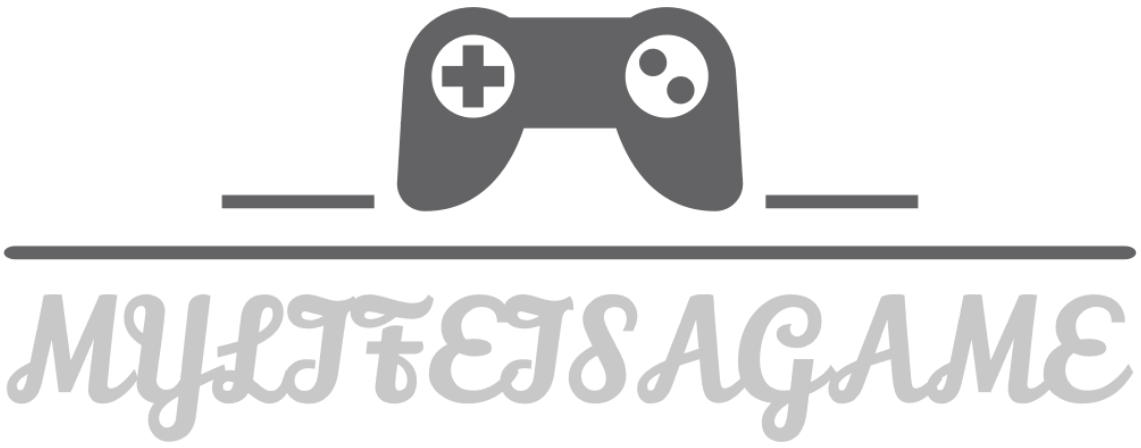


Real Time API Integration: My Life Is A Game

Israel Lozano Callado
Grado en Ingeniería Informática
Java EE

Antoni Oller Arcas

01/2023

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Real Time API Integration: My Life Is A Game
Nombre del autor:	Israel Lozano Callado
Nombre del consultor:	Antoni Oller Arcas
Fecha de entrega (mm/aaaa):	01/2023
Área del Trabajo Final:	Java EE
Titulación:	Grado en Ingeniería Informática

Resumen del Trabajo (máximo 250 palabras):

El objetivo de este proyecto es el análisis y realización de pruebas de concepto de diversas soluciones para la optimización de comunicaciones entre sistemas, basados en APIs, con tal de determinar una arquitectura basada en los escenarios más habituales en la actualidad.

Para conseguir cumplir este objetivo, se han hecho diversos estudios para las diferentes tecnologías usándolas en el desarrollo de nuestras aplicaciones o servicios, para comparar y testear su comportamiento. Así mismo como reto personal se quiere llevar a la práctica de forma real y completa todos los conocimientos y competencias adquiridos en las asignaturas del itinerario del grado de Ingeniería del software.

Las tecnologías que se han estudiado son servicios REST para generar nuestra API, Generación de eventos a través de Kafka, securización y control de nuestra API mediante Oauth y sistema de caché usando Redis.

Al final, decidir qué usar y qué no depende de los sistemas que queremos integrar. Las soluciones estudiadas para este TFG han demostrado ser útiles para solucionar problemas específicos pero fácilmente escalables para otras soluciones.

Abstract (in English, 250 words or less):

The objective of this project is the analysis and proof of concept of various solutions for the optimization of communications between systems, based on APIs, in order to determine an architecture based on the most common scenarios today.

In order to achieve this objective, various studies have been carried out for the different technologies, using them in the development of our applications or services, to compare and test their behavior. Likewise, as a personal challenge, we want to put into practice in a real and complete way all the knowledge and skills acquired in the subjects of the itinerary of the Software Engineering degree.

The technologies that have been studied are REST services to generate our API, event generation through Kafka, security and control of our API through Oauth and cache system using Redis.

In the end, deciding what to use and what not depends on the systems we want to integrate. The solutions studied for this TFG have proven to be useful to solve specific problems but easily scalable for other solutions.

