UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

HexaGame, biblioteca online de videojuegos mediante mediante Arquitectura Hexagonal

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Selim, Alen

Tutor/a: Soler Bayona, José Vicente

CURSO ACADÉMICO: 2022/2023

Resumen

Este proyecto consiste en analizar, diseñar e implementar la fase inicial de una biblioteca online orientada a videojuegos. El principal objetivo es agilizar la revisión del catálogo de videojuegos que un cliente tiene en posesión, que muchos de ellos hoy en día utilizan cuadernos, tablas Excel o simplemente ni siquiera tienen un control real de su colección.

Los clientes de Hexagames son aquellos coleccionistas que desean tener apuntado hasta el más mínimo detalle de su colección de videojuegos, tener a unos pocos clics un comparador de precios y poder acceder de forma sencilla a los juegos que desea incorporar a su colección en un futuro.

A diferencia de otras aplicaciones, Hexagames intenta ser lo más completa posible frente a sus competidores, ya que algunos de ellos incorporan funcionalidades asombrosas, pero tienen un coste muy elevado debido a que no llegan a tener una competencia real.

En los últimos años, hay una tendencia al alza orientando el desarrollo de las aplicaciones hacia el entorno web debido a la gran versatilidad que ofrecen los nuevos lenguajes de programación, más y más herramientas comienzan a estar muy orientadas a este entorno, y, sobre todo, el gran avance que tenemos hoy en día con internet.

Hexagames está desarrollada con 2 de los frameworks más potentes que son AngularJS y SpringBoot, junto a otros lenguajes y tecnologías punteras. Como continuidad del proyecto después de la entrega del TFG, se pretende incorporar 2 nuevas tecnologías punteras como son SwiftUI y Kotlin para la implementación de las versiones móviles de forma nativa.

Palabras Clave

Arquitectura, Hexagonal, Angular, Spring Boot, Videojuegos.

Abstract

This project involves analyzing, designing, and implementing the initial phase of an online library oriented towards video games. The main objective is to streamline the review of the video games catalog that a client possesses, as many of them currently use notebooks, Excel spreadsheets, or simply do not have real control over their collection.

Hexagames' clients are collectors who want to have every detail of their video games collection recorded, have a price comparator just a few clicks away, and easy access to games they wish to add into their collection in the future.

Unlike other applications, Hexagames strives to be as comprehensive as possible compared to its competitors, as some of them offer amazing features but come at a high cost due to a lack of real competition.

In recent years, there has been a growing trend towards web-based applications development due to the versatility offered by new programming languages. More and more tools are becoming highly oriented towards this environment, especially given the significant role that the internet plays today.

Hexagames is developed using two of the most powerful frameworks available today, which are AngularJS and SpringBoot, along with other cutting-edge languages and technologies. As a continuation of this project after the TFG delivery, the plan is to incorporate two additional cutting-edge technologies, named SwiftUI and Kotlin, for the implementation of native mobile versions.

Key words

Architecture, Hexagonal, Angular, Spring Boot, Videogames.

Índice

1. Introducción

1.1 Presentación

Mi nombre es Alen, soy desarrollador de un framework implementado con Spring Boot y me apasiona mucho el mundo de la programación. Cada día me dedico a buscar y a aprender las últimas novedades que giran en torno a Spring Boot, nuevas características de la arquitectura de software o bien sencillamente aprender nuevos lenguajes por mi cuenta.

1.2 Motivación

Una de mis otras grandes pasiones es coleccionar videojuegos y hasta la fecha, usaba una tabla Excel para tener control sobre mi colección. Un día, junto a unos amigos también coleccionistas, todos compramos el mismo juego y nos dimos cuenta horas más tarde que ya lo teníamos comprado. A raíz de esto, surgió la idea de tener una aplicación para poder tener un mejor control de nuestras colecciones. Con esta situación nació HexaGames.

Como valor añadido, los amigos coleccionistas que tengo también son desarrolladores y queremos llevar HexaGames mucho más allá, teniendo el objetivo de crear una app propia para móviles iOS como Android.

1.3 Marco del proyecto

Este Trabajo Fin de Grado consiste en realizar un análisis y diseño completo de una aplicación web que consistirá en una biblioteca online de videojuegos para que los coleccionistas puedan conocer y actualizar fácilmente su biblioteca de juegos. También consistirá en el desarrollo de la primera fase de HexaGames.

La aplicación web será una herramienta encargada de poder dar de alta videojuegos separados por plataformas (PlayStation, Xbox o Nintendo), editar un videojuego, consultar la lista de videojuegos en posesión o bien eliminar un videojuego. También tendrá un sistema de autenticación mediante usuario y contraseña.

1.4 Objetivo

El principal objetivo es el análisis, diseño y desarrollo de la primera fase de la aplicación web, que corresponderá al desarrollo del backend, que será agnóstico a cualquier base de datos que utilicemos, aunque en este proyecto tengamos PostgreSQL y la implementación inicial del frontend.

El proyecto ha sido concebido de manera agnóstica en cuanto a su despliegue en la nube y a la base de datos, abarcando tanto el backend como el frontend. Esta cuidadosa planificación garantiza que la implementación de la solución pueda llevarse a cabo de manera fluida en cualquier entorno de nube, permitiendo una transición sin contratiempos mientras preservamos la funcionalidad y la estética del proyecto en todo momento.

El sistema debe ser capaz de dar de alta usuarios nuevos, dar de alta videojuegos, editarlos y eliminarlos, visualización de los videojuegos de cada usuario en tablas y separados por plataformas en su fase inicial facilitando una interfaz sencilla y cómoda para el usuario.

El proyecto se divide en varios subobjetivos:

- Diseño de los modelos de datos en el backend, con los cuales tendremos también las tablas de base de datos.
- Implementación del contrato Swagger con los distintos endpoints para comenzar el desarrollo del backend.
- Desarrollo del backend empleando la arquitectura hexagonal.
- Desarrollo del frontend mediante la arquitectura clean en tres capas.

1.5 Estructura del documento

En el segundo punto se realiza un análisis de los lenguajes y tecnologías utilizadas dentro del proyecto y una explicación de cómo se han utilizado e implementado dentro del desarrollo del proyecto junto a varias pruebas de investigación realizadas.

En el tercer punto se analiza con detenimiento la arquitectura hexagonal del backend y la arquitectura por capas del frontend, y, como se compone la arquitectura del proyecto.

En el cuarto punto se realiza un estudio de mercado de potenciales competidores en los cuales se destaca una aplicación por encima de las demás competencias.

En el quinto punto se realiza un análisis de la aplicación, detallando análisis funcional, no funcional y describiendo los casos de uso.

El sexto punto se compone de los diagramas de flujo y los diagramas de bases de datos que se han implementado.

En el séptimo punto se detalla cómo se ha conseguido la agnosticidad en el despliegue de la aplicación en cualquier nube.

En el octavo punto se detallan las pruebas que se han implementado en el desarrollo de la aplicación para garantizar la calidad y funcionamiento del código.

En el noveno punto se detalla la relación entre las asignaturas cursadas con el proyecto desarrollado.

Y por último en el décimo punto veremos las conclusiones extraídas durante el desarrollo del proyecto y el trabajo futuro que se realizará a continuación de la entrega del TFG.

2. Lenguajes y herramientas

2.1 Angular

Angular [1] es un framework de desarrollo de aplicaciones web de código abierto, creado y mantenido por Google. Se utiliza para construir aplicaciones de una sola página (Single Page Applications o SPAs) que se ejecutan en el navegador web. Angular es uno de los frameworks más populares para el desarrollo frontend, junto con React y Vue.js. Como se señala en el libro "*Angular in Action*" de Jeremy Wilken [2]:

"*Angular ha evolucionado y se ha convertido en una herramienta esencial para el desarrollo web moderno. Su arquitectura basada en componentes, su sistema de enlace de datos bidireccional y su amplia comunidad de desarrolladores lo convierten en una elección sólida para proyectos web de todos los tamaños y complejidades*".

Características clave de Angular:

1. **Componentes**: Angular utiliza el concepto de componentes para dividir la interfaz de usuario en piezas reutilizables y modulares. Cada componente representa una parte de la interfaz y puede contener su propia lógica y plantilla.

2. **Data Binding**: Angular proporciona un enlace bidireccional entre los datos y la interfaz de usuario. Esto significa que los cambios en los datos se reflejan automáticamente en la interfaz de usuario y viceversa, lo que simplifica el manejo de la lógica de la aplicación.

3. **Inyección de dependencias**: Angular utiliza un sistema de inyección de dependencias para gestionar la creación y la inyección de objetos y servicios en toda la aplicación. Esto ayuda a mantener la modularidad y facilita las pruebas unitarias.

4. **Directivas**: Las directivas son etiquetas especiales que se utilizan para extender la funcionalidad de HTML. Angular proporciona directivas integradas y permite crear directivas personalizadas.

5. **Routing**: Angular incluye un sistema de enrutamiento para gestionar la navegación dentro de la aplicación y cambiar dinámicamente las vistas sin recargar toda la página.

6. **Gestión de formularios**: Angular ofrece un poderoso módulo para la creación y validación de formularios, lo que facilita la captura y validación de datos del usuario.

7. **Pipes**: Los pipes (tuberías) son transformadores de datos que permiten formatear los valores mostrados en la interfaz de usuario.

8. **Módulos**: Angular organiza la aplicación en módulos, lo que permite dividir la aplicación en piezas más pequeñas y facilita el mantenimiento y la colaboración en equipos.

Angular se basa en el lenguaje TypeScript, que es una extensión de JavaScript que añade características adicionales y tipado estático para mejorar el desarrollo y la detección de errores tempranos. Esta combinación de tecnologías permite crear aplicaciones web robustas, escalables y de alto rendimiento.

2.2 Spring Boot

Spring Boot [3] es un framework de código abierto para el desarrollo de aplicaciones Java que facilita la creación de aplicaciones con Spring de manera rápida y sencilla. Está basado en el

popular framework Spring, que proporciona un conjunto de herramientas y bibliotecas para desarrollar aplicaciones empresariales en Java.

Como se menciona en "*Spring Boot in Action*" de Craig Walls [4]:

"*Spring Boot ha revolucionado la forma en que los desarrolladores abordan el desarrollo de aplicaciones web en Java. Ofrece un enfoque basado en convenciones en lugar de configuraciones, reduciendo la necesidad de código repetitivo y proporcionando un sólido ecosistema para construir aplicaciones escalables y listas para producción.*"

Características clave de Spring Boot:

1. **Facilidad de configuración**: Spring Boot permite una configuración automática y por convención, lo que significa que muchas configuraciones se realizan automáticamente basándose en las dependencias presentes en el proyecto. Esto reduce la necesidad de configurar manualmente cada componente de la aplicación.

2. **Incorporación de dependencias**: Spring Boot utiliza el concepto de "starter dependencies" (dependencias de inicio) para proporcionar conjuntos de bibliotecas y configuraciones preconfiguradas para diferentes tipos de aplicaciones (por ejemplo, aplicaciones web, aplicaciones de datos, aplicaciones de seguridad, etc.). Esto facilita el inicio de nuevos proyectos y reduce la complejidad al agregar funcionalidades adicionales.

3. **Incorporación de servidor embebido**: Spring Boot incluye un servidor web embebido, como Tomcat, Jetty o Undertow, lo que significa que no es necesario desplegar la aplicación en un servidor externo. Con un simple comando, la aplicación se puede ejecutar directamente como una aplicación independiente.

4. **Actuación y monitorización**: Spring Boot proporciona herramientas para medir y mejorar el rendimiento de la aplicación. Además, ofrece integración con herramientas de monitorización y gestión de logs.

5. **Gestión de propiedades**: Spring Boot facilita la gestión de propiedades de la aplicación, permitiendo configurarlas en diferentes entornos (desarrollo, pruebas, producción) y modificarlas mediante propiedades externas o variables de entorno.

6. **Soporte para gestión de dependencias**: Spring Boot integra herramientas de gestión de dependencias como Maven o Gradle, lo que facilita la gestión de las bibliotecas y dependencias utilizadas en el proyecto.

7. **Seguridad**: Spring Boot proporciona características de seguridad integradas, como autenticación y autorización, para proteger las aplicaciones y los servicios.

En resumen, Spring Boot es una herramienta poderosa y popular en el mundo de desarrollo en Java. Su enfoque en la facilidad de configuración, productividad y el desarrollo de aplicaciones empresariales robustas lo ha convertido en una elección común para el desarrollo de aplicaciones web, servicios RESTful, microservicios y otros tipos de aplicaciones Java.

2.3 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto y gratuito. Es conocido por su robustez, escalabilidad, y cumplimiento con los estándares SQL. PostgreSQL es muy utilizado en el mundo empresarial y en aplicaciones web debido a su capacidad para manejar grandes volúmenes de datos y sus características avanzadas.

