Documentación Técnica — Periodo de Prueba (Turing IA)

Este documento consolida la documentación técnica del reto descrito en **“PERIODO DE PRUEBA DESARROLLO DE SOFTWARE (1).pdf”**, ubicado en la raíz del repositorio. El objetivo es presentar una visión clara, verificable y alineada a buenas prácticas sobre la solución desarrollada (frontend + backend + base de datos), así como sus alcances, arquitectura y entregables.

Tabla de contenidos

---

1) Objetivo del reto

Desarrollar una aplicación web **end-to-end** (Frontend + Backend) a partir de un wireframe, demostrando dominio en:

- **HTML/CSS/JavaScript y React** (interfaz moderna y responsiva).
- Manipulación del DOM y consumo de APIs.
- Backend con **CRUD**, autenticación (login) y persistencia en base de datos.
- Arquitectura limpia, validaciones y documentación técnica.

Como criterio de calidad, la solución debe contemplar:

- Integración completa **Frontend ↔ API**.
- Interactividad real (búsqueda, filtros y carga incremental).
- Estilo visual profesional y enfoque en experiencia de usuario.

---

2) Alcance del proyecto

Frontend

- Implementado en **React + Vite**.
- Estética tipo **Netflix** (tema oscuro), con:
    - Navegación clara y componentes reutilizables.
    - Filtros por categoría, búsqueda en tiempo real y paginación/carga incremental.
    - Diseño **responsive** (mobile/tablet/desktop).

Backend

- Implementado en **Node.js + Express**.
- Persistencia con **SQLite** usando **Sequelize** como ORM.
- Funcionalidades principales:
    - Autenticación con JWT.
    - CRUD de películas (con restricciones por rol).
    - Endpoints de categorías y resumen de datos.

Base de datos

- Modelo normalizado con **3 tablas principales** y relaciones en **3NF**.
- Incluye **seed** para datos iniciales (categorías, películas y usuarios).

Fuera de alcance (por tiempo)

- Registro/autogestión de usuarios.
- Administración avanzada (gestión dinámica de roles, panel extendido).
- CI/CD automatizado.

---

3) Requisitos del sistema

- **Node.js 18+**
- **NPM 9+**
- Sistema operativo: Windows/macOS/Linux

---

4) Instalación y ejecución local

Backend

cd backend

npm install

npm run seed

npm run dev

La API quedará disponible en: http://localhost:3000

Frontend

cd frontend

npm install

npm run dev

El frontend quedará disponible en: http://localhost:5173

---

5) Arquitectura del proyecto

Estructura general:

/frontend -> React + Vite

/backend -> Express + SQLite (Sequelize)

/controllers

/routes

/models

/middlewares

/validators

Diagrama lógico (alto nivel)

Usuario → UI React (Navbar + Galería + Admin Panel)
UI React → API Express (Auth, Movies, Categories)
API Express → SQLite (Users, Categories, Movies)

La separación por capas permite mantener responsabilidades claras (rutas, controladores, modelos, validadores y middlewares), facilitando mantenimiento y escalabilidad.

---

6) Modelo de datos y base de datos (3NF)

Tablas

- **Users**: id, username, password, role
- **Categories**: id, name
- **Movies**: id, title, description, poster_url, category_id, release_year

Relaciones

- **Categories (1) → (N) Movies**
- **Users** se utiliza para autenticación y control de acceso por rol.

Seed

El script de seed genera:

- Mínimo **3 categorías**

- Mínimo **5 películas**
- Usuarios iniciales con roles: **admin** y **user**

---

7) Endpoints principales

Autenticación

- POST /api/auth/login

Películas (requiere token)

- GET /api/movies
- GET /api/movies/:id
- POST /api/movies *(solo admin)*
- PUT /api/movies/:id *(solo admin)*
- DELETE /api/movies/:id *(solo admin)*

Categorías (requiere token)

- GET /api/categories

Otros

- GET /api/data *(resumen para dashboard)*
- GET /health *(verificación de estado)*

Se incluye colección lista para pruebas en Postman: postman_collection.json.

