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: 7A

Materia: Programación III (M1)

Ejercicios 1 Y 2

Docente:

Ing. Carlos Márquez

Estudiantes:

Juan Echenique

V- 30.829.758

Caracas, 12 de diciembre de 2025

INTRODUCCIÓN

En el presente documento se expone el proceso de elaboración de los ejercicios sobre el lenguaje de programación TypeScript, y Node.js con el objetivo de facilitar su comprensión y destacar sus principales características, brindado al desarrollador las practicas mínimas necesarias en estas tecnologías y aplicaciones dentro del desarrollo web moderno.

Documentación Paso a Paso: Aplicación de Animales Favoritos

Desarrollado por: Juan Luis Echenique Gonzalez

Este documento detalla el funcionamiento, estructura y configuración del proyecto "Ejercicio 1-2 Programación III".

1. Descripción del Proyecto

Esta es una aplicación web construida con **Node.js** y **TypeScript** que permite a los usuarios:

Insertar Un nombre de un animal u objeto captarlo y mostrarlo en la siguiente pagina

Tecnologías Utilizadas

- Backend: Node.js + Express
- Lenguaje: TypeScript (Tipado estático)
- Vistas: EJS (Motor de plantillas HTML)
- Estilos: CSS (Archivos estáticos en `public`)

2. Requisitos Previos

Para ejecutar este proyecto necesitas tener instalado:

- [Node.js](https://nodejs.org/) (versión 16 o superior recomendada).

```
C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII> node install

node:internal/modules/cjs/loader:1424
  throw err;
  ^

Error: Cannot find module 'C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-Pr
ogramacionIII\install'
    at Module._resolveFilename (node:internal/modules/cjs/loader:1421:15)
    at defaultResolveImpl (node:internal/modules/cjs/loader:1059:19)
    at resolveForCJSWithHooks (node:internal/modules/cjs/loader:1064:22)
    at Module._load (node:internal/modules/cjs/loader:1227:37)
    at TracingChannel.traceSync (node:diagnostics_channel:328:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:245:24)
    at Module.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:154:5)
    at node:internal/main/run_main_module:33:47 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v24.12.0

C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII>
```

Imagen 1. Instalación de Node.js

3. Instalación y Configuración

Sigue estos pasos para preparar el entorno:

Paso 1: Instalar dependencias

Abre una terminal en la carpeta raíz del proyecto y ejecuta:

```npm install```

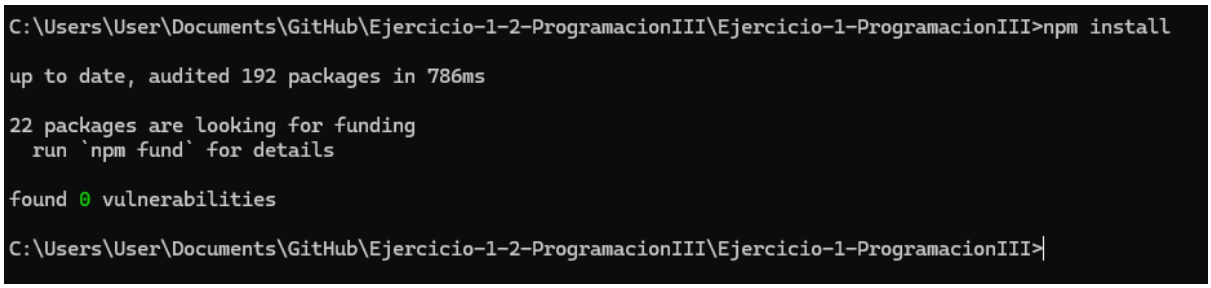

Imagen 2. Instalación de npm

Esto descargará todas las librerías listadas en “package.json” (como express, ejs, typescript, etc.) dentro de la carpeta “node_modules”.

Paso 2: Scripts Disponibles

El archivo package.json define los siguientes comandos útiles:

- Modo Desarrollo: “npm run dev”
    - Usa “nodemon” para reiniciar el servidor automáticamente cuando haces cambios.
- Construir (Compilar): “npm run build”
    - Compila el código TypeScript a JavaScript en la carpeta “dist”.
    - Copia las vistas y archivos estáticos a “dist”.
- Limpiar: “npm run clean”
```

- Borra la carpeta “dist” para una compilación limpia.

