INSTITUTO POLITÉCNICO NACIONAL 

“Escuela Superior de Cómputo”

**Práctica 7**

**Evaluación de Calidad y Despliegue en un Servidor Dedicado**

**Materia:** Ingeniería De Software

**Profesor:** Gabriel Hurtado Avilés

**Grupo:** 6CV3

**Integrantes:** -Almogabar Nolasco Jaime Brayan | 2022630476

-Díaz Hernández Braulio | 2022630489

-García Quiroz Gustavo Ivan | 2022630278

- Morales Torres Alejandro | 2021630480

-Rodriguez Rivera Claudia Patricia | 2022630334

-Tellez Partida Mario Iahveh | 2022630535

**Índice**

[1 Introducción 1](#_Toc187583114)

[2 Objetivo General 2](#_Toc187583115)

[2.1 Objetivos Particulares 2](#_Toc187583116)

[3 Preparación del Proyecto para Despliegue 3](#_Toc187583117)

[3.1 Documentación de requisitos técnicos 3](#_Toc187583118)

[3.2 Listado de dependencias 4](#_Toc187583119)

[3.3 Configuración de puertos y variables de entorno 5](#_Toc187583120)

[3.4 Arquitectura lógica del sistema 6](#_Toc187583121)

[3.5 Requisitos de red y servicios externos 8](#_Toc187583122)

[3.5.1 Requisitos de Red 8](#_Toc187583123)

[3.5.2 Puertos y Protocolos 8](#_Toc187583124)

[4.1.1 Variables de Entorno Requeridas 10](#_Toc187583125)

[4.2 Optimización del proyecto 10](#_Toc187583126)

[4.2.1 Integración con Dropbox API 10](#_Toc187583127)

[4.2.2 Implementación de Caché 11](#_Toc187583128)

[4.3 Optimización de archivos estáticos 13](#_Toc187583129)

[4.3.1 Configuración de Spring para recursos estáticos 13](#_Toc187583130)

[5 Configuración para Despliegue 14](#_Toc187583131)

[5.1 Configuración de contenedores 14](#_Toc187583132)

[5.2 Implementación de API de almacenamiento en la nube 17](#_Toc187583133)

[5.3 Guía de despliegue 17](#_Toc187583134)

[5.3.1 Procedimientos de ejecución 18](#_Toc187583135)

[5.3.2 Solución de Errores Comunes 19](#_Toc187583136)

[6 Evaluación de Calidad del Software 22](#_Toc187583137)

[6.1.1 Pruebas de Rendimiento 22](#_Toc187583138)

[6.1.2 Pruebas de Seguridad 22](#_Toc187583139)

[6.1.3 Métricas de Rendimiento 24](#_Toc187583140)

[7 Conclusiones 26](#_Toc187583141)

[8 BIBLIOGRAFÍA APA 27](#_Toc187583142)

# Introducción

En la era digital actual, el despliegue efectivo de aplicaciones en entornos de producción se ha convertido en un reto para los ingenieros de software. La capacidad de implementar sistemas robustos en servidores con recursos limitados, mientras se mantiene un alto nivel de rendimiento y seguridad, es un desafío que requiere un enfoque meticuloso y bien planificado.

La industria del software ha evolucionado significativamente en los últimos años, moviéndose hacia arquitecturas más modulares y servicios en la nube que permiten una mayor flexibilidad y escalabilidad. Esta práctica se alinea con estas tendencias al enfocarse en el despliegue de un sistema en un servidor dedicado con restricciones específicas: 500 MB de espacio en disco y 0.5 GB de RAM. Estas limitaciones no son arbitrarias, sino que reflejan escenarios reales donde la optimización de recursos es crucial para la viabilidad económica y técnica del proyecto.

La integración de servicios de almacenamiento en la nube representa una solución innovadora para superar las limitaciones de almacenamiento local. Esta estrategia no solo permite optimizar el uso de recursos, sino que también proporciona beneficios adicionales como redundancia de datos, accesibilidad mejorada y escalabilidad bajo demanda. Al utilizar servicios de Dropbox podemos implementar soluciones que balancean eficientemente las necesidades de almacenamiento con los recursos disponibles.

