República Bolivariana de Venezuela

Ministerio del Poder Popular Para la Educación Universitaria

Universidad Nacional Experimental de las Telecomunicaciones e Informática

PNF: Ingeniería en Informática

Trayecto: 3 Sección: 6y7B

Materia: Programación III

Ejercicio Practico 1

Estudiante:

Jonathan Alvarado

V- 22.749.638

Caracas, 12 de diciembre de 2025

Introducción

En este proyecto se realizaron dos ejercicios usando tecnologías como Node.js, Express, JavaScript, TypeScript, Html, Css y una base de datos Json. A pesar de que son ejercicios prácticos que pueden ser muy simples, pueden ser más complejos ya que existen muchas formas de abordarlo, y ese puede ser un punto de inflexión.

El primer ejercicio puede parecer simple pero el hecho de abordar la tarea de usar node para registrar o capturar un nombre y luego mostrarlo en pantalla puede ser complejo, siempre que no se cuente con el conocimiento general. Para esta tarea se implementó el uso principal de Node.js y Express como Backend, Html, Css y Javascript se encargaron del Frontend.

Por último, el segundo ejercicio fue el más complejo, ya que se debe usar TypeScript con la característica enum, se integró con Node.js y Express, para el Backend, Html y Css para el Frontend y por último Json para guardar en base de datos.

1. Softwares

1.1 Node.js: Es un entorno de ejecución de JavaScript basado en eventos. Esta diseñado para construir aplicaciones web o de red, que se puede ejecutar del lado del servidor.

1.2 Express: Es un framework de Node.js flexible que proporciona un conjunto solido de funciones y librerías para aplicaciones web y móviles.

1.3 Json: JavaScript Object Notation, es un formato estándar para representar datos estructurados en la sintaxis de JavaScript.

1.4 JavaScript: JavaScript es el lenguaje de programación que se suele usar para añadir características especiales a los sitios web, como interactividad entre el usuario y el portal.

1.5 TypeScript: Es un super conjunto de JavaScript que agrega sintaxis adicionales y se caracteriza por usar tipado estático.

1.6 Html: Es un lenguaje de etiqueta que se usa para el desarrollo web, con este puedes definir la estructura y organizar el contenido de manera que los navegadores lo puedan mostrar e interpretar correctamente.

1.7 Css: Se encarga de dar estilos a los sitios web, como colores, fuentes, márgenes, disposición. Para que las páginas puedan ser entendibles y llamativas.

1.7 Visual Studio Code: Es un editor de código gratuito para programadores, de código abierto y multiplataforma. Se interfaz es muy reconocible, posee muchas extensiones y permite trabajar con varios tipos de lenguajes de programación.

2. Ejercicio 1

Solicitar el nombre de un animal que sea su favorito y mediante *NODE,* intercepta y recarga otra página html que muestra dicho nombre, requiere dos vistas.

2.1 Herramientas y Entorno:

2.1.1 Node.js: Se utilizo como framework principal, con este se puede gestionar el Backend de todo el proyecto en sí.

2.1.2 Express: Proporciona un conjunto de librerías que facilitan el entorno para manejar el backen en conjunto a node.

2.1.3 JavaScript: Se escogió para dar vida al sitio web y se vea más interactivo.

2.1.4 Html: Es la estructura principal de cada sitio web, para una tarea como esta es lo indicado.

2.1.5 Css: Son necesarios los estilos para cada sitio web, con esto cobra vida y hace que sea más entendible para el usuario.

2.1.6: Vs Code: Es la herramienta adecuada para gestionar todos los lenguajes de programación y frameworks, ya que cuenta con una amplia librería.

2.2 Estructura del Proyecto:

2.2.1 Ejercicio 1

Se realizo escogiendo cada uno de los frameworks a trabajar, se organizó en carpetas cada una con contenidos específicos. Css, para cada hoja de estilo, db para almacenar la base de datos Json, html en donde se encuentran los archivos para los ejercicios 1 y 2, js se encuentran los archivos de JavaScript y TypeScript.

Luego en node_mudules, se encuentran instaladas todas las librerías de Express que son necesarias para que Node y Express puedan trabajar en conjunto. Tenemos index.html que es la página principal del sitio web “Imagen 2”. Index.js es el centro del proyecto, donde se encuentra el Backend, por último, tenemos archivos. json donde se almacena información referente al proyecto, como la versión y configuraciones que se hacen al iniciar por primera vez.

