Índice

[Contexto de la aplicación 2](#_Toc197697802)

[Mundo real del problema 2](#_Toc197697803)

[Qué aplicaciones existen 2](#_Toc197697804)

[Mejoras de la aplicación respecto a las existentes 2](#_Toc197697805)

[Casos de Uso - F 2](#_Toc197697806)

[Requisitos funcionales y no funcionales 2](#_Toc197697807)

[Funcionales 2](#_Toc197697808)

[No funcionales 3](#_Toc197697809)

[Diseño 3](#_Toc197697810)

[GUI 3](#_Toc197697811)

[UI (vistas) 3](#_Toc197697812)

[UX (usabilidad) 3](#_Toc197697813)

[Diagrama navegación 3](#_Toc197697814)

[Reutilización (fragmentos futuros) 3](#_Toc197697815)

[Arquitectura 3](#_Toc197697816)

[Despliegue 3](#_Toc197697817)

[Componentes 3](#_Toc197697818)

[Base de datos 3](#_Toc197697819)

[Paquetes, Interfaces y Clases 3](#_Toc197697820)

[Plan de pruebas (cómo) 3](#_Toc197697821)

[Implementación 4](#_Toc197697822)

[Entorno de Desarrollo 4](#_Toc197697823)

[Implantación/Puesta en producción 4](#_Toc197697824)

[Capturas de la ejecución de la funcionalidad 4](#_Toc197697825)

[Información sobre la versión y software necesario 4](#_Toc197697826)

[Elementos destacables del desarrollo 4](#_Toc197697827)

[Conclusiones 4](#_Toc197697828)

[Bibliografía 4](#_Toc197697829)

[Anexos 5](#_Toc197697830)

# Contexto de la aplicación

## Mundo real del problema

Muchos jugadores acumulan videojuegos en diferentes formatos y plataformas, lo que dificulta su organización. Esta aplicación busca centralizar toda esta información en un único lugar, resolviendo problemas como la desorganización, dificultad de localización, y falta de estadísticas.

## Qué aplicaciones existen

Actualmente existen algunas aplicaciones que ayudan a gestionar colecciones de videojuegos, como GG App, HowLongToBeat o Backloggd. Estas apps permiten realizar un seguimiento, pero no ofrecen funcionalidades como el almacenamiento físico. Además, todas son aplicaciones web, esta es una alternativa a nivel de aplicación para la gente que prefiera tener la aplicación a mano, y que no sea como puede ocurrir con las otras, que un fallo en la web puede evitar que accedes a tu biblioteca durante mucho tiempo. Aquí únicamente necesitas conexión a internet y que la web donde se aloja la base de datos funcione, no dependes de las web del servidor de la empresa o posibles ataques a la misma.

## Mejoras de la aplicación respecto a las existentes

La aplicación permite localizar juegos físicos, mostrar estadísticas personalizadas (sobre nosotros, como los géneros más jugados, consola para la que más juegos tienes…), además de funcionalidades como exportar colecciones en PDF y conectar con bases de datos en la nube. Y como mencionamos anteriormente, no dependes tanto de factores externos como el servidor de una web.

## Casos de Uso - F

• Inicio de sesión  
• Añadir videojuegos a la biblioteca  
• Añadir videojuegos a wishlist  
• Filtrar juegos de tu biblioteca  
• Exportar la biblioteca

[AÑADIR DIAGRAMAS DE CASOS DE USO AQUÍ]

## Requisitos funcionales y no funcionales

### Funcionales

• Login obligatorio  
• Permitir guardar, añadir y eliminar (eliminar en progreso) videojuegos.  
• Gestión de wishlist  
• Estadísticas  
• Exportación a PDF  
• Cuentas con bibliotecas propias.

### No funcionales

• Conexión a APIs  
• Conexión a la BBDD en línea  
• Contraseñas hasheadas  
• Escalabilidad

# Diseño

## GUI

### UI (vistas)

Pantallas: login, registro, biblioteca, wishlist, estadísticas, detalle del juego.