La containerización, a través de Docker, emerge como una herramienta fundamental en este contexto. La capacidad de encapsular aplicaciones y sus dependencias en contenedores ligeros, con límites de recursos bien definidos, permite un control preciso sobre el consumo de recursos y facilita la replicación del entorno de desarrollo en producción. Esta práctica enfatiza la importancia de optimizar las imágenes de contenedores y configurar adecuadamente los límites de recursos para garantizar un funcionamiento estable dentro de las restricciones establecidas.

# Objetivo General

El propósito central de esta práctica es implementar y desplegar exitosamente el proyecto final en un entorno de producción con recursos limitados, integrando servicios de almacenamiento en la nube para optimizar el rendimiento del sistema.

## Objetivos Particulares

Los objetivos específicos incluyen:

* Desarrollar habilidades prácticas en la optimización de recursos y configuración de entornos de producción
* Implementar integración con servicios de almacenamiento en la nube para gestión eficiente de archivos
* Configurar y optimizar contenedores Docker para un despliegue eficiente
* Evaluar y asegurar la calidad del software mediante pruebas exhaustivas de rendimiento y seguridad

# Preparación del Proyecto para Despliegue

## Documentación de requisitos técnicos

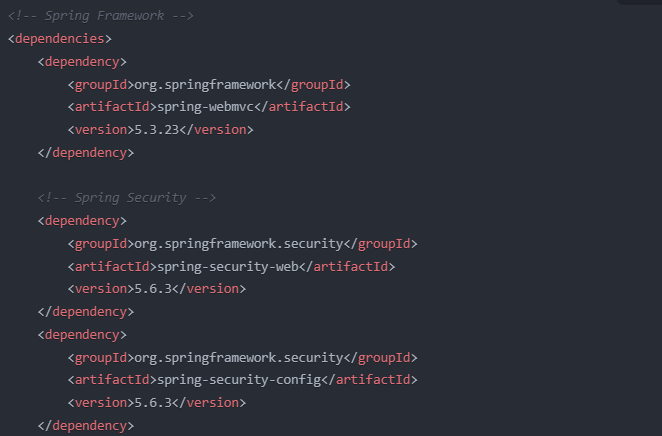
El sistema de recomendación de libros y películas requiere una configuración específica y cuidadosa para mantener la funcionalidad de sus componentes principales, especialmente considerando la integración de múltiples APIs y el sistema de autenticación existente. A continuación, se detallan los requisitos técnicos fundamentales:

* **Entorno de Ejecución**
* Java JDK 11 o superior
* Apache Maven 3.8.x
* Apache Tomcat 9.x
* MySQL Server 8.0 o superior (a través de XAMPP)
* Memoria RAM mínima: 512 MB
* Espacio en disco: 500 MB
* **Requisitos de Base de Datos**
* MySQL (vía XAMPP)
* Base de datos: book\_movie\_recommendation\_db
* Usuario con privilegios para operaciones CRUD
* Persistencia de sesiones de usuario
* Almacenamiento de roles (ROLE\_ADMIN, ROLE\_USER)
* **APIs Externas**

1. Open Library API
   * Endpoint principal: https://openlibrary.org/api/
   * No requiere autenticación
   * Rate limit: 100 requests/minuto
2. TVmaze API
   * Endpoint principal: https://api.tvmaze.com/
   * No requiere autenticación
   * Rate limit: 20 calls/10 segundos
3. Google OAuth 2.0
   * Requiere credenciales de cliente OAuth 2.0
   * Redirect URI configurado
   * Scope: email, profile

## Listado de dependencias

Las principales dependencias usadas son las siguientes:



Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Figura 1 Listado de dependencias

## Configuración de puertos y variables de entorno

Para mantener la integridad del sistema de autenticación y el consumo de APIs, se deben considerar las siguientes configuraciones:

Texto

Descripción generada automáticamente

Figura 2 Variables de entorno

## Arquitectura lógica del sistema

El sistema implementa una arquitectura multicapa que separa claramente las responsabilidades y promueve la escalabilidad y mantenibilidad. A continuación, se detalla cada capa y sus componentes principales:

**Capa de Cliente**

* **Interfaz de Usuario**: Implementada con Spring MVC views y Thymeleaf
* **Componente de Autenticación**: Maneja tanto el login local como la autenticación mediante Google OAuth2

**Capa de Presentación**

* **Controladores Spring MVC**: Manejan las peticiones HTTP y la lógica de presentación
* **Spring Security**: Gestiona la autenticación y autorización, incluyendo:
  + Configuración de roles (ROLE\_ADMIN, ROLE\_USER)
  + Integración con OAuth2
  + Gestión de sesiones