- Producción: “npm start”

- Ejecuta el código compilado desde “dist”.

```json
{
  "name": "ejercicio-1-2-programacioniii",
  "version": "1.0.0",
  "description": "Aplicación web para mostrar animales favoritos usando Node.js, Express, TypeScript y EJS",
  "main": "dist/app.js",
  ▷Depurar
  "scripts": {
    "dev": "nodemon src/app.ts",
    "build": "tsc && copyfiles -u 1 src/views/**/* src/public/**/* dist",
    "start": "node dist/app.js",
    "clean": "rm -rf dist",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "express",
    "typescript",
    "ejs",
    "programacioniii"
  ],
  "author": "Tu Nombre",
  "license": "MIT",
  "dependencies": {
    "ejs": "^3.1.9",
    "express": "^4.18.0"
  },
  "devDependencies": {
    "@types/ejs": "^3.1.2",
    "@types/express": "^4.17.0",
    "@types/node": "^20.0.0",
    "copyfiles": "^2.4.1",
    "nodemon": "^3.1.11",
    "ts-node": "^10.9.2",
    "typescript": "^5.0.0"
  }
}
```

Imagen 3. package.json

4. Estructura del Proyecto

La estructura de directorios está organizada de la siguiente manera:

- “/src”: Código fuente TypeScript.
  - “/controllers”: Lógica de negocio.
    - “AnimalController.ts”: Gestiona los datos de animales y las respuestas a las peticiones (Renderizado y API).
  - “/models”: (Actualmente vacío, la lógica de datos se simplificó en el Controlador).
  - “/types”: Definiciones de tipos TypeScript.

        - “index.ts”: Interfaces (`IAnimal`), Enums (`AnimalCategory`), y otros tipos.
    - “/views”: Plantillas HTML con EJS.
        - “index.ejs”, “animal-info.ejs”, “error.ejs”.
    - “app.ts”: Punto de entrada principal. Configura Express y el servidor.
    - “/public”: Archivos estáticos (CSS, Imágenes, JS del cliente).
- “/dist”: Código compilado (JavaScript) listo para producción.
- “package.json”: Configuración de dependencias y scripts.
- “tsconfig.json”: Configuración del compilador TypeScript.

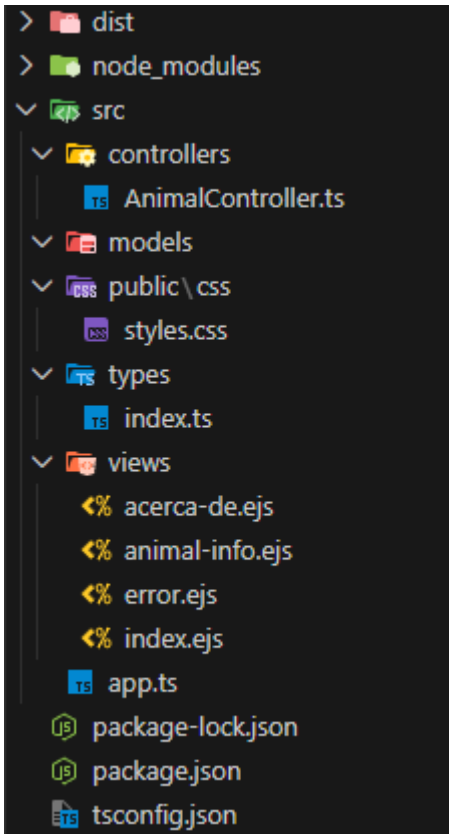

Imagen 4. Estructura

5. Explicación del Código (Paso a Paso)

Paso A: Definición de Tipos (“src/types/index.ts”)

Aquí definimos la estructura de nuestros datos para asegurar que siempre tengan el formato correcto.

