Movie app

Producto: Movie api

Modulo: 1

*Documento de requerimiento funcional.*

**Fecha:** 10 de junio de 2025

| **Fecha** | **Autor** | **Versión** | **Referencia de Cambios** |
| --- | --- | --- | --- |
|  |  |  |  |
| 10/06/2024 | Carlos Andrés Montes | 1.0 | Inicio de manual de requerimiento |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contenido

[1. Ficha técnica 4](#_Toc200426192)

[2. Requisitos e instalación de la plataforma: 4](#_Toc200426193)

[2.1 Backend 5](#_Toc200426194)

[2.2 Front End 5](#_Toc200426195)

[3. Tecnologías mas importantes Utilizadas. 5](#_Toc200426196)

[3.1 Tecnologias en el Backend 5](#_Toc200426197)

[3.2 Tecnologías en el front-end 6](#_Toc200426198)

[4. Base de datos. 7](#_Toc200426199)

[5. Estrategias 8](#_Toc200426200)

[5.1 Dockerizar: 8](#_Toc200426201)

[4.2 Autorización: 8](#_Toc200426202)

[4.3 Interceptors: 8](#_Toc200426203)

[4.4 Carga perezosa: 9](#_Toc200426204)

[4.5 Paginación: 9](#_Toc200426205)

[4.6 Formularios reactivos: 9](#_Toc200426206)

# Ficha técnica

|  |  |
| --- | --- |
| ***<Nombre de Requerimiento>*** | |
| OD | *<Orden de desarrollo>* |
| Dominio |  |
| Área |  |
| Módulo |  |
| Responsable de desarrollo. | Carlos Montes |
| Tiempo estimado desarrollo. |  |
| Responsable de pruebas. |  |
| Tiempo pruebas |  |
| ODs asociados |  |
| Capas involucradas |  |
| Tipo de desarrollo. |  |
| Cliente |  |
| Contrato. |  |

# Requisitos e instalación de la plataforma:

## Backend

* Clonar el proyecto de github del siguiente enlace: <https://github.com/carlosmontesengineer/nest-movie-app.git>
* Descargar Docker de la página oficial e instalarlo: <https://www.docker.com/>
* Docker debe estar corriendo en el equipo para que la base de datos configurada en el backend pueda levantar un contenedor por medio de un archivo llamado “docker-compose.yml” que está en la raíz del backend
* Abrir la consola en la raíz del proyecto y ejecutar el siguiente comando: “npm install”
* Se debe copiar el archivo: .env.template y renombrarlo a: .env Luego debemos cambiar las variables de entorno, ya que esta es solo una plantilla, las variables de entorno se deben solicitar por otro medio.
* Levantar la base de datos con el siguiente comando: “docker-compose up -d”. Esto creará un contenedor en Docker con la imagen de postgres.
* Por último debemos ejecutar en la consola el siguiente comando: npm run start. Esto debe correr nuestro backend en el puerto 3000.

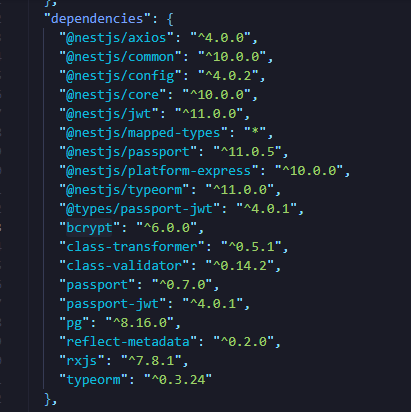
## Front End

* Clonar proyecto del siguiente enlace: <https://github.com/carlosmontesengineer/angular-movies-front.git>
* ejecutar: ``` npm install``` para instalar los módulos de node.
* ejecutar: ``` npm run start ``` para correr el proyecto.