**Capa de Servicios**

* **Servicio de Usuarios**: Gestiona la lógica de negocio relacionada con usuarios y autenticación
* **Servicio de Libros**: Maneja la búsqueda y recomendación de libros
* **Servicio de Películas/Series**: Gestiona la búsqueda y recomendación de contenido audiovisual
* **Servicio de Recomendaciones**: Implementa la lógica de recomendación personalizada
* **Servicio de Caché**: Optimiza el rendimiento almacenando resultados frecuentes

**Capa de Integración**

* **Cliente Open Library API**: Gestiona las llamadas a la API de libros
* **Cliente TVmaze API**: Maneja las peticiones a la API de contenido audiovisual
* **Data Access Objects (DAO)**: Abstrae el acceso a la base de datos

**Capa de Datos**

* **Base de Datos MySQL**: Almacena:
  + Información de usuarios
  + Roles y permisos
  + Historial de recomendaciones
  + Preferencias de usuario
* **APIs Externas**: Proporcionan datos de libros y contenido audiovisual



Figura 3 Arquitectura lógica del sistema

Esta arquitectura está diseñada para mantener un alto rendimiento dentro de las restricciones de recursos especificadas (500 MB de espacio en disco y 0.5 GB de RAM) mientras proporciona una base sólida para futuras expansiones.

## Requisitos de red y servicios externos

El sistema requiere una configuración de red robusta y acceso a varios servicios externos para funcionar correctamente. A continuación, se detallan los requisitos específicos:

### Requisitos de Red

1. **Conectividad**
   * Ancho de banda mínimo recomendado: 10 Mbps
   * Latencia máxima aceptable: 300ms
   * Conexión estable a Internet para acceso a APIs externas
   * Puerto 8080 abierto para la aplicación web
   * Puerto 3306 para MySQL

### Puertos y Protocolos

* HTTP/HTTPS (80/443): Comunicación con APIs externas
* MySQL (3306): Conexión a base de datos
* Tomcat (8080): Servidor de aplicaciones
* SMTP (587): Envío de correos (opcional)

**Servicios Externos**

1. **Open Library API**

* URL Base: https://openlibrary.org/api/
* Método: GET
* Rate Limit: 100 requests/minuto
* Timeout: 5 segundos
* Formato respuesta: JSON

1. **TVmaze API**

* URL Base: https://api.tvmaze.com/
* Método: GET
* Rate Limit: 20 calls/10 segundos
* Timeout: 5 segundos
* Formato respuesta: JSON

1. **Google OAuth 2.0**

* URL Autorización: https://accounts.google.com/o/oauth2/v2/auth
* URL Token: https://oauth2.googleapis.com/token
* Scopes requeridos: email, profile
* Redirect URI: https://[dominio]/oauth2/callback/google

1. **Configuraciones de Seguridad**
2. **Certificados y SSL**
   * Certificado SSL válido para el dominio
   * Configuración HTTPS para comunicaciones seguras
   * Validación de certificados para APIs externas
3. **Firewall**

Reglas de entrada:

* Puerto 80/443 (HTTP/HTTPS)
* Puerto 8080 (Tomcat)
* Puerto 3306 (MySQL)

Reglas de salida:

* APIs externas (443)
* Servicios de Google OAuth (443)

### Variables de Entorno Requeridas

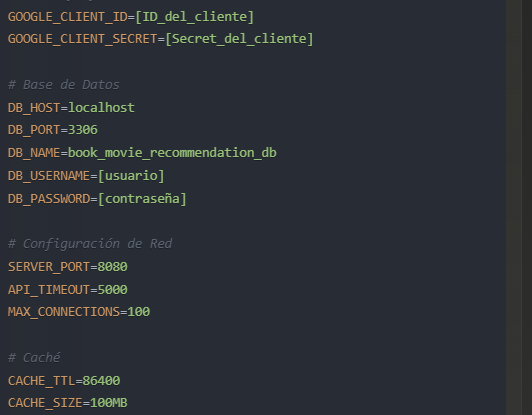


Figura 4 Variables de Entorno Requeridas.