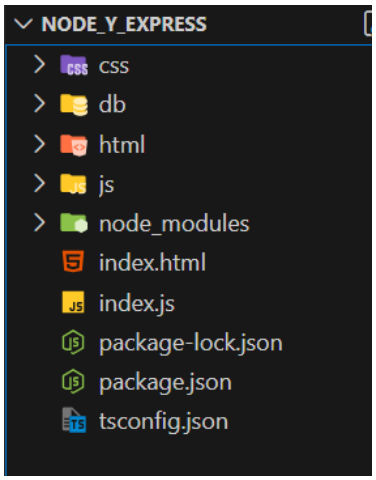

Imagen 1. Estructura del Proyecto en Vs Code

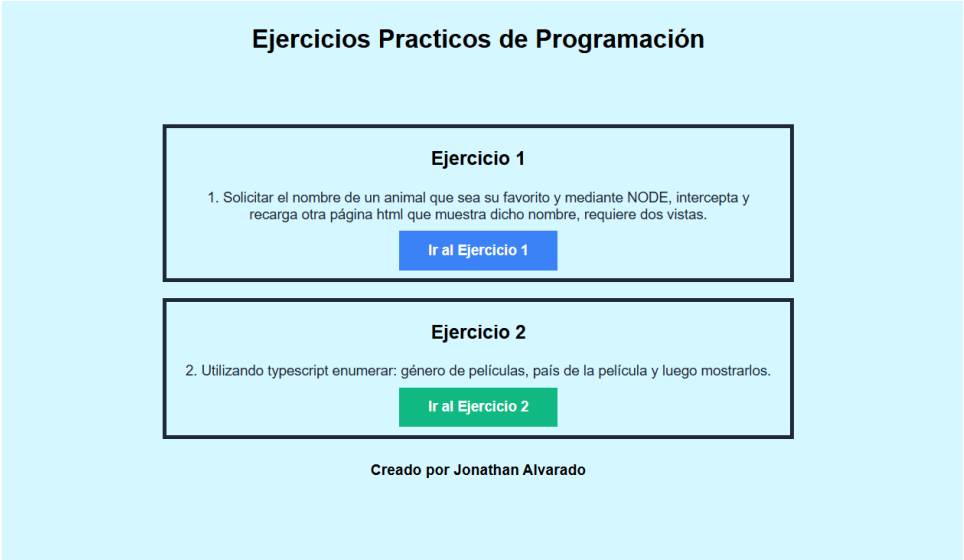

Imagen 2. Vista principal, Index.html

Este es el primer archivo que se muestra al ejecutar el servidor, es la página principal, donde se muestra el título, dos cards donde la primera es para el ejercicio 1 y la otra segunda para el ejercicio 2, cada uno cuenta con un botón que permite ir a la actividad.

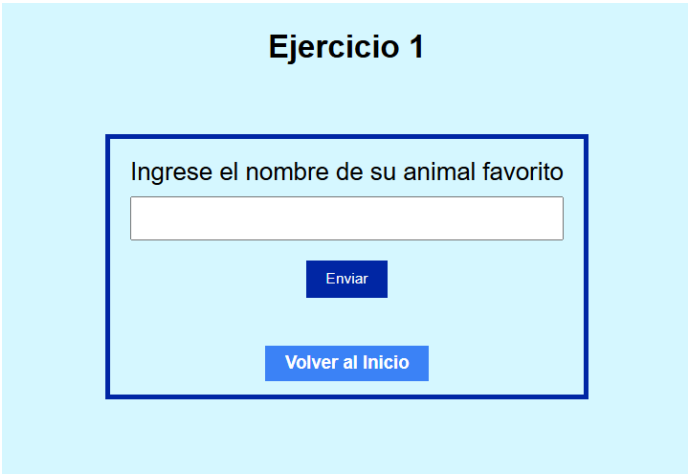

Imagen 3. Ejercicio 1

En este se puede apreciar un recuadro que contiene un texto, formulario y dos botones, uno para enviar con la función submit y el otro para volver a la página principal. La actividad indica que se debe escribir el nombre de un animal y este se debe capturar por node y mostrar en pantalla.

