Sustentación API cine
Yuleibis Noemí Armenta Cruz Y Adriana Carolina Rodríguez Caicedo

Problemática que resuelve nuestro programa

En el mundo del cine, actores, directores, películas y teatros son entidades que necesitan **organización y gestión eficiente**.
Sin un sistema, la información estaría dispersa o dependería de documentos manuales, hojas de cálculo o incluso solo del conocimiento de las personas encargadas.

Esto trae problemas como:

- Dificultad para **consultar información** (ejemplo: "¿Cuál es la nacionalidad de un actor?" o "¿Qué capacidad tiene cierto teatro?").
- Posibilidad de **duplicar datos** (dos películas con el mismo ID, dos directores repetidos, etc.).
- Imposibilidad de **eliminar o actualizar registros de manera controlada**.
- Falta de un **punto centralizado de acceso** para integrar esta información con otros sistemas (como apps móviles, sistemas de boletería, etc.).

Solución que aporta nuestro programa

Nuestra aplicación **resuelve esa problemática creando una API REST con NestJS** que permite:

1. **Gestionar actores**

- Registrar actores con datos validados (nombre, fecha de nacimiento, nacionalidad).
- Consultar todos los actores o uno específico.
- Evitar duplicados por id.
- Eliminar actores cuando sea necesario.

2. **Gestionar directores**

- Registrar directores con datos validados.
- Consultar directores existentes.
- Controlar duplicados y eliminar registros.

3. **Gestionar películas**

- Registrar películas con validaciones en el título, año (1900 hasta el año actual) y género.
- Evitar IDs duplicados.

- Consultar todas las películas o una en específico.
- Eliminar películas de forma controlada.

4. **Gestionar teatros**

- Registrar teatros/cines con nombre, ubicación validada y capacidad limitada entre 100 y 5000.
- Evitar duplicados por ID.
- Consultar todos los teatros o uno en particular.
- Eliminar teatros por ID.

En pocas palabras

Nuestro programa **resuelve el problema de la gestión centralizada y validada de datos de un sistema de cine**.
Permite:

- Crear registros con datos consistentes gracias a las validaciones (class-validator).
- Consultar la información fácilmente mediante endpoints REST.
- Eliminar registros de manera controlada y con mensajes claros.
- Tener una base sólida para escalar a un sistema más grande (por ejemplo, relacionar películas con actores o directores, o teatros con películas que proyectan).

Se crearon 4 entidades, todas con validación tipo DTO para manejar de manera organizada a los directores, actores, películas y salas de teatro.
Se demostró que las validaciones y el código de manera general funciona exitosamente, además de contar con los requisitos que se solicitaron en primer lugar:

Movie:
Rutas: 6

- GET /movie → obtener todas las películas
- GET /movie/:id → obtener película por ID
- POST /movie → crear una película
- DELETE /movie/:id → eliminar película por ID
- (Y en el archivo .http se incluye 2 eliminaciones como prueba: /movie/1 y /movie/2)
- **Verbos:** 3 (GET, POST, DELETE)

Director:
Rutas: 4

- GET /director → obtener todos los directores
- GET /director/:id → obtener director por ID
- POST /director → crear un director
- DELETE /director/:id → eliminar director por ID
- **Verbos:** 3 (GET, POST, DELETE)

Actor:
Rutas: 4

- GET /actor → obtener todos los actores
- GET /actor/:id → obtener actor por ID
- POST /actor → crear un actor
- DELETE /actor/:id → eliminar actor por ID
- **Verbos:** 3 (GET, POST, DELETE)

Theater:
Rutas: 4

- GET `/theater` → obtener todos los teatros
- GET `/theater/:id` → obtener teatro por ID
- POST `/theater` → crear un teatro
- DELETE `/theater/:id` → eliminar teatro por ID
- **Verbos:** 3 (GET, POST, DELETE)

En resumen:

- **Movie:** 6 rutas → 3 verbos
- **Director:** 4 rutas → 3 verbos
- **Actor:** 4 rutas → 3 verbos
- **Theater:** 4 rutas → 3 verbos

Total, General:
Rutas: **6 + 4 + 4 + 4** = 18 rutas
Verbos: **3 + 3 + 3 + 3** = 12 verbos

Estructura de las carpetas:

```
mini-api
├── dist/                    # Archivos compilados de TypeScript a JavaScript
├── node_modules/            # Dependencias del proyecto
│
├── src
│   ├── actor/               # Módulo de Actor
│   ├── director/            # Módulo de Director
│   ├── movie/               # Módulo de Movie
│   ├── theater/             # Módulo de Theater
│   │
│   ├── app.controller.spec.ts   # Archivo de pruebas para AppController
│   ├── app.controller.ts        # Controlador principal de ejemplo
│   ├── app.module.ts            # Módulo raíz donde importas los demás módulos
│   ├── app.service.ts           # Servicio principal de ejemplo
```

```
│   └── main.ts              # Punto de entrada de la aplicación
│
├── test/                    # Pruebas unitarias y de integración
│
├── .gitignore
├── .prettierrc              # Configuración de formato de código
├── eslint.config.mjs        # Configuración de ESLint
├── nest-cli.json            # Configuración de Nest CLI
├── package-lock.json
├── package.json             # Dependencias y scripts
├── README.md
├── tsconfig.build.json      # Configuración de compilación
└── tsconfig.json            # Configuración de TypeScript
```