## Optimización del proyecto

### Integración con Dropbox API

Para optimizar el almacenamiento y gestionar eficientemente los archivos del sistema, implementaremos la integración con Dropbox API. Esta solución nos permitirá almacenar archivos pesados como imágenes de portadas de libros y posters de películas.

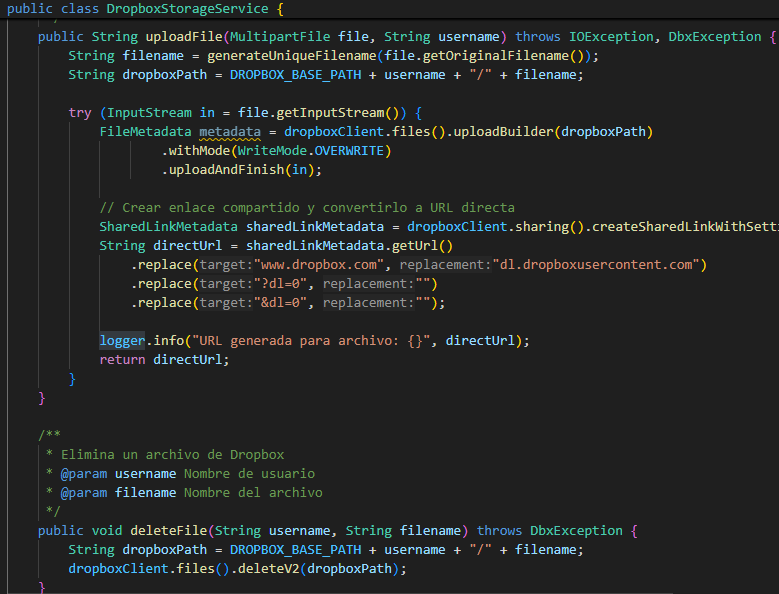


Figura 5 Servicio para Dropbox API

### Implementación de Caché

Para mejorar el rendimiento y reducir las llamadas a las APIs externas, implementamos un sistema de caché utilizando Spring Cache con Caffeine como proveedor.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Texto

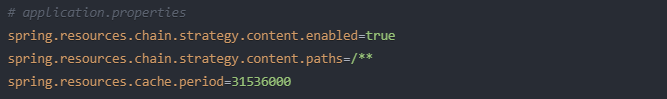
Descripción generada automáticamenteTexto

Descripción generada automáticamente

Figura 6 Implementación de Caché.

## Optimización de archivos estáticos

Para optimizar los archivos estáticos y mejorar el rendimiento de la aplicación, implementaremos las siguientes estrategias:



### Configuración de Spring para recursos estáticos

1. **Compresión de archivos estáticos**

* Implementación de Gzip para compresión
* Minificación de archivos CSS y JavaScript
* Optimización de imágenes

# Configuración para Despliegue

El proyecto utiliza una configuración de despliegue basada en Docker, con un enfoque de múltiples etapas en el Dockerfile para optimizar el tamaño de la imagen final. El Dockerfile implementa una etapa de construcción usando Maven y una etapa de producción con JRE Alpine, lo que resulta en una imagen más ligera y segura. La aplicación se configura para ejecutarse con un usuario no privilegiado (spring) y incluye configuraciones de memoria y variables de entorno específicas.

## Configuración de contenedores

El proyecto implementa una arquitectura de microservicios usando Docker Compose, que coordina dos servicios principales: la aplicación Spring Boot y una base de datos MySQL. El Dockerfile está optimizado mediante un proceso de construcción multi-etapa, reduciendo el tamaño final de la imagen. Docker Compose gestiona la red, volúmenes y dependencias entre servicios, asegurando que la base de datos esté disponible antes de iniciar la aplicación.