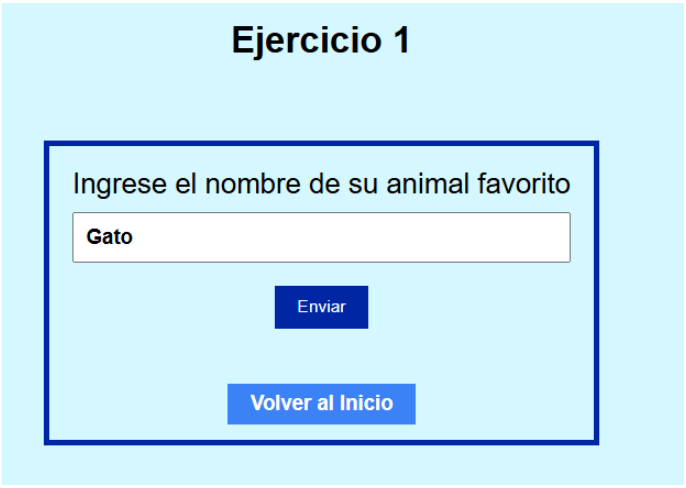

Imagen 4. Agregando datos

Imagen 5. Mostrando datos en pantalla

El programa cumple con la actividad. Se uso JavaScript en conjunto con Html y Css para crear un modal y que los botones sean interactivos para dar dinamismo a la aplicación.

2.2.2 Ejercicio 2

Para el segundo ejercicio se debe utilizar TypeScript con la característica enum para mostrar en pantalla: genero de películas y país, pero se llevó un poco más allá y se agregaron id y titulo. Esto con el fin de mostrar en pantalla una lista de películas que se deseen guardar, siendo esto posible ya que se usó una base de datos con Json para almacenar información.

Imagen 6. Ejercicio 2

Se puede apreciar el titulo principal, así como el formulario para agregar título, una lista para el genero y otra para el país, además, de un botón “Guardar Película”. Más abajo se encuentra el botón “Volver al Inicio” “Imagen 2 como referencia” y por último esta la tabla, donde se muestran todas las películas almacenadas.

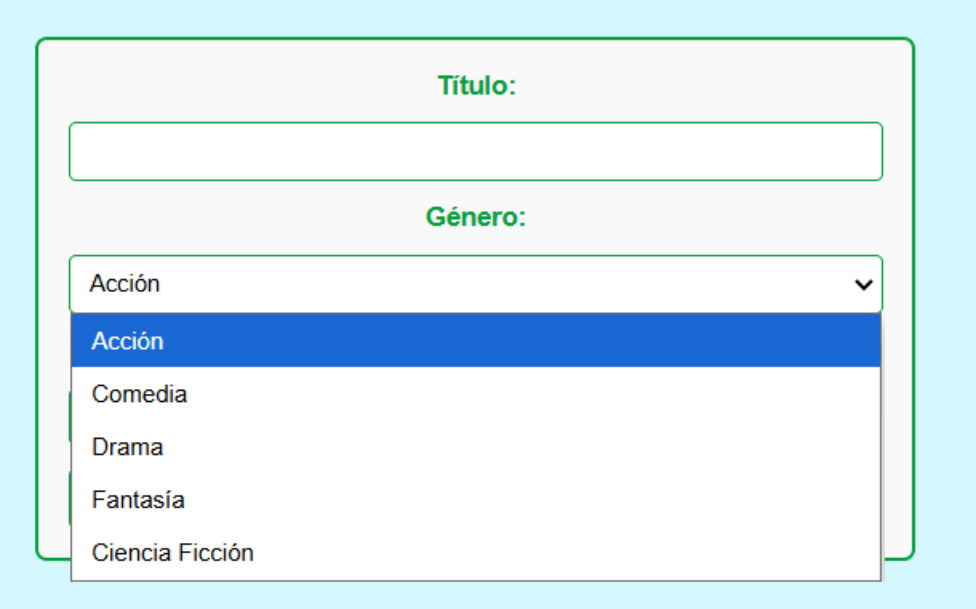

Imagen 7. Lista de Genero

Imagen 8. Lista de Países

ID	Título	País	Género
1	Origen	Estados Unidos	Ciencia Ficción
2	Amélie	Francia	Comedia
3	duro de matar	Estados Unidos	Acción
4	harry potter	Venezuela	Fantasía

Imagen 9. Tabla, se muestran las películas en pantalla

2.3 Código:

2.3.1 Index.html: En este documento se encuentra toda la estructura general de la página principal. Está conformado por un head donde se guardan todos los valores predefinidos de html:5 así como las rutas para las etiquetas y el titulo que se muestra en la barra de navegación.