Palabras clave (entre 4 y 8):

Real Time API Integration

Índice

1. Introducción

Introducción a la creación de una infraestructura de API en tiempo real

Reflejando el auge de las aplicaciones basadas en API, el tiempo real se está convirtiendo en una fuerza emergente en el desarrollo de aplicaciones modernas. Está impulsando la mensajería instantánea, las transmisiones en vivo, la geolocalización, el big data y las transmisiones sociales. Pero, ¿qué es el tiempo real y qué significa realmente? ¿Qué tipos de software y tecnología están impulsando esta industria?

Sumerjámonos en ello.

¿Qué es tiempo real?

El tiempo real podría definirse en un sentido temporal más relativo. Podría significar que un cambio en A se sincroniza con un cambio en B. O podría significar que un cambio en A desencadena inmediatamente un cambio en B. O bien... podría significar que A le dice a B que algo cambió, pero B no hace nada. O... ¿significa que A les dice a todos que algo cambió, pero no le importa quién escuche?

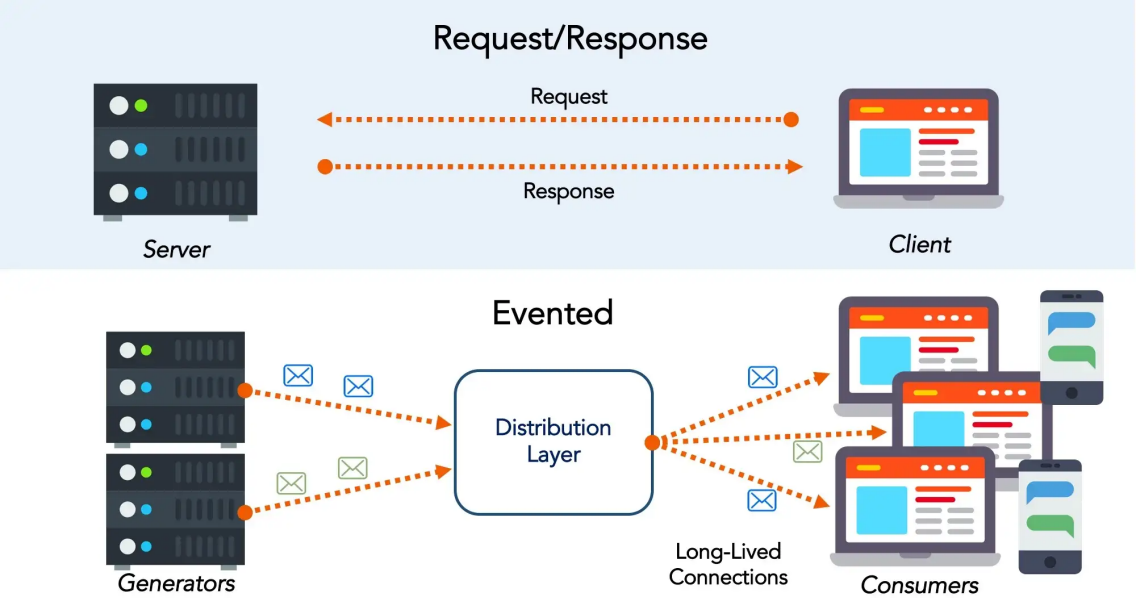
Profundicemos un poco más. En tiempo real no significa necesariamente que algo se actualice al instante (de hecho, no existe una definición singular de "al instante"). Entonces, no nos centramos en el efecto, sino en el mecanismo. *El tiempo real se trata de enviar datos lo más rápido posible: es una comunicación automatizada, sincrónica y bidireccional entre puntos finales a una velocidad de unos pocos cientos de milisegundos.*

- Síncrono significa que ambos puntos finales tienen acceso a los datos al mismo tiempo.
- Bidireccional significa que los datos se pueden enviar en cualquier dirección.
- Los puntos finales son emisores o receptores de datos (teléfono, tablet, servidor).
- Unos pocos cientos de milisegundos es una métrica un tanto arbitraria, ya que los datos no se pueden entregar al instante, pero se alinea más estrechamente con lo que los humanos perciben como tiempo real.

