SENATI

PLAN DE TRABAJO DEL ESTUDIANTE

RDA
RECURSO DIDÁCTICO PARA EL APRENDIZAJE

DATOS DEL ESTUDIANTE

Apellidos y Nombres:	**Alcala Meneses Luis Orlando**	**ID:**	**1463684**
Dirección Zonal/CFP:	**Ica – Ayacucho**		
Carrera:	Ingeniería de Software con Inteligencia Artificial	Semestre:	**V**
Curso/ Mód. Formativo:	**FullStack Developer Software**		
Tema de Trabajo Final:	**Backend**		

INFORME DE LA TAREA 8

Estructura del Informe – Proyecto "VideoClub" (Backend)

1. Introducción

Explicación sencilla de la finalidad del proyecto: crear un backend funcional conectado a MySQL para gestionar una tabla de películas mediante operaciones básicas (GET, POST, PUT y DELETE).

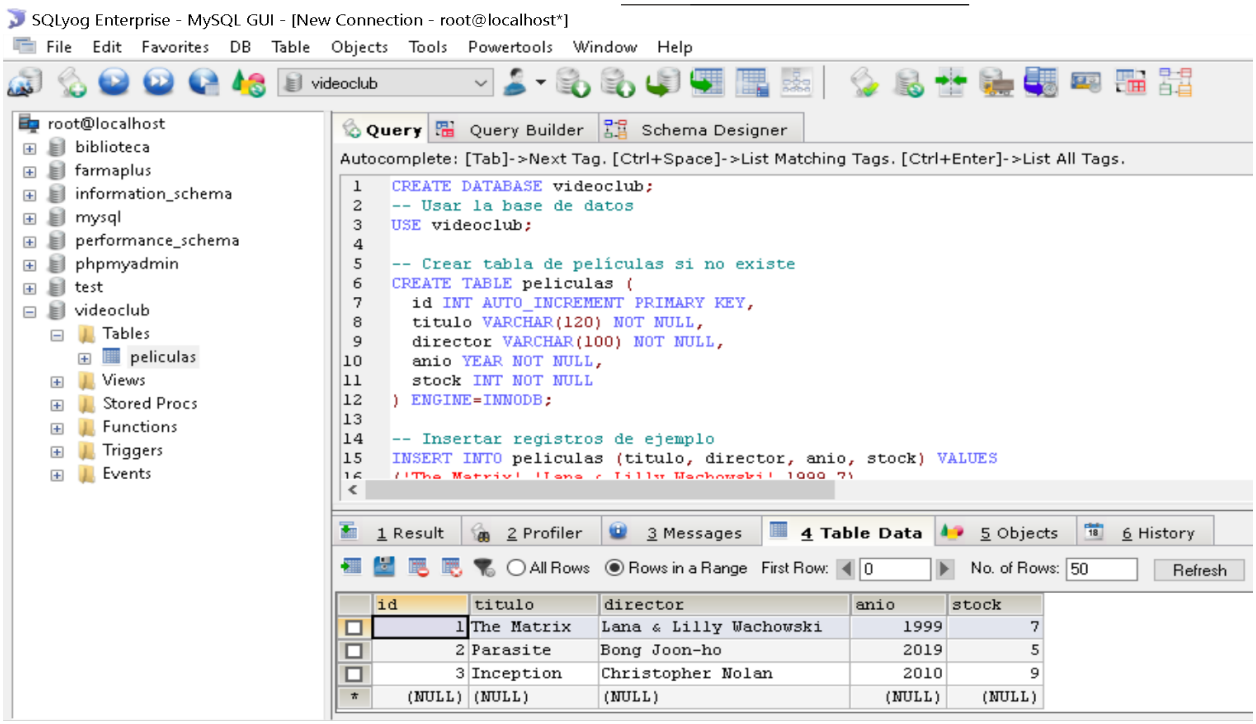
2. Preparación del Entorno

- **Instalación de MySQL.**
- **Instalación de Node.js y creación del proyecto (npm init -y).**
- **Instalación de librerías necesarias (express, mysql2, cors, dotenv, etc.).**