Se compone de un body, donde se aloja el header, este se usa con h1 para colocar todos los títulos en los tres documentos html. También cuenta con un main, este es el apartado principal de la estructura del body aquí se alojan en el section las cards 1 y 2 especificando el contenido del ejercicio, cuenta con un título “h2 y p” para el párrafo. Y por último un div, donde se encuentra el button y el “a href” para ir al ejercicio uno o dos dependiendo de la card. Y por último un h3 donde se encuentra el autor de dicha actividad.

```
<!--Titulo-->
<header>
  <h1 class="title">Ejercicios Practicos de Programación</h1>
</header>
<!--Contenido-->
<main>
  <section>
    <!--Card 1 para Ejercicio 1-->
    <div class="card">
      <div class="content-1">
        <h2>Ejercicio 1</h2>
        <p>
          1. Solicitar el nombre de un animal que sea su favorito y mediante
          NODE, intercepta y recarga otra página html que muestra dicho
          nombre, requiere dos vistas.
        </p>
      </div>
      <div class="footer">
        <button class="btn-1">
          <a href="/html/ejercicio_1.html">Ir al Ejercicio 1</a>
        </button>
      </div>
    </div>
```

Imagen 10. Parte 1 index.html

```
<!--Card 2 para Ejercicio 2-->
<div class="card">
    <div class="content-2">
        <h2>Ejercicio 2</h2>
        <p>
        2. Utilizando typescript enumerar: género de películas, país de la
        película y luego mostrarlos.
        </p>
    </div>
    <div class="footer">
        <button class="btn-2">
  <a href="/html/ejercicio_2.html">Ir al Ejercicio 2</a>
</button>
    </div>
</div>
<h3>Creado por Jonathan Alvarado</h3>
</section>
```

Imagen 11. Parte 2 Index.html

2.3.2 Css: Este documento cuenta con todos los estilos, ya sean root para almacenar las paletas de colores, body, en este se guardan características generales como el fondo. Header, se guardan los estilos aplicados el título y h1, en el main se justifica, alinea y distribuye los espacios para los elementos que se encuentran en él.

Elementos como h2, h3 y p disponen de propiedades similares como el espaciado interno. Y Luego, tenemos las card siendo dos que comparten las mismas propiedades de distribución, bordes y un máximo de ancho; los btn cumplen con funciones similares solo que cada uno tiene un enlace a paginas diferentes, por eso se le coloca un color único para identificar.

```
* {
    /*Estilo global*/
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: Arial, Helvetica, sans-serif;
}

:root {
    /*Paleta de colores*/
    --color-text-primary: #1f2936;
    --color-text-secondary: #ffffff;
    --color-text-placeholder: #798eae;
    --color-bg-primary: #d5f7ff;
    --color-bg-btn-1: #3b82f6;
    --color-bg-btn-2: #10b981;
    --color-bg-btn-1-hover: #2563eb;
    --color-bg-btn-2-hover: #059669;
}
```

```
/*Color del Fondo*/
body {
    background-color: var(--color-bg-primary);

}

/*Titulo*/
header {
    margin-top: 50px;
    margin-bottom: 50px;
    text-align: center;
}

/*Apartado Principal*/
main {
    display: flex;
    align-items: center;
```

Imagen 12. Css

2.3.3 Ejercicio_1.html: Aquí se aloja el primer programa de la actividad propuesta. Este se conforma por: Html, Css, JavaScript, Node.js, Express. El Html cumple con la función de crear un formulario donde se ingrese un dato y luego sea capturado por Node y Express para ser almacenado temporalmente y mostrarlo en pantalla. También cuenta con un apartado para el modal, que se encuentra previamente desactivado, hasta que se completen los requisitos que son agregar un dato al input y se presione enviar. Css cumple con la función de dar interactividad con el entorno además de desactivar temporalmente al modal.

Este html se compone de tres partes logrando que se cumpla con la actividad. Primera parte el formulario, este comienza desde el div “form-conten”, label es el texto que indicando al usuario el dato a ingresar en el form, los datos ingresan por input y se envían al sistema usando button del tipo submit.