Según el sitio web oficial de PostgreSQL [5]:

"*PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto que ha ganado una reputación sólida por su arquitectura probada, su confiabilidad, integridad de datos y características avanzadas. Se utiliza en una amplia variedad de aplicaciones empresariales y es conocido por su capacidad para gestionar grandes conjuntos de datos y cargas de trabajo complejas.*"

Características clave de PostgreSQL:

1. **Modelo relacional**: PostgreSQL utiliza el modelo relacional, que organiza los datos en tablas con filas y columnas, permitiendo establecer relaciones entre ellas mediante claves primarias y foráneas.

2. **SQL completo**: PostgreSQL soporta el lenguaje SQL completo, lo que significa que cumple con los estándares SQL y también ofrece extensiones y características propias.

3. **Escalabilidad**: PostgreSQL es conocido por su capacidad para manejar grandes cantidades de datos y operaciones concurrentes. Es apto para aplicaciones con altos volúmenes de tráfico y datos.

4. **Transacciones ACID**: PostgreSQL garantiza la integridad de los datos mediante el soporte de transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que garantiza que las operaciones sean consistentes y seguras incluso en escenarios de fallos o errores.

5. **Extensiones y funciones**: PostgreSQL permite crear y usar extensiones, lo que facilita la personalización y ampliación de sus capacidades. Además, ofrece una amplia variedad de funciones y operadores incorporados que hacen más eficiente el trabajo con los datos.

6. **Tipos de datos avanzados**: Además de los tipos de datos estándar, PostgreSQL ofrece una amplia gama de tipos de datos avanzados, incluyendo tipos geométricos, tipos de datos para direcciones IP, matrices, tipos JSON y JSONB, entre otros.

7. **Índices y optimización de consultas**: PostgreSQL ofrece una variedad de índices y herramientas para optimizar consultas, lo que mejora significativamente el rendimiento de las operaciones de búsqueda y filtrado.

8. **Replicación y alta disponibilidad**: PostgreSQL permite configurar replicación para garantizar la alta disponibilidad de los datos y la tolerancia a fallos.

9. **Soporte para funciones y procedimientos almacenados**: PostgreSQL admite funciones almacenadas y procedimientos almacenados que permiten ejecutar lógica de negocios directamente en la base de datos.

En resumen, PostgreSQL es una opción sólida y confiable para sistemas de bases de datos que requieren rendimiento, escalabilidad y un enfoque en la integridad y seguridad de los datos. Es ampliamente utilizado en una variedad de aplicaciones, desde pequeñas aplicaciones personales hasta grandes sistemas empresariales.

2.4 Docker

Docker es una plataforma de contenedorización que se utiliza para desarrollar, enviar y ejecutar aplicaciones de manera consistente y eficiente en entornos de desarrollo, pruebas y producción. Los contenedores Docker proporcionan un entorno ligero y aislado en el que se pueden empaquetar aplicaciones y todas sus dependencias, lo que facilita su portabilidad y distribución.

Como lo describe Solomon Hykes, el fundador de Docker [6]:

"*Docker simplifica el desarrollo, la implementación y la ejecución de aplicaciones al permitir que los contenedores ligeros y portátiles se ejecuten de manera consistente en cualquier entorno. Esta abstracción de nivel de sistema operativo ha cambiado la forma en que construimos, entregamos y ejecutamos aplicaciones.*"

Aquí hay conceptos clave relacionados con Docker:

1. **Contenedor:** Un contenedor Docker es una instancia aislada y ejecutable de una aplicación junto con su entorno y dependencias, todo encapsulado en una unidad autónoma. Los contenedores son ligeros, eficientes y se ejecutan de manera consistente en cualquier entorno que admita Docker.

2. **Imagen:** Una imagen de Docker es un paquete de solo lectura que contiene el código de la aplicación, sus dependencias y la configuración necesaria para ejecutarla. Las imágenes son la base para crear contenedores. Puedes pensar en ellas como plantillas preconfiguradas para contenedores.

3. **Docker Engine:** Es el componente principal de Docker que gestiona la creación y ejecución de contenedores. Incluye un servidor y una interfaz de línea de comandos (CLI) para interactuar con Docker.

4. **Dockerfile:** Un Dockerfile es un archivo de texto que contiene instrucciones para construir una imagen de Docker. Define cómo se configura y se construye un contenedor a partir de una imagen base.

5. **Registro Docker:** Un registro es un repositorio en línea que almacena imágenes de Docker. Docker Hub es el registro público más utilizado, pero también puedes configurar registros privados para almacenar y distribuir imágenes de manera segura.

6. **Orquestación:** Docker Swarm y Kubernetes son herramientas de orquestación que permiten administrar y escalar contenedores en clústeres de servidores para entornos de producción a gran escala.

Las ventajas de Docker incluyen:

- **Portabilidad:** Los contenedores Docker son independientes del entorno, lo que facilita su ejecución en diferentes sistemas operativos y entornos de desarrollo.

- **Aislamiento:** Los contenedores proporcionan un alto grado de aislamiento entre aplicaciones, lo que evita conflictos de dependencias y problemas de compatibilidad.

- **Rapidez y eficiencia:** Los contenedores son más livianos que las máquinas virtuales, lo que permite una rápida creación y puesta en marcha de aplicaciones.

- **Escalabilidad:** Docker facilita la escalabilidad horizontal, lo que significa que puedes aumentar o reducir fácilmente la cantidad de contenedores según la demanda.

Docker se ha convertido en una tecnología fundamental en el desarrollo y la administración de aplicaciones, especialmente en entornos de microservicios y DevOps, donde la portabilidad y la eficiencia son cruciales.

2.5 ADFS

ADFS

ADFS es una sigla que se refiere a "Active Directory Federation Services", en español, "Servicios de Federación de Active Directory". ADFS es un servicio de Microsoft que permite a las organizaciones establecer confianza entre diferentes sistemas de autenticación, lo que facilita la autenticación única (Single Sign-On o SSO) y la federación de identidades.

Según la documentación oficial de Microsoft [7]:

"*ADFS es una solución de servidor que permite compartir identidades de forma segura entre organizaciones y aplicaciones. Facilita la colaboración empresarial y el acceso a recursos externos, permitiendo a los usuarios acceder a aplicaciones y servicios con un único inicio de sesión.*"

Las características y funciones principales de ADFS son las siguientes:

1. **Autenticación Única (SSO)**: ADFS permite a los usuarios acceder a múltiples aplicaciones y servicios con una sola autenticación. Esto significa que los usuarios inician sesión una vez y pueden acceder a todas las aplicaciones y recursos que tengan permisos sin necesidad de volver a autenticarse.
2. **Federación de Identidad**: ADFS establece confianza entre organizaciones o dominios diferentes, lo que permite que los usuarios de una organización accedan a recursos en otra organización sin necesidad de crear y mantener múltiples cuentas y contraseñas.
3. **Integración con Active Directory**: ADFS se integra estrechamente con Active Directory, lo que facilita la administración de identidades y la sincronización de usuarios y grupos entre diferentes sistemas.

4. **Seguridad**: ADFS utiliza estándares de seguridad como SAML (Security Assertion Markup Language) y WS-Federation para garantizar la seguridad de las transacciones de autenticación y autorización.
5. **Soporte para Aplicaciones Variadas**: ADFS es compatible con una variedad de aplicaciones, incluyendo aplicaciones web, aplicaciones móviles y servicios en la nube.
6. **Control de Acceso**: Permite a los administradores definir políticas de acceso para determinar quién tiene acceso a qué recursos y en qué condiciones.

   ADFS es especialmente útil en entornos empresariales donde se utilizan múltiples aplicaciones y servicios que requieren autenticación, ya que simplifica el proceso de inicio de sesión para los usuarios y mejora la seguridad al centralizar la gestión de identidades y el control de acceso. También es una tecnología fundamental en la implementación de SSO y la federación de identidades en organizaciones que utilizan servicios en la nube y aplicaciones de terceros.

2.6 Envoy

"Envoy" se refiere comúnmente a Envoy Proxy, que es un servidor proxy de código abierto diseñado para ser utilizado en entornos de microservicios y contenedores. Envoy Proxy es un componente clave en la infraestructura de servicios moderna y se utiliza para gestionar el tráfico de red entre servicios en una arquitectura de microservicios.

Según Matt Klein, el creador de Envoy [8]:

"*Envoy es un proxy de servicio de alto rendimiento que facilita la comunicación entre los servicios de aplicaciones. Ofrece una arquitectura extensible y una variedad de características para resolver problemas comunes en el mundo de las aplicaciones distribuidas.*"

Las características y funciones importantes de Envoy:

1. **Proxy de Red**: Envoy actúa como un intermediario entre servicios, enrutando las solicitudes de manera eficiente, manejando la carga de trabajo de red y facilitando la comunicación entre microservicios.
2. **Balanceo de Carga**: Envoy ofrece capacidades de balanceo de carga para distribuir las solicitudes de manera uniforme entre las instancias de un servicio, mejorando así la escalabilidad y la redundancia.
3. **Descubrimiento de Servicios**: Puede integrarse con sistemas de descubrimiento de servicios como Consul o Kubernetes para mantener una lista actualizada de los servicios disponibles y sus ubicaciones.
4. **Seguridad**: Envoy ofrece funciones de seguridad, como el cifrado de comunicaciones y la autenticación de servicios, para proteger el tráfico entre los servicios.
5. **Observabilidad**: Proporciona métricas y registros detallados que permiten el monitoreo y la depuración de problemas de red y rendimiento.
6. **Flexibilidad**: Envoy es altamente configurable y admite una variedad de protocolos, lo que lo hace adecuado para una amplia gama de aplicaciones y casos de uso.

Envoy Proxy es especialmente popular en el ámbito de las arquitecturas de microservicios y contenedores, donde la gestión del tráfico entre servicios es esencial para garantizar un funcionamiento suave y escalable de las aplicaciones distribuidas. Fue desarrollado inicialmente por Lyft y se ha convertido en un proyecto de código abierto ampliamente adoptado, respaldado por la Cloud Native Computing Foundation (CNCF).

2.7 Kafka

Kafka es una plataforma de streaming de datos de código abierto desarrollada por Apache Software Foundation. Se utiliza para la ingesta, almacenamiento, procesamiento y transmisión en tiempo real de flujos de datos, lo que lo convierte en una herramienta fundamental en la arquitectura de datos en tiempo real y en el procesamiento de eventos.

Según Jay Kreps, uno de los creadores de Kafka [9]:

"*Kafka se diseñó desde el principio para ser una plataforma de transmisión de eventos distribuida y altamente escalable. Permite a las organizaciones gestionar eficazmente flujos de datos en tiempo real y desbloquear el potencial de aplicaciones en tiempo real.*"

Las características clave de Kafka son las siguientes:

1. **Publicación y Suscripción**: Kafka se basa en un modelo de publicación y suscripción, donde los productores de datos envían mensajes a "temas" y los consumidores se suscriben a estos temas para recibir y procesar los mensajes.

2. **Escalabilidad y Tolerancia a Fallos**: Kafka es altamente escalable y tolerante a fallos. Permite agregar nodos a un clúster para aumentar la capacidad y garantizar la disponibilidad de datos incluso en situaciones de fallo.

3. **Retención de Datos**: Kafka retiene los datos durante un período configurable, lo que permite el procesamiento retrospectivo de eventos pasados o la recuperación en caso de interrupciones.

4. **Procesamiento en Tiempo Real**: Se utiliza en combinación con herramientas de procesamiento en tiempo real, como Apache Flink o Apache Spark Streaming, para procesar eventos a medida que llegan.

5. **Flujo de Datos Multitema**: Permite organizar los datos en múltiples temas, lo que facilita la gestión de diferentes flujos de datos en una sola plataforma.

6. **Durabilidad**: Kafka garantiza que los datos se almacenan de manera duradera antes de ser consumidos, lo que reduce la pérdida de datos en caso de fallos.

7. **Ecosistema Rico**: Kafka cuenta con un ecosistema de herramientas y bibliotecas que amplían su funcionalidad, incluyendo Confluent (una plataforma empresarial basada en Kafka), Kafka Streams (para procesamiento de eventos en Kafka) y Connectors (para integrar Kafka con otras fuentes y destinos de datos).

Kafka se utiliza en una amplia variedad de aplicaciones, como el seguimiento de registros de aplicaciones, la ingesta de datos de sensores en tiempo real, la transmisión de datos en aplicaciones de análisis y la integración de sistemas en tiempo real. Es especialmente valioso en escenarios donde se requiere una alta fiabilidad y escalabilidad en la gestión de flujos de datos.