---

8) Seguridad y validaciones

La solución incluye medidas de seguridad y validación para asegurar integridad y control de acceso:

- Autenticación basada en **JWT**.
- Autorización por rol:
    - **admin**: CRUD completo.
    - **user**: acceso de solo lectura (GET).

- Validación de payload con **Joi** en operaciones **POST/PUT** de películas.
- Middlewares dedicados para autenticación y autorización.
- Contraseñas encriptadas con **bcryptjs** (incluidas en el proceso de seed).

---

9) Funcionalidades del frontend

- Navbar con efecto dinámico (transparencia → sólido al hacer scroll).
- Sección **Hero** con contenido destacado y CTAs.
- **Búsqueda en tiempo real** por título.
- **Filtros por categorías** consumidos desde la API.
- Paginación por carga incremental (botón **"Cargar más"**).
- Transiciones suaves al cambiar filtros/categorías.
- Panel de administración con acciones de **Editar/Eliminar** y alta de películas.
- Diseño **100% responsivo**.

---

10) Flujo de datos y autenticación

1. El usuario ingresa credenciales en la pantalla de login.
2. La API responde con:
    - **JWT**
    - rol del usuario
3. El frontend conserva el token **en memoria** y lo envía en cada request.
4. El backend valida el token y aplica autorización según el rol.
5. La galería consume GET /api/movies aplicando búsqueda, filtros y paginación.

Filtros y paginación (query params)

- q: filtra por título
- category: filtra por nombre de categoría
- limit y offset: controlan carga incremental (Cargar más)

---

11) UI/UX y componentes clave

Navbar

- Buscador en tiempo real
- Cambio visual por scroll
- Acciones rápidas: “Mi lista” / “Salir” (según implementación)

Hero

- Película destacada con fondo visual
- CTAs: “Reproducir” / “Ver detalles”

Galería

- Cards responsivas con **CSS Grid**
- Acciones admin visibles sobre cada tarjeta
- Transiciones suaves al cambiar filtros/categorías

Panel Admin

- Formulario para crear/actualizar películas
- Select de categorías sincronizado con API
- Manejo de errores y validaciones reflejadas desde backend

---

12) Lineamientos de calidad y buenas prácticas

- Arquitectura por capas: routes, controllers, models
- Validaciones centralizadas (Joi)
- Middlewares para autenticación/autorización
- Mensajes de error claros y consistentes
- Estructura modular, lista para escalar

---

13) Riesgos y mitigaciones

- **Datos inválidos** → validación en backend + feedback visible en UI
- **Errores de autenticación** → JWT + control por rol
- **Rendimiento** → paginación + límites por request
- **Escalabilidad** → separación de responsabilidades + ORM

---

14) Roadmap y mejoras futuras

- Persistencia de sesión (por ejemplo, localStorage con manejo seguro)
- CRUD con modal y edición inline
- Búsqueda avanzada (por año, multi-categoría, etc.)
- Gestión de usuarios en panel admin
- Pruebas unitarias y E2E + pipeline CI/CD

---

15) Evidencias y entregables

Conforme a los lineamientos del documento base, se consideran:

- Avances diarios compartidos en Google Drive.
- Reportes/actividades diarias de progreso.
- Video demostrativo de funcionamiento (plus).
- Repositorio en GitHub con commits claros y descriptivos (plus).

---

16) Lineamientos del periodo de prueba

- Reporte diario en dos horarios:
    - **09:00 am**
    - **06:30 pm**
- Duración: **3 días**
- Subida de avances a la carpeta compartida en **Google Drive**
- Contacto responsable:

- **Mayte Lopez**
- Correo: mlopez@turing-ia.com
- Teléfono: +52 5616631953

---

17) Notas finales

- Las validaciones de Joi requieren que poster_url incluya protocolo (por ejemplo: https://...).
- Para regenerar datos de prueba, ejecutar: npm run seed