- **`interface IAnimal`**: Define qué propiedades *debe* tener un animal

```
// INTERFACE - Datos del formulario
export interface IAnimalFormData {
  nombreAnimal: string;
  emailUsuario?: string;
  comentario?: string;
}
```

Imagen 5. Interfas

- “interface IApiResponse”: Estandariza cómo responde nuestra API (success, message, data).

```
// INTERFACE - Respuesta de la API
export interface IApiResponse<T = any> {
  success: boolean;
  message: string;
  data?: T;
  timestamp: Date;
  code: number;
}
```

Imagen 6 interface IApiResponse

6. Ejecución del Proyecto

1. En la terminal, ejecuta:

   Npm run build

```
C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII>npm run build

> ejercicio-1-2-programacioniii@1.0.0 build
> tsc && copyfiles -u 1 src/views/**/* src/public/**/* dist
```

Imagen 7 Compilacion de Proyecto

2. Ejecute el siguiente comando:

Npm start

```
C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII>npm start

> ejercicio-1-2-programacioniii@1.0.0 start
> node dist/app.js

============================================================
🚀 Ejercicio 1-2 - Programación III v1.0.0
============================================================
✅ Servidor corriendo en: http://localhost:3000
📁 Entorno: development
📁 Directorio de vistas: C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-Prog
ramacionIII\dist\views
📁 Archivos estáticos: C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-Progra
macionIII\dist\public
============================================================
📋 Rutas disponibles:
   • GET  /              - Página principal
   • POST /buscar-animal - Buscar animal
   • GET  /animal/:nombre - Ver animal específico
   • GET  /acerca-de     - Información del proyecto
============================================================
🧘 Presiona Ctrl+C para detener el servidor
============================================================
```

Imagen 8 Servidor ejecutado e iniciado

3. Abre tu navegador web y ve a `http://localhost:3000`.

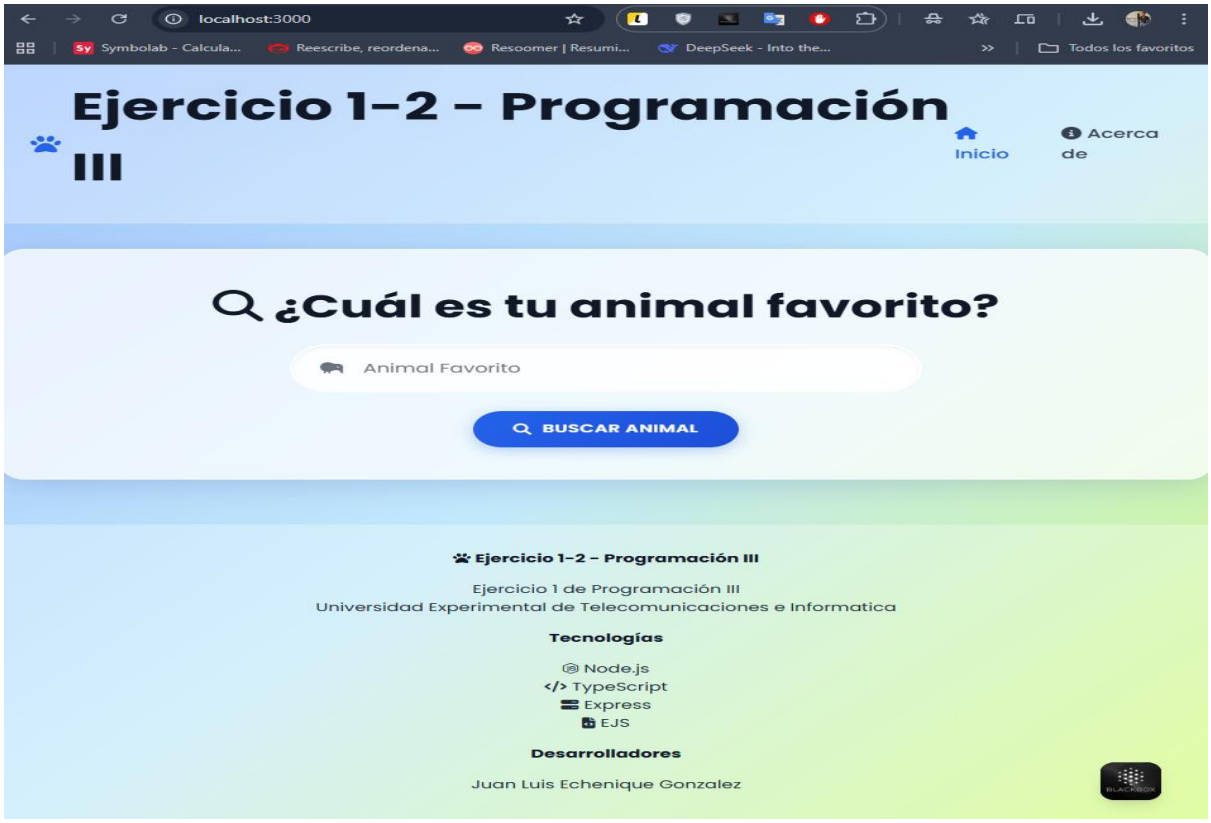

Imagen 9 Proyecto en ejecución

Documentación Paso a Paso: Aplicación de Películas Favoritos

1. Descripción del Proyecto

Este proyecto es una aplicación web sencilla desarrollada para demostrar el uso de TypeScript, específicamente el uso de **Enumeraciones (Enums), tipado estático y manipulación del DOM.