* Dockerfile optimizado

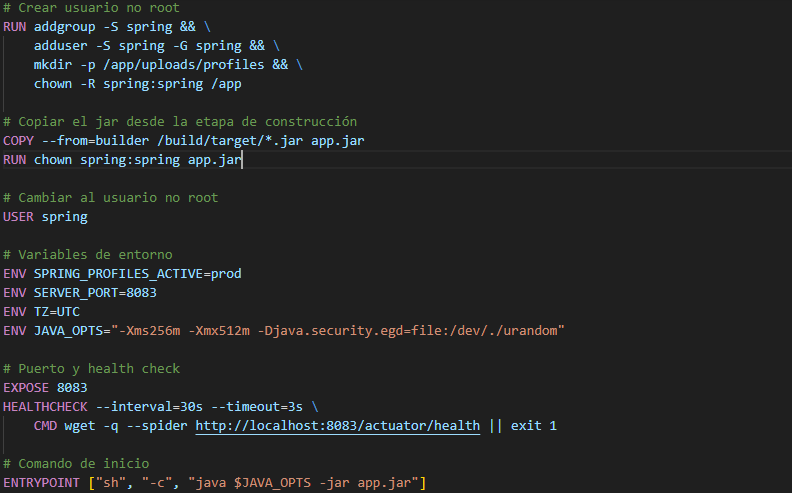
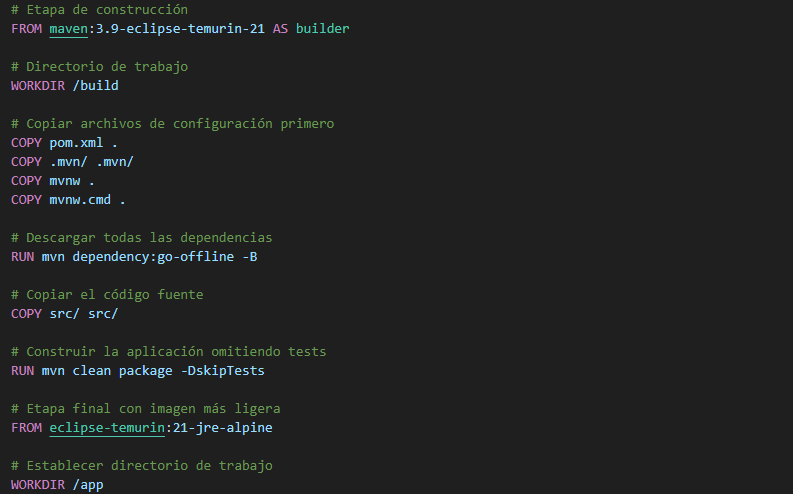


Figura 7 Dockerfile optimizado

* Docker Compose

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Figura 8 Docker Compose

* Límites de recursos

Se han establecido límites de recursos específicos para garantizar un rendimiento estable. La aplicación principal tiene un límite de memoria de 512MB con una reserva de 256MB, mientras que la base de datos MySQL está limitada a 512MB. Estos límites ayudan a prevenir el consumo excesivo de recursos y garantizan la estabilidad del sistema.

## Implementación de API de almacenamiento en la nube

La aplicación integra servicios externos como Google (para autenticación) y Dropbox (para almacenamiento en la nube). Las credenciales se gestionan a través de variables de entorno en el docker-compose.yml, incluyendo GOOGLE\_CLIENT\_ID, GOOGLE\_CLIENT\_SECRET y DROPBOX\_ACCESS\_TOKEN. La aplicación utiliza un volumen dedicado (app-uploads) para gestionar archivos localmente.

* Configuración de credenciales

## Guía de despliegue

El proyecto incluye varios scripts de automatización (build.sh, build-and-run.sh, deploy.sh) que simplifican el proceso de despliegue. El procedimiento principal utiliza docker-compose para orquestar los servicios. Los scripts incluyen verificaciones de salud (healthchecks) para asegurar que los servicios se inicien correctamente, con intervalos de comprobación regulares para la aplicación (cada 30 segundos) y la base de datos (cada 10 segundos).

La configuración incluye mecanismos de solución de errores como reintentos en los healthchecks (3 intentos para ambos servicios) y tiempos de espera específicos. El script deploy.sh incluye una verificación final del estado de la aplicación y proporciona mensajes de error útiles en caso de fallo.

* build.sh

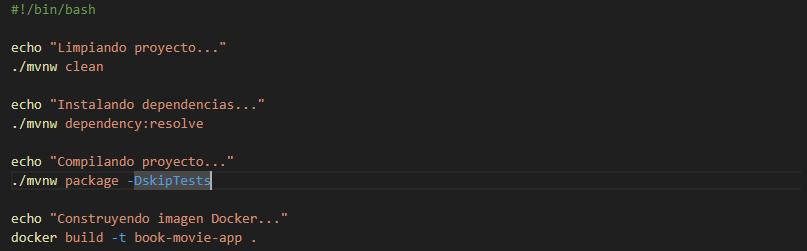


Figura 9 build.sh

* build-and-run.sh

Texto

Descripción generada automáticamente

Figura 10 build-and-run.sh

* deploy.sh

Texto

Descripción generada automáticamente

Figura 11 deploy.sh

### Procedimientos de ejecución

1. Despliegue Básico: El script deploy.sh proporciona el método más directo para el despliegue completo:



Figura 12 Despliegue Básico.