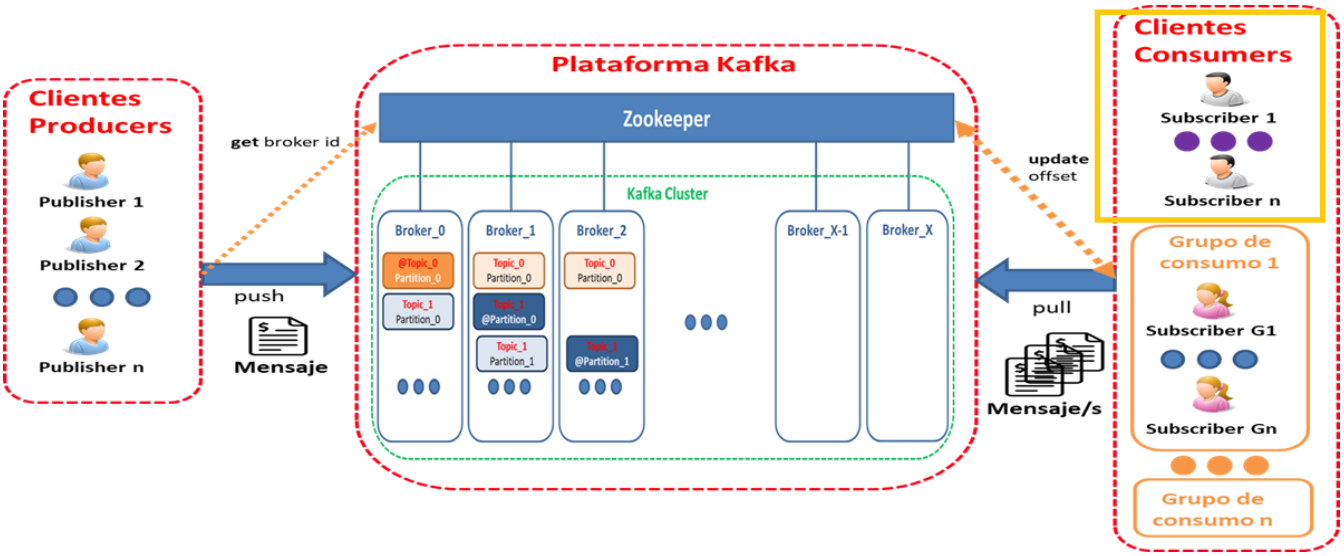


Figura 1: pequeño detalle del funcionamiento interno de Kafka

2.8 Apache Maven

Maven es una herramienta de gestión de proyectos de código abierto ampliamente utilizada en el desarrollo de software. Fue desarrollada por la Apache Software Foundation y se utiliza para facilitar la construcción, la gestión de dependencias y la administración de proyectos de software en una variedad de lenguajes de programación, aunque es más comúnmente asociada con proyectos en Java.

Según la documentación oficial de Apache Maven [10]:

"*Maven es una herramienta de construcción y gestión de proyectos que brinda un enfoque coherente y una forma efectiva de gestionar proyectos de software. Con Maven, los desarrolladores pueden centrarse en escribir código, mientras Maven se encarga de tareas como la gestión de dependencias, la compilación y la distribución.*"

Las características clave de Maven incluyen:

1. **Gestión de Dependencias**: Maven simplifica la gestión de dependencias al proporcionar un sistema de administración de dependencias y un repositorio centralizado para bibliotecas comunes. Los desarrolladores pueden especificar las dependencias de su proyecto en un archivo de configuración (pom.xml) y Maven se encargará de descargar automáticamente las bibliotecas necesarias y gestionar las versiones.

2. **Ciclo de Vida de Construcción**: Define un ciclo de vida de construcción estándar que incluye fases como compilación, empaquetado, prueba y despliegue. Esto permite la automatización de tareas comunes de desarrollo, como la compilación de código fuente, la ejecución de pruebas y la generación de artefactos desplegables.

3. **Repositorio Central**: Utiliza un repositorio centralizado (Central Repository) que almacena bibliotecas y plugins de uso común. Esto facilita la reutilización de bibliotecas y plugins por parte de múltiples proyectos y evita la necesidad de descargar las mismas bibliotecas una y otra vez.

4. **Convenciones y Estructura de Proyectos**: Maven promueve la organización de proyectos siguiendo convenciones y estándares, lo que facilita la legibilidad y la consistencia del código. Los proyectos Maven suelen seguir una estructura de directorios predefinida.

5. **Extensibilidad**: Maven es altamente extensible y permite la creación de plugins personalizados para satisfacer las necesidades específicas de un proyecto. Esto significa que los desarrolladores pueden agregar funcionalidades personalizadas a sus flujos de trabajo de construcción.

En resumen, Maven es una herramienta que automatiza muchas de las tareas comunes de construcción y gestión de proyectos en el desarrollo de software, lo que simplifica el proceso de desarrollo y facilita la gestión de dependencias, la compilación y la distribución de aplicaciones.

Como la arquitectura hexagonal necesita varios módulos para garantizar la agnosticidad de bases de datos y para crear el núcleo del domino de la aplicación, vamos a crear un proyecto Maven Multimódulo.

2.8.1 Maven Multimódulo

Un proyecto Maven multimodular es un tipo de estructura de proyecto en la que un proyecto principal (padre) contiene varios subproyectos (módulos) relacionados. Cada módulo puede representar una parte independiente de la funcionalidad del proyecto, como un componente de software, una biblioteca, una aplicación o cualquier otro componente lógico.

Las características clave de un proyecto Maven multimodular incluyen:

1. **Jerarquía de Proyectos**: En un proyecto multimodular, existe una jerarquía de proyectos en la que un proyecto principal (padre) actúa como el contenedor para uno o más subproyectos (módulos).
2. **Gestión de Dependencias**: El proyecto principal puede gestionar las dependencias comunes que son compartidas por los módulos, lo que evita la duplicación de configuraciones de dependencias en cada módulo. Esto facilita la coherencia en la gestión de dependencias.
3. **Construcción y Despliegue Conjuntos**: Puedes construir y desplegar todos los módulos en conjunto desde el proyecto principal. Esto significa que puedes compilar, probar y empaquetar todos los módulos en una sola operación.
4. **Independencia de Módulos**: Aunque los módulos pueden estar relacionados, son independientes en términos de desarrollo y construcción. Cada módulo tiene su propio POM

(Project Object Model) que especifica sus propias dependencias, configuración y objetivos de construcción.

5. **Facilita la Organización**: Los proyectos multimodulares son útiles para organizar y gestionar proyectos complejos o aplicaciones grandes con múltiples componentes. Cada módulo puede centrarse en una parte específica de la funcionalidad, lo que facilita la colaboración entre equipos o desarrolladores.

2.9 Herramientas para Testing Unitario

JUnit y Mockito son dos herramientas ampliamente utilizadas en el desarrollo de software en el entorno Java para realizar pruebas unitarias y pruebas de integración. Cada una de estas herramientas cumple una función específica en el proceso de pruebas de software:

1. **JUnit**:
    - **¿Qué es JUnit?**: JUnit es un marco de pruebas unitarias de código abierto diseñado para Java. Proporciona un entorno para escribir y ejecutar pruebas unitarias de manera automatizada.
    - **Función Principal**: JUnit permite a los desarrolladores escribir pruebas para verificar que las unidades individuales de código, como métodos o clases, funcionen correctamente. Estas pruebas se ejecutan de forma automatizada y proporcionan retroalimentación rápida sobre si el código cumple con las expectativas definidas.
    - **Características Clave**:
        - Define anotaciones como **@Test** para marcar métodos de prueba.
        - Proporciona aserciones (assertions) para verificar resultados esperados.
        - Permite la organización de pruebas en suites.
        - Facilita la ejecución de pruebas en IDEs y herramientas de construcción como Maven y Gradle.
2. **Mockito**:
    - **¿Qué es Mockito?**: Mockito es un marco de pruebas de código abierto que se utiliza para crear objetos simulados (mocks) en pruebas unitarias. Los mocks se utilizan para simular el comportamiento de componentes o dependencias externas en las pruebas unitarias.

- **Función Principal**: Mockito permite crear objetos simulados que actúan como sustitutos de componentes del sistema real. Estos mocks se utilizan para aislar la unidad bajo prueba y verificar su comportamiento en un entorno controlado.
- **Características Clave**:
  - Facilita la creación de mocks de manera sencilla.
  - Permite definir comportamientos esperados para los mocks.
  - Facilita la verificación de llamadas a métodos en los mocks.

2.10 Aplicación de las tecnologías en el proyecto

En HexaGames, el proyecto frontend está hecho en Angular, concretamente la versión 14.2.12 en la cual usé principalmente la librería Material para implementar la visualización de las tablas de datos.

El proyecto backend, está hecho con Spring Boot 3.1, utilizando la arquitectura hexagonal que detallaré más adelante.

Como base de datos, se ha utilizado PostgreSQL mediante una imagen Docker.

Arrancando la imagen Docker y con la estructura inicial del proyecto Maven Multimódulo definiendo los Model Objects, que son clases java que son entidades y se utilizan para comunicar el microservicio con la base de datos, se han escrito los scripts SQL para inicializar el proyecto backend y la base de datos.

Después, se han definido las clases de domino que será el núcleo de la aplicación y unos sencillos endpoints para comenzar la investigación de la agnosticidad sobre el módulo repository.

La primera prueba realizada fue migrar el motor de la base de datos de PostgreSQL a OracleDB y los resultados fueron un éxito. Simplemente con tener la imagen de Oracle arrancada, cambiamos la cadena de conexión a base de datos del microservicio y funciona exactamente igual.

Una vez asegurada la agnosticidad en este punto, se han terminado de implementar todos los endpoints necesarios del microservicio. Una vez llegados a este punto, comenzamos a implementar todos los test unitarios para garantizar calidad y funcionalidad del código y una vez terminada esta tarea podemos comenzar con el desarrollo del proyecto frontend.

Se ha configurado Envoy para la seguridad una vez el microservicio se despliegue en nube y mediante el servicio de ADFS el microservicio valida la audiencia del token recibido por parte del frontend para poder garantizar la seguridad dentro de la aplicación.

En la aplicación frontend, se ha requerido aprender desde 0 el lenguaje de Angular y eso dificultó más el desarrollo del proyecto en ese aspecto.

3. Arquitectura

En términos generales, la aplicación consiste en un frontend que realiza varias llamadas a la API mediante el protocolo REST utilizando varios tipos de llamadas como GET, POST, PUT o DELETE.

La arquitectura general de la aplicación es la siguiente:

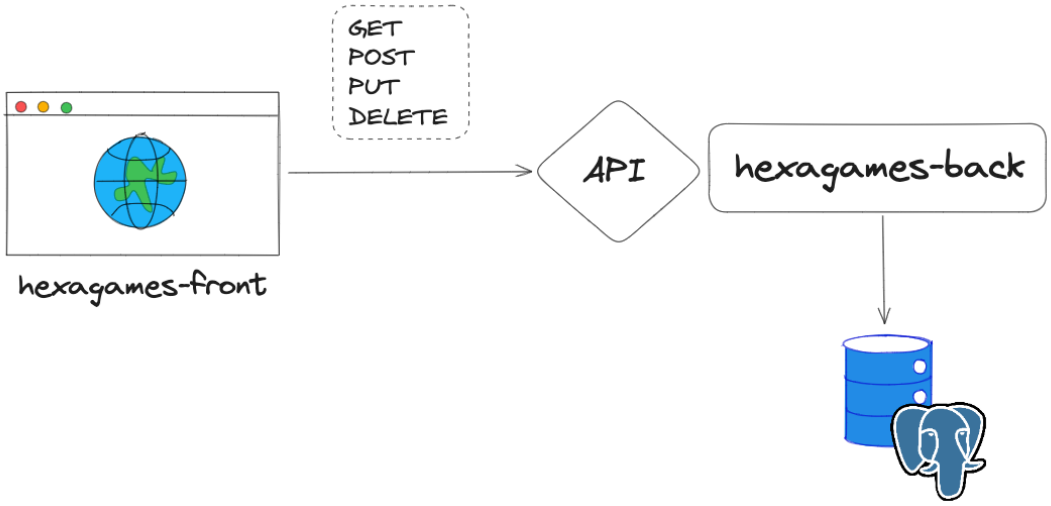


Figura 2: Arquitectura de la solución Hexagames

3.1 Frontend

En la capa frontend tenemos una arquitectura clean basada en tres capas.