# Tecnologías mas importantes Utilizadas.

## 3.1 Tecnologias en el Backend

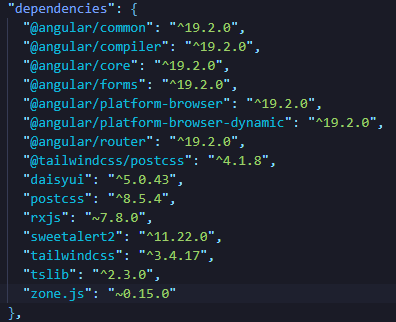
* **Docker:** Se utilizó Docker en la plataforma de forma tal que sea mas fácil el levantamiento de la base de datos. La base de datos hecha en postgres no estará en el equipo si no en una imagen de Docker. En la raíz del backend hay un archivo llamado: “Docker-compuse.ym” el cual ejecuta una configuración que levanta y configura toda la base de datos con tan solo un comando. Esto hace que el despliegue sea mucho más fácil.
* **Postgres:** Se utiliza postgres como base de datos ya que se integra bien con Docker y es un sistema de gestión de base de datos potente.
* **TypeOrm:** Se utiliza type ORM para mapear la base de datos preconstruida en el backend y asi hacer mas
* **Class Validator:** Se utiliza class validator para validar la entrada de datos al backend, esto permite la integridad de los datos.
* **Passport:** Se utiliza pasport para laauthenticacion del usuario, se encarga de devolver un token para poder authenticar al usuario
* **Bcrypt:** Se utiliza bcrypt para encriptar las contraseñas de los usuarios.

****

## Tecnologías en el front-end

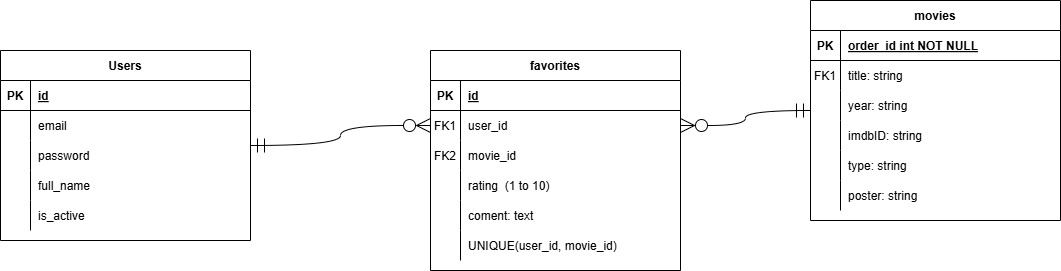
En el front-end se está utilizando Angular en la versión 19. Se están utilizando los nuevos recursos como componentes standalone y se usan más que todo señales para agilizar el rendimiento de la página, aunque no se está utilizando la detección Zoneless. Se usaron herramientas como los resources, que me permiten trabajar de una forma más fácil, sobre todo se utilizaron rxsourse para trabajar de forma directa con los observables. Se utilizó la nueva forma de directivas como el @for, el @if y el @Empty y otros

* **Tailwindcss** : Se utiliza talwind como framework de hojas de estilos ya que tiene muchas clases individuales que hacen mas fácil el diseño responsive y la implementación de css por medio de clases.
* **Daisyui:** Se utiliza para agilizar el diseño en el frontend ya que Daisyui tiene muchos componentes predefinidos que se pueden utilizar mas fácilmente.
* **sweetalert2** : Se utiliza Sweet alert que es un framework que me facilita crear notificaciones al usuario de una forma sencilla.



# Base de datos.

Como se mencionó anteriormente la base de datos está hecha en postgres, específicamente en la versión14.3. La BD está montada en un contenedor de docker y se preconstruye desde un archivo .yml con la configuracion previa.



* Hay una tabla de usuarios que se utiliza para la autenticación,
* Un usuario tiene cero o muchos favoritos.
* Una película es favorita de varios usuarios y a su vez la tabla favoritos tiene el comentario y el rating de la película.

# Estrategias

## 5.1 Dockerizar:

Al utilizar Docker, nos olvidamos de instalaciones tediosas. Basta con tener Docker instalado en el equipo; al ejecutar el backend, este se construye automáticamente. No es necesario tener PostgreSQL instalado localmente, ya que se configura automáticamente dentro de un contenedor en Docker. Además, se evitan conflictos de versiones en el equipo.

## 4.2 Autorización:

Para la autenticación de los usuarios se utilizó **JSON Web Token (JWT)**, ya que es una estrategia segura que no requiere almacenar sesiones en el servidor, lo que reduce la carga y mejora el rendimiento. Se implementaron **guards** y **decoradores** para extraer el usuario a partir del token enviado en la cabecera de las peticiones HTTP, permitiendo así verificar su autenticación. De esta forma, también se protegieron las rutas en los controladores.

En el frontend, igualmente se utilizaron **guards** para proteger las rutas, verificando si el usuario está autenticado. El token se almacena en el **local storage**, lo que permite mantener la sesión activa incluso si el navegador se cierra o la página se actualiza, siempre y cuando el token siga siendo válido.

