UNIVERSIDAD TECMILENIO

CAMPUS LAS TORRES

“INNOVACION CON PROPOSITO DE VIDA”

ACTIVIDAD 2

MATERIA: FULL STACK

ALUMNO: RODRIGO MEJIA ROCHA

MATRICULA: T03028726

CARRERA: IDS

https://github.com/bomejia/GESTORDETAREAS.git

Introducción

Lo que se quiso poner en práctica en esta actividad básicamente fue desarrollar una web app interactiva para gestionar tareas un tipo to-do list, utilizando HTML, CSS y JavaScript, sin frameworks ni librerías externas.

La app permite al usuario agregar nuevas tareas, marcarlas como completadas, editar su descripción y eliminarlas. Además, se implementó persistencia mediante LocalStorage para que las tareas no e pierdan al cerrar o recargar el navegador.

Con esta actividad se aplicaron de manera practica los conceptos clave que hemos visto lo que llevamos del curso: manipulación del DOM, algunas características más modernas de JavaScript, programación orientada a objetos y diseño responsivo básico.

Objetivos cumplidos

De acuerdo con las instrucciones de la actividad y la rúbrica de trabajo en canvas:

- Estructurar la página HTML con un campo de entrada, botón para agregar y lista de tareas.
- Crear clases en JavaScript para representar tareas individuales y el gestor general.
- Manipular el DOM dinámicamente para reflejar cambios en tiempo real.
- Utilizar características ES6+ como let/const, funciones flecha, template literals y forEach.
- Validar entradas (no permitir tareas vacías) y hacer la aplicación responsiva.
- Implementar el desafío adicional: almacenamiento persistente con LocalStorage.

Tecnologías y herramientas utilizadas

- **HTML5**: Estructura semántica de la página.
- **CSS3**: Diseño responsivo con flexbox, sombras, transiciones y media queries.

- **JavaScript ES6+**: Lógica completa de la aplicación (clases, eventos, DOM, LocalStorage).
- **LocalStorage**: API nativa del navegador para guardar y recuperar datos.
- Editor de código: Visual Studio Code.

Descripción del diseño e interfaz

La interfaz se diseñó para ser simple, limpia y moderna:

- Fondo con degradado (azul-morado) para un estilo atractivo y moderno .
- Contenedor centrado con bordes redondeados, sombra suave y padding cómodo.
- Input y botón en una fila (en pantallas grandes); en dispositivos celulares se apilan verticalmente.
- Cada tarea aparece como una tarjeta con:
    - Texto normal o tachado (si está completada).
    - Botones: toggle completado (✓ o ↺), Editar y Eliminar.
- Efectos hover en tarjetas y botones para mejorar la usabilidad.
- Validación visual: alerta si se intenta agregar tarea vacía.

El diseño es completamente responsivo y se ve bien en celulares, tablets y computadoras.

Implementación detallada

1. JavaScript básico y manipulación del DOM Se utilizaron métodos nativos como:

- document.getElementById() para seleccionar el input, botón y lista.
- addEventListener() para capturar clics en botones y la tecla Enter.
- createElement(), appendChild() y innerHTML para generar y actualizar elementos de forma dinámica.
- textContent y clases CSS para tachar tareas completadas.

Todos los cambios en la lista se reflejan inmediatamente sin recargar la página.

2. Características modernas de ES6+ Se aplicaron consistentemente:

- const y let para declarar variables (nunca var).
- Funciones flecha (() => {}) en todos los manejadores de eventos y en forEach.
- Template literals (` ... ${variable} ... `) para construir el HTML de cada tarea en el método render().
- Método forEach para recorrer el arreglo de tareas y crear los elementos li.
- Uso de trim() para limpiar espacios en blanco en las entradas.

3. Programación Orientada a Objetos (POO) Se crearon dos clases principales:

- **Clase Tarea** Representa una sola tarea. Propiedades: nombre (string), completada (boolean), id (número único). Métodos:
    - toggleCompletada(): invierte el estado completado/incompleto.
    - editar(nuevoNombre): actualiza el nombre si no está vacío.
- **Clase GestorDeTareas** Administra la colección completa. Propiedades: tareas (array de instancias de Tarea). Métodos principales:
    - agregarTarea(nombre): valida, crea nueva Tarea, agrega al array, guarda y renderiza.
    - eliminarTarea(id), editarTarea(id, nuevoNombre), toggleTarea(id).
    - render(): limpia la lista ul y genera todos los li dinámicamente.
    - guardarTareas() y cargarTareas(): serializa/deserializa con JSON y LocalStorage.