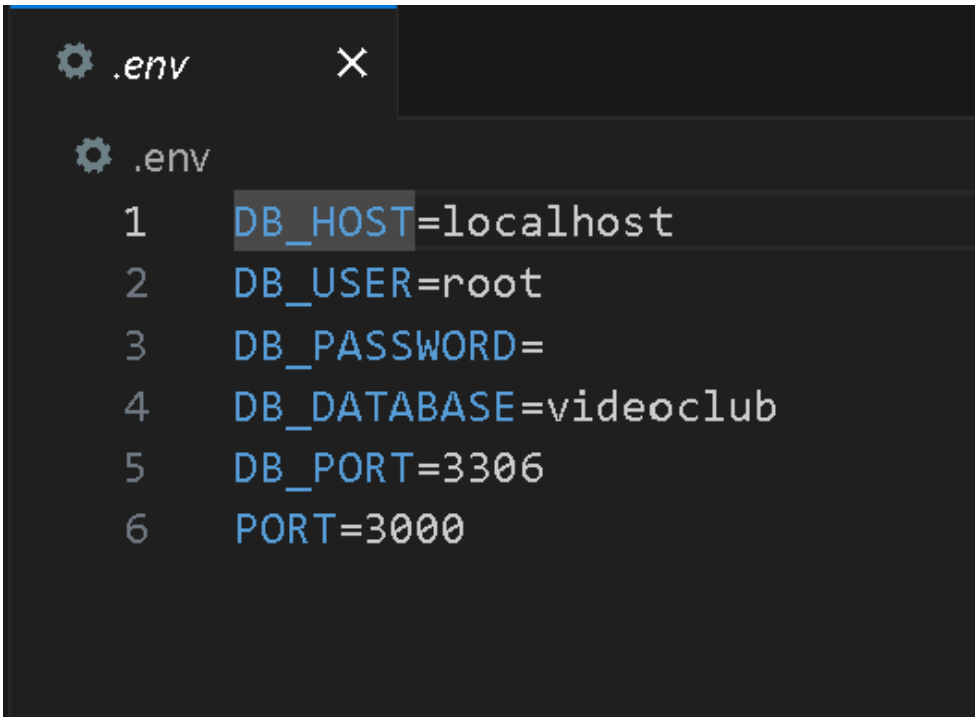
3. Diseño de la Base de Datos

- **Creación de la base de datos videoclub.**
- **Creación de la tabla peliculas con sus campos:**
  - **id (PRIMARY KEY, auto_increment)**
  - **titulo (VARCHAR 120)**
  - **director (VARCHAR 100)**
  - **anio (YEAR)**
  - **stock (INT)**

Creación de la BD para su uso en el Backend (Trata sobre VideoClub y contiene su tabla Peliculas)



3.1 Configurando las Credenciales y Creando la Conexión Inicial a la BD:



4. Configuración del Servidor Backend

- **Estructura de carpetas: controllers, routes, config.**
- **Archivo db.js → conexión con MySQL.**
- **Archivo server.js → configuración del servidor con Express.**

📂 Archivo db.js — Conexión con la Base de Datos

Este archivo permite establecer la conexión entre el servidor backend y la base de datos MySQL usando un pool de conexiones. Esto mejora el rendimiento porque mantiene conexiones abiertas y reutilizables en lugar de abrir una nueva cada vez que se hace una consulta.

db.js

config > db.js > ...

```
1   // Acceso a datos:
2   // Acceder al archivo .env
3   require('dotenv').config()
4
5   // Administrar la BD (promesa = proceso en curso...)
6   const mysql = require('mysql2/promise')
7
8   // Pool de conexiones = acceso
9   const pool = mysql.createPool({
10    host: process.env.DB_HOST,
11    user: process.env.DB_USER,
12    password: process.env.DB_PASSWORD,
13    database: process.env.DB_DATABASE,
14    port: process.env.DB_PORT
15  })
16
17  // Aprovechar el recurso en otra parte de la App
18  module.exports = pool
```

Explicación paso a paso sobre db.js :

1. **Se usa dotenv para cargar variables de entorno de manera segura.**
2. **mysql2/promise permite manejar la conexión con MySQL usando promesas, facilitando un código más limpio y asincrónico.**
3. **Se crea un pool de conexiones para evitar problemas de rendimiento con múltiples peticiones simultáneas.**
4. **El objeto pool se exporta para ser usado en cualquier parte de la aplicación (por ejemplo, en los controladores).**

🧠 Archivo peliculaController.js — Lógica del CRUD

Este archivo contiene la lógica de negocio de la aplicación, es decir, las funciones que interactúan directamente con la base de datos MySQL para crear, listar, buscar, actualizar y eliminar registros de la tabla peliculas.