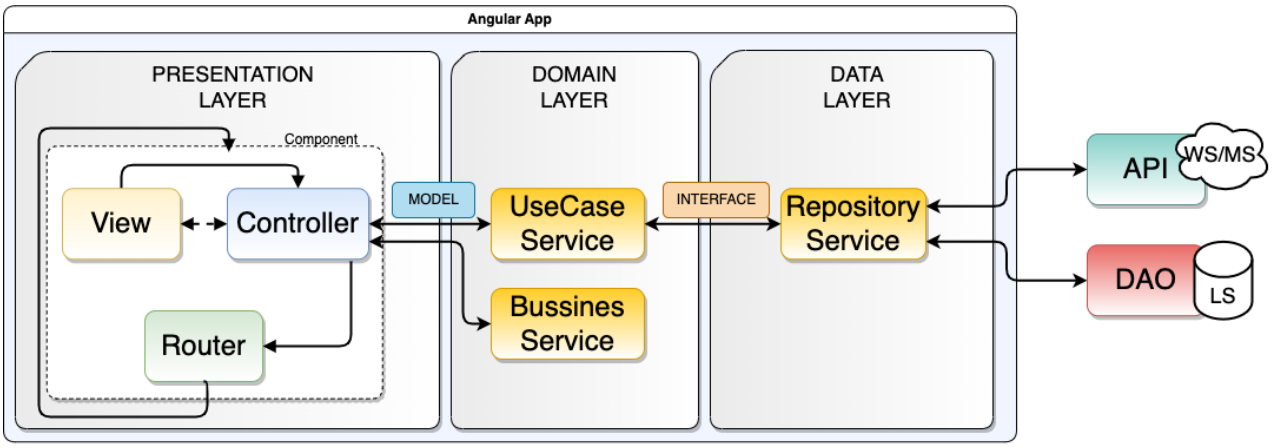


Figura 3: Ilustración de la arquitectura de tres capas

3.1.1 Capa de presentación

Es la capa donde se presentan los datos al usuario y se capturan las interacciones del usuario con la aplicación.

Esta deberá estar formada por un componente de tipo página y 0 o N componentes hijos (dependiendo de la complejidad).

Sus funciones son:

- Presentar los datos al usuario.
- Recoger las acciones del usuario (Si es de un componente hijo por sus @output)
- Recoger y/o presentar datos de formularios.
- Recoger eventos de rutas.
- Recoger cambios de estado.
- Enviar información a los componentes hijos (por los @inputs)
- Enviar toda esta información a la capa de dominio.

3.1.2 Capa de dominio

Es la capa que se encarga de la lógica de negocio, de hacer de intermediaría con la capa de datos, hacer gestiones de estado (si aplica) y hacer de intercomunicador entre componentes desacoplados (que no tengan relación padre-hijo).

Esta capa estará compuesta de Servicios Angular que pueden ser inyectados en root, si la lógica es compartida en toda la aplicación. O en el módulo de la página padre, si es compartida solo por una pantalla o pantallas hijas de esta.

Cuando su función sea de intermediaría con la capa de datos, la comunicación será por interfaces y la comunicación con la capa de presentación por modelos de datos (ver apartado anterior).

3.1.3 Capa de datos

Es la capa encargada de obtener y almacenar los datos de la aplicación.

Podrá obtener los dados desde API con servicios REST, o por Websockets.

Deberá tener un Servicio Angular con tantos métodos exponga la API de servicio. Ejemplo: Los servicios generados con la herramienta de Swagger estarían dentro de esta capa.

Podrá almacenar los datos en el Storage del navegador si fuera necesario, exponiendo los métodos para su lectura y escritura.

3.2 Backend

En la capa backend tenemos una arquitectura clean + hexagonal.

3.2.1 Clean Architecture

Clean Architecture es un conjunto de principios cuya finalidad principal es ocultar los detalles de implementación a la lógica de dominio de la aplicación.

De esta manera mantenemos aislada la lógica, consiguiendo tener una lógica mucho más mantenible y escalable en el tiempo.

En Clean Architecture, una aplicación se divide en responsabilidades y cada una de estas responsabilidades se representa en forma de capa. De esta forma tenemos capas exteriores y capas interiores:

- Las capa más exterior representa los detalles de implementación
- Las capas más interiores representan el dominio, incluyendo lógica de aplicación y lógica negocio empresarial.

Las principales características de Clean Architecture frente a otras arquitecturas es la regla de dependencia.

La regla de dependencia nos dice que un círculo interior nunca debe conocer nada sobre un círculo exterior. Sin embargo, los círculos exteriores si pueden conocer círculos interiores.

La lógica de dominio es lo que menos va a cambiar con el tiempo, por lo tanto, debemos evitar que tenga dependencias de detalles de implementación que van a cambiar con más frecuencia.

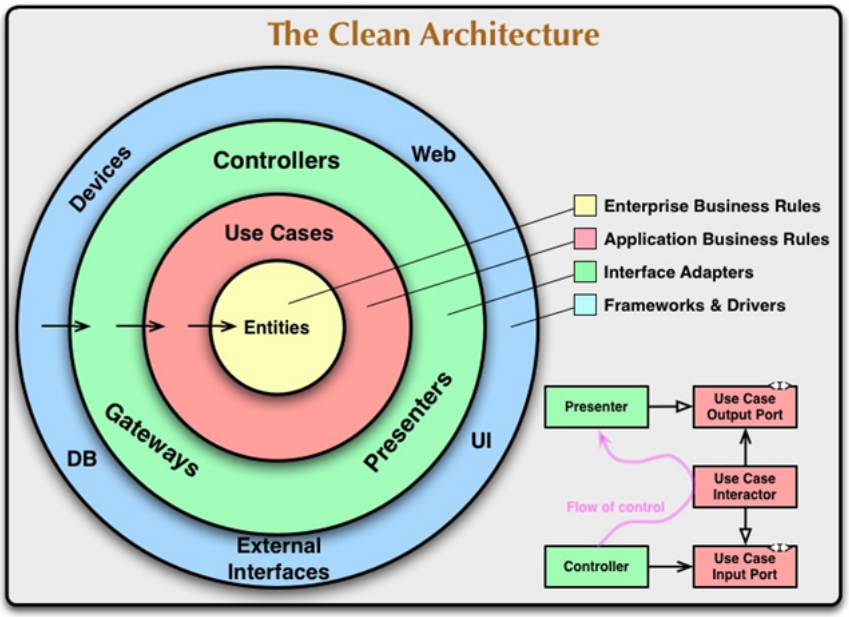


Figura 4: Representación de la Clean Architecture

Toda arquitectura que implemente una Clean Architecture produce sistemas que son:

- **Independiente de Frameworks:** La arquitectura no depende de la existencia de ninguna biblioteca de software llena de características. Esto permite utilizar estos frameworks como herramientas, en lugar de tener que acoplar el sistema a sus restricciones, limitándolo.
- **Testeable:** Las reglas de negocio se pueden probar sin la interfaz de usuario, la base de datos, el servidor web o cualquier otro elemento externo.
- **Independiente de la interfaz de usuario:** La interfaz de usuario puede cambiar fácilmente, sin cambiar el resto del sistema. Una interfaz de usuario web podría reemplazarse por una interfaz de usuario de consola, por ejemplo, sin cambiar las reglas de negocio.
- **Independiente de la base de datos:** Sería posible cambiar Oracle o PostgreSQL por Mongo, BigTable, CouchDB u otra cosa. Las reglas de negocio no están vinculadas a la base de datos.
- **Independiente de cualquier sistema externo:** De hecho, las reglas de negocio simplemente no saben nada sobre el mundo exterior.

3.2.2 Arquitectura Hexagonal

La arquitectura Hexagonal es una propuesta de implementación de una Clean Architecture. Define principalmente 2 tipologías de elementos dentro de la arquitectura, los Puertos y los Adaptadores.

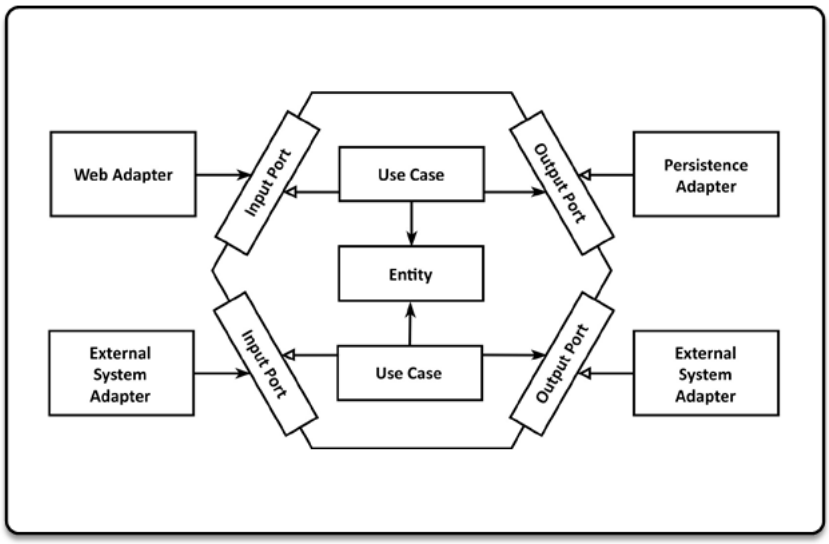
Propone que nuestro dominio sea el núcleo de las capas y que este no se acople a nada externo. En lugar de hacer uso explícito y mediante el principio de inversión de dependencias nos acoplamos a contratos (interfaces o puertos) y no a implementaciones concretas.

A partir de aquí, definiremos los siguientes conceptos como:

- Puerto: definición de una interfaz pública.
- Adaptador: especialización de un puerto para un contexto concreto.

Al ser una arquitectura que fomenta que nuestro dominio sea el núcleo de todas las capas (o regiones), y que no se acople a nada externo, encaja muy bien con la idea de Domain Driven Design (DDD).

Podríamos decir que DDD se basa en la Arquitectura Hexagonal como pilar central en términos de arquitectura.



Figuras 5: Representación de la arquitectura hexagonal

3.2.3 Aplicación de la Hexagonal en el proyecto

El proyecto backend es un proyecto Maven multimódulo, en el que los distintos módulos cumplen un papel en el diagrama.

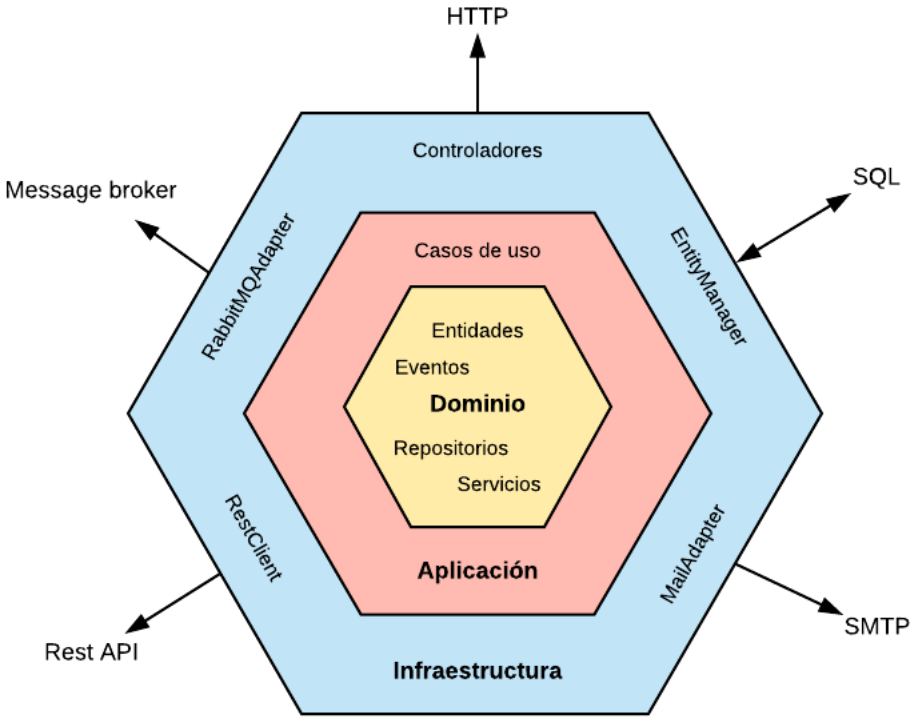


Figura 6: Hexágono representando la unión de los módulos

- **Aplicación:** En este módulo desarrollamos nuestra lógica de negocio y deberemos definir nuestros puertos de entrada y salida.
- **EntityManager** (repository): La responsabilidad de este módulo es la de persistir los datos. Será nuestro adaptador y lo implementaremos usando la tecnología JPA.
- **RestClient** (API-Rest): Este módulo definirá la implementación de los clientes de servicios web ofrecidos por otras aplicaciones siguiendo la aproximación de Contract First. Contendrá las clases y objetos generados a partir del contrato y ofrecerá como servicio los métodos definidos en el Swagger. Más adelante se define como se genera las clases y objetos a partir del contrato.

Aparte de los módulos anteriores, existe un módulo **(Boot)** que definirá el ensamblado de los distintos módulos de la aplicación en un paquete *.jar*. Contiene la clase que ejecuta el

microservicio, agrupa los módulos funcionales (api-rest, repository, etc.) y contiene la configuración de la aplicación y de plataformas cloud (GCP, Openshift, etc.).

