Práctica 2: "Lista de tareas dinámica con almacenamiento y control de eventos múltiples"

Objetivos:

- Aplicar la manipulación avanzada del DOM.
- Utilizar múltiples tipos de eventos del DOM.
- Modularizar el código usando buenas prácticas de JavaScript.
- Usar localStorage para guardar el estado.
- Interactuar con el HTML desde Node.js con un servidor simple (opcional).

Herramientas:

- Visual Studio Code
- Navegador (Chrome, Firefox, etc.)
- Live Server o un servidor local con Node.js

Paso a paso

Paso 1: Estructura básica del proyecto

1. Crea una carpeta llamada todo-avanzado.
2. Dentro, crea los archivos:
    - index.html
    - styles.css
    - app.js
3. Estructura básica de index.html:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Lista de Tareas Avanzada</title>
  <link rel="stylesheet" href="styles.css"/>
</head>
<body>
```

```
  <h1>Lista de Tareas</h1>
  <form id="task-form">
    <input type="text" id="task-input" placeholder="Escribe tu tarea..." required/>
    <button type="submit">Agregar</button>
  </form>
  <ul id="task-list"></ul>

  <button id="clear-tasks">Eliminar todas</button>

  <script src="app.js"></script>
</body>
</html>
```

Paso 2: Estilo básico en styles.css

```
body {
  font-family: Arial;
  padding: 20px;
  background-color: #f4f4f4;
}

h1 {
  color: #333;
}

form {
  margin-bottom: 20px;
}

input[type="text"] {
  padding: 10px;
  width: 300px;
```

```
}

button {
  padding: 10px;
  margin-left: 5px;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  background: #fff;
  padding: 10px;
  margin-top: 5px;
  border-left: 5px solid green;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Paso 3: Lógica en app.js

1. Agrega las siguientes funcionalidades:
    - Agregar tarea
    - Eliminar tarea individual
    - Marcar como completada
    - Guardar tareas en localStorage
    - Cargar tareas al iniciar

```
const taskForm = document.getElementById('task-form');
const taskInput = document.getElementById('task-input');
```

```
const taskList = document.getElementById('task-list');
const clearBtn = document.getElementById('clear-tasks');

// Cargar tareas desde localStorage
document.addEventListener('DOMContentLoaded', loadTasks);

// Agregar tarea
taskForm.addEventListener('submit', function (e) {
  e.preventDefault();
  addTask(taskInput.value);
  taskInput.value = '';
});

// Eliminar todas las tareas
clearBtn.addEventListener('click', clearTasks);

// Delegar eventos en lista
taskList.addEventListener('click', handleTaskClick);

function addTask(text) {
  const li = document.createElement('li');
  li.innerHTML = `
    <span>${text}</span>
    <button class="delete">X</button>
  `;
  taskList.appendChild(li);
  saveTasks();
}

function handleTaskClick(e) {
  const li = e.target.closest('li');
```

```
  if (!li) return;

  if (e.target.classList.contains('delete')) {
    li.remove();
  } else {
    li.classList.toggle('done');
  }
  saveTasks();
}

function clearTasks() {
  taskList.innerHTML = '';
  saveTasks();
}

function saveTasks() {
  const tasks = [];
  document.querySelectorAll('#task-list li').forEach(li => {
    tasks.push({
      text: li.querySelector('span').textContent,
      done: li.classList.contains('done')
    });
  });
  localStorage.setItem('tasks', JSON.stringify(tasks));
}

function loadTasks() {
  const tasks = JSON.parse(localStorage.getItem('tasks')) || [];
  tasks.forEach(t => {
    addTask(t.text);
    if (t.done) {
```

```
    const last = taskList.lastChild;
    last.classList.add('done');
  }
 });
}
```

Paso 4: Experimentar con eventos

Haz que los alumnos identifiquen:

- submit
- click
- DOMContentLoaded
- Delegación de eventos con e.target

También pueden usar el inspector del navegador para analizar cómo cambia el DOM al interactuar con los botones.

Paso 5 (opcional): Montar un servidor Node.js básico

Crea un archivo server.js:

javascript

CopiarEditar

```
const http = require('http');
const fs = require('fs');
const path = require('path');

const server = http.createServer((req, res) => {
 let filePath = '.' + req.url;
 if (filePath === './') filePath = './index.html';

 const extname = String(path.extname(filePath)).toLowerCase();
 const mimeTypes = {
  '.html': 'text/html',
  '.js': 'text/javascript',
```

```
  '.css': 'text/css',
 };

 const contentType = mimeTypes[extname] || 'application/octet-stream';

 fs.readFile(filePath, (error, content) => {
  if (error) {
   res.writeHead(500);
   res.end('Error interno');
  } else {
   res.writeHead(200, { 'Content-Type': contentType });
   res.end(content, 'utf-8');
  }
 });
});

server.listen(3000, () => console.log('Servidor en http://localhost:3000'));
```

Resultado esperado

Aplicación funcional de lista de tareas:

- Que responde a eventos.
- Que guarda los datos.
- Que manipula dinámicamente el DOM.
- Desde un servidor Node.js.

Actividad propuesta:

Actividad Propuesta: “Calculadora de gastos personales”

Objetivo:

Crear una aplicación web que permita al usuario registrar y visualizar sus gastos personales, usando manipulación del DOM y eventos, y sirviendo el sitio desde un servidor en Node.js.

Descripción de la actividad:

El estudiante deberá desarrollar una aplicación web que permita:

1. **Agregar un gasto**, ingresando:
    a. Descripción
    b. Monto
    c. Categoría (comida, transporte, ocio, etc.)
2. **Visualizar todos los gastos** en una tabla dinámica debajo del formulario.
3. Mostrar el **total acumulado** de todos los gastos en tiempo real.
4. Permitir **eliminar gastos individuales** con un botón.
5. Estilizar la tabla para que se vean claras las categorías y montos.
6. Todo el contenido debe generarse/modificarse dinámicamente mediante **JavaScript en el navegador**, sin recargar.
7. Usar **Node.js** para servir todos los archivos necesarios (HTML, CSS, JS).

Requisitos técnicos:

- Utilizar Node.js con Express o el módulo `http` para servir los archivos estáticos.
- Manejo del DOM para crear elementos de tabla (`<tr>`, `<td>`) al agregar gastos.
- Eventos `submit`, `click` y `change`.
- Cálculo dinámico del total de gastos.
- Buena estructura de carpetas y separación de archivos (HTML, CSS, JS).