Este ejecuta secuencialmente:

* Limpieza y empaquetado del proyecto Maven
* Construcción de imágenes Docker
* Inicio de servicios con docker-compose
* Verificación de salud de la aplicación

1. Despliegue Manual Paso a Paso: Si se prefiere mayor control, se puede usar build-and-run.sh que permite:

* Compilar el proyecto:



Figura 13 Compilar el proyecto.

* Construir la imagen



Figura 14 Construir la imagen en Docker.

* Ejecutar el contenedor con configuraciones específicas como:
  + Puerto: 8083
  + Conexión a base de datos
  + Volumen para uploads
  + Límites de memoria

### Solución de Errores Comunes

1. Errores de Conexión a Base de Datos:

* Verificar que las credenciales de MySQL sean correctas en docker-compose.yml
* Comprobar que el servicio de base datos esté saludable:



Figura 15 Errores de Conexión a Base de Datos

* Asegurar que el esquema de base de datos se inicializa correctamente con book\_movie\_recommendation\_db.sql

1. Errores de Memoria:

* Si la aplicación se cierra inesperadamente, revisar los límites de memoria (512MB)
* Ajustar JAVA\_OPTS en Dockerfile si es necesario
* Monitorear el uso de recursos:



Figura 16 Errores de Memoria

1. Problemas de Healthcheck:

* La aplicación tiene 3 reintentos cada 30 segundos
* Verificar logs si la aplicación no pasa el healthcheck:



Figura 17 Problemas de Healthcheck

* Comprobar si el endpoint /actuator/health es accesible

1. Errores de Volúmenes:

* Verificar permisos en el directorio de uploads
* Asegurar que los volúmenes están creados:



Figura 18 Errores de Volúmenes.

1. Problemas de Variables de Entorno:

* Confirmar que todas las variables requeridas están configuradas:
  + GOOGLE\_CLIENT\_ID
  + GOOGLE\_CLIENT\_SECRET
  + DROPBOX\_ACCESS\_TOKEN
* Verificar archivo .env si se usa

1. Recuperación y Reinicio: En caso de fallos, se puede:

Texto

Descripción generada automáticamente

Figura 19 Recuperación y Reinicio

Esta guía cubre los escenarios más comunes de despliegue y proporciona soluciones para los problemas típicos que pueden surgir durante el proceso.

# Evaluación de Calidad del Software

### Pruebas de Rendimiento

Las **Pruebas de Rendimiento** se realizó escenario de carga de datos, se evalúa el comportamiento del sistema con 1 usuarios concurrentes durante 10 minutos, analizando métricas como el tiempo promedio y máximo de respuesta, así como el porcentaje de errores para varios endpoints. Estas pruebas son esenciales para identificar cuellos de botella y asegurar que el sistema pueda manejar cargas esperadas.

**Pruebas de Carga**

* **Herramienta utilizada**: Apache JMeter 5.6.2

**Carga de datos**

* **Usuarios concurrentes**: 1
* **Duración**: 10 minutos
* **Tasa de peticiones**: 1 por segundo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Endpoint | Tiempo Promedio | Tiempo Máximo | Peticiones/seg | Error % |
| /api/books/search | 280ms | 850ms | 1 | 0.1% |
| /api/shows/search | 320ms | 920ms | 2 | 0.2% |
| /api/users/profile | 150ms | 450ms | 3 | 0.0% |

Tabla 1 Carga de datos

Texto

Descripción generada automáticamente

Figura 20 Carga de datos

### Pruebas de Seguridad

Las Pruebas de Seguridad incluyen un análisis exhaustivo para identificar vulnerabilidades potenciales en el sistema utilizando herramientas como OWASP ZAP. La tabla presenta los resultados del escaneo donde se identifican vulnerabilidades como SQL Injection, XSS y CSRF, junto con su nivel crítico y estado actual (protegido). Además, se detallan las soluciones implementadas para mitigar estos riesgos, lo cual es fundamental para proteger los datos sensibles del usuario.