Como podemos observar en las siguientes imágenes, a la izquierda es el scaffolding del proyecto y a la derecha en el pom principal, los módulos que componen la aplicación.

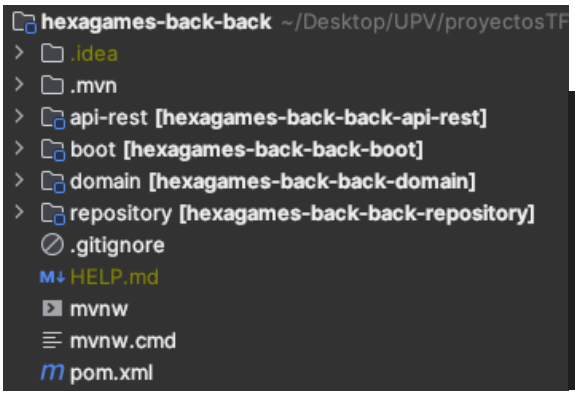


Figura 7: Estructura de carpetas del proyecto

Figura 8: Los diferentes módulos

3.2.4 Conclusión

Lo que podemos extraer de la Arquitectura Hexagonal es lo siguiente:

- Busca la **separación completa del código** de negocio (dominio) de los accidentes tecnológicos.
- Emplea **Adaptadores/Puertos** para la interacción con sistemas externos (BDDD, colas, APIs, etc.) al dominio.
- Son los adaptadores los que dependen del dominio (**inversión de dependencias**), no al revés.

3.3 Construcción de la base de datos

Para la construcción de la base de datos, he usado una imagen Docker de PostgreSQL para levantar un contenedor y acceder desde el backend mediante la siguiente cadena de conexión:

- localhost:5432/hexagames

La imagen Docker es la siguiente:

```yaml
version: '3'
services:
  tfg_hexagames:
    container_name: tfg_hexagames
    image: postgres:11
    environment:
      POSTGRES_USER: sa
      POSTGRES_PASSWORD: root
      POSTGRES_DB: hexagames
    volumes:
      - postgresql-tfg:/var/lib/postgresql-tfg
      - postgres11-tfg-data:/var/lib/postgresql-tfg/data
    ports:
      - "5432:5432"

volumes:
  postgresql-tfg:
  postgres11-tfg-data:
    driver: local
```

Figura 9: Fichero docker-compose.yaml que arranca la imagen de PostgreSQL

En el desarrollo de aplicaciones modernas, especialmente en entornos de microservicios, mantener una base de datos coherente y sincronizada con la evolución del código es crucial. Para abordar este desafío, se recurre a herramientas como Flyway, una solución que permite la administración automatizada y controlada de migraciones de esquemas de bases de datos.

Introducción a Flyway

Flyway es una herramienta de migración de bases de datos que ofrece una forma estructurada y segura de gestionar cambios en la estructura de la base de datos a medida que la aplicación evoluciona. La herramienta proporciona una solución para evitar problemas de compatibilidad y asegurar que la base de datos se mantenga en sincronía con la aplicación, especialmente en el contexto de microservicios.

Uso de Flyway para Generar Tablas en un Microservicio

La generación y evolución de tablas de bases de datos en un microservicio puede ser gestionada eficazmente a través de Flyway. A continuación, se presenta un resumen del proceso de uso de Flyway para generar tablas en un microservicio:

1. **Instalación y Configuración**: Flyway se integra en el proyecto como una dependencia y se configura con los detalles de conexión a la base de datos y la ubicación de las migraciones. Esta configuración se puede realizar mediante un archivo de configuración o propiedades en el archivo de construcción.

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

Figura 10: Dependencia de Flyway

2. **Creación de Migraciones**: En el directorio designado para las migraciones, se crean archivos SQL individuales para cada cambio en el esquema de la base de datos. Cada archivo representa una migración y sigue un formato de nomenclatura que incluye la versión y el nombre de la migración.

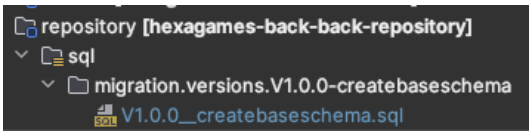


Figura 11: Migración que contiene la inicialización de las tablas

3. **Desarrollo de Scripts SQL**: Dentro de cada archivo de migración, se escriben scripts SQL que describen los cambios necesarios en la base de datos. Estos scripts deben ser reversibles y capaces de aplicar o revertir los cambios según sea necesario.

```
CREATE TABLE public.playstation (
    id int4 NOT NULL GENERATED ALWAYS AS IDENTITY,
    console varchar NULL,
    "name" varchar NULL,
    price int4 NULL,
    shop_name varchar NULL,
    status varchar NULL,
    purchased bool NULL,
    description varchar NULL,
    CONSTRAINT playstation_pk PRIMARY KEY (id)
);

CREATE TABLE public.switch (
    id int4 NOT NULL GENERATED ALWAYS AS IDENTITY,
    "name" varchar NULL,
    price int4 NULL,
    shop_name varchar NULL,
    status varchar NULL,
    purchased bool NULL,
    description varchar NULL,
    CONSTRAINT switch_pk PRIMARY KEY (id)
);
```

Figura 12: Script sql que contiene la construcción de las tablas

4. **Ejecución de Migraciones**: Al ejecutar la aplicación o el comando correspondiente de Flyway, la herramienta detecta las migraciones pendientes y las aplica en orden, manteniendo así la consistencia entre la estructura de la base de datos y el código de la aplicación.

```
Database: jdbc:postgresql://localhost:5432/hexagames (PostgreSQL 11.15)
Successfully validated 1 migration (execution time 00:00.019s)
Current version of schema "flyway": << Empty Schema >>
Migrating schema "flyway" to version "1.0.0 - createbaseschema"
Successfully applied 1 migration to schema "flyway", now at version v1.0.0 (execution time 00:00.066s)
```

Figura 13: Primer arranque del plugin y validación de los scripts

5. **Control de Versiones y Auditoría**: Flyway registra las migraciones que se han aplicado a la base de datos, lo que permite un seguimiento preciso de la versión actual y garantiza que las mismas migraciones no se apliquen repetidamente.

flyway_schema_history | Enter a SQL expression to filter results (use Ctrl+Space)

Grilla

	installed_rank	version	description	type	script	checksum
1	1	1.0.0	createbaseschema	SQL	V1.0.0-createbaseschema/V1.0.0__createbaseschema.sql	-1.468.790.799

Figura 14: Tabla del registro de la migración

En resumen, Flyway se ha convertido en una herramienta esencial para garantizar la integridad y la coherencia entre el esquema de la base de datos y el código de la aplicación en entornos de microservicios. Su enfoque controlado y automatizado de las migraciones de bases de datos ayuda a prevenir problemas de compatibilidad y simplifica la administración de cambios a lo largo del ciclo de vida del software.

Integrar Flyway en el proceso de desarrollo y despliegue de microservicios brinda una solución eficaz para mantener bases de datos actualizadas, coherentes y compatibles, lo que contribuye a un desarrollo más fluido y a la entrega exitosa de aplicaciones confiables y escalables.

4. Estudio de mercado

Se han observado varios ejemplos de páginas web, las cuales tienen la misma funcionalidad que Hexagames, pero ninguna incorpora una idea de un comparador de precios y la amplia personalización que se ofrece desde esta aplicación.

A continuación, dejo un ejemplo de lo que a mi parecer es una aplicación completa que puede realmente llegar a ser competencia de Hexagames.

4.1 CLZ Games

CLZ Games [11] es un software proporcionado por Collectorz.com en la que ofrece un servicio para catalogar videojuegos en forma de aplicación para móviles, página web o una aplicación de escritorio exclusivamente para usuarios Windows.

Las principales características de la aplicación son comunes a cualquier plataforma a la que el cliente se suscriba, teniendo un precio de 14,95€ para usuarios en aplicación móvil y 29,95€ tanto para página web como para aplicación de escritorio. Cabe destacar que ofrecen un plan de aplicación móvil + página web por 39,90€, quedando claro que pagando una modalidad no tienes acceso al resto. También hay que mencionar que los pagos antes expuestos, son para una suscripción anual.

¿Cuáles son las características que ofrece?

El servicio de CLZ Games, ofrece la portada de los videojuegos de tu colección, día de lanzamiento, publicador, descripción y tráiler entre muchas otras descripciones acerca de tu videojuego.

Otra característica muy sorprendente, es que puedes añadir un juego a tu colección fácilmente escaneando el código de barras y automáticamente la aplicación se encarga de buscar cuál es el juego, su precio y las demás descripciones. Es una característica muy curiosa ya que desde la aplicación móvil es bastante práctica.

Finalmente, una característica muy esperada, es la posibilidad de editar manualmente cualquier campo de un videojuego de tu colección pudiendo añadir notas personalizadas, ubicaciones, día de compra, etc.

En resumen, es un software muy potente pero el cual me parece excesivamente caro y sin una versión freemium, una versión gratuita con funcionalidades limitadas y que está repleta de publicidad. No obstante, lo poco que he podido observar el software, me parece muy completo y muy intuitivo de utilizar, pero con una interfaz un poco obsoleta a lo que se acostumbra a ver hoy en día.

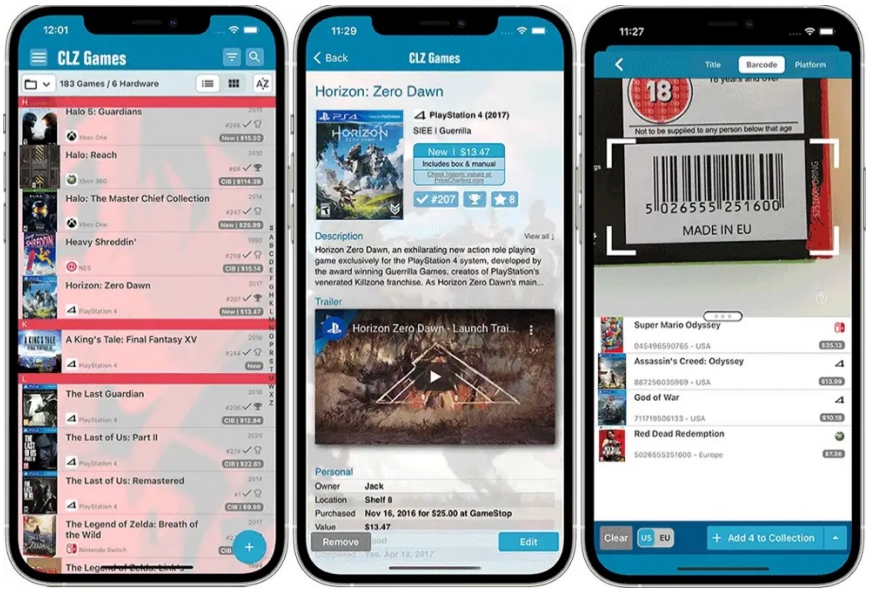


Figura 15: Interfaz de CLZ Games en un dispositivo móvil

5. Análisis

Esta etapa es crítica para el éxito del proyecto de software, ya que sienta las bases para todo el proceso de desarrollo y garantiza que el software cumpla con las expectativas de los usuarios y las metas del negocio.

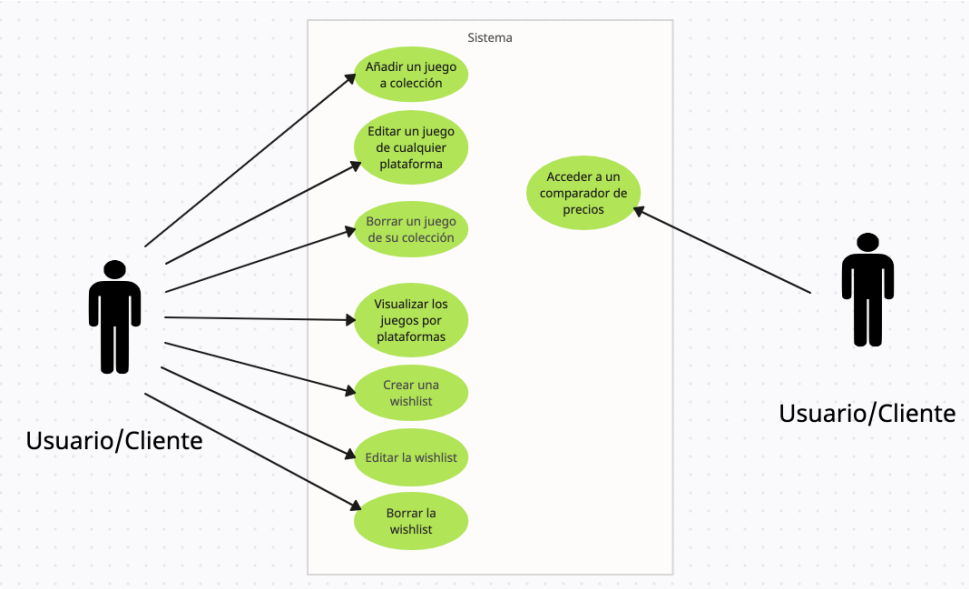


Figura 16: Diagrama de casos de uso

5.1 Análisis Funcional

- Un usuario debe poder registrarse en la aplicación
- Un usuario debe poder iniciar sesión en la aplicación
- Un usuario podrá consultar la plataforma de videojuegos que quiera
- Un usuario podrá editar o borrar un videojuego que tenga registrado del catálogo de una plataforma
- Un usuario podrá registrar un videojuego en la plataforma que desee
- Un usuario podrá consultar su lista de deseados
- Un usuario podrá añadir o eliminar un videojuego de su lista de deseados
- Un usuario podrá comparar precios de varias tiendas para un videojuego.