### UX (usabilidad)

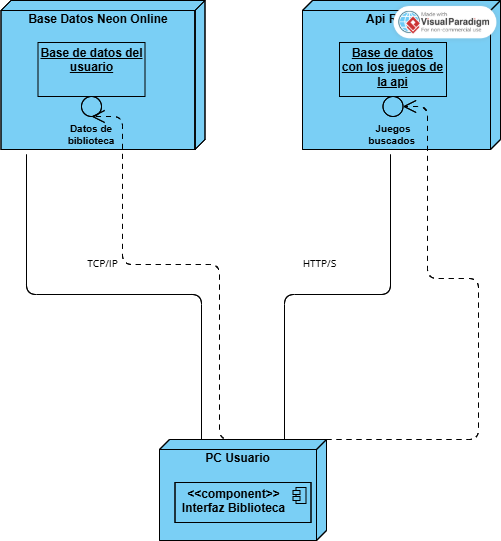
Diseño con JavaFX y Scene Builder, navegación intuitiva con controladores para cada pantalla.

### Diagrama navegación

### 

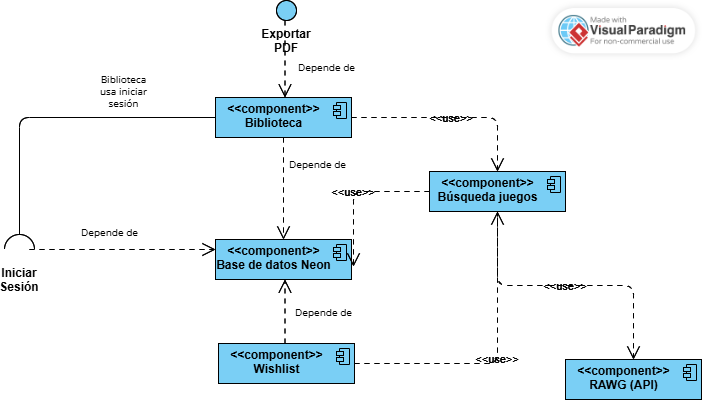
## Arquitectura

### Despliegue

Aplicación de escritorio con base de datos en la nube y consultas a una api:  