## 4.3 Interceptors:

Se usaon interceptors para enviar el token en la cabecera en todas las peticiones, debido que para casi todas se necesita enviar el token

## 4.4 Carga perezosa:

Se utilizó carga perezosa para optimizar el funcionamiento y la rapidez de la página, por que retrasa la carga de componentes que no se necesitan. Esto lo hace más rápido.

## 4.5 Paginación:

Se utilizó un componente en la carpeta shared para realizar la paginación por medio de signals, se le envia el numero de paginas y la pagina actual desde el componente padre para que este se construya.

## 4.6 Formularios reactivos:

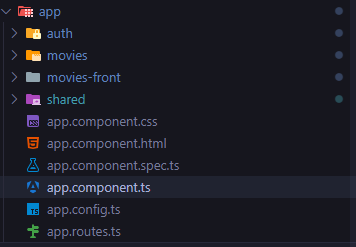
Se utiizaron formularios reactivos para poder validar los datos inmediatamente, se muestran los errores debajo de los input gracias al archivo form Utils

## 4.7 Resources y signals:

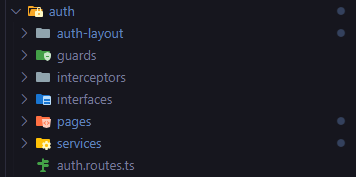
Los resources me permiten realizar peticiones automáticas, se mejora eficientemente el manejo de los datos. Los rxResource permiten obtener los observables y manejar los datos sin tener que pasar por un suscribe. Las señales permiten sincronizar los datos de forma reactiva.

## 4.8 Estructura de carpetas

La estructura de carpetas en el frontend de maneja de la siguiente forma:

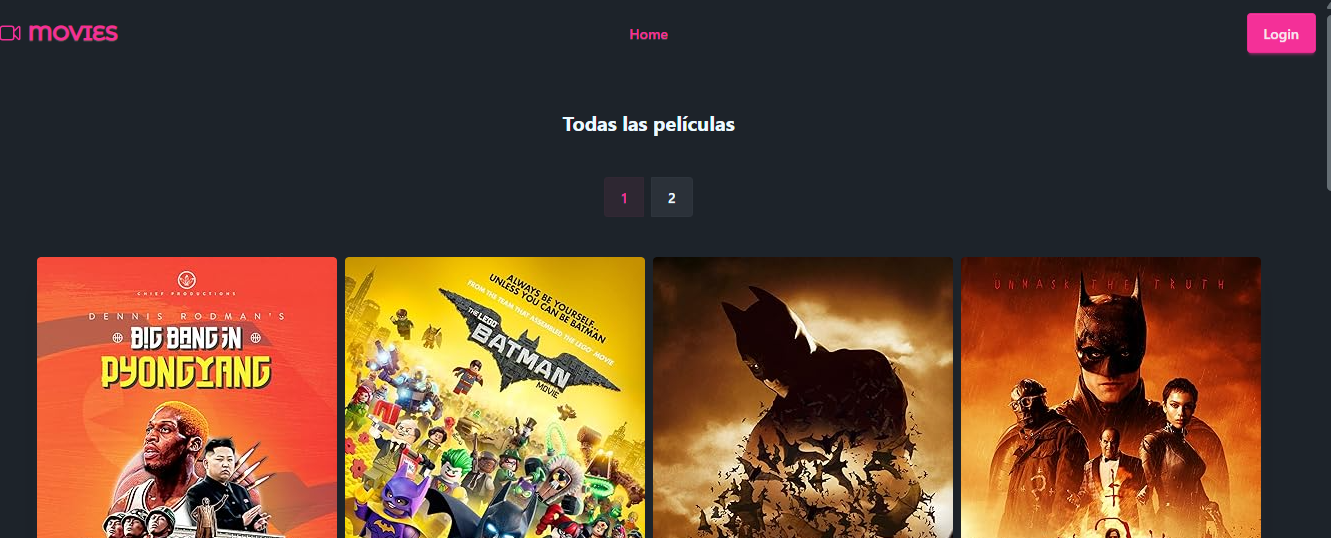


* En la carpeta auth esta todo lo referente a la autenticación del usuario.
* En movies está lo referente a la lógica de las películas
* Movies-front, aquí esta todo lo referente al diseño de las paginas de forma visual
* Auth esta otra carpeta llamada layout, este componente se utiliza para organizar el router outlet y poder realizar la carga perezosa, igualmente para el layout de movies-front
* Cada uno de los submodelos generalmente tiene una estructura de esta forma:

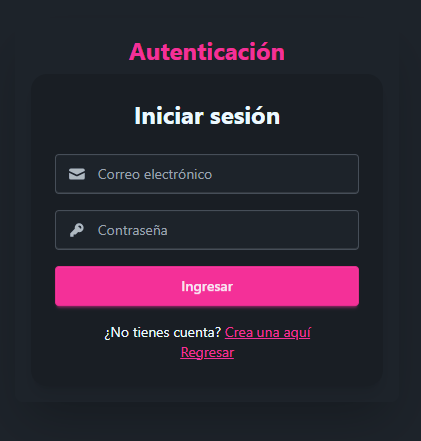


# Funcionamiento de la plataforma

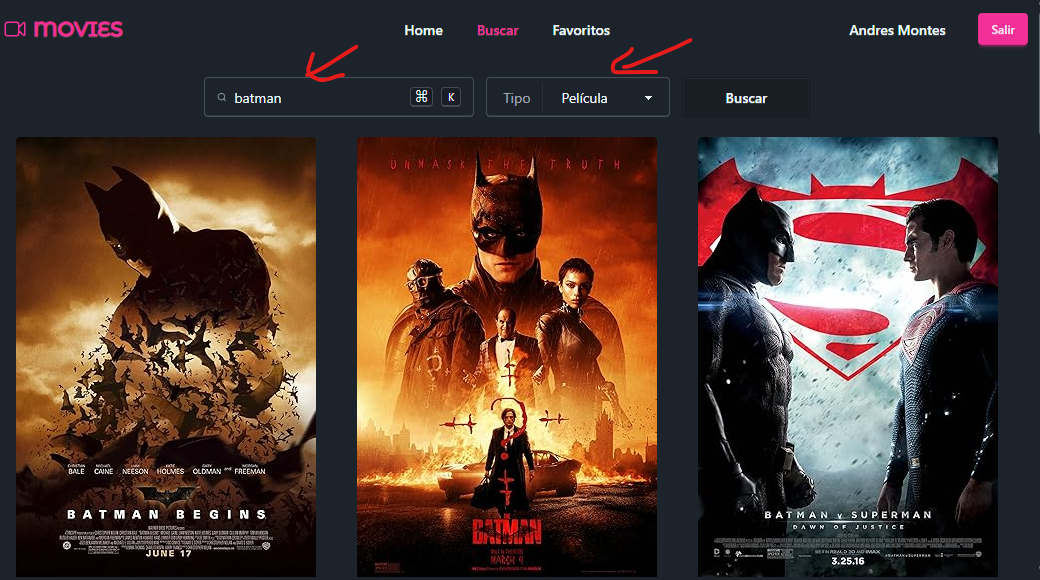
## Inicialmente empieza en la página home donde se muestran todas las películas guardadas en la base de datos en la tabla Movies, de la base de datos local. (No las de omdb).



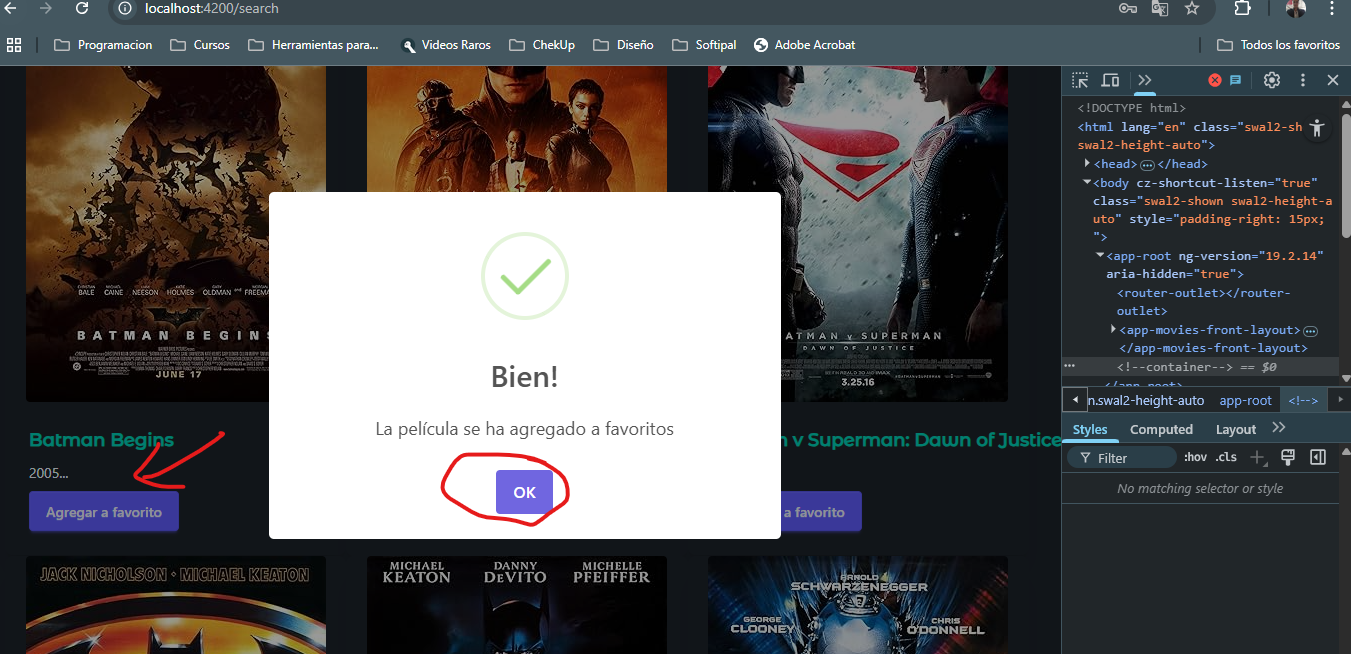
* En la parte superior derecha hacemos click en login para loguearnos.