5.2 Análisis No Funcional

Rendimiento:

- Cualquier operación de la aplicación no debería tardar más de 1 segundo en completarse.

Usabilidad:

- Debe tener una interfaz intuitiva y fácil de aprender a utilizarse.

Seguridad:

- Un cliente no podrá acceder a información de otros clientes.

5.3 Casos de uso

En un diagrama de casos de uso, se representan los actores y los casos de uso para modelar las interacciones entre un sistema y sus usuarios o partes externas. Los conceptos clave son:

1. **Actor:** Un actor es un rol o entidad externa que interactúa con el sistema. Los actores pueden ser personas, otros sistemas, dispositivos, o incluso el propio sistema en algunos casos. Los actores se representan como figuras externas al sistema y se utilizan para mostrar quiénes están involucrados en las interacciones con el sistema. Los actores desencadenan casos de uso al interactuar con el sistema y pueden estar asociados con uno o varios casos de uso.
2. **Caso de uso:** Un caso de uso es una representación de una interacción específica entre un actor y el sistema. Describe una funcionalidad o un conjunto de acciones que el sistema realiza para lograr un objetivo específico para el actor. Los casos de uso ayudan a capturar los requisitos funcionales del sistema desde la perspectiva del usuario. Cada caso de uso se representa como una elipse en el diagrama de casos de uso.

La relación entre actores y casos de uso en un diagrama de casos de uso se representa mediante líneas de comunicación llamadas "relaciones de asociación." Estas relaciones muestran qué

actores participan en qué casos de uso y cómo interactúan. Las relaciones de asociación se pueden clasificar de la siguiente manera:

- **Asociación simple:** Una línea sólida que conecta un actor a un caso de uso, indicando que el actor está involucrado en ese caso de uso.
- **Inclusión:** Se utiliza para mostrar que un caso de uso (llamado caso de uso incluido) se incluye en otro caso de uso más grande (llamado caso de uso base). Esto significa que el caso de uso base ejecutará el caso de uso incluido en algún punto de su flujo.
- **Extensión:** Se utiliza para mostrar que un caso de uso (llamado caso de uso extendido) puede extender otro caso de uso (llamado caso de uso base) en ciertas condiciones. Esto implica que el caso de uso extendido puede agregarse opcionalmente al flujo del caso de uso base bajo ciertas circunstancias.

5.3.1 Registro

· Descripción:
  - Este caso de uso permite al usuario registrarse en la aplicación
· Actores:
  - Usuario
· Flujo de eventos:
  - El caso de uso se inicia al entrar en la aplicación por primera vez
· Flujo básico:
  - El usuario introduce su nombre de usuario y contraseña
  - Pulsa en el botón de registrarse y tendrá su cuenta creada
· Flujo alternativo:
  - No existe flujo alternativo
· Precondiciones:
  - No existen precondiciones
· Postcondiciones:
  - Que el usuario se haya podido registrarse correctamente en la aplicación

5.3.2 Iniciar sesión

- Descripción:
  - Este caso de uso permite al usuario iniciar sesión en la aplicación
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al entrar en la aplicación
- Flujo básico:
  - El usuario introduce su nombre de usuario y contraseña
  - Pulsa en el botón de iniciar sesión y accederá al menú principal
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar registrado en la aplicación
- Postcondiciones:
  - Que el usuario pueda acceder correctamente a la aplicación

5.3.3 Consultar plataforma PlayStation

- Descripción:
  - Este caso de uso permite al usuario acceder a la tabla de visualización para la plataforma PlayStation
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en la card de PlayStation
- Flujo básico:
  - El usuario pulsa en la card de PlayStation y accederá a la tabla de su colección de PlayStation
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:

- Estar en el menú principal

· Postcondiciones:

- No existen postcondiciones

5.3.4 Editar videojuego en plataforma PlayStation

· Descripción:

- Este caso de uso permite al usuario editar un videojuego de la plataforma PlayStation

· Actores:

- Usuario

· Flujo de eventos:

- El caso de uso se inicia al clicar en el icono del lápiz del videojuego en cuestión

· Flujo básico:

- El usuario pulsa en el icono del lápiz

· Flujo alternativo:

- No existe flujo alternativo

· Precondiciones:

- Estar visualizando la tabla de PlayStation

· Postcondiciones:

- Visualizar el videojuego con los datos actualizados

5.3.5 Borrar videojuego en plataforma PlayStation

· Descripción:

- Este caso de uso permite al usuario borrar un videojuego de la plataforma PlayStation

· Actores:

- Usuario

· Flujo de eventos:

- El caso de uso se inicia al clicar en el icono de la papelera del videojuego en cuestión

· Flujo básico:

- El usuario pulsa en el icono de la papelera

· Flujo alternativo:

- No existe flujo alternativo

· Precondiciones:
  - Estar visualizando la tabla de PlayStation
· Postcondiciones:
  - Borrar correctamente el videojuego de la base de datos

5.3.6 Consultar plataforma Xbox

· Descripción:
  - Este caso de uso permite al usuario acceder a la tabla de visualización para la plataforma Xbox
· Actores:
  - Usuario
· Flujo de eventos:
  - El caso de uso se inicia al clicar en la card de Xbox
· Flujo básico:
  - El usuario pulsa en la card de Xbox y accederá a la tabla de su colección de Xbox
· Flujo alternativo:
  - No existe flujo alternativo
· Precondiciones:
  - Estar en el menú principal
· Postcondiciones:
  - No existen postcondiciones

5.3.7 Editar videojuego en plataforma Xbox

· Descripción:
  - Este caso de uso permite al usuario editar un videojuego de la plataforma Xbox
· Actores:
  - Usuario
· Flujo de eventos:
  - El caso de uso se inicia al clicar en el icono del lápiz del videojuego en cuestión

- Flujo básico:
  - El usuario pulsa en el icono del lápiz
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar visualizando la tabla de Xbox
- Postcondiciones:
  - Visualizar el videojuego con los datos actualizados

5.3.8 Borrar videojuego en plataforma Xbox

- Descripción:
  - Este caso de uso permite al usuario borrar un videojuego de la plataforma Xbox
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el icono de la papelera del videojuego en cuestión
- Flujo básico:
  - El usuario pulsa en el icono de la papelera
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar visualizando la tabla de Xbox
- Postcondiciones:
  - Borrar correctamente el videojuego de la base de datos

5.3.9 Consultar plataforma Switch

- Descripción:
  - Este caso de uso permite al usuario acceder a la tabla de visualización para la plataforma Switch
- Actores:

  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en la card de Switch
- Flujo básico:
  - El usuario pulsa en la card de Switch y accederá a la tabla de su colección de Switch
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar en el menú principal
- Postcondiciones:
  - No existen postcondiciones

5.3.10 Editar videojuego en plataforma Switch

- Descripción:
  - Este caso de uso permite al usuario editar un videojuego de la plataforma Switch
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el icono del lápiz del videojuego en cuestión
- Flujo básico:
  - El usuario pulsa en el icono del lápiz
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar visualizando la tabla de Switch
- Postcondiciones:
  - Visualizar el videojuego con los datos actualizados

5.3.11 Borrar videojuego en plataforma PlayStation

- Descripción:

- Este caso de uso permite al usuario borrar un videojuego de la plataforma Switch

· Actores:
  - Usuario
· Flujo de eventos:
  - El caso de uso se inicia al clicar en el icono de la papelera del videojuego en cuestión
· Flujo básico:
  - El usuario pulsa en el icono de la papelera
· Flujo alternativo:
  - No existe flujo alternativo
· Precondiciones:
  - Estar visualizando la tabla de Switch
· Postcondiciones:
  - Borrar correctamente el videojuego de la base de datos

5.3.12 Añadir videojuego al catálogo

· Descripción:
  - Este caso de uso permite al usuario añadir un videojuego a la base de datos
· Actores:
  - Usuario
· Flujo de eventos:
  - El caso de uso se inicia al clicar en el botón de añadir videojuego
· Flujo básico:
  - El usuario pulsa en el botón de Añadir videojuego
· Flujo alternativo:
  - No existe flujo alternativo
· Precondiciones:
  - Estar en el menú principal
· Postcondiciones:
  - Que se añada y se visualice correctamente el videojuego

5.3.13 Visualizar la wishlist

- Descripción:
  - Este caso de uso permite al usuario acceder a la wishlist
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el botón Wishlist
- Flujo básico:
  - El usuario pulsa en el botón de la wishlist lo que permite la navegación hacia la pantalla de la wishlist
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar en el menú principal
- Postcondiciones:
  - No existen postcondiciones

5.3.14 Añadir videojuego a la wishlist

- Descripción:
  - Este caso de uso permite al usuario añadir un videojuego a la wishlist
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el botón de añadir videojuego
- Flujo básico:
  - El usuario pulsa en el botón de Añadir videojuego
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar en la pantalla de la wishlist
- Postcondiciones:
  - Que se añada y se visualice correctamente el videojuego

5.3.15 Borrar videojuego de la wishlist

- Descripción:
  - Este caso de uso permite al usuario borrar un videojuego de la wishlist
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el icono de la papelera del videojuego en cuestión
- Flujo básico:
  - El usuario pulsa en el icono de la papelera
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar visualizando la wishlist
- Postcondiciones:
  - Borrar correctamente el videojuego de la wishlist

5.3.16 Acceder al comparador de precios

- Descripción:
  - Este caso de uso permite al usuario comparar el precio de un videojuego
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia al clicar en el botón de ver precios
- Flujo básico:
  - El usuario pulsa en el botón de ver precios
- Flujo alternativo:
  - No existe flujo alternativo
- Precondiciones:
  - Estar en la pantalla de la wishlist
- Postcondiciones:
  - No existen postcondiciones

6. Diagramas

6.1 Diagrama de flujo

Los elementos básicos que se incluyen son:

1. **Símbolos:** Los símbolos son formas geométricas que representan diferentes tipos de acciones o decisiones en el proceso. Los símbolos más comunes incluyen rectángulos para representar acciones, rombos para representar decisiones, óvalos para representar el inicio o el final del proceso, y otros símbolos específicos para operaciones como entrada/salida de datos.

2. **Flechas:** Las flechas conectan los símbolos y muestran la secuencia en la que se realizan las acciones. Indican la dirección del flujo del proceso, es decir, el orden en que se ejecutan las actividades.

El propósito principal de un diagrama de flujo es:

1. **Visualización:** Proporciona una representación visual clara y concisa de un proceso, lo que facilita la comprensión de cómo funciona y cuáles son las etapas involucradas.

2. **Análisis y optimización:** Permite identificar posibles cuellos de botella, ineficiencias o áreas de mejora en un proceso, lo que es esencial para la mejora continua y la optimización.

3. **Documentación:** Sirve como una forma de documentar un proceso para futuras referencias, capacitación de personal o comunicación con otros equipos.

4. **Comunicación:** Facilita la comunicación entre diferentes partes interesadas, como diseñadores, programadores, gerentes de proyectos y usuarios, al proporcionar una representación visual común del proceso.

