


Facultad de Ciencias
UNAM

Ingeniería de Software

Proyecto Final
CodeFlix - Comedia

López Villalba Cielo

Martínez Leal José María

Agapito Bautista Erick Israel

Matu Hérnandez Diana

Gómez Elizalde Alexys

CONTENIDO

METODOLOGÍA

Para este proyecto de cine de comedia se decidió adoptar la metodología ágil Scrum con el objetivo de garantizar una gestión eficiente y altamente adaptable. Dado que el sector del entretenimiento requiere una respuesta rápida a las preferencias del público, Scrum nos permite realizar entregas iterativas e incrementales, asegurando que las funcionalidades más valiosas para el cliente estén operativas en el menor tiempo posible. A través de los Sprints y una comunicación constante, el equipo puede identificar riesgos tempranamente, integrar cambios sin afectar el cronograma general y mantener un enfoque de mejora continua, lo cual es vital para ofrecer una experiencia de usuario de alta calidad.

ROLES

Las asignaciones de acuerdo a la metodología de Scrum son de la siguiente manera:

Cielo - Desarrolladora Front-end

José María - Scrum master

Erick Israel - Desarrolladora Back-end

Diana - Product Owner

Alexys - Diseñador

HISTORIAS DE USUARIO:

Esta sección detalla las necesidades del usuario final desde una perspectiva ágil, enfocándose en el valor que cada interacción aporta. Para el proyecto CodeFlix, las historias de usuario se centran en la accesibilidad y la experiencia de consumo de contenido. Los usuarios requieren, principalmente, poder visualizar

un listado de películas de comedia, acceder a detalles específicos como sinopsis y calificaciones, y realizar búsquedas eficientes por título o actor. Asimismo, se enfatiza la necesidad de un diseño intuitivo que facilite la navegación. Estas historias sirven como la base para definir las funcionalidades críticas del MVP.

Las Historias de usuario:

+ Como usuario quiero poder ver una lista de películas de comedía
+ Como usuario que poder ver la sinopsis y calificación de las películas
+ Como usuario quiero poder buscar las películas por título o actor
+ Como usuario quiero que el diseño sea fácil de entender y de usar

CASOS DE USO:

Aquí se describen formalmente las interacciones paso a paso entre los actores (Usuario) y el sistema CodeFlix. Se documentan los flujos principales para las operaciones críticas del negocio.
Actor: Usuario ingresa al sitio web
Flujo principal:

- El usuario accede a la página y decide si seguir como invitado o registrarse
- El usuario crea su cuenta
- El usuario ingresa correo y contraseña
- El sistema crea la cuenta
- El sistema envía correo de confirmación
- El sistema ingresa a la página al usuario
-

Actor: Usuario ve lista de películas
Flujo principal:

- El usuario accede a la página principal del sitio
- El sistema muestra la lista de películas disponibles con su título, portada y año.
- El usuario puede desplazarse, filtrar o seleccionar una película para ver más detalles.

Actor: Usuario visualiza la sinopsis y calificación de la película
Flujo principal:

- El usuario selecciona una película de la lista.

- El sistema muestra la ficha técnica con título, año, sinopsis, calificación y reseñas.
- El usuario puede regresar al catálogo o consultar otras películas.
- Si el usuario lo desea puede calificar la película que seleccionó.

Actor: Usuario busca las películas de su preferencia
Flujo principal:

- El usuario accede a la barra de búsqueda.
- Escribe el nombre de una película o actor.
- El sistema filtra las coincidencias y muestra los resultados correspondientes.
- El usuario selecciona una película de los resultados para ver su información.

REQUISITOS FUNCIONALES Y NO FUNCIONALES

Este segmento establece las especificaciones técnicas y operativas obligatorias del sistema.