Explicación paso a paso sobre peliculaController.js :

Explicación clara para tu informe PDF:

1. **crearPelicula** → Inserta una nueva película validando primero que los campos requeridos no estén vacíos.
2. **obtenerPeliculas** → Retorna un listado completo de las películas registradas.
3. **obtenerPeliculaPorId** → Busca una película específica mediante su ID.
4. **actualizarPelicula** → Modifica uno o más campos de una película existente.
5. **eliminarPelicula** → Elimina una película según el ID enviado por URL.

📂 Archivo peliculaRoutes.js — Definición de Rutas de la API

Este archivo define las rutas principales que permiten interactuar con el backend de VideoClub.
Cada ruta está asociada a una función específica del controlador (peliculaController.js), lo que facilita la organización del código siguiendo el patrón MVC (Modelo – Vista – Controlador).

peliculaRoutes.js

routes > peliculaRoutes.js > ...

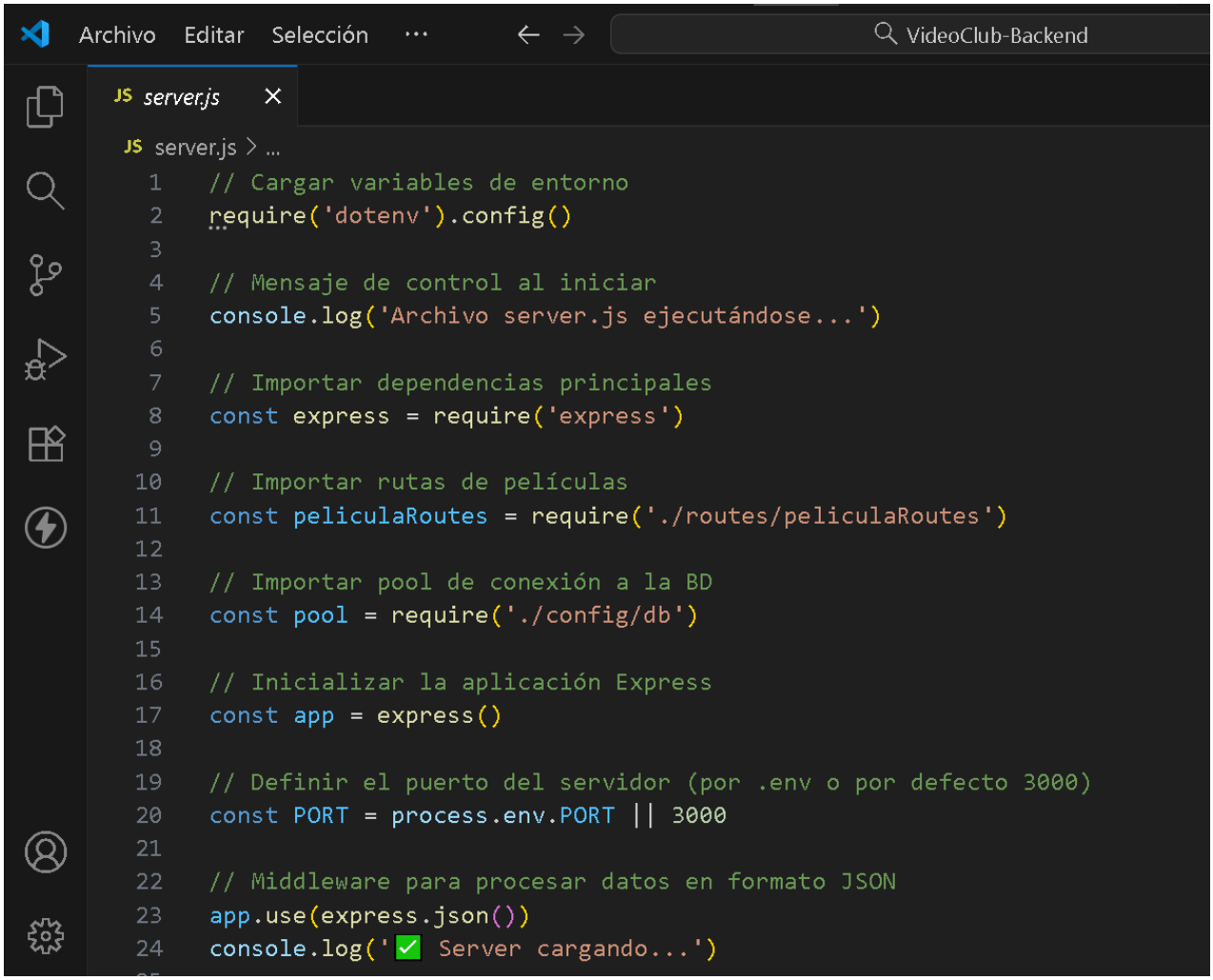
```
1   const express = require('express');
2   const router = express.Router();
3   const peliculaController = require('../controllers/peliculaController');
4
5   // Crear película
6   router.post('/', peliculaController.crearPelicula);
7
8   // Listar todas las películas
9   router.get('/', peliculaController.obtenerPeliculas);
10
11  // Obtener película por ID
12  router.get('/:id', peliculaController.obtenerPeliculaPorId);
13
14  // Actualizar película
15  router.put('/:id', peliculaController.actualizarPelicula);
16
17  // Eliminar película
18  router.delete('/:id', peliculaController.eliminarPelicula);
19
20  // Exportar el router
21  module.exports = router;
```