Con esta definición y sus advertencias en mente, exploremos el concepto de enviar datos. Centrándonos en dos puntos base

Comunicaciones asíncronas

Comenzaremos contrastando el envío de datos con la "solicitud-respuesta" (request-response). La solicitud-respuesta es la forma más fundamental en que se comunican los sistemas informáticos. La computadora A envía una solicitud de algo a la computadora B y la computadora B responde con una respuesta. En otras palabras, puede abrir un navegador y escribir "reddit.com". El navegador envía una solicitud a los servidores de Reddit y responden con la página web.



En un modelo de envío de datos, los datos se envían al dispositivo de un usuario en lugar de que el usuario los extraiga (los solicite). Por ejemplo, el correo electrónico push moderno permite a los usuarios recibir mensajes de correo electrónico sin tener que verificarlos manualmente. De manera similar, podemos examinar el impulso de datos en un sentido más continuo, en el que los datos se transmiten continuamente. Cualquiera que tenga acceso a un canal o frecuencia en particular puede recibir esos datos y decidir qué hacer con ellos.

Infraestructura en Tiempo Real como Servicio

“La infraestructura como servicio (IaaS) es una oferta estandarizada y altamente automatizada, en la que los recursos informáticos, complementados con capacidades de almacenamiento y redes, son propiedad de un proveedor de servicios y los alojan, y se ofrecen a los clientes bajo demanda. Los clientes pueden auto aprovisionarse de esta infraestructura mediante una interfaz gráfica de usuario basada en web que sirve como consola de administración de operaciones de TI para el entorno general. **El acceso API a la infraestructura también se puede ofrecer como una opción**”.

A menudo usamos PaaS (plataforma como servicio) y SaaS (software como servicio), entonces, ¿en qué se diferencian de IaaS?

- Infraestructura como servicio (IaaS): el hardware lo proporciona un proveedor externo y lo administra usted.
- Plataforma como servicio (PaaS): tanto el hardware como la capa de su sistema operativo se administran por usted.
- Software como servicio (SaaS): se proporciona una capa de aplicación para la plataforma y la infraestructura (que se administra por usted).

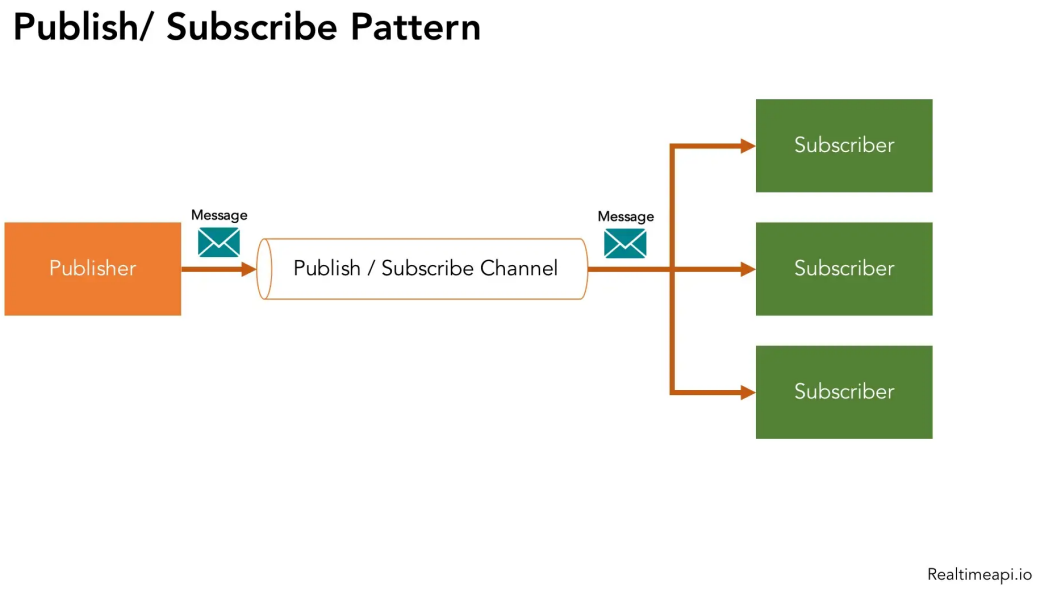
Para potenciar el tiempo real, las aplicaciones requieren un sistema cuidadosamente diseñado de servidores, API, balanceadores de carga, etc. En lugar de construir estos sistemas internamente, las organizaciones encuentran que es más rentable y eficiente en recursos comprar gran parte de esta infraestructura sistémica y luego conducirla internamente. Estos sistemas, por lo tanto, no son solo IaaS, sino que generalmente brindan una plataforma y una capa de software para ayudar con la administración. En términos fundamentales, su beneficio principal es que proporcionan una infraestructura en tiempo real, ya sea que la aloje internamente o confíe en una instancia administrada.

