JS
Relación de proyecto a elegir

Introducción

El objetivo a la hora de realizar **uno de estos proyectos** es poner en práctica todo lo aprendido en los contenidos anteriormente vistos. Antes de usar cualquier framework o librería de frontend es ideal saber afrontar este tipo de proyectos usando JS a secas, aunque puedes ayudarte de librerías externas si lo deseas.

Formato de entrega

Al igual que en ocasiones anteriores debemos subir el proyecto para que pueda ser corregido a Github Pages con un enlace público navegable.

Propuesta 1: Juego de Buscaminas (Minesweeper)

Objetivo del ejercicio:

- Practicar la generación y manipulación dinámica del DOM.
- Utilizar lógica más avanzada para manejar estados de celdas (abiertas, cerradas, banderas, minas).
- Implementar funciones recursivas o de recorrido para descubrir espacios vacíos automáticamente.
- (Opcional) Usar localStorage para guardar el mejor tiempo del jugador.

Ejercicio:

1 Generar el Tablero

- Pide al usuario que elija el tamaño del tablero (ej. 8x8, 16x16) y la cantidad de minas.
- Crea dinámicamente una cuadrícula con celdas HTML.
- Ubica las minas de manera aleatoria.

2 **Lógica del Juego**

- Al hacer clic en una celda:
- Si contiene una mina, mostrar un mensaje de “Game Over” y revelar todas las minas.
- Si no contiene mina, revela el número de minas adyacentes.
- Si es 0 (celda vacía), revela automáticamente todas las celdas adyacentes (y así sucesivamente) hasta topar con celdas que tengan un número mayor que 0.
- Al hacer clic derecho en una celda, se coloca o quita una bandera (para marcar que hay una mina).

3 **Sistema de Victoria**

- El juego finaliza con éxito si el usuario descubre todas las celdas sin mina.
- Muestra un mensaje de felicitación y el tiempo total de juego.

4 **(Opcional) Persistencia con** localStorage

- Guarda el mejor tiempo para un tamaño de tablero determinado.
- Si el usuario gana la partida en menor tiempo, actualiza ese valor.

5 **Extras (ideas de mejora)**

- Añadir un contador de minas.

- Agregar diferentes niveles de dificultad (Fácil, Medio, Difícil).
- Posibilidad de reiniciar el juego sin recargar la página.

Propuesta 2: Resolución de Sudoku (Sudoku Solver)

Objetivo del ejercicio:

- Practicar la implementación de algoritmos más complejos (por ejemplo, backtracking).
- Dominar la manipulación de estructuras de datos en JavaScript.
- (Opcional) Crear una interfaz para jugar y/o resolver tableros de Sudoku dinámicamente.

Ejercicio:

1 **Generación o Entrada de un Tablero**

- Opción A: Permite al usuario ingresar un Sudoku incompleto en un formulario (9x9), dejando celdas vacías en forma de 0 o un espacio en blanco.
- Opción B: Genera automáticamente un Sudoku predefinido o uno de ejemplo.

2 **Algoritmo de Resolución (Backtracking)**

Implementa una función solveSudoku(tablero) que utilice backtracking para:

1. Encontrar una celda vacía.
2. Intentar dígitos del 1 al 9, revisando las reglas del Sudoku (fila, columna y cuadrante 3x3).
3. Si un dígito es válido, se coloca y se continua con la siguiente celda.
4. Si en algún punto no se puede avanzar, se deshace la decisión (backtrack) y se prueba con el siguiente dígito.

Cuando todas las celdas estén llenas correctamente, el Sudoku está resuelto.

3 **Validaciones del Tablero**

Añade una función que valide si el Sudoku inicial es correcto (sin repeticiones en filas, columnas ni sub-cuadrantes).

Controla que el usuario no introduzca caracteres inválidos.

4 **Interfaz Gráfica (Opcional, pero recomendado)**

- Un grid 9x9 que permita al usuario ver o ingresar los números.
- Un botón “Resolver” que ejecute tu función de backtracking y muestre el tablero solucionado.
- Mensajes de error si el Sudoku es inválido o no se puede resolver.

5 **(Opcional) Persistencia y Mejoras**

- Guardar el estado actual del Sudoku en localStorage para continuar la sesión más tarde.
- Incluir un temporizador y almacenar los tiempos de resolución.
- Implementar un generador de Sudoku aleatorio y con distintos niveles de dificultad.

