🎯 ACTIVIDAD PRÁCTICA: "Gestor de Tareas Interactivo" - To Do List

📋 Consigna

Desarrolla la funcionalidad JavaScript para una aplicación web de gestión de tareas personales. El HTML y CSS ya están proporcionados. Tu tarea es implementar la lógica JavaScript para manipulación del DOM, manejo de eventos y programación orientada a la interacción del usuario. El proyecto incluye funcionalidades avanzadas como contador de tareas, animaciones y accesibilidad.

🎯 Objetivos de Aprendizaje

1. Manipulación del DOM (Nivel Intermedio)

- Crear elementos dinámicamente con createElement()

- Agregar y remover elementos del DOM con appendChild() y remove()

- Modificar contenido de elementos con textContent

- Gestionar clases CSS con classList.add(), classList.remove(), classList.toggle()

2. Manejo de Eventos (Nivel Intermedio)

- Conectar eventos de click, input y teclado

- Usar addEventListener() para múltiples tipos de eventos

- Implementar eventos de teclado para accesibilidad

- Manejar validación en tiempo real

3. Programación Funcional (Nivel Básico)

- Crear funciones reutilizables

- Separar responsabilidades en funciones específicas

- Implementar validación robusta

- Manejar estados de la aplicación

4. UX/UI y Accesibilidad (Nivel Básico)

- Implementar feedback visual inmediato

- Agregar animaciones y transiciones

- Soporte completo de teclado

- Atributos ARIA para accesibilidad

📝 Tareas Específicas

Tarea 1: Configuración Básica del DOM

1. Obtener referencias a elementos principales usando getElementById()

2. Configurar el evento DOMContentLoaded para asegurar que el DOM esté listo

3. Crear variables para los elementos: input, botón agregar, lista de tareas

Tarea 2: Funciones Reutilizables

1. Crear función createDeleteButton() para generar botones de eliminar

2. Crear función createTaskSpan() para generar elementos de texto de tarea

3. Implementar atributos de accesibilidad (aria-label, tabindex, role)

Tarea 3: Sistema de Validación

1. Crear función validateInput() para validar entrada del usuario

2. Implementar feedback visual (cambios de color, placeholder dinámico)

3. Conectar validación en tiempo real con evento input

Tarea 4: Gestión de Tareas **/ todolist no les gusta a los recruiters**

1. Implementar función agregarTarea() con validación y animaciones

2. Agregar animaciones de entrada para nuevas tareas

3. Conectar eventos de click y teclado (Enter) para agregar tareas

Tarea 5: Contador de Tareas

1. Crear función updateTaskCounter() para mostrar estadísticas

2. Mostrar total de tareas, completadas y pendientes

3. Actualizar contador automáticamente al agregar/eliminar/marcar tareas

Tarea 6: Funcionalidades Avanzadas

1. Implementar función limpiarCompletadas() para eliminar tareas completadas

2. Agregar botón "Limpiar Completadas" dinámicamente

3. Implementar animaciones de eliminación

Tarea 7: Accesibilidad y UX

1. Agregar soporte de teclado para marcar tareas como completadas

2. Implementar atributos ARIA para lectores de pantalla

3. Agregar animaciones y transiciones suaves

💡 Estructura del Código JavaScript

Variables Globales:

javascript

const taskInput = document.getElementById('taskInput');

const addTaskBtn = document.getElementById('addTaskBtn');

const taskList = document.getElementById('taskList');

Funciones Principales:

javascript

// Funciones reutilizables

function createDeleteButton() { / ... / }

function createTaskSpan(text) { / ... / }

// Funciones de gestión

function validateInput() { / ... / }

function agregarTarea() { / ... / }

function updateTaskCounter() { / ... / }

function limpiarCompletadas() { / ... / }

// Event listeners

addTaskBtn.addEventListener('click', agregarTarea);

taskInput.addEventListener('input', / validación en tiempo real /);

taskInput.addEventListener('keypress', / agregar con Enter /);