La aplicación permite al usuario:

1. Ingresar el nombre de una película.
2. Seleccionar un Género de una lista predefinida (Enum).
3. Seleccionar un País de una lista predefinida (Enum).
4. Agregar la película a una lista visual en pantalla.

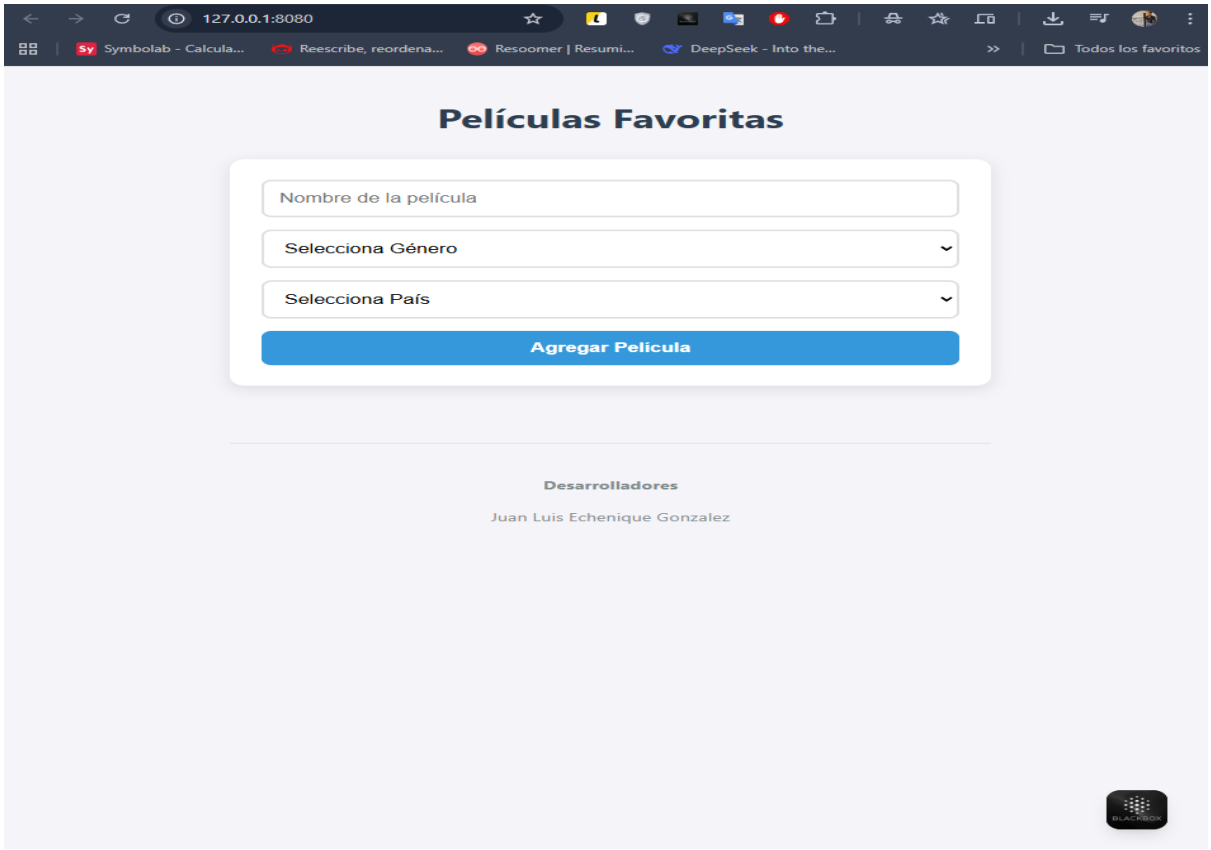

Imagen 10 Funciones del Proyecto 2

2. Tecnologías Utilizadas

* **HTML5**: Estructura de la página.