Segunda parte “botton” el usuario al presionar regresara a la pagina principal, index.html. Y la ultima parte el modal, este se compone de un contenedor que guarda un span que se usa para asignar el botón cerrar “X”, un h2 donde indica en pantalla el dato a mostrar y luego “p” aquí se muestra el dato o valor que se guarda desde el formulario en sistema y lo muestra en pantalla. Además, se puede apreciar el Script que llama al documento js que almacena las funciones del modal y este pueda funcionar.

```
<body>
  <header>
    <h1>Ejercicio 1</h1>
  </header>
  <main>
    <!--Formulario:Escribe un nombre de un animal para guardarlo y enviarlo al modal-->
    <div class="form-content">
      <label for="animal">Ingrese el nombre de su animal favorito</label>
      <form action="/animal" method="POST" id="form">
        <input type="text" id="animal" name="animal" required />
        <button type="submit" class="modalbtn" id="abrirModalBtn">
          Enviar
        </button>
      </form>
      <!--Boton Volver-->
      <botton class="backbtn">
        <a href="/index.html">Volver al Inicio</a>
      </botton>
    </div>

    <!--Modal: muestra el nombre de un animal en pantalla-->
    <div class="modal" id="modal">
      <div class="modal-content">
        <span class="cerrar" id="cerrarModalBtn">X</span>
        <h2>Tu animal favorito es:</h2>
        <p></p>
      </div>
    </div>
  </main>
  <!--Script Modal y Captura-->
  <script src="/js/script.js"></script>
</body>
```

Imagen 13. Ejercicio_1.html

2.3.4 Script.js: Este documento se encarga de gestionar el modal, abrir o cerrar de pendiendo de que condiciones se estén cumpliendo para activar las funciones. Dichas funciones se componen por constantes, variables y métodos. Tenemos una lista amplia de const como el de formulario, modal, cerrar el botón modal y mostrar texto en el modal usando “p”.

La primera función esta asignada al formulario, esta se conecta al servidor con node y express usando fetch y guardando en data Json para captura y enviarla de regreso para mostrar los resultados en pantalla. Luego tenemos las abrir modal, ocultar so se hace click en la pantalla y por último cerrar el modal usando un botón.

```
// Esperar a que el DOM esté cargado
document.addEventListener('DOMContentLoaded', () => {
  const form = document.getElementById('form');
  const modal = document.getElementById('modal');
  const cerrarModalBtn = document.getElementById('cerrarModalBtn');
  const modalText = modal.querySelector('p');

  // Capturar envío del formulario
  form.addEventListener('submit', async (e) => {
    e.preventDefault(); // evitar recarga

    const formData = new FormData(form);
    const response = await fetch('/animal', {
      method: 'POST',
      body: new URLSearchParams(formData)
    });

    const data = await response.json();
    // mostrar el animal en el modal
    modalText.textContent = data.animal;

    // Abrir modal
   abrirModalBtn.addEventListener("click", function() {
    modal.style.display = "flex";
})
  });
  // ocultar el modal al hacer clic fuera del contenido
 modal.addEventListener('click', function(e) {
 if (e.target === modal) {
   modal.style.display = 'none';
 }
});

  // Cerrar modal
  cerrarModalBtn.addEventListener('click', () => {
    modal.style.display = 'none';
  });
});
```

Imagen 14. Script.js

2.3.5 index.js: Este es el archivo principal para node.js aquí se encuentran las conexiones para express, asi podremos manejar los servidores. Con path se manejan las ruras y fs trabaja con archivos. El puerto del localhost para comprobar si el servidor presenta o no problemas, responde desde el puerto 3000 de manera satisfactoria. Las apps son para comprobar los datos estáticos, del formulario y del tipo Json.

```
// Importar Express al proyecto para habilitar el servidor local
const express = require('express')
const path = require('path')
const fs = require('fs')
const app = express()
const port = process.env.PORT || 3000

// Middleware para servir archivos estáticos y parsear cuerpos de solicitudes
app.use(express.static(path.join(__dirname)))
app.use(express.urlencoded({ extended: true }))
app.use(express.json())

// Servir index.html correctamente
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'))
})

// Endpoint simple para /animal (usado por ejercicio_1)
app.post('/animal', (req, res) => {
  const animal = req.body && req.body.animal ? req.body.animal : ''
  res.json({ animal })
})
```

Imagen 15. Index.js

2.3.6 ejercicio_2.html: Este archivo contiene la estructura que se uso para la realización del ejercicio 2. Comenzando por el formulario, esta toma datos importantes como el título de la película que pasa por un input, luego tenemos las selecciones o listas, que son para país y géneros de las películas, son selecciones simples y luego tenemos un botón tipo “submit” para enviar la información tomada al servidor, este la lee y la guarda en un archivo Json, este cumple la función de una base de datos.

