lorca utiliza las metáforas y la alegoría como elemento principal sobre el que construye sus poemas, al analizar profundamente descubrimos que siempre recurre a las mismas metáforas.

El color no se utiliza aquí solo como elemento decorativo, sino como un recurso semántico: cada color representa un símbolo literario presente en el poema.

Desde el punto de vista técnico: - Los **colores de fondo y de texto** deben definirse en **clases CSS**. - JavaScript no decide colores, únicamente decide qué clase aplicar según el contenido del poema.

4.1 Palabras clave y colores asociados

Se debe detectar el texto del poema las siguientes **palabras clave** y aplicar el **color de fondo correspondiente**:

- **luna** → fondo **negro**
- **guardia civil** → fondo **verde militar**
- **sangre** → fondo **granate oscuro**
- **caballo** → fondo **color tierra**
- **cuchillo / navaja** → fondo **plateado**

El color del texto debe elegirse siempre de forma que exista **contraste suficiente** con el fondo y se garantice la legibilidad.

Si el poema contiene varias palabras clave, solo debe aplicarse **un único estilo**, siguiendo un orden de prioridad definido previamente.

Si el poema no contiene ninguna de estas palabras, se aplicará un **estilo por defecto** con colores aleatorios.

5 Extra opcional: símbolo *aurora*

De forma **voluntaria**, se puede añadir el símbolo `aurora`, que tendrá **prioridad absoluta sobre todos los demás**.

Cuando el poema contenga la palabra `aurora`, el fondo debe mostrar un **degradado animado**.

5.1 CSS necesario para el extra

```css
.aurora {
  background: linear-gradient(120deg,
    #ff005d,
    #ff9f1c,
    #ffee32,
    #3cff00,
    #00e5ff,
    #7a00ff,
    #ff00c8
  );
  background-size: 600% 600%;
  animation: aurora 10s linear infinite;
  color: black;
}

@keyframes aurora {
  0%   { background-position: 0% 50%; }
  50%  { background-position: 100% 50%; }
  100% { background-position: 0% 50%; }
}
```

---

6 Consejos

6.1 Centrado de la página (layout clásico)

Toda la página debe estar centrada **horizontalmente**.

El centrado se consigue mediante:

- Un contenedor con un ancho máximo definido.
- Márgenes automáticos a izquierda y derecha.

Desde el punto de vista conceptual:

- `margin: 0 auto` **no centra texto**, centra **bloques**.
- El navegador reparte automáticamente el espacio sobrante a ambos lados del contenedor.

6.2 Animación del cambio de color

Para evitar que los cambios de color sean bruscos, se utiliza la propiedad **`transition`** en CSS.

Esta propiedad indica al navegador que los cambios en determinadas propiedades visuales deben hacerse de forma progresiva.

Ejemplo conceptual:

```
body {
  transition: background-color 0.6s, color 0.6s;
}
```

Gracias a esta transición: - Cuando JavaScript cambia una clase o un color, - el navegador interpola automáticamente entre el color anterior y el nuevo.

No es necesario programar animaciones en JavaScript: **CSS se encarga de todo el efecto visual**.

6.3 Búsqueda de una palabra

Para buscar si determinada palabra determinada se debe utilizar la función **`includes`** de la clase string

```
texto = "hoy nieva";

texto.includes("nieva");    //TRUE
texto.includes("llueve");   //FALSE
```

6.4 Generación del color aleatorio

Para la generacióndel color aleatorio de puede utilizar la siguiente función

```
const randomColor = () => '#' + Math.floor(Math.random() * 16777215).toString(16);
```

6.5 Conteo versos y estrofas

El recuento de versos y estrofas debe hacerse **a partir del texto del poema**, no con valores escritos a mano. A continuación se dan algunas **pistas técnicas** para abordar este problema.

6.5.1 Conteo de versos

Un verso puede considerarse, a efectos prácticos en esta tarea, como **una línea de texto no vacía**.

Una estrategia habitual consiste en: - Separar el texto por saltos de línea. - Eliminar las líneas vacías o formadas solo por espacios. - Contar cuántas líneas válidas quedan.

Ejemplo orientativo:

```
const versos = texto
  .split('\n')
  .filter(linea => linea.trim() !== '')
  .length;
```

Este enfoque es suficiente para la mayoría de los poemas del *Romancero gitano* y evita errores frecuentes.

6.5.2 Conteo de estrofas

Una estrofa puede entenderse como un **bloque de versos separado de otros bloques por una línea en blanco**.

Una posible estrategia es: - Separar el texto usando dos (o más) saltos de línea consecutivos. - Eliminar bloques vacíos. - Contar los bloques resultantes.

Ejemplo orientativo:

```
const estrofas = texto
  .split(/\n\s*\n/)
  .filter(bloque => bloque.trim() !== '')
  .length;
```

Entrega: un único archivo HTML funcional.

7 Recomendaciones para afrontar la tarea

Al enfrentarnos a un proyecto de cierta complejitud, el como afrontarlo resulta de cricial para el desarollo del mismo por ello a a continuación dejo un esquema de implementación modular. Sería mejor no utilizarlo, pero puede ser un buen punto de partida si no se sabe como empezar la app, no es el único proceso de implementación válido ni es más correcto que cualquier otro

1. Estructura de HTML básica contadores de versos y estrofas *placeholders* de poema y título y botón
2. Formateo CSS parte estática: centrado, tamaño (todo lo que no sea color)
3. Definir clases con colores predeterminados
4. Lógica interna con JS

- Buscar palabra
- Contar estrofas
- Conteo versos
- Selección de clase
- Generar color aleatorio si procede

5. Integración con DOM