Todo se reduce a la simple verdad de que el tiempo real es difícil por varias razones:

- Demanda de tiempo de actividad del cliente: los clientes que dependen de las actualizaciones en tiempo real se darán cuenta de inmediato cuando su red no esté funcionando.
- Escalabilidad horizontal: debe poder manejar cargas volátiles y masivas en su sistema o correr el riesgo de tiempo de inactividad. Por lo general, esto se logra mediante una escalabilidad horizontal inteligente y sistemas que pueden administrar millones de conexiones simultáneas.
- Complejidad arquitectónica: mantener un sistema en tiempo real con buen rendimiento no solo es complejo, sino que requiere una amplia experiencia y conocimientos. Esto es costoso de comprar, especialmente en el mercado de ingeniería de alta demanda actual.
- Contingencias: inevitablemente, su sistema experimentará algún tiempo de inactividad, ya sea debido a un pico de carga anticipado o a una función recientemente lanzada. Por lo tanto, es importante contar con múltiples contingencias de tiempo de actividad para asegurarse de que el sistema sepa qué hacer, en caso de que su mecanismo principal en tiempo real no funcione.
- Cola: cuando envía una gran cantidad de datos, es probable que necesite un mecanismo de cola intermedio para asegurarse de que sus procesos de back-end no se sobrecarguen con una mayor carga de mensajes.

IaaS de aplicaciones en tiempo real

La infraestructura de aplicaciones en tiempo real envía datos a navegadores y clientes. Por lo general, utiliza mensajes de publicación/suscripción, webhooks y/o websockets, y es independiente de la API principal de una aplicación o servicio. Estas soluciones son las mejores para las organizaciones que buscan mensajería en tiempo real sin necesidad de crear sus propias API en tiempo real.



Estos sistemas también tienen herramientas de administración de software/plataforma mejor construidas además de sus ofertas de infraestructura. Por ejemplo, los principales proveedores tienen herramientas de configuración integradas como controles de acceso, delegación de eventos, herramientas de depuración y configuración de canales.

Beneficios de la aplicación en tiempo real IaaS

- Velocidad: típicamente diseñada explícitamente para entregar datos con latencias bajas a los dispositivos de los usuarios finales, incluidos teléfonos inteligentes, tabletas, navegadores y computadoras portátiles.
- Múltiples SDK para una integración más sencilla.
- Utiliza plataformas de entrega de datos en tiempo real distribuidas globalmente.
- Adaptadores de múltiples protocolos.
- Bien probado en entornos de producción.
- Mantiene la configuración interna al mínimo.

1.1 Objetivos del Trabajo

Según lo visto en el apartado anterior, el objetivo principal de nuestro proyecto va a consistir en el desarrollo de toda la arquitectura e infraestructura mínima necesaria para el desarrollo de una API en tiempo real. Para ello nos fijaremos los siguientes objetivos:

1. Creación de un API de publicación
    a. Siguiendo las directrices OpenAPI Specification[1]
    b. Seguridad de acceso siguiendo el estándar de autenticación OAuth 2.0 *[2]*
    c. Asegurar el número de peticiones que puede recibir nuestro API para no sobrecargar el sistema y defender frente ataques.
    d. Seguir un enfoque de arquitectura hexagonal
2. Mensajería Publish/Service
    a. Habilitar un canal de suscripción para notificaciones
    b. Integrar con proveedor de notificaciones a suscriptores (En nuestro caso usaremos Firebase *[3]*
3. Interfaz de usuario
    a. Generar una interfaz mínima de visualización del contenido
    b. Garantizar acceso securizado para nuestros usuarios
    c. Habilitar