Luego que tenemos la información en data, este va directamente a la tabla, que tiene 4 campos, id, titulo, país y género. Cada vez que se gurde una nueva información se mostrara en una nueva posición fácil de identificar. Referencias, imagen 7,8 y 9. Además de que también cuenta con un botón para volver al inicio y pasar a la siguiente actividad si así se desea.

```html
<body>
  <header><h1>Ejercicio 2</h1></header>
  <main>
    <!-- Formulario para agregar película -->
    <form id="formPelicula">
      <label for="titulo">Título:</label>
      <input type="text" id="titulo" name="titulo" required />

      <!--Seleccion en lista para el genero-->
      <label for="genero">Género:</label>
      <select id="genero" name="genero" required>
        <option value="Acción">Acción</option>
        <option value="Comedia">Comedia</option>
        <option value="Drama">Drama</option>
        <option value="Fantasía">Fantasía</option>
        <option value="Ciencia Ficción">Ciencia Ficción</option>
      </select>
      <!--Seleccion en lista para el el pais-->
      <label for="pais">País:</label>
      <select id="pais" name="pais" required>
        <option value="Venezuela">Venezuela</option>
        <option value="Estados Unidos">Estados Unidos</option>
        <option value="España">España</option>
        <option value="Japón">Japón</option>
        <option value="Francia">Francia</option>
        <option value="Gran Bretaña">Gran Bretaña</option>
      </select>
      <!--Boton para guardar la pelicula-->
      <button type="submit">Guardar Película</button>
    </form>
```

Imagen 16. Ejercicio_2.html

2.3.8 pelicuas.ts: En este archivo se guardan los enum de TypeScript, todos los valores agrupados pertenecen a los mismos de la selección del formulario en html, ya que el propósito es representar lista de opciones para así evitar errores y hacer que sea más legible el código. Luego tenemos las funciones, la primera es para mostrar automáticamente los datos a la tabla, esto es posible gracias a la función de cargar películas, esta guarda toda la información que viene del Backend. Y por último la función de guardar nuevas películas, esto es lo que hace es recopilar la información en datos específicos que se muestran en pantalla luego ser capturados.

```typescript
enum Genero {
  Accion = "Acción",
  Comedia = "Comedia",
  Drama = "Drama",
  Fantasia = "Fantasía",
  CienciaFiccion = "Ciencia Ficción"
}

enum Pais {
  Venezuela = "Venezuela",
  EstadosUnidos = "Estados Unidos",
  España = "España",
  Japon = "Japón",
  Francia = "Francia",
  GranBretaña = "Gran Bretaña"
}

interface Pelicula {
  id?: number;
  titulo: string;
  genero: Genero;
  pais: Pais;
}
```

Imagen 17. Película.js TypeScript enum

```typescript
// Función para renderizar películas en la tabla
function renderPeliculas(peliculas: Pelicula[]) {
  const tbody = document.querySelector("#peliculas tbody") as HTMLTableSectionElement;
  tbody.innerHTML = "";
  peliculas.forEach(p => {
    const fila = document.createElement("tr");
    fila.innerHTML = `
      <td>${p.id}</td>
      <td>${p.titulo}</td>
      <td>${p.pais}</td>
      <td>${p.genero}</td>
    `;
    tbody.appendChild(fila);
  });
}

// Cargar películas desde el backend
async function cargarPeliculas() {
  const response = await fetch("/peliculas");
  const peliculas: Pelicula[] = await response.json();
  renderPeliculas(peliculas);
}
```

Imagen 18. Funciones render y cargar

```
// Guardar nueva película
document.addEventListener("DOMContentLoaded", () => {
  cargarPeliculas();

  const form = document.getElementById("formPelicula") as HTMLFormElement;
  form.addEventListener("submit", async e => {
    e.preventDefault();
    const titulo = (document.getElementById("titulo") as HTMLInputElement).value;
    const genero = (document.getElementById("genero") as HTMLSelectElement).value as Genero;
    const pais = (document.getElementById("pais") as HTMLSelectElement).value as Pais;

    await fetch("/peliculas", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ titulo, genero, pais })
    });

    form.reset();
    cargarPeliculas();
  });
});
```