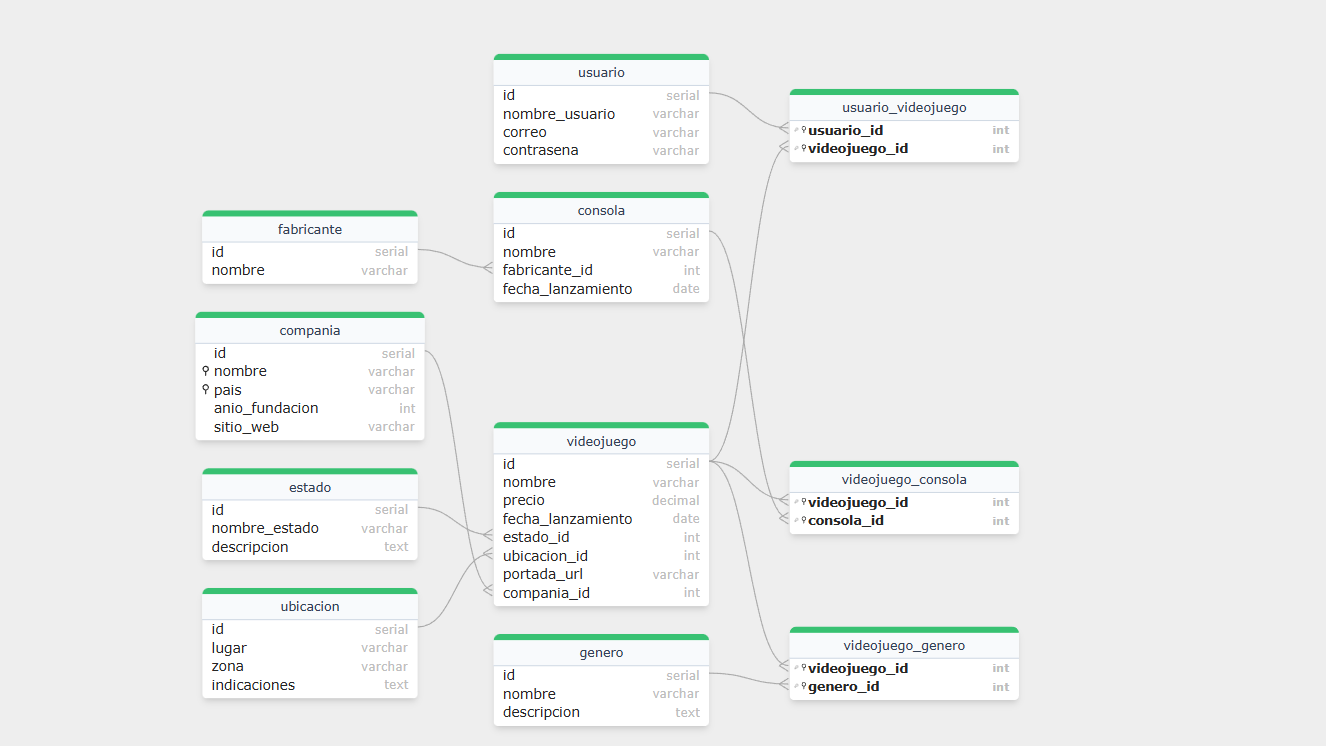
### Componentes

MVC + Spring Boot. Backend arranca junto con JavaFX. También se ha desarrollado un cliente REST utilizando Retrofit para consumir la API de RAWG. Esta API proporciona información detallada y actualizada sobre videojuegos y es fundamental para llenar la base de datos del usuario sin necesidad de introducir manualmente todos los datos. Para el acceso a datos, se emplea Spring Data JPA con Hibernate como implementación. Esto permite abstraer las consultas SQL y aplicar principios de diseño como la separación de responsabilidades entre capas. La comunicación con APIs externas se gestiona mediante una arquitectura basada en Retrofit junto con OkHttp para el manejo de peticiones HTTP. Esto proporciona una forma robusta y mantenible de integrar servicios REST.



## Base de datos

Base de datos PostgreSQL alojada en Neon. Se utiliza Hibernate como ORM de la aplicación, además de JPA para simplificar las consultas PostgreSQL (CRUD) además de anotaciones.



## Paquetes, Interfaces y Clases

En este apartado, además de los paquetes clases e interfaces, me gustaría comentar el uso de Lombok para ahorrar código (constructores y getter y setters sobretodo)

Paquetes: api, dentro de api: retrofit, configuraciones, controlador,modelo, dentro de modelo: ids,repositorio,servicio,utils, resources, vista   
Clases dentro de api: RAWGContenedorPlataforma,RAWGGenero,RAWGPlataforma,RAWGRespuesta,RAWGServicioy RAWGVideojuego.

Clases dentro de retrofit: RAWGCliente

Clases dentro de configuraciones: ConfiguracionBean y ConfiguracionRestTemplate

Clases dentro de controlador: BibliotecaControlador,ControladorBusqueda, LogInControlador y RegistroControlador (faltan clases por crear)

Clases dentro de modelo: Compania, Consola, Estado, Fabricante, Genero, Ubicacion, Usuario, UsuarioVideojuego, Usuario, Videojuego, VideojuegoConsola, VideojuegoGenero.

Clases dentro de ids: UsuarioVideojuegoID, VideojuegoConsolaID, VideojuegoGeneroID

## Plan de pruebas (cómo)

Pruebas unitarias con JUnit, conexión con APIs, pruebas manuales.  
AÑADIR EL PLAN DE PRUEBAS CUANDO SE TENGA.