* **CSS3**: Estilos y diseño responsivo (Flexbox/Grid).

* **TypeScript**: Lógica de programación, tipos y seguridad.

* **Node.js & NPM**: Gestión de dependencias y automatización de tareas.

3. Estructura del Proyecto

Ejercicio-2-ProgramacionIII/

├── dist/ # Código JavaScript generado (compilado)

│ └── app.js

├── src/ # Código Fuente TypeScript

│ └── app.ts

├── index.html # Interfaz principal

├── style.css # Estilos visuales

├── package.json # Configuración de dependencias y scripts

└── tsconfig.json # Configuración del compilador TypeScript

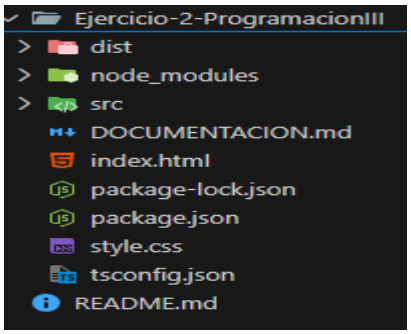

Imagen 11 Estructura del Proyecto 2

4. Detalle de Archivos

4.1. Configuración (“tsconfig.json”)

Este archivo configura cómo TypeScript transforma el código a JavaScript comprensible por el navegador.

* target: ES2017 (Para soportar “Object.values”).

* outDir: ./dist (Carpeta de salida).

* strict: true (Modo estricto para evitar errores).

Imagen 12 Archivo tsconfig.json

4.2. Interfaz (`index.html`)

Contiene:

* Un formulario con inputs y selects.

* Un contenedor vacío (`#movieList`) donde se inyectan las películas.

* Referencia al script compilado (`./dist/app.js`).

```html
<body>
    <div class="container">
        <h1>Películas Favoritas</h1>
        <div class="form-section">
            <input type="text" id="movieName" placeholder="Nombre de la película" class="input-field">

            <select id="genreSelect" class="input-field">
                <option value="">Selecciona Género</option>
            </select>

            <select id="countrySelect" class="input-field">
                <option value="">Selecciona País</option>
            </select>

            <button id="addBtn" class="btn">Agregar Película</button>
        </div>

        <div id="movieList" class="movie-list">
            <!-- Las películas añadidas aparecerán aquí -->
        </div>

        <footer>
            <h4>Desarrolladores</h4>
            <p>Juan Luis Echenique Gonzalez</p>
        </footer>
    </div>
    <script src="./dist/app.js"></script>
</body>
```

Imagen 13 Archivo index.html

4.4. Lógica (`src/app.ts`)

Es el corazón de la aplicación.

Enums: Define `Genre` y `Country` para restringir los valores posibles.

```typescript
// 1. DEFINICIÓN DE ENUMS
// Los enums nos permiten definir un conjunto de constantes con nombre.
// Aquí definimos los Géneros permitidos para las películas.
enum Genre {
    Action = "Acción",
    Drama = "Aventura",
    Comedy = "Comedia",
    Horror = "Suspenso",
    SciFi = "Ciencia Ficción",
}

// Aquí definimos los Países permitidos.
enum Country {
    USA = "Estados Unidos",
    UK = "Alemania",
    France = "Francia",
    Japan = "Japón",
    Spain = "España",
    Brazil = "Mexico"
}
```

Imagen 14 Comando enums

Clase Movie: Molde para crear objetos de tipo película.

```
// Usamos una clase para definir la estructura de lo que es una "Película" en nuestro sistema.
// El constructor utiliza la sintaxis abreviada de TypeScript para definir e inicializar propiedades a la vez.
class Movie {
    constructor(
        public title: string,   // Nombre de la película
        public genre: Genre,    // Debe ser uno de los valores del Enum Genre
        public country: Country // Debe ser uno de los valores del Enum Country
    ) { }
}
```

Imagen 15 Comando de Clase Movie

Funciones:

`initOptions()`: Genera dinámicamente las opciones del menú <select> basándose en los Enums.

```
/**
 * Inicializa las opciones de los menús desplegables (selects)
 * Recorre los valores de los Enums y crea etiquetas <option> para el HTML.
 */
function initOptions() {
    // Llenar el select de Géneros
    // Object.values(Genre) nos devuelve un array con los valores: ["Acción", "Aventura", ...]
    Object.values(Genre).forEach((genre: string) => {
        const option = document.createElement('option');
        option.value = genre;
        option.textContent = genre;
        genreSelect.appendChild(option);
    });

    // Llenar el select de Países
    Object.values(Country).forEach((country: string) => {
        const option = document.createElement('option');
        option.value = country;
        option.textContent = country;
        countrySelect.appendChild(option);
    });
}
```