Imagen 19. Funciones Guardar

2.3.9 Index.js: Para este ejercicio se complementó este archivo agregando nuevas funcionalidades, que se encuentran ligadas, el TypeSctipt y Html. Ya que desde aquí parte toda la lógica del Backend o de servidores locales. En este caso las funciones son muy puntuales, es cargar una nueva película desde el archivo Json que se verán en la tabla que esta ubicada en el archivo Html. Luego tenemos las funciones de valor por defecto, estas están ligadas al archivo json, mostrando la estructura, id, titulo, país y género. Y por último tenemos la función de guardar las películas en formato Json, en caso de exista un error pasa una alerta.

```
// Persistencia ligera: archivo JSON en /db/peliculas.json
const peliculasFile = path.join(__dirname, 'db', 'peliculas.json')

// Cargar películas desde el archivo JSON
function loadPeliculas() {
  try {
    if (fs.existsSync(peliculasFile)) {
      const raw = fs.readFileSync(peliculasFile, 'utf8')
      const data = JSON.parse(raw)
      if (Array.isArray(data)) return data
    }
  } catch (err) {
    console.error('Error leyendo peliculas.json:', err)
  }
  // valor por defecto si no existe o falla la lectura
  return [
    { id: 1, titulo: 'Origen', genero: 'Ciencia Ficción', pais: 'Estados Unidos' },
    { id: 2, titulo: 'Amélie', genero: 'Comedia', pais: 'Francia' }
  ]
}

// Guardar películas en el archivo JSON
function savePeliculas(list) {
  try {
    const dir = path.dirname(peliculasFile)
    if (!fs.existsSync(dir)) fs.mkdirSync(dir, { recursive: true })
    fs.writeFileSync(peliculasFile, JSON.stringify(list, null, 2), 'utf8')
  } catch (err) {
    console.error('Error guardando peliculas.json:', err)
  }
}
```

Imagen 20. Funciones de index,js

Conclusión

Este fue un trabajo que se logró completar e implementar, tuvo su taza de dificultad ya que se carece de ciertos conocimientos y experiencias con los frameworks, pero a la vez se disfruto del logro de culminar cada ejercicio. Cada uno tuvo su área de dificultad. En el ejercicio 1 el mayor reto fue comprender la lógica para mostrar en pantalla el nombre. Y en el ejercicio 2 fue implementar la base de datos y que muestre el funcionamiento del servidor.

Cada practica tiene un gran margen de mejora, lo ideal era hacer el uso puntual del código, para optimizar los procesos y no tener código espagueti. Ya que express cuenta con una gran librería y una cantidad voluminosa de datos, extender el código complicaría una tarea tan puntual y especifica.

Se cumplió con la meta de usar Node.Js se capturaron los datos y se mostraron en pantalla de la manera adecuada. Express no presento problemas desde su instalación y los módulos no afectaron el área de trabajo. El uso de Html fue el adecuado, se agregó Css y JavaScript para dar estilos y funcionalidad a las páginas. Y el uso de Typescript fue el indicado solo que se agregaron unos parámetros extras para complementar el trabajo y así culminar con grandes logros.

Bibliografías

Node.js. (s.f.). About Node.js. Recuperado de https://nodejs.org/es/about

Express.js. (s.f.). Welcome to Express. Recuperado de https://expressjs.com/

García de Zúñiga, F. G. (2025, 17 de julio). ¿Qué es Visual Studio Code y cuáles son sus ventajas? Arsys. Recuperado de https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas

W3Schools. (s.f.). How to Create Modal Boxes. Recuperado de https://www.w3schools.com/howto/howto_css_modals.asp

W3Schools. (s.f.). JavaScript Form Validation. Recuperado de https://www.w3schools.com/js/js_validation.asp

Mozilla Developer Network. (s.f.). HTML. https://developer.mozilla.org/es/docs/Web/HTML

Jain, S. (s.f.). Manejo de formularios del lado del cliente con JavaScript explicado con código de ejemplo. FreeCodeCamp. Recuperado de https://www.freecodecamp.org/espanol/news/manejo-de-formularios-del-lado-del-cliente-con-javascript-explicado-con-codigo-de-ejemplo/

Anexos

Jonathan Alvarado (2025). Proyecto ejemplo [Repositorio Github]. Github. https://github.com/Jonathan-5367/node_y_express.git

Para correr este programa se debe iniciar “npm install” y Luego “npm start”.