xplicación paso a paso sobre peliculaRoutes:

1. **Importación de dependencias:**
   Se importa express para crear un router que gestionará las rutas relacionadas con las películas.
2. **Asociación de rutas y controladores:**
   - **POST /api/peliculas** → crear nueva película
   - **GET /api/peliculas** → listar todas
   - **GET /api/peliculas/:id** → obtener una específica
   - **PUT /api/peliculas/:id** → actualizar
   - **DELETE /api/peliculas/:**id → eliminar
3. **Exportación del router:**
   Esto permite que las rutas se usen en server.js con la línea:
   app.use('/api/peliculas', peliculaRoutes);

🧭 Archivo server.js — Configuración del Servidor Backend

Este archivo es el corazón del proyecto. Se encarga de levantar el servidor con Express, conectarlo a la base de datos y registrar las rutas de la API.



server.js

```
// Middleware para registrar acceso a /api/peliculas
app.use('/api/peliculas', (req, res, next) => {
  console.log('📡 Entrando a /api/peliculas')
  next()
})

// Rutas principales de la API
app.use('/api/peliculas', peliculaRoutes)

// Ruta de prueba para verificar que Express responde
app.get('/', (req, res) => {
  res.send('Servidor funcionando correctamente')
})

// ✅ Verificar conexión a la base de datos al iniciar
pool.query('SELECT 1')
  .then(() => console.log('✅ Conexión a la base de datos correcta'))
  .catch(err => console.error('❌ Error de conexión a la BD:', err))

// Iniciar el servidor y escuchar en el puerto indicado
app.listen(PORT, () => {
  console.log(`Servidor iniciado en http://localhost:${PORT}`)
})
```

Explicación paso a paso sobre server.js:

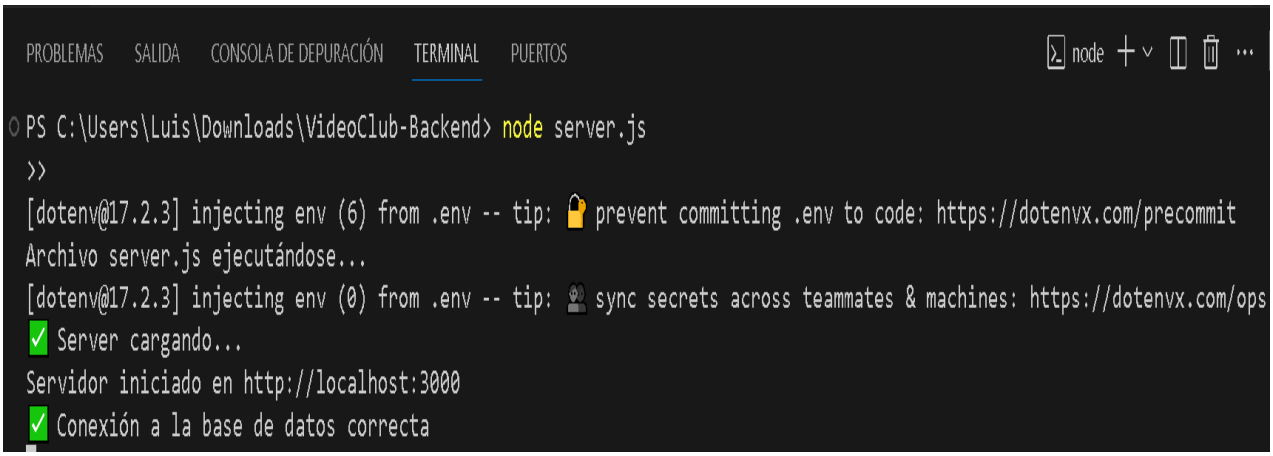
1. **Carga de variables de entorno:**
   Se usa dotenv para mantener credenciales seguras y no escribirlas directamente en el código.
2. **Inicialización de Express:**
   Express permite levantar un servidor rápido y modular.
3. **Middleware:**
   express.json() interpreta los datos enviados en formato JSON.
   También hay un middleware de consola que muestra cada acceso a

Consola:

Evidencia en consola de la correcta conexión del servidor backend con la base de datos MySQL.

El mensaje "✅ Conexión a la base de datos correcta" confirma que el pool establecido en db.js funciona correctamente.

El mensaje "🚀 Servidor iniciado en http://localhost:3000" indica que el servidor Express se está ejecutando sin errores y está listo para recibir peticiones en el puerto configurado.

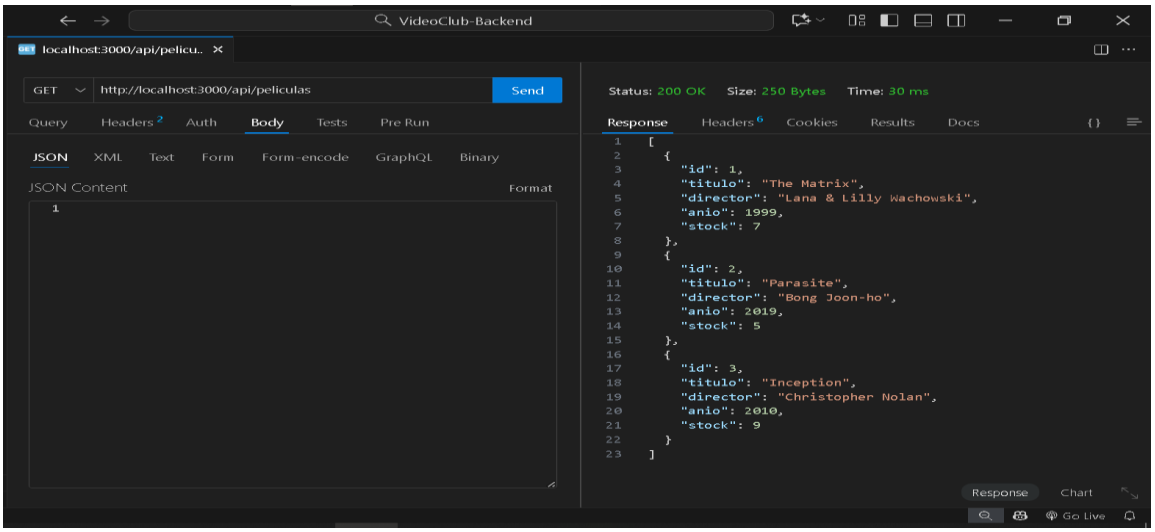


5. Endpoints REST Implementado

GET /api/peliculas → listar películas.

