

## **Práctica: "Lista de Tareas Interactiva con Node.js, DOM y Eventos"**

### **Objetivo general:**

Desarrollar una pequeña aplicación web en la que los usuarios puedan crear, completar y eliminar tareas. Esta práctica combina el uso de Node.js como servidor local, junto con la manipulación del DOM y el uso de eventos en JavaScript del lado del cliente.

### **PASO A PASO EXPLICADO**

#### **Paso 1: Crear el proyecto**

1. **Abrir Visual Studio Code**

Abre el programa VS Code, que será el entorno donde escribirás y ejecutarás el código.

2. **Crear una nueva carpeta**

Dentro de tu computadora, crea una carpeta llamada lista-tareas. Esta carpeta contendrá todos los archivos del proyecto.

3. **Abrir la carpeta en VS Code**

En Visual Studio Code, selecciona Archivo > Abrir carpeta y elige la carpeta lista-tareas.

4. **Abrir una terminal integrada**

Pulsa Ctrl + ñ o ve a Terminal > Nueva terminal. Esto abrirá una línea de comandos integrada dentro de VS Code.

5. **Inicializar el proyecto con Node.js**

Escribe el siguiente comando y presiona Enter:

```
npm init -y
```

Este comando crea un archivo package.json con la configuración mínima para tu proyecto Node.js.

6. **Instalar Express (servidor web)**

Express es una librería que permite crear servidores web fácilmente. Instálalo con:

```
npm install express
```

#### **Paso 2: Crear la estructura de archivos**

Ahora necesitas crear varios archivos dentro de tu carpeta del proyecto.

1. Dentro de la carpeta principal lista-tareas, crea una subcarpeta llamada public.

Esta carpeta contendrá los archivos que verás en el navegador (HTML, CSS, JS del cliente).

2. Dentro de public, crea tres archivos:

- index.html: la estructura de la página.
- style.css: los estilos visuales.

- script.js: la lógica en el navegador (DOM y eventos).
3. En la carpeta principal (fuera de public), crea un archivo llamado server.js. Este será el archivo principal que ejecutará el servidor con Node.js.

### **Paso 3: Código del servidor (Node.js con Express)**

Abre el archivo server.js y escribe el siguiente código:

```
const express = require('express');

const app = express();

const path = require('path');

// Sirve archivos estáticos desde la carpeta 'public'
app.use(express.static('public'));

// Ruta principal que entrega el HTML
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, '/public/index.html'));
});

// Ejecuta el servidor en el puerto 3000
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

#### **¿Qué hace este código?**

- Importa Express y configura el servidor.
- Permite que se accedan los archivos dentro de la carpeta public.
- Al acceder a la raíz (<http://localhost:3000>), muestra el archivo index.html.
- El servidor se ejecuta en el puerto 3000.

Para iniciar el servidor, ejecuta este comando en la terminal:

```
node server.js
```

Luego abre tu navegador y visita:

<http://localhost:3000>

#### **Paso 4: Crear el archivo HTML (estructura base de la app)**

Abre el archivo public/index.html y escribe:

```
<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="UTF-8">

  <title>Lista de Tareas</title>

  <link rel="stylesheet" href="style.css">

</head>

<body>

  <h1>Mi Lista de Tareas</h1>

  <form id="form-tarea">

    <input type="text" id="input-tarea" placeholder="Escribe una tarea" required>

    <button type="submit">Agregar</button>

  </form>

  <ul id="lista-tareas"></ul>

  <script src="script.js"></script>

</body>

</html>
```

#### **¿Qué contiene este archivo?**

- Un formulario con un campo de texto para escribir tareas y un botón para agregarlas.
- Una lista (<ul>) donde se mostrarán las tareas.
- Un enlace al archivo CSS para los estilos.
- Un enlace al archivo JavaScript donde estará la lógica de la aplicación.

#### **Paso 5: Agregar estilo con CSS**

Abre el archivo public/style.css y agrega lo siguiente:

```
body {

  font-family: sans-serif;
```

```
max-width: 600px;

margin: auto;
}

li {
  display: flex;
  justify-content: space-between;
  margin: 5px 0;
  padding: 5px;
  background: #f1f1f1;
  border-radius: 4px;
}
```

```
.completed {
  text-decoration: line-through;
  color: gray;
}
```

### ¿Qué hace este CSS?

- Mejora la presentación del texto y los elementos.
- Da un diseño de "lista" a los elementos <li>.
- Define cómo se ve una tarea cuando está completada (tachada y gris).

### Paso 6: Lógica en JavaScript (DOM + Eventos)

Abre el archivo public/script.js y escribe:

```
document.addEventListener('DOMContentLoaded', () => {

  const form = document.getElementById('form-tarea');
  const input = document.getElementById('input-tarea');
  const lista = document.getElementById('lista-tareas');

  form.addEventListener('submit', (e) => {
    e.preventDefault();
```

```
const texto = input.value.trim();
if (texto === "") return;

const li = document.createElement('li');

const span = document.createElement('span');
span.textContent = texto;

// Evento para marcar como completada
span.addEventListener('click', () => {
  span.classList.toggle('completed');
});

// Botón de eliminar
const botonEliminar = document.createElement('button');
botonEliminar.textContent = '✖';
botonEliminar.addEventListener('click', () => {
  li.remove();
});

li.appendChild(span);
li.appendChild(botonEliminar);
lista.appendChild(li);

input.value = "";
});
});
```


### ¿Qué hace este código?

- Espera a que se cargue el documento (DOMContentLoaded).
- Detecta cuando se envía el formulario.
- Crea dinámicamente un nuevo <li> con la tarea escrita.

- Permite que al hacer clic en la tarea se tache (evento click).
- Agrega un botón de eliminar, que borra la tarea de la lista.

**Resultado final esperado:**

Cuando el alumno abra la página, verá:

- Un campo para escribir tareas.
- Al agregar una tarea, aparece debajo como un ítem de la lista.
- Si hace clic en el texto, se tacha como completado.
- Puede eliminar una tarea con el botón .

**Actividades:**

1. **Personaliza los estilos:** cambia colores, usa íconos o similar.