- Requisitos Funcionales: Definen *qué* debe hacer el sistema, incluyendo el registro de usuarios, la visualización de catálogos con imágenes, la búsqueda por múltiples criterios (año, título, actor), la gestión de reseñas y calificaciones, y la generación de recomendaciones de películas similares .
- Requisitos No Funcionales: Definen *cómo* debe comportarse el sistema, estableciendo estándares de calidad como un diseño adaptable (responsive) a móviles y PC, tiempos de respuesta menores a 2 segundos, disponibilidad del 99% (24/7), moderación automática de comentarios ofensivos y políticas de respaldo diario de la base de datos

Actores:
-Usuario.
Funcionales

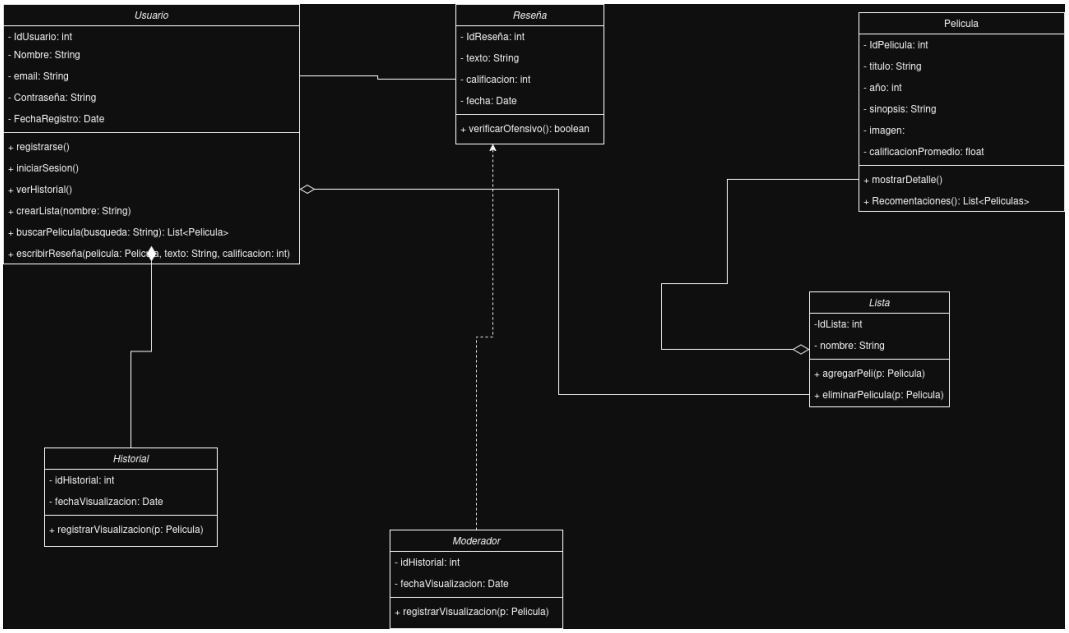
1. Mostrar un catalogo de peliculas de comedia que contenga imagen y título
2. El sistema debe permitir al usuario registrarse e iniciar sesión.
3. Permitir la búsqueda de películas de comedia por año, título o actor.
4. El usuario debe poder dejar reseñas y calificaciones después de ver una película.
5. Mostrar el detalle de la sinopsis de cada película al seleccionarla.
6. El sistema debe mostrar recomendaciones de películas similares.
7. El usuario debe poder ver su historial de películas.

No funcionales

1. La página web será adaptable a celulares y computadoras.
2. Tendrá una interfaz clara y colorida.
3. La respuesta por consulta será rápida, de al menos 2 segundos.
4. Solo los usuarios registrados pueden dejar reseñas.
5. Las reseñas deben moderarse automáticamente contra lenguaje ofensivo.
6. El sistema debe funcionar 24/7 con mínimo 99% de disponibilidad.
7. Las imágenes y videos deben almacenarse en un servidor o CDN para optimizar el rendimiento.
8. El sistema debe mantener copias de seguridad automáticas diarias de la base de datos.
9. El sistema debe ser escalable para integrar futuras secciones como “Eventos” o “Comunidad”.