La siguiente figura representa el diagrama de flujo de Hexagames:

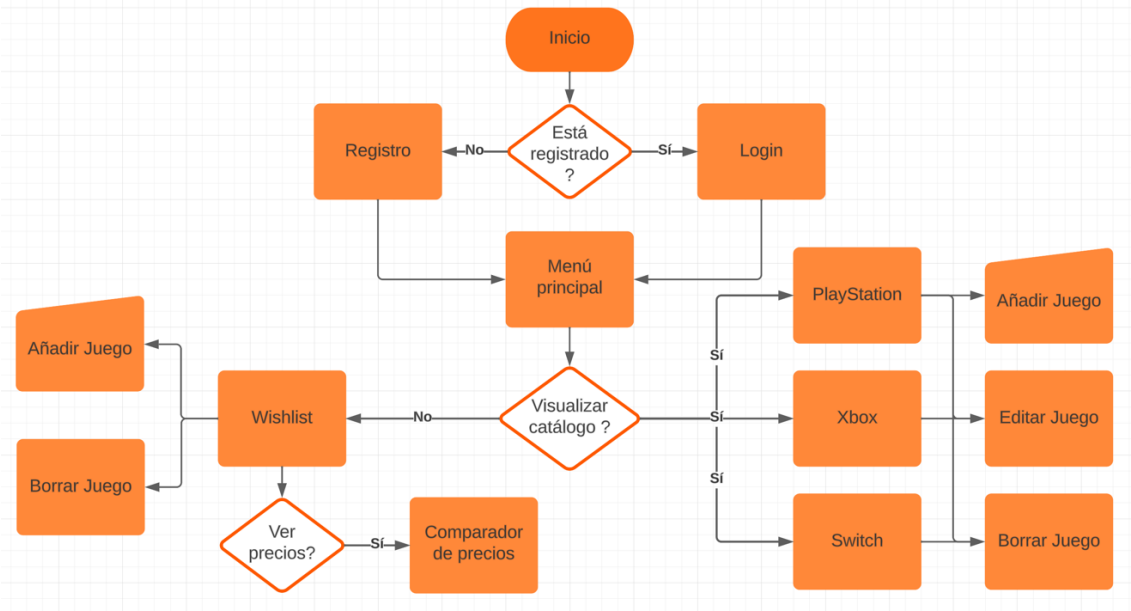


Figura 17: Diagrama de flujo

6.2 Diagrama de base de datos

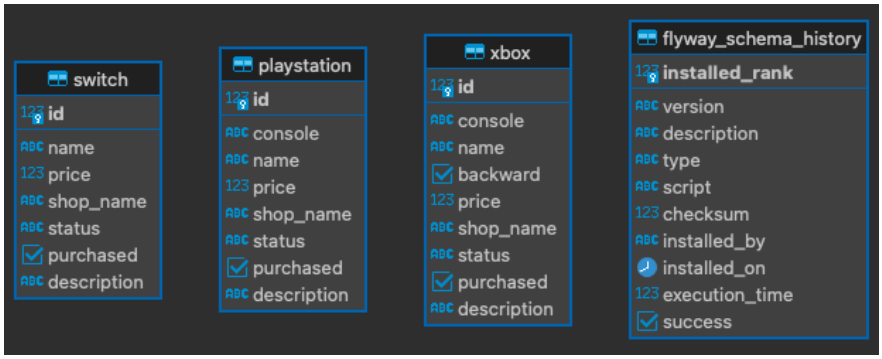


Figura 18: Diagrama de base de datos

Como podemos observar en la figura, tanto la tabla switch como playstation y xbox comparten algunos atributos, pero no se ha utilizado la herencia para hacer más sencilla la funcionalidad utilizando Hibernate y JPA en el microservicio.

Los atributos comunes:

- id es el identificador de cada videojuego
- name es el nombre de cada videojuego
- price es el precio pagado por cada videojuego

- shop_name es la tienda donde se ha comprado el videojuego
- status es el estado del videojuego
- purchased es si actualmente se encuentra en la colección del usuario
- description es la descripción que tiene cada videojuego

Los atributos que no son comunes:

- console identifica la plataforma de la consola
- backward, sólo en xbox, indica si los juegos son compatibles con las consolas de nueva generación

La tabla de flyway_schema_history se genera al usar el plugin previamente detallado para la generación de las tablas y de la base de datos.

7. Configuración y agnosticidad en nube

La agnosticidad en nube quiere decir que los proyectos pueden desplegarse y arrancar sin importar que nube estemos utilizando para ofrecer los servicios.

Como ejemplo pondré el proyecto de backend.

En el proyecto backend, desde el diseño de la base de datos, se ha hecho una investigación para que el microservicio sufra el menor cambio posible ante la decisión de una migración de base de datos (por ejemplo, cambiar de una base de datos relacional a otra).

El mismo concepto se ha aplicado ahora en el entorno cloud, podemos desplegar el microservicio en Google Cloud y si mañana se decide realizar una migración a una nube AWS sólo tendríamos que cambiar el nombre del nuevo namespace y con eso tendríamos el microservicio operativo nuevamente en la nueva nube. Esto se debe a que las nubes utilizan Kubernetes.

Kubernetes, comúnmente abreviado como K8s, es una plataforma de código abierto diseñada para la orquestación y gestión de contenedores en aplicaciones distribuidas. Fue desarrollado originalmente por Google y luego donado a la Cloud Native Computing Foundation (CNCF), donde ha ganado una gran comunidad de desarrolladores y usuarios.

Las características principales de Kubernetes incluyen:

1. **Orquestación de Contenedores**: Kubernetes permite administrar la implementación, actualización y escalabilidad de contenedores de manera eficiente. Puedes desplegar contenedores Docker u otros formatos de contenedor en clústeres de servidores físicos o virtuales.
2. **Escalabilidad Automática**: Kubernetes puede escalar automáticamente la cantidad de réplicas de un contenedor en función de la carga de trabajo o la demanda, lo que garantiza un rendimiento óptimo sin intervención manual.
3. **Balanceo de Carga**: Ofrece balanceo de carga integrado para distribuir el tráfico entre las instancias de un servicio, lo que mejora la disponibilidad y la redundancia.

4. **Autoreparación**: Kubernetes monitorea continuamente el estado de los contenedores y los servicios. Si detecta un fallo, puede reiniciar automáticamente los contenedores o crear nuevas instancias para reemplazar las que fallaron.

5. **Despliegues Controlados**: Permite realizar despliegues progresivos y controlados, lo que facilita la actualización de aplicaciones sin tiempo de inactividad.

6. **Gestión Declarativa**: Las configuraciones de despliegue y servicio se definen en archivos YAML o JSON, lo que permite una gestión declarativa de la infraestructura y aplicaciones.

7. **Portabilidad**: Kubernetes es compatible con múltiples proveedores de nube y se puede ejecutar en entornos locales o en nubes públicas, lo que facilita la portabilidad de las aplicaciones.

8. **Ecosistema Amplio**: Kubernetes cuenta con un ecosistema de herramientas y extensiones que amplían su funcionalidad, como Helm para administrar paquetes de aplicaciones y Prometheus para el monitoreo.

Kubernetes se ha convertido en la plataforma líder para la gestión de contenedores en aplicaciones modernas y es ampliamente utilizado en el desarrollo de aplicaciones en la nube, la implementación de microservicios y la administración de infraestructuras escalables y automatizadas.

Se adjuntan varios ejemplos de configuración de Kubernetes para el despliegue del microservicio.

```yaml
springboot-service:
  cloudProvider: gke
  environment: dev
  secretManager: conjur
  ingress:
    enabled: true
    hosts:
      - host: host.com
        port: http
        path: /api/hexagames/v1
    tls: tls-crt-wildcard
  customSecret:
    enabled: true
    mountAsVolume: true
    readOnly: true
    name: hexagame-secrets
    mountPath: /var/run/secrets/
    data:
      - remoteSecret: db-app-pass-hexagames-app-hexagames
        k8sKey: database-password
      - remoteSecret: db-app-hexagames-app-hexagames
        k8sKey: database-user
    conjurAuthnLogin: host/dev-hexagames
    conjurConnectConfigMap: conjur-cm
    serviceAccountName: conjur-sa
  extraEnv:
    - name: SPRING_PROFILES_ACTIVE
      value: dev
    - name: JAVA_MONITORING_OPTS_EXT
      value: -Dotel.otlp.span.timeout=4000 -Dotel.propagators=b3multi -Dotel.export
    - name: CONSOLE_LOG_PATTERN
      value: '{"level": "%level", "date": "%d{yyyy-MM-ddTHH:mm:ss.SSSZ}", "applicat
    - name: JAVA_OPTS_EXT
      value: -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005,
    - name: CLOUD_SQL_INSTANCE_ID
      value: databasehost
    - name: CLOUD_SQL_CREDENTIALS_FILE
      valueFrom:
        secretKeyRef:
          key: json-credentialssql.json
          name: hexagames-secrets
    - name: CLOUD_SQL_DATABASE
      value: hexagames
```

Figura 19: Fichero de configuración de nube (Parte 1)

```
      - name: CLOUD_SQL_DATABASE_USERNAME
        valueFrom:
          secretKeyRef:
            key: database-user
            name: hexagames-secrets
      - name: CLOUD_SQL_DATABASE_PASSWORD
        valueFrom:
          secretKeyRef:
            key: database-password
            name: hexagames-secrets
    extraVolumes:
      - name: service-account-credentials
        secret:
          secretName: sa-json-hexagames
    overrideContainerPort: 8080
    service:
      name: hexagames-back-back
      internalPort: 8081
      portName: http
    envoy:
      enabled: true
      args:
        - -i
        - /config/swagger.yaml
        - -p
        - '"/api/hexagames/v1:"'
        - --corsAllowOrigin
        - '"https?://.+\\.host\\.(com|es)"'
        - --corsAllowOrigin
        - '"https?://localhost:4200"'
      env:
        - name: O2E_ISSUER_ADFS
          value: "adfsissuer"
        - name: O2E_JWKS_URI_ADFS
          value: "adfsjwks"
        - name: O2E_AUDIENCES_ADFS
          value: "audience"
      swaggerFiles:
        - api-rest/contracts/swagger.yaml
    resources:
      requests:
        memory: "768Mi"
        cpu: "100m"
      limits:
        memory: "850Mi"
```

Figura 20: Fichero de configuración para despliegue en nube (Parte 2)

8. Pruebas

Las pruebas de rendimiento, unitarias e integración son componentes esenciales de las estrategias de prueba en el desarrollo de software. A continuación, explicaré cada una de ellas:

1. **Pruebas Unitarias**:
    - **Definición**: Las pruebas unitarias son el nivel más bajo de pruebas en el desarrollo de software. Se centran en evaluar unidades individuales de código, como funciones, métodos o clases, de forma aislada.
    - **Objetivo**: El objetivo principal de las pruebas unitarias es garantizar que cada componente de software funcione correctamente de acuerdo con su especificación. Esto ayuda a detectar y corregir errores en un nivel temprano del desarrollo y mejora la calidad del código.
    - **Herramientas**: Se suelen utilizar marcos de pruebas unitarias, como JUnit para Java o NUnit para .NET, para automatizar las pruebas y proporcionar resultados claros.
    - **Ejemplo**: Si tienes una función que suma dos números, una prueba unitaria podría verificar que la función devuelve el resultado correcto cuando se le pasan números específicos como entrada.
2. **Pruebas de Integración**:
    - **Definición**: Las pruebas de integración se centran en evaluar cómo interactúan diferentes módulos, componentes o servicios de un sistema cuando se combinan.
    - **Objetivo**: El objetivo principal de las pruebas de integración es identificar problemas de interoperabilidad entre las partes del sistema y asegurarse de que funcionen de manera conjunta de manera efectiva.
    - **Herramientas**: Puedes utilizar herramientas de pruebas de integración específicas o marcos de trabajo que faciliten la configuración y ejecución de estas pruebas.
    - **Ejemplo**: Si estás desarrollando un sistema de comercio electrónico, una prueba de integración podría verificar que el proceso de compra interactúa correctamente con la gestión de inventario y la base de datos de usuarios sin errores.

3. **Pruebas de Rendimiento**:

    - **Definición**: Las pruebas de rendimiento evalúan cómo se comporta un sistema en términos de velocidad, capacidad y estabilidad bajo diversas condiciones, como carga de usuarios, tráfico de red o volumen de datos.
    - **Objetivo**: El objetivo principal de las pruebas de rendimiento es asegurarse de que el software pueda manejar la demanda real o prevista sin problemas de rendimiento, como tiempos de respuesta lentos o bloqueos del sistema.
    - **Herramientas**: Se utilizan herramientas de pruebas de rendimiento, como Apache JMeter o LoadRunner, para simular condiciones de uso del sistema y recopilar métricas de rendimiento.
    - **Ejemplo**: Si tienes una aplicación web, las pruebas de rendimiento podrían simular un alto tráfico de usuarios concurrentes para verificar que el servidor web y la base de datos puedan manejar la carga sin problemas.

Para las pruebas unitarias me he centrado en el proyecto backend en el cual he utilizado JUnit5 y Mockito para realizar las pruebas unitarias del microservicio.

Para mostrar como ejemplo la cobertura de código en el proyecto, tomamos como ejemplo la clase HexaGamesControllerAdapter.java que está situada en el módulo api-rest.