Esta estructura encapsula la lógica y hace el código más organizado y reutilizable.

4. Persistencia con LocalStorage Cada vez que se agrega, edita, elimina o cambia el estado de una tarea, se ejecuta localStorage.setItem("tareas", JSON.stringify(this.tareas)). Al iniciar la app, se carga con JSON.parse(localStorage.getItem("tareas")) y se reconstruyen las instancias de Tarea.

Funcionamiento de la aplicación

Figura 1: Pantalla inicial (sin tareas).

Figura 2: Agregando una tarea nueva.

Figura 3: Lista con tareas, una marcada como completada (texto tachado).

Figura 4: Editando el texto de una tarea (usando prompt).

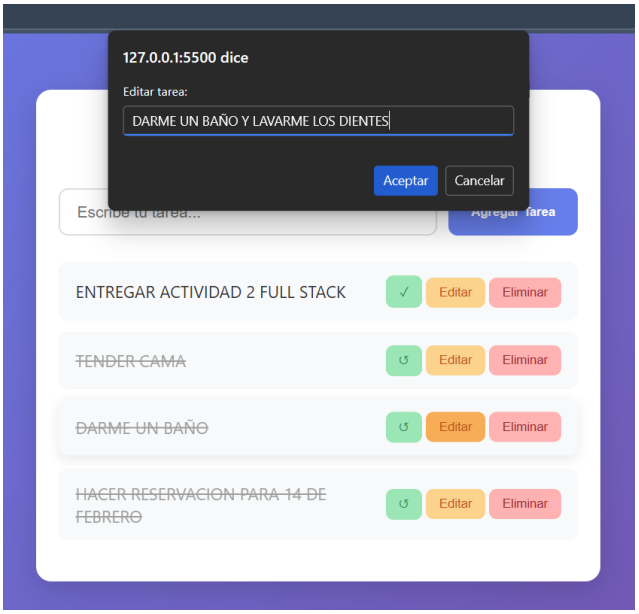

Figura 5: Eliminando una tarea (antes y después).

Figura 6: Vista responsiva en dispositivo móvil (simulado en DevTools).

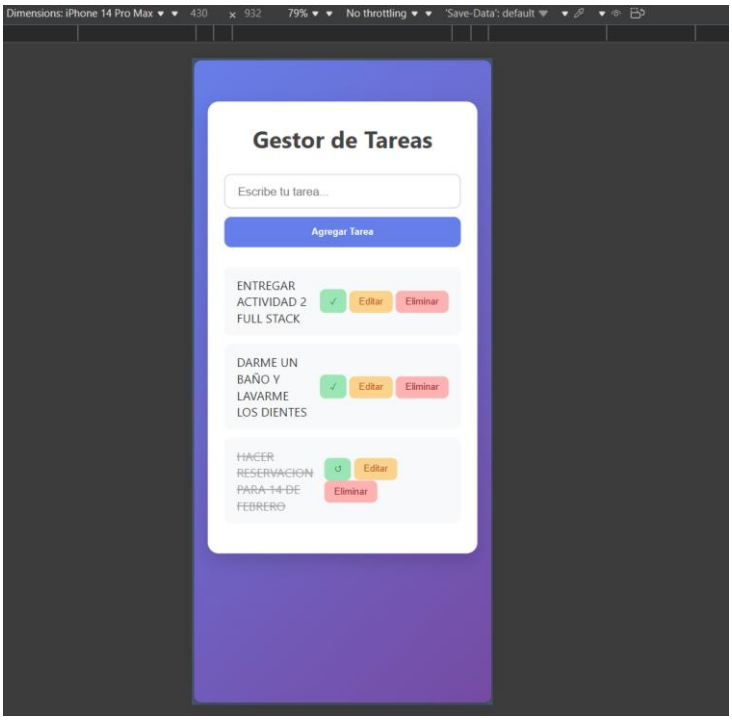

Figura 7: Recarga de página – las tareas persisten gracias a LocalStorage.

Conclusiones

La aplicación cumple con todos los requisitos de la actividad, incluyendo el desafío adicional de persistencia. Se aplicaron correctamente los conceptos de JavaScript básico, ES6+ y POO, logrando una interfaz funcional, atractiva y responsiva.

Este proyecto fortaleció mis habilidades en manipulación del DOM, diseño de clases y manejo de datos en el navegador, conocimientos que serán base para temas más avanzados en Full Stack.