Imagen 16 Comando de funciones

`addMovie()`: Captura los datos, valida que no estén vacíos y crea el objeto.

```
/**
 * Función que se ejecuta al hacer clic en el botón "Agregar Película".
 * Captura los datos, valida y crea el objeto.
 */
function addMovie() {
    // Capturamos los valores actuales del formulario
    const title = movieNameInput.value;
    const genre = genreSelect.value as Genre;     // Forzamos el tipo a Genre
    const country = countrySelect.value as Country; // Forzamos el tipo a Country

    // Validación simple: verificar que no haya campos vacíos
    if (title.trim() === '' || genreSelect.value === '' || countrySelect.value === '') {
        alert('Por favor completa todos los campos');
        return; // Detenemos la ejecución si hay error
    }

    // Instanciamos un nuevo objeto Película
    const newMovie = new Movie(title, genre, country);

    // Llamamos a la función encargada de pintar en pantalla
    renderMovie(newMovie);

    // Limpiamos el formulario para la siguiente entrada
    movieNameInput.value = '';
    genreSelect.selectedIndex = 0;
    countrySelect.selectedIndex = 0;
}
```

Imagen 17 Comando de funciones

* `renderMovie()`: Genera el HTML de la tarjeta y lo inserta en la página.

```
/**
 * Crea el elemento visual (HTML) para una película y lo agrega a la lista.
 * @param movie El objeto película a mostrar
 */
function renderMovie(movie: Movie) {
    const card = document.createElement('div');
    card.className = 'movie-card'; // Clase CSS para estilos

    // Insertamos el HTML dentro de la tarjeta usando Template Strings (``)
    card.innerHTML = `
        <div class="movie-info">
            <h3>${movie.title}</h3>
            <div class="movie-details">
                <span class="tag">🎬 ${movie.genre}</span>
                <span class="tag">🌍 ${movie.country}</span>
            </div>
        </div>
    `;

    // .prepend() agrega el elemento AL PRINCIPIO de la lista (nueva película arriba)
    movieList.prepend(card);
}

// 4. EVENTOS
// Esperamos a que el HTML termine de cargar para iniciar las opciones
document.addEventListener('DOMContentLoaded', initOptions);
// Escuchamos el clic en el botón
addBtn.addEventListener('click', addMovie);
```

Imagen 18 Comando de funciones

5. Cómo Ejecutar el Proyecto

1. En la terminal, ejecuta:

Npm run build

```
C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII>npm run build

> ejercicio-1-2-programacioniii@1.0.0 build
> tsc && copyfiles -u 1 src/views/**/* src/public/**/* dist
```

Imagen 19 Compilacion de Proyecto

2. Ejecute el siguiente comando:

Npm start

```
C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-ProgramacionIII>npm start

> ejercicio-1-2-programacioniii@1.0.0 start
> node dist/app.js

============================================================
🚀 Ejercicio 1-2 - Programación III v1.0.0
============================================================
✅ Servidor corriendo en: http://localhost:3000
📁 Entorno: development
📁 Directorio de vistas: C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-Prog
ramacionIII\dist\views
📁 Archivos estáticos: C:\Users\User\Documents\GitHub\Ejercicio-1-2-ProgramacionIII\Ejercicio-1-Progra
macionIII\dist\public
============================================================
📋 Rutas disponibles:
   • GET  /              - Página principal
   • POST /buscar-animal - Buscar animal
   • GET  /animal/:nombre - Ver animal específico
   • GET  /acerca-de     - Información del proyecto
============================================================
🛑 Presiona Ctrl+C para detener el servidor
============================================================
```