Propuesta 3: CatGallery con The Cat API

api: https://thecatapi.com/

Objetivo del Ejercicio

- Practicar peticiones a una API REST usando **fetch** (o tu biblioteca preferida, como Axios).
- Manejar y presentar datos dinámicamente en el DOM.
- Implementar funcionalidades de guardado en **localStorage** (por ejemplo, para "favoritos").
- Manejar paginación o carga de más resultados.
- Mejorar la experiencia de usuario con funciones de filtrado y/o clasificación, si lo deseas.

Descripción General

1 **Página Principal**

- Debe mostrar un contenedor con varias imágenes de gatos obtenidas de The Cat API.
- Cada imagen debe tener un botón o ícono para marcarla como "favorita".

2 **Favoritos**

- Cuando el usuario marca una imagen como favorita, la información correspondiente (por ejemplo, la URL de la imagen o un ID) se guarda en **localStorage**.
- En la parte superior (o en una sección aparte), muestra o permite al usuario acceder a la galería de favoritos.
- El usuario debe poder **eliminar** de favoritos las imágenes que ya no desee, actualizando así el localStorage.

3 **Paginación o "Ver más"**

- Puedes mostrar inicialmente 9 (o la cantidad que desees) imágenes de gatos.
- Incluye un botón "Ver más" que cargue otras 9 imágenes siguientes desde la API y las añada a la galería.
- Alternativamente, puedes implementar un **scroll infinito**: cuando el usuario llegue al final de la página, se cargan más imágenes automáticamente.

4 **Manejo de Errores y Estados**

- Muestra un **estado de carga** (un texto o spinner) mientras la petición a la API se realiza.
- Si ocurre un error (por ejemplo, conexión fallida), notifícalo al usuario con un mensaje claro.

5 **(Opcional) Filtrado o Búsqueda Avanzada**

- The Cat API soporta parámetros de filtrado por raza, tipo de imagen (jpg, png, gif), etc.
- Agrega un **formulario** con selectores para razas, o checkboxes para el tipo de archivo, para refinar la búsqueda.

6 **(Opcional) Información Extra**

- Cada imagen de gato a veces contiene metadatos sobre la raza o su historia. Muestra datos como “breed name”, “temperament”, “origin” en un modal emergente o en un recuadro aparte.

Propuesta 4 Rick and Morty Explorer

api: https://rickandmortyapi.com/

Objetivo: Crear una aplicación web que consuma datos de la API de Rick and Morty y permita al usuario explorar personajes, episodios o localizaciones de la serie.

Propuesta de Desafío:

1 **Búsqueda y Filtrado de Personajes**

- Mostrar un listado inicial de personajes con sus datos principales (nombre, imagen, estado, especie, etc.).
- Incluir un buscador (input) para filtrar personajes por nombre.
- Agregar filtros opcionales por **estado** (Alive, Dead, Unknown) o **especie**.

2 **Detalle de Personaje**

- Permitir que al hacer clic en un personaje, se muestre la información completa.
- Mostrar datos extra como localización, origen y lista de episodios en los que aparece.

3 **Visualización de Episodios (Opcional)**

- Incluir una sección para episodios, mostrando información relevante (nombre del episodio, fecha de emisión, personajes que participan, etc.).
- Permitir ver la lista de personajes de un episodio y enlazar a su detalle.

4 **Paginación o Carga Dinámica**

- La API de Rick and Morty ofrece datos paginados. Implementar la paginación o un botón “Cargar más” para obtener más resultados.

5 **Favoritos (Opcional con LocalStorage)**

- Agregar una funcionalidad para “marcar” personajes o episodios como favoritos.
- Guardar y persistir esa lista de favoritos usando localStorage.
- Mostrar una sección o página independiente con la lista de favoritos.

6 **Manejo de Errores y Estados**

- Mostrar feedback al usuario mientras se cargan los datos (spinner o mensaje de “Cargando…”).

- Informar si ocurre algún error al consumir la API (problema de red, endpoint incorrecto, etc.).

7 **Diseño y Responsividad**

- Procurar un diseño agradable y adaptado a distintos tamaños de pantalla (móvil, tableta, desktop).