🔧 Conceptos Técnicos a Implementar

Manipulación del DOM:

- document.createElement()

- element.appendChild()

- element.remove()

- element.textContent

- element.classList.add/remove/toggle()

Eventos:

- addEventListener('click', callback)

- addEventListener('input', callback)

- addEventListener('keypress', callback)

- addEventListener('DOMContentLoaded', callback)

Validación y UX:

- element.value.trim()

- element.style.property

- element.setAttribute()

- setTimeout() para animaciones

Accesibilidad:

- aria-label

- aria-describedby

- tabindex

- role

- Soporte de teclado (Enter, Espacio)

🎨 Estilos CSS (Ya Proporcionados)

Clases Disponibles:

- .task-item - Estilo base para cada tarea

- .task-text - Texto de la tarea (clickeable)

- .delete-btn - Botón de eliminar

- .completed - Tarea completada

- .task-counter - Contador de estadísticas

- .clear-btn - Botón limpiar completadas

Funcionalidades CSS Incluidas:

- ✅ Transiciones suaves para hover

- ✅ Animaciones de entrada para nuevas tareas

- ✅ Animaciones de eliminación

- ✅ Efectos visuales para tareas completadas

- ✅ Responsive design para móviles

- ✅ Estilos de accesibilidad

Nota: No necesitas modificar el CSS, solo enfócate en la lógica JavaScript.

🚀 Funcionalidades Esperadas

Comportamiento del Input:

- ✅ Validación en tiempo real con feedback visual

- ✅ Agregar tarea con Enter o click en botón

- ✅ Limpiar input automáticamente después de agregar

- ✅ Focus automático después de agregar tarea

Comportamiento de las Tareas:

- ✅ Click en texto para marcar/desmarcar como completada

- ✅ Soporte de teclado (Enter/Espacio) para marcar completada

- ✅ Animación suave al marcar como completada

- ✅ Botón eliminar con animación de deslizamiento

Contador de Tareas:

- ✅ Mostrar total de tareas

- ✅ Mostrar tareas completadas

- ✅ Mostrar tareas pendientes

- ✅ Actualización automática en tiempo real

Funcionalidades Extra:

- ✅ Botón "Limpiar Completadas" (se agrega automáticamente)

- ✅ Animaciones fluidas para todas las interacciones

- ✅ Diseño responsive para móviles

- ✅ Accesibilidad completa

💡 Consejos de Desarrollo

Orden de Implementación:

1. Primero: Configurar variables y DOMContentLoaded

2. Segundo: Implementar funciones reutilizables

3. Tercero: Crear sistema de validación

4. Cuarto: Implementar agregar tareas (las animaciones CSS ya están listas)

5. Quinto: Agregar contador de tareas

6. Sexto: Implementar limpiar completadas

7. Séptimo: Agregar accesibilidad y conectar todos los eventos

Recuerda: El CSS ya maneja las animaciones, solo necesitas agregar/remover las clases correctas.

Debugging:

- Usa console.log() para verificar que los elementos se obtienen correctamente

- Verifica que los eventos se conectan correctamente

- Revisa la consola del navegador para errores

- Prueba la funcionalidad en diferentes dispositivos

Errores Comunes:

- No usar DOMContentLoaded puede causar errores

- Olvidar usar trim() puede causar problemas con espacios

- No remover todas las clases antes de agregar la nueva

- No actualizar el contador después de cambios

🎯 Resultado Final

Al completar el ejercicio, tendrás:

- ✅ Una aplicación de gestión de tareas completamente funcional

- ✅ Sistema de validación robusto con feedback visual

- ✅ Contador de tareas en tiempo real

- ✅ Animaciones y transiciones profesionales

- ✅ Accesibilidad completa

- ✅ Diseño responsive

- ✅ Código JavaScript limpio y bien estructurado

- ✅ Comprensión profunda de manipulación del DOM