Element	Class, %	Method, %	Line, %
com.hexagames.back.controllers.adapte	100% (1/1)	100% (10/10)	100% (29/29)
HexaGamesControllerAdapter	100% (1/1)	100% (10/10)	100% (29/29)

Figura 21: Cobertura de código del módulo api-rest

```java
@Test
void login() {
    var expected = new ResponseEntity<>(HttpStatus.OK);
    var login = new Login( username: "admin",  password: "admin");

    given(gamesDriveIn.login(login)).willReturn( t: true);
    assertEquals(expected, hexaGamesControllerAdapter.login(new Login( username: "admin",  password: "admin")));

    var expectedUnauthorized = new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
    assertEquals(expectedUnauthorized, hexaGamesControllerAdapter.login(new Login( username: "admin",  password: "1234")));
}

new *
@Test
void getAllPsGames() {
    var games = new SuccessGetPlaystationCollection(new ArrayList<>());
    var expected = ResponseEntity.ok(games);

    assertEquals(expected.getStatusCode(), hexaGamesControllerAdapter.getAllPsGames( id: "1").getStatusCode());
}
```

Figura 22: Ejemplo de 2 tests unitarios

Como vamos a darle continuidad al proyecto una vez esté en un entorno PREProductivo vamos a hacerle unas pruebas de rendimiento para observar que el flujo de peticiones concurrentes es el esperado.

También, una vez lo tengamos en un entorno Cloud, ejecutaré las pruebas de integración con el entorno.

9. Relación del trabajo desarrollado con los estudios cursados

La relación que existe entre las asignaturas que he cursado con este proyecto es bastante fuerte, debido que han servido para sentar las bases de los conocimientos necesarios para poder afrontar el último reto del Grado de Informática.

Asignaturas como Bases de Datos son esenciales en el desarrollo del backend ya que involucra la interacción con bases de datos. Esta asignatura es crucial para entender cómo trabajar con bases de datos, como PostgreSQL, y entender principalmente el funcionamiento interno que poseen y sobre todo poder aprender a manejar el lenguaje SQL.

La asignatura Interfaces Persona Computador ayuda a diseñar una interfaz de usuario amigable y atractiva para el frontend.

Asignaturas como Ingeniería del Software, del primer trimestre de tercero, nos introduce al desarrollo de funcionalidades y casos de prueba de un proyecto más completo y seleccionar la rama que lleva el mismo nombre que la asignatura profundiza mucho más en el aprendizaje de esos conocimentos.

Dentro de la rama, asignaturas como Análisis y especificación de requisitos o Desarrollo de software dirigido por modelos nos ayudan paso a paso a aprender a realizar los análisis necesarios antes de abordar el desarrollo de nuevos proyectos y nos ayuda a entender las necesidades que se tiene a la hora de afrontar un nuevo proyecto y que seamos capaces de identificarlas.

Otras asignaturas como, Calidad del Software o Análisis, validación y depuración de software nos ayudan a crear las pruebas necesarias para validar y asegurar la calidad de nuestro código y asegurándonos de su correcto funcionamiento.

Diseño de software y Mantenimiento y evolución de software nos permite perfeccionar nuestras buenas prácticas a la hora de escribir código, nos enseñan herramientas para que nuestro software tenga un mejor ciclo de vida implementando el uso de patrones de diseño, patrones arquitectónicos, etc. Nos proporciona las herramientas para poder detectar errores a tiempo y no arrastrarlos hasta el final del desarrollo y que después resulte en un costoso retrabajo con una gran repercusión en el presupuesto final del proyecto.

Finalmente, asignaturas como Proceso de software o Proyecto de ingeniería de software nos permite acercarnos a lo que sería un desarrollo de un software mucho más realista. En estas asignaturas aprendemos a trabajar mucho mejor con los compañeros, a poner en valor la comunicación constante con el equipo y que tengamos pinceladas ya más realistas en cuanto a un trabajo profesional se refiere.

10. Continuidad y Conclusiones

Junto a varios amigos, tenemos idea de continuar este proyecto más allá del proyecto de TFG, ya que este proyecto es como la V0, y la idea es sacar una aplicación nativa tanto para Android como para iOS, así como, reforzar el desarrollo realizado en la parte de Angular.

Algunos ejemplos de una funcionalidad extra que se le brindará:

1. Cambio en la autenticación debido a que ahora se utiliza BasicAuth y tenemos en idea integrar un servicio ADFS con autenticación mediante tokens y en el backend implementar una pieza llamada Envoy que sirve para todo el flujo de seguridad en la parte backend y que está conectado con el microservicio.
2. Implementar como antes mencioné las aplicaciones nativas.
3. Implementar la funcionalidad de comparador de precios para adquirir nuevos videojuegos. Está funcionalidad puede ser una de las más complejas debido a que se implementarán varios microservicios conectados con APIs públicas, se hará uso de Kafka y SSE (Server-Side Events) para poder ofrecer la mejor experiencia al usuario.
4. Implementación de métricas para poder monitorizar todas las actividades del microservicio y poder prevenir errores y proporcionar un mantenimiento excepcional
5. Ofrecer la funcionalidad de crear una wishlist de los videojuegos que el cliente desea comprar en un futuro, que integrado con Kafka y SSE, podrá recibir en tiempo real las notificaciones de bajadas de precio u otras ofertas relacionadas con el producto en cuestión.

Como conclusión del TFG, me ha servido mucho afianzar todos los conocimientos referentes al desarrollo de un proyecto y más aún afianzar varios de los conceptos que ya poseía sobre Arquitectura y desarrollo. Respecto a la parte de frontend, esta era la primera toma real que tenía de programar un proyecto frontend en nuevas tecnologías y la verdad que me ha servido mucho, tengo ya nociones más fuertes respecto a su funcionamiento y, podría decir también, que si me tengo que enfrentar a un proyecto real más sencillo, tendría la capacidad de analizar y mejorar o mantener dicho proyecto.

Puede que este proyecto no sea tan extenso como los demás, pero hay muchas horas de investigación y de aprendizaje que han servido para que desarrolle de la mejor forma posible el proyecto.

11. Referencias

[1] Documentación oficial Angular <<https://angular.io/>>

[2] Jeremy Wilken, <<Angular in Action>>

[3] Documentación oficial SpringBoot <<https://spring.io/>>

[4] Craig Walls, <<Spring Boot in Action>>

[5] Página Oficial PostgreSQL, <<https://www.postgresql.org/>>

[6] Solomon Hykes, Página Oficial, <<https://www.docker.com/>>

[7] ADFS, Documentación oficial, <<https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/ad-fs-overview>>

[8] Matt Klein, Creador de Envoy, Documentación Oficial <<https://www.envoyproxy.io/>>

[9] Jay Kreps, Documentación Oficial Kafka <<https://kafka.apache.org/documentation.html>>

[10] Maven, Documentación Oficial <<https://maven.apache.org/>>

[11] CLZ Games, Página Oficial <<https://www.collectorz.com/game/clz-games>>

12. Apéndice de términos

12.1 Server-Sent Events

SSE (Server-Sent Events) se refiere a una tecnología que permite a los servidores enviar actualizaciones de forma asincrónica a los navegadores web. Puedes incluir en tu TFG que SSE es una técnica útil para crear aplicaciones web en tiempo real, como chats en vivo o paneles de control de datos en tiempo real, ya que permite que el servidor envíe información nueva a los clientes de manera automática sin que estos tengan que hacer solicitudes repetitivas. SSE utiliza el protocolo HTTP y es más eficiente que las solicitudes periódicas AJAX para ciertos casos de uso de aplicaciones web en tiempo real.

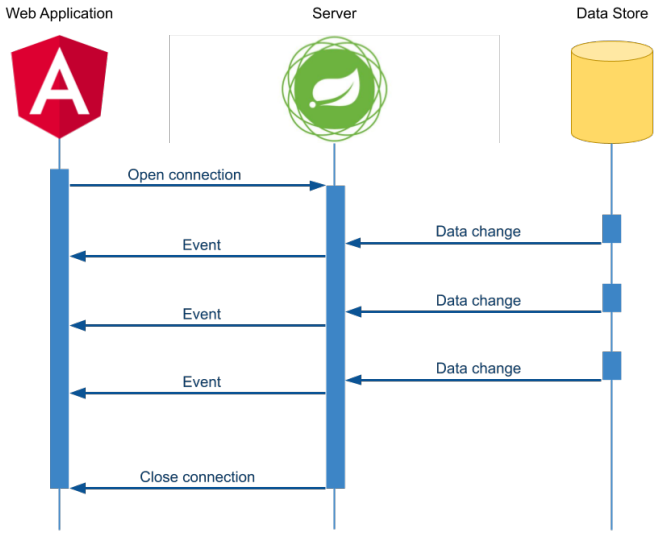


Figura 23: Funcionamiento de Server-Sent Events

12.2 Wishlist

Una "wishlist" (lista de deseos en español) es una lista que una persona crea para enumerar y llevar un registro de los productos, objetos o experiencias que le gustaría tener o recibir en el futuro. Esta lista puede incluir una variedad de elementos, como regalos para ocasiones

especiales, artículos que la persona planea comprar eventualmente o cosas que le gustaría adquirir en algún momento.

Las wishlists son especialmente comunes en el contexto de compras en línea y comercio electrónico, donde los usuarios pueden agregar productos a sus listas de deseos en sitios web de tiendas en línea. Esto les permite guardar artículos que les interesan y, en muchos casos, recibir notificaciones o recordatorios cuando haya descuentos, ofertas especiales o cuando un artículo vuelva a estar disponible.

En resumen, una wishlist es una herramienta que ayuda a las personas a organizar y mantener un registro de sus deseos y preferencias en términos de compras o regalos, facilitando la toma de decisiones y la planificación de futuras adquisiciones.

12.3 Investigaciones sobre el proyecto

Como refuerzo de aprendizaje, y, por poner en valor todo el esfuerzo realizado en este proyecto dejo algunos de los artículos más interesantes en los que base todos los fundamentos de este proyecto.

Clean Architecture:

https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

https://developers.redhat.com/articles/2023/08/08/implementing-clean-architecture-solutions-practical-example

https://betterprogramming.pub/the-clean-architecture-beginners-guide-e4b7058c1165

Hexagonal Architecture:

https://tsh.io/blog/hexagonal-architecture/#:~:text=Hexagonal%20architecture%20is%20a%20pattern,databases%20from%20the%20core%20application.

https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c

https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f

Cloud Agnostic:

https://medium.com/@cuemby/qu%C3%A9-es-ser-cloud-agnostic-y-por-qu%C3%A9-debes-usarlo-3ae2b8c3553

https://www.thoughtworks.com/es-es/insights/blog/kubernetes-exciting-future-developers-and-infrastructure-engineering-0

13. ODS

Aplicar los Objetivos de Desarrollo Sostenible (ODS) a un proyecto de software como una biblioteca en línea de videojuegos con funciones de wishlist y comparador de precios es una excelente manera de asegurarse de que el proyecto tenga un impacto positivo en la sociedad y el medio ambiente. Aquí algunas relaciones de los ODS con el proyecto:

1. **ODS 3: Salud y Bienestar:** La aplicación promueve la salud y el bienestar al proporcionar información sobre los videojuegos, sus clasificaciones de contenido y posiblemente consejos sobre el tiempo de juego saludable.
2. **ODS 9: Industria, Innovación e Infraestructura:** El proyecto de software en sí mismo representa la innovación en la industria del entretenimiento y la tecnología. Además, alentar a los usuarios a comparar precios podría ayudar a fomentar una competencia justa en la industria de los videojuegos.
3. **ODS 12: Producción y Consumo Responsables:** Al promover la responsabilidad en el consumo al incluir información sobre los impactos ambientales de los videojuegos se alenta a los usuarios a tomar decisiones informadas sobre sus compras.
4. **ODS 17: Alianzas para lograr los objetivos:** Para lograr este objetivo, se busca colaborar con organizaciones o empresas que promuevan los ODS o trabajar en alianza con sitios web que comparten información sobre juegos y precios.

ODS	Objetivo de Desarrollo Sostenible	Relación con el proyecto
ODS 3	Salud y Bienestar	La aplicación promueve la salud y el bienestar al proporcionar información sobre los videojuegos, sus clasificaciones de contenido y posiblemente consejos sobre el tiempo de juego saludable.
ODS 9	Industria, innovación e infraestructura	El proyecto de software en sí mismo representa la innovación en la industria del

		entretenimiento y la tecnología. Además, alentar a los usuarios a comparar precios podría ayudar a fomentar una competencia justa en la industria de los videojuegos.
ODS 12	Producción y Consumo Responsables	Al promover la responsabilidad en el consumo al incluir información sobre los impactos ambientales de los videojuegos se alenta a los usuarios a tomar decisiones informadas sobre sus compras.
ODS 17	Alianzas para lograr los objetivos	Para lograr este objetivo, se busca colaborar con organizaciones o empresas que promuevan los ODS o trabajar en alianza con sitios web que comparten información sobre juegos y precios.