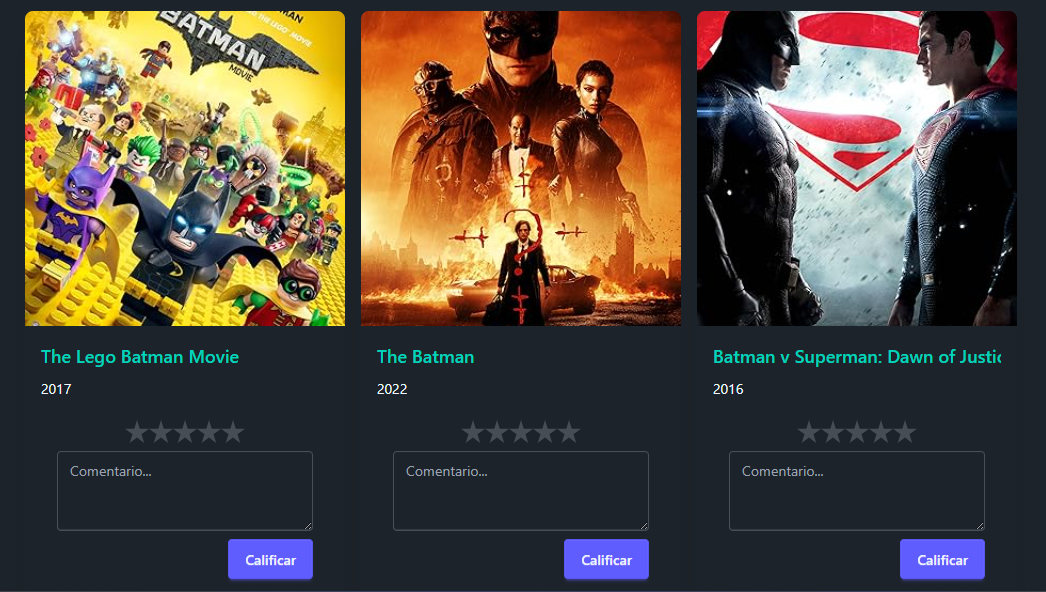
* Después de iniciar sesión en la pestaña de buscar podemos consultar las películas de Omdb para posteriormente ser agregadas a favoritos.



* Debemos hacer click en agregar a favoritos



* Luego salen las favoritas en la pestaña de favoritos, en donde se pueden realizar calificaciones y comentarios.



* El sistema al guardar un favorito lo primero que hace es revisar si la película ya esta en la base de datos en la tabla movies. Si no está la crea, y si esta entonces con el uuid de la película agrega el favorito. De este modo se va poblando la nuestra base de datos de películas a medida que los usuarios van guardando sus favoritos.

# 7.1 EndPoints

El backend utiliza Data Transfer Objects (DTOs) con validaciones que garantizan la integridad de los datos enviados a los endpoints. En el repositorio se incluye un archivo JSON con una colección de Postman que contiene ejemplos de todos los endpoints implementados, facilitando su prueba y documentación.