G100 Desarrollos, Plaza de la Paz,
Puerto Interior #102, Silao, Gto.

Descripción

Nombre de la Plataforma: CineMagic

Descripción: CineMagic es una plataforma digital diseñada para los amantes del cine, que ofrece una experiencia única para la compra de boletos, visualización de horarios, y acceso a contenido exclusivo de películas. Esta plataforma es intuitiva y está orientada a mejorar la experiencia del usuario antes, durante y después de su visita al cine. CineMagic está preparada para escalar y adaptarse a una creciente comunidad de cinéfilos.

Especificación

Usuarios Esperados:

1. Espectadores: Individuos que desean una experiencia cinematográfica sin inconvenientes. Pueden explorar películas, ver horarios y comprar boletos
2. Administradores de Cine: Gerentes y personal del cine que usan la plataforma para actualizar horarios de funciones, analizar tendencias de ventas de boletos (ver ventas)

Funcionalidades y Servicios:

- Cartelera de películas con horarios actualizados.
- Compra de boletos con selección de asiento.
- Sistema de calificaciones y reseñas de películas por parte de los usuarios.

Transacciones: Las transacciones son cruciales en CineMagic. Los usuarios pueden comprar boletos Se almacena información detallada de cada compra, incluyendo selección de película, horario, asiento, y detalles de pago.

Requerimientos

1. **Normalización y Estructura de Bases de Datos:**

En el modelo de la BD se deben aplicar las reglas de normalización de bases de datos, hasta 3NF para minimizar la redundancia. También se debe considerar la integridad referencial, el uso correcto de claves primarias y foráneas, así como la implementación de índices para optimizar la recuperación de datos.

2. **Principios de Programación Orientada a Objetos:**

Se debe considerar la aplicación de los cuatro principios fundamentales de POO: encapsulamiento, abstracción, herencia y polimorfismo. Contar con una adecuada estructura de las clases y objetos para confirmar la correcta aplicación de estos principios, el uso de interfaces y clases abstractas, y la implementación de patrones de diseño para resolver problemas comunes, así como empaquetamiento de las clases en las capas adecuadas.

3. **Implementación de Spring Boot:**

Se debe considerar una adecuada configuración y estructura del proyecto Spring Boot, incluyendo el uso de anotaciones como @RestController, @Service, @Repository, y la separación de responsabilidades. Así como, la configuración de los beans y el uso de diferentes capas como controladores, servicios, y acceso a datos.

4. **Manejo de Excepciones y Validación de Datos:**

Es necesario considerar una estrategia de manejo de excepciones globales con @ControllerAdvice o @RestControllerAdvice y la implementación de validaciones de entrada usando @Valid y la anotación @Validator. Así como, la cobertura de casos de error y la claridad de los mensajes de error proporcionados al cliente.

5. **Cobertura de Pruebas Unitarias:**

Se deben generar las pruebas unitarias con JUnit para garantizar que cubran una amplia gama de casos de uso. Comprobar la implementación de Mockito para simular dependencias. Las pruebas deben ser capaces de ejecutarse de manera independiente y en cualquier entorno.

6. **Seguridad con Spring Security y JWT:**

Considerar la configuración de Spring Security para la autenticación y autorización, y cómo se generan, distribuyen y validan los JWT. Verificar la seguridad de las rutas, el cifrado de contraseñas, y la gestión de permisos para diferentes roles de usuarios.

7. **Uso y Manejo de Git:**

Considerar el uso efectivo de Git, incluyendo la estructura de ramas, estrategias de merge, y resolución de conflictos. Comprobar el uso de commits semánticos, mensajes descriptivos, y la aplicación de pull requests para revisión de código antes de la integración en la rama principal.

METAPHORCE
ConTIgo al futuro

Evaluación

#	Descriptor	Porcentaje	Cumplimiento
1	Estructura de la BD	10%	
2	POO	10%	
3	SpringBoot	40%	
4	Excepciones	10%	
5	Pruebas Unitarias	10%	
6	Seguridad	10%	
7	GIT	10%	
TOTAL		**100%**	