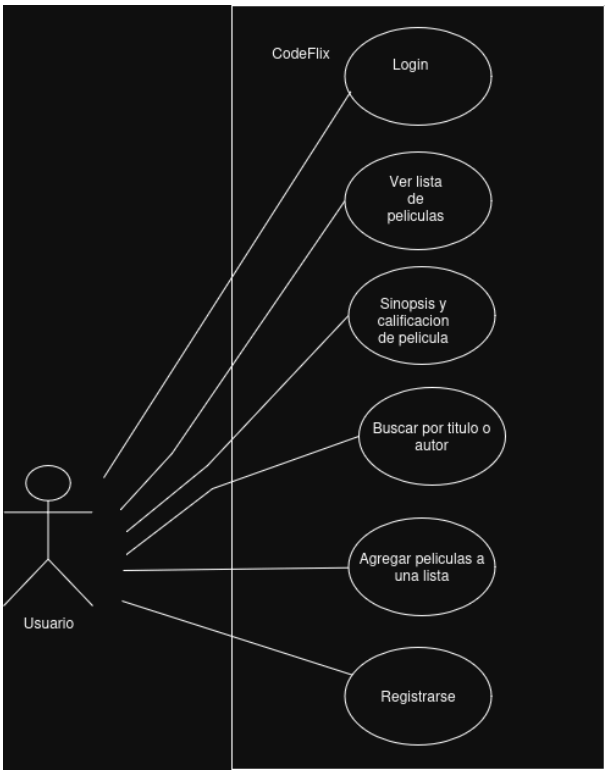
CLASES UML

Esta sección presenta el modelo de datos y la estructura lógica del software a través de un Diagrama de Clases. Se definen las entidades principales y sus relaciones:



CASOS DE USO

Esta sección presenta el modelo de datos y la estructura lógica del software a través de un Diagrama de Clases. Se definen las entidades principales y sus relaciones



ARQUITECTURA

Tras el análisis de los requisitos funcionales y no funcionales, y considerando el plazo y el tamaño del equipo, se ha optado por una Arquitectura Monolítica en Capas (N-Tier), implementada específicamente como un modelo de 3 capas. Esta elección proporciona un balance ideal entre una estructura organizada, velocidad de desarrollo y escalabilidad controlada, siendo la más pragmática para los objetivos del proyecto.

Análisis y Justificación de la Arquitectura Seleccionada

Alternativas Consideradas: Microservicios

Se evaluó la adopción de una arquitectura de microservicios, conocida por su alta escalabilidad y desacoplamiento. Sin embargo, para este proyecto se descartó por las siguientes razones:

- Complejidad Operacional: Introduce una sobrecarga significativa en la gestión de despliegues, comunicación entre servicios (API Gateway, service discovery) y monitorización, que excede las necesidades del proyecto.
- Tiempo de Desarrollo: La configuración inicial y la definición de los límites de cada servicio consumirían una parte considerable del mes asignado al proyecto.
- Cohesión del Dominio: Las funcionalidades del sistema (usuarios, películas, reseñas) están altamente relacionadas. Separarlas en servicios independientes en esta etapa sería una optimización prematura.

Ventajas de la Arquitectura Monolítica en Capas

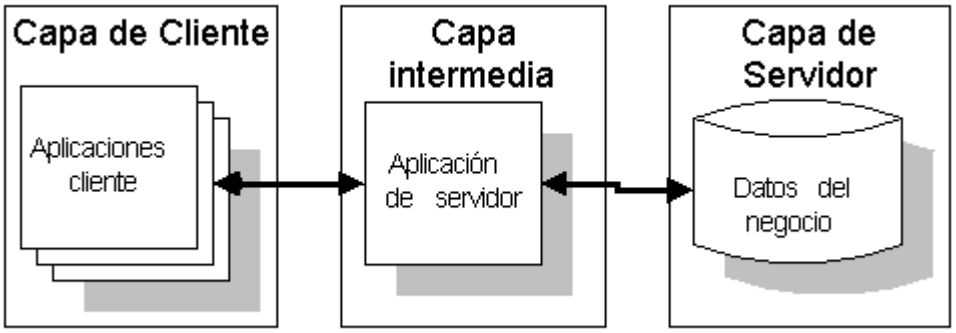
La arquitectura monolítica en capas fue seleccionada por ser la más adecuada, gracias a:

- Simplicidad: Todo el código base se desarrolla y despliega como una única unidad, simplificando el proceso de compilación, pruebas y puesta en marcha.
- Desarrollo Rápido: Al no existir latencia de red entre componentes, el desarrollo es más directo y rápido, ideal para un proyecto con un plazo ajustado.
- Separación de incumbencias (Separation of Concerns): La división en capas lógicas (Presentación, Lógica de Negocio, Datos) permite al equipo

trabajar en paralelo sobre diferentes partes de la aplicación sin interferir entre sí, manteniendo el código organizado y mantenible.

Descripción de la Arquitectura en 3 Capas

La arquitectura divide la aplicación en tres niveles lógicos principales. Cada capa tiene una responsabilidad definida y solo se comunica con las capas adyacentes, promoviendo un flujo de datos estructurado y seguro.



Capa de Presentación (Frontend) Es la interfaz con la que interactúa el usuario final. Su responsabilidad principal es la renderización de la interfaz gráfica y la captura de las entradas del usuario.

- Responsabilidades:
    - Mostrar el catálogo de películas, los detalles, la sinopsis y las reseñas.
    - Presentar formularios de registro, inicio de sesión y búsqueda.
    - Gestionar la experiencia de usuario, asegurando un diseño adaptable (responsive) para computadoras y celulares.
    - Comunicarse exclusivamente con la Capa de Lógica de Negocio a través de peticiones HTTP a su API REST.

Capa de Lógica de Negocio (Backend / API Server) Actúa como el cerebro del sistema. Procesa todas las entradas, aplica las reglas de negocio y coordina las operaciones entre el frontend y la base de datos.

- Responsabilidades:

    - Exponer una API RESTful como punto de entrada seguro para todas las solicitudes del frontend.
    - Gestionar la autenticación (registro, inicio de sesión) y autorización (distinguir entre usuario y administrador) de los actores del sistema.
    - Implementar la lógica de búsqueda, recomendaciones y gestión de películas (Añadir, Editar, Eliminar).
    - Validar y procesar los datos antes de persistirlos, incluyendo la moderación automática de reseñas.
    - Orquestar la comunicación con la Capa de Datos.

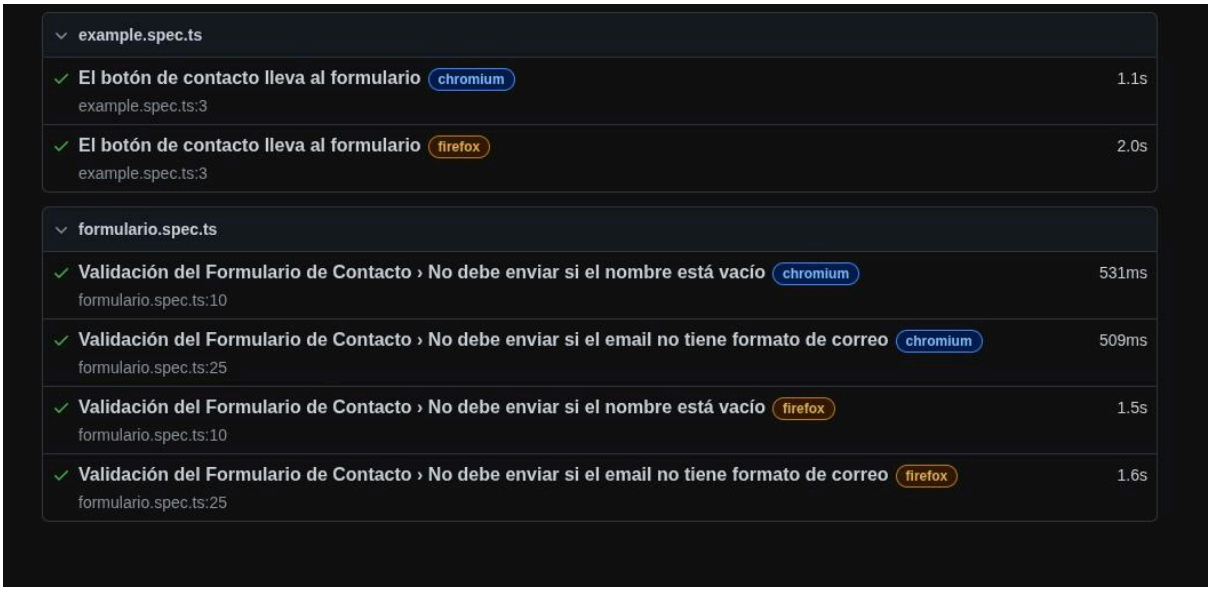
Capa de Datos (Persistencia) Es la responsable del almacenamiento y la gestión de toda la información del sistema de manera permanente y segura.