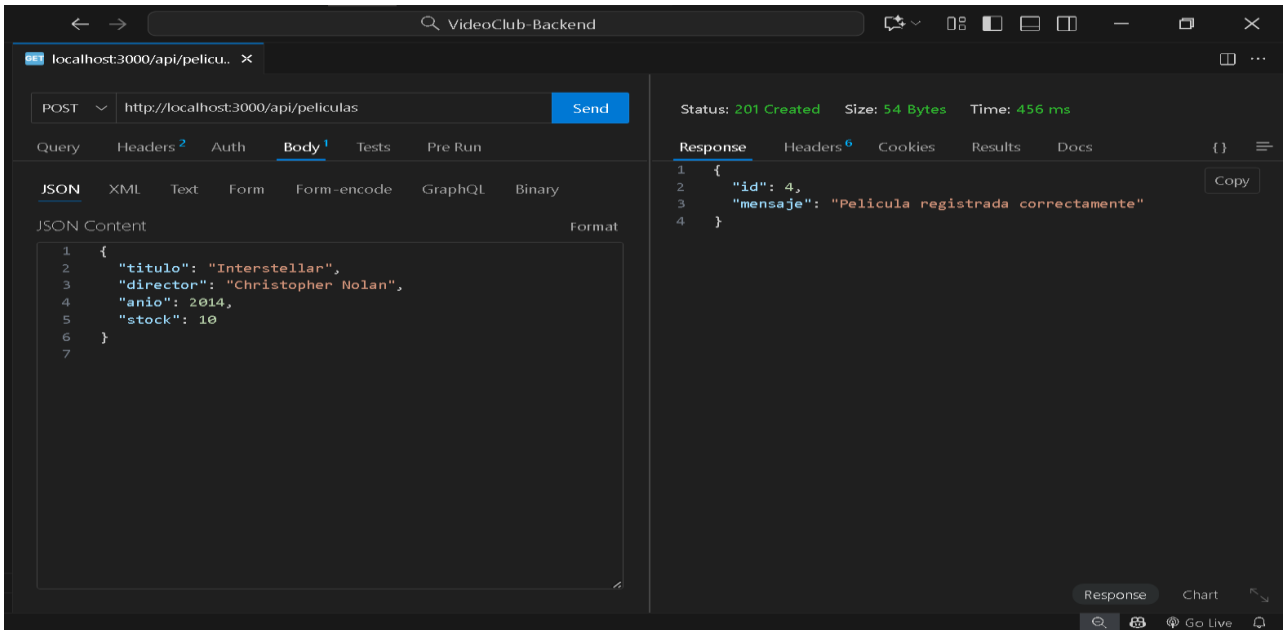
Este endpoint permite obtener un listado completo de todas las películas almacenadas en la base de datos.

Cuando se realiza una solicitud GET a esta ruta, el servidor responde con un arreglo en formato JSON que contiene cada película registrada, incluyendo sus atributos: id, titulo, director, anio y stock.



POST /api/peliculas — Registrar película

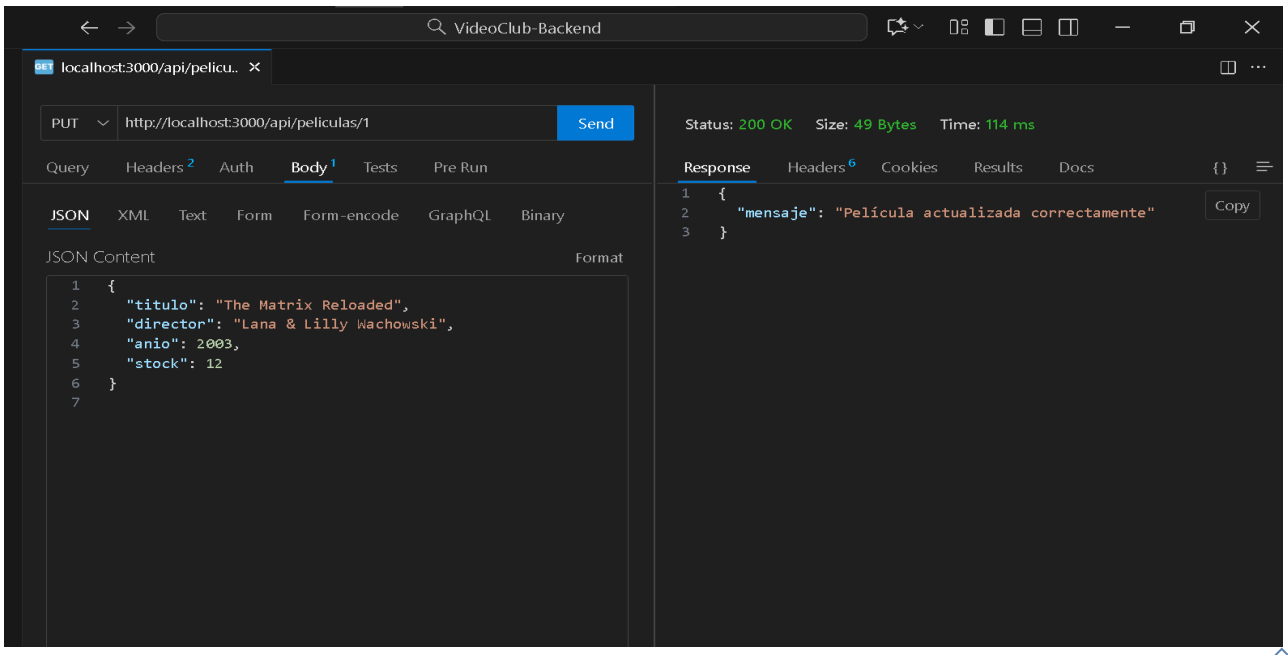
Descripción. Este endpoint crea un nuevo registro en la tabla peliculas. El cliente envía un JSON con titulo, director, anio y stock.