Imagen 20 Servidor ejecutado e iniciado

5. El navegador se abrirá automáticamente mostrando la aplicación.

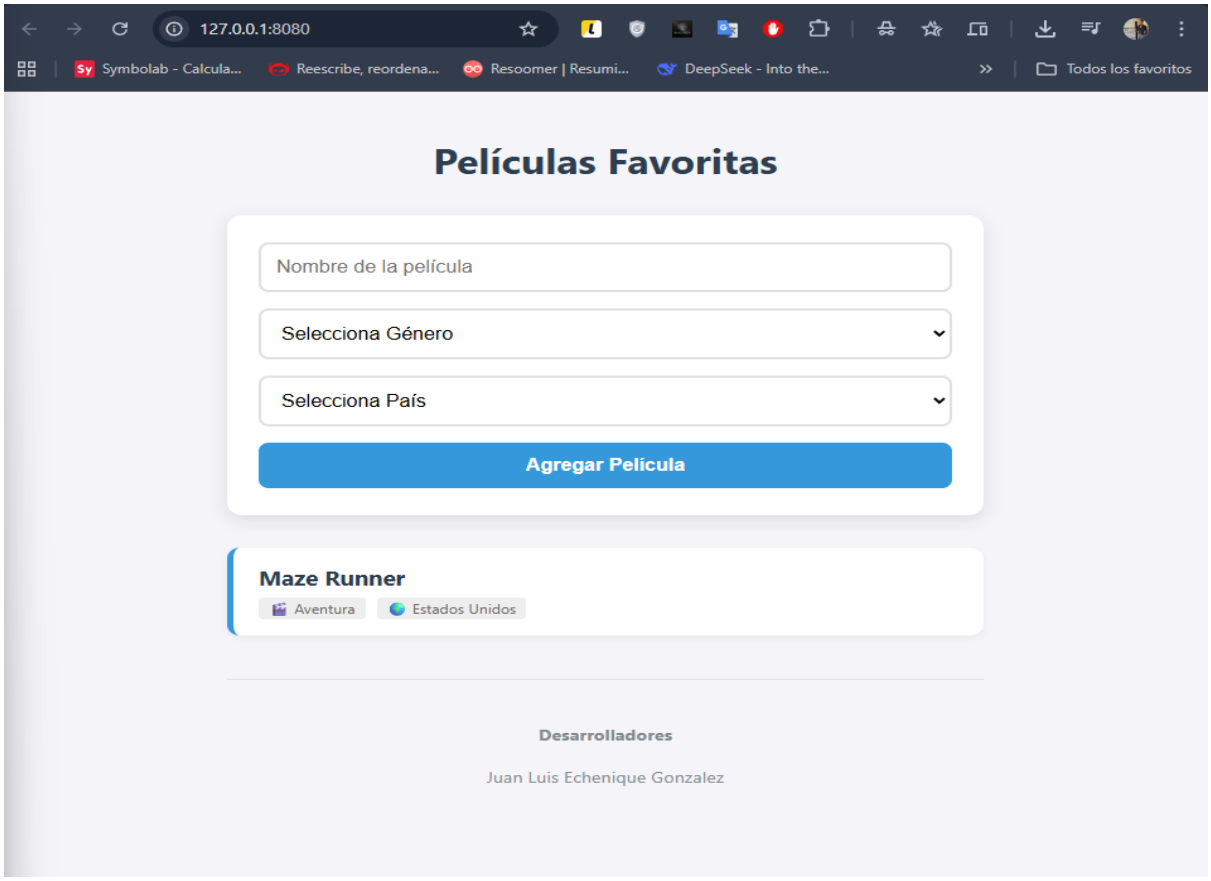

Imagen 21 Proyecto ejecutado e iniciado

Repositorio del Proyecto

https://github.com/Jun1308/Ejercicio-1-2-ProgramacionIII

CONCLUSION

Ambos ejercicios cumplen satisfactoriamente con el objetivo de integrar tecnologías modernas en el desarrollo web, demostrando un manejo adecuado de herramientas como Express, EJS, enums, interfaces, clases y la manipulación dinámica del DOM. Lo que demuestra que el estudiante obtuvo conocimientos en la sesión dicótica impartida, preparándose para la siguientes actividades mas adelantes

BIBLIOGRAFÍA

https://www.youtube.com/watch?v=X0gcwHrBnqM&list=PLTd5ehIj0goPbPaN9VEoQQVUwZN2eXdB5&index=31

https://www.youtube.com/watch?v=RhXLQxLcF70&list=PLvRPaExkZHFn_EGzNOXsWqtAAGl8ZCbUo&index=4

https://www.youtube.com/watch?v=4W3UWjyyVkQ