- Responsabilidades:
    - Abstraer el acceso a la base de datos, proveyendo métodos para consultar y manipular la información.
    - Almacenar los datos de usuarios (con contraseñas debidamente encriptadas), películas, reseñas y calificaciones.
    - Asegurar la integridad y consistencia de los datos.
    - Gestionar las copias de seguridad automáticas diarias.

La arquitectura monolítica en 3 capas es la solución más robusta, adecuada y pragmática para los requerimientos del proyecto. Proporciona una estructura sólida que facilita el desarrollo, las pruebas y el mantenimiento, al tiempo que cumple con todos los requisitos funcionales y no funcionales establecidos. Este diseño sienta una base firme para futuras expansiones del sistema,

PRUEBAS

Se realizaron pruebas End-to-End (E2E) que simulan el comportamiento de un usuario real navegando por el sitio. Validan que todo el sistema (Frontend + Backend/API) funcione en conjunto correctamente. No les importa el código interno, sino que la "experiencia" funcione. Tambien se realizaron pruebas Unitarias (Unit Testing) que aíslan una pequeña parte del código (un componente o función) para verificar que funcione bien por sí sola, sin depender del navegador o de internet. Son rápidas y detectan errores de lógica.




```
erickisrael@fedora:~/Documentos/ingenieria/main/IS_CodeFlix_Comedia/CodeFlix$ npm run test:unit

> codeflix-comedy@0.0.1 test:unit
> vitest run


 RUN  v4.0.14 /home/erickisrael/Documentos/ingenieria/main/IS_CodeFlix_Comedia/CodeFlix

 ✓ src/tests/titulo.test.ts (1 test) 34ms
   ✓ El Header muestra el título CodeFlix 33ms

 Test Files  1 passed (1)
      Tests  1 passed (1)
   Start at  18:49:56
   Duration  1.24s (transform 611ms, setup 0ms, import 1.08s, tests 34ms, environment 0ms)
```


```
erickisrael@fedora:~/Documentos/ingenieria/main/IS_CodeFlix_Comedia/CodeFlix$ npm run test:unit

> codeflix-comedy@0.0.1 test:unit
> vitest run


 RUN  v4.0.14 /home/erickisrael/Documentos/ingenieria/main/IS_CodeFlix_Comedia/CodeFlix

 ✓ src/tests/titulo.test.ts (1 test) 34ms
   ✓ El Header muestra el título CodeFlix 33ms

 Test Files  1 passed (1)
      Tests  1 passed (1)
   Start at  18:49:56
   Duration  1.24s (transform 611ms, setup 0ms, import 1.08s, tests 34ms, environment 0ms)
```

ENLACES DEL PROYECTO

Google Drive : Documentos individuales, presentaciones y diagramas.
Trello : Tablero del proyecto.
Repositorio : Código e imágenes.
Figma : Mockup del proyecto.