PUT /api/peliculas/:id — Actualizar película

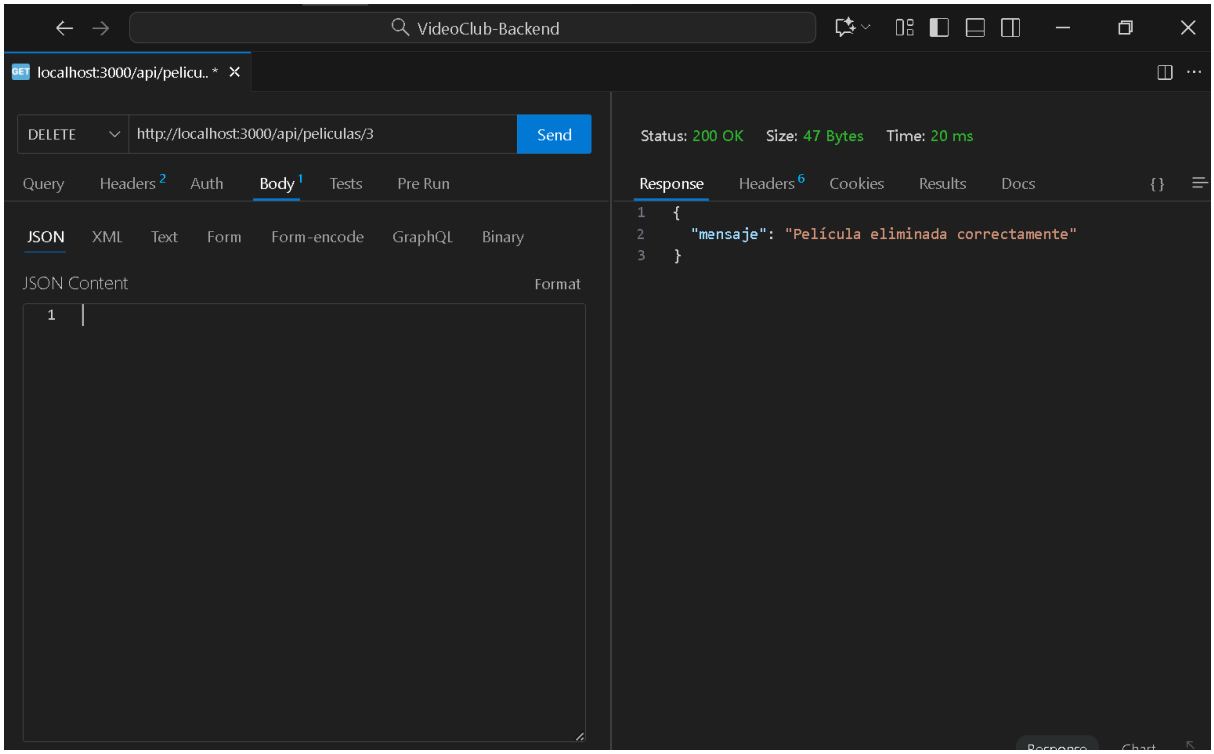
Descripción. Actualiza los datos de una película existente indicada por :id.
URL de ejemplo. http://localhost:3000/api/peliculas/1