# Implementación

## Entorno de Desarrollo

Java 22, pues es la versión que es más compatible con las dependencias que necesito utilizer. Visual Studio Code, ya que es un entorno conocido y que contiene todo lo necesario para probar y desarrollar la aplicación, Maven para la gestion de dependencias y no tener que gestionarlas manualmente, Neon como sgbe pues usa postgresql, que es un lenguaje que funciona bien con JPA, aloja la BBDD on-line, para poder acceder a ella sin necesidad de tener la base de datos a nivel local, y tiene una versión gratuita que es suficiente para el Desarrollo de esta app y GitHub para llevar el control de versions, además permite realizar backups para volver a una versión anterior. Para la exportación en formato PDF se usa OpenPDF, Jackson para el manejo de JSON (para las respuestas a las consultas a la api RAWG), Jakarta Validation e Hibernate Validation para las validaciones. Se usa RetroFit para el consume de la api.

## Implantación/Puesta en producción

La aplicación se ejecuta localmente mediante Maven con conexión a una base de datos alojada en la nube. Cuando se desea añadir un videojuego, se realiza una consulta a una api con videojuegos (RAWG), se puede buscar mediante nombre, consola etc. Si pulsas el botón de agregar, se agregará a tu biblioteca. También tiene un sistema de log in que permite tener varios usuarios con sus propias bibliotecas.

# Capturas de la ejecución de la funcionalidad

[AÑADIR CAPTURAS DE EJECUCIÓN Y PRUEBAS FUNCIONALES AQUÍ]

# Información sobre la versión y software necesario

• Java 22  
• Visual Studio Code  
• Neon DB  
• Maven  
• GitHub  
• Scene Builder  
• Conexión a Internet para uso completo

# Elementos destacables del desarrollo

• Integración con APIs para la búsqueda de videojuegos y agregarlos en la bbdd, para no tener necesidad de crear una base de datos propia de donde añadir todos los videojuegos, si no ir rellenando nuestra base de datos extrayendo los datos de la api con los datos de los videojuegos.

• Base de datos en la nube para poder tener acceso a la misma en cualquier situación (siempre y cuando se tenga internet) y evitar problemáticas como perder la base de datos debido a un formateo o tener que reinstalar el SO.

• Estadísticas basadas en nuestra biblioteca y exportación de la biblioteca en formato pdf para poder mostrarla de manera sencilla.

• Uso de JavaFX moderno para poder tener unas interfaces simples pero funcionales

• Modularidad y escalabilidad en caso de querer añadir nuevas funcionalidades.

# Manual de usuario

EN PROCESO

# Conclusiones

EN PROCESO

# Bibliografía

• OpenAI. (2025). \*ChatGPT\*. Para consultas de errores y código básico Recuperado de https://chatgpt.com/   
• Google. (2025). \*Google Search\*. Para buscar diferentes informaciones Recuperado de https://www.google.com/   
• Gemini. (2025). \*Gemini AI\*. Para consultas de errores, usabilidad de interfaces y código básico Recuperado de https://gemini.google.com/  
• Stack Overflow. (2025). \*Stack Overflow\*. Para preguntar y buscar respuestas en los foros Recuperado de https://stackoverflow.com/   
• Neon. (n.d.). \*Neon Docs\*. Para consultas de cómo conectar la app a su BBDD y realizar operaciones Recuperado de https://neon.tech/docs   
• Oracle. (n.d.). \*Java Documentation\*. Para dudas básicas de programación en java Recuperado de https://docs.oracle.com/en/java/   
• RAWG. (n.d.). \*RAWG Video Games Database API\*. Para consultar lo que ofrece la aplicación en cuanto a datos y cómo usarla Recuperado de https://rawg.io/apidocs   
• Visual Paradigm. (n.d.). \*Visual Paradigm Diagram Maker\*. Para la creación de diversos diagramas. Recuperado de https://www.visual-paradigm.com/

# Anexos