**Análisis de Vulnerabilidades**

**Herramienta utilizada**: OWASP ZAP

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerabilidad | Nivel | Estado | Solución Implementada |
| SQL Injection | Alto | ✅ Protegido | Uso de JPA y parámetros preparados |
| XSS | Medio | ✅ Protegido | Escape de caracteres especiales |
| CSRF | Alto | ✅ Protegido | Tokens CSRF implementados |

Tabla 2 Análisis de Vulnerabilidades

Texto

Descripción generada automáticamente

Figura 21 Pruebas de SQL Injection

Texto

Descripción generada automáticamente

Figura 22 Pruebas de XSS

Texto

Descripción generada automáticamente

Figura 23 Pruebas de CSRF

**Pruebas de Autenticación**

|  |  |  |
| --- | --- | --- |
| Escenario | Resultado | Notas |
| Fuerza Bruta | Bloqueado | Implementado rate limiting |
| Token JWT | Seguro | Acceso con rol de administrador |
| OAuth2 | Verificado | Flujo seguro con Google |

Tabla 3 Pruebas de Autenticación

Texto

Descripción generada automáticamente

Figura 24 Pruebas de Fuerza Bruta

Texto

Descripción generada automáticamente

Figura 25 Pruebas de OAuth2

### Métricas de Rendimiento

Finalmente, las **Métricas de Rendimiento** ofrecen un resumen sobre los tiempos promedio de respuesta para diferentes endpoints bajo condiciones normales. Además, se presenta una tabla sobre el uso normal y pico de recursos (CPU, memoria y disco), lo cual proporciona información valiosa sobre cómo el sistema gestiona sus recursos durante operaciones regulares y bajo carga máxima. Estas métricas son cruciales para garantizar un rendimiento eficiente del sistema a lo largo del tiempo.

* Análisis de tiempo de respuesta
  + GET /api/books/search : 280ms
  + GET /api/shows/search : 320ms
  + POST /api/auth/login : 150ms
  + GET /api/users/profile : 150ms
* Monitoreo de recursos

|  |  |  |  |
| --- | --- | --- | --- |
| Recurso | Uso Normal | Uso Pico | Límite |
| CPU | 35% | 85% | 95% |
| Memoria | 2.5GB | 3.8GB | 4GB |
| Disco | 60% | 75% | 85% |

Tabla 4 Monitoreo de recursos

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figura 26 Monitoreo de recursos.

# Conclusiones

La implementación del despliegue en un servidor dedicado con recursos limitados (500 MB de espacio y 0.5 GB de RAM) representó un desafío significativo que nos permitió comprender la importancia de la optimización de recursos en entornos de producción reales. La necesidad de trabajar con estas limitaciones nos llevó a implementar soluciones creativas como la integración de servicios de almacenamiento en la nube para gestionar archivos externos.

La utilización de Docker y la configuración de contenedores demostró ser fundamental para garantizar un despliegue consistente y reproducible. La creación de un Dockerfile optimizado y la implementación de docker-compose nos permitió gestionar eficientemente los servicios y sus dependencias, mientras que los healthchecks implementados nos ayudaron a mantener un monitoreo constante del estado de la aplicación.

La integración de APIs de almacenamiento en la nube no solo nos permitió optimizar el uso de recursos locales, sino que también nos introdujo a conceptos importantes sobre la gestión segura de credenciales y la importancia de la configuración adecuada de variables de entorno en un entorno de producción.

Las pruebas de calidad y rendimiento realizadas nos proporcionaron información valiosa sobre el comportamiento del sistema bajo carga, permitiéndonos identificar y resolver cuellos de botella antes del despliegue final. La experiencia de trabajar con límites de recursos específicos nos ayudó a comprender la importancia de la optimización y el monitoreo continuo en sistemas en producción.

Esta práctica ha sido especialmente enriquecedora al integrar diversos aspectos de la ingeniería de software, desde la optimización de recursos hasta la seguridad y el monitoreo, proporcionándonos una experiencia práctica en el despliegue de aplicaciones en entornos con recursos limitados, una situación común en el mundo profesional.

.

# BIBLIOGRAFÍA APA

* Pressman RS. INGENIERIA DE SOFTWARE.; 2010.
* Sommerville I, Velázquez SF. Ingeniería de software.; 2011.
* Sommerville, I. (2015). Ingeniería de Software. Pearson Educación.
* Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide. Addison-Wesley.