Flujo interno. La ruta PUT /api/peliculas/:id llama a actualizarPelicula en el controlador, que valida campos, arma dinámicamente el UPDATE y ejecuta:



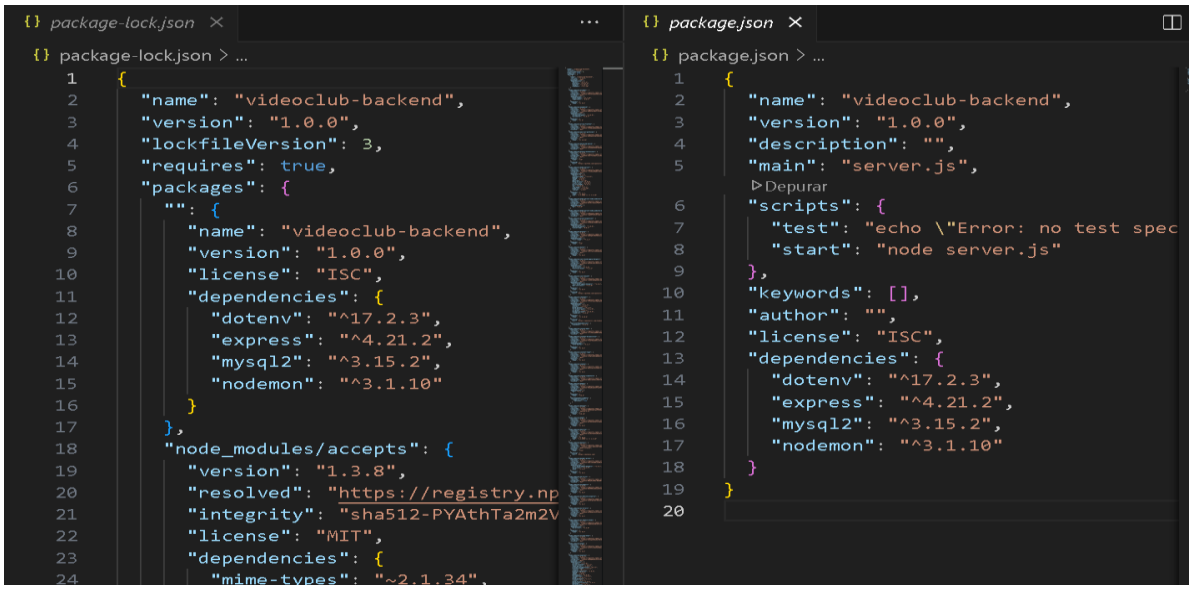
🔴 DELETE /api/peliculas/:id — Eliminar película

Descripción. Elimina definitivamente la película cuyo :id se indica en la URL.
Ejemplo. DELETE http:///localhost:3000/api/peliculas/3
Body. No requiere cuerpo.



6) Requisitos y dependencias

Node.js 18+, MySQL 8.
Paquetes: express, mysql2, dotenv, cors, (opcional: nodemon).
package.json (extracto):



7) Instrucciones de instalación y ejecución

1. npm install
2. Crear archivo .env con PORT, DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE, DB_PORT.
3. Crear BD y tabla (script del punto 3).
4. Ejecutar: npm run dev o npm start.
5. Probar en http://localhost:3000/api/peliculas.

8) Arquitectura y patrón usado

Arquitectura MVC ligera:

- config/ conexión BD (infraestructura)
- controllers/ lógica de negocio (CRUD)
- routes/ capa de rutas (API REST)
  Ventaja: separación de responsabilidades, fácil mantenimiento y escalabilidad.

9) Conclusiones

“El proyecto implementa un backend REST funcional con Node.js, Express y MySQL, cumpliendo CRUD completo sobre la entidad películas. La arquitectura modular (config–controllers–routes) facilita mantenimiento y crecimiento. El sistema queda listo para ser consumido por un frontend o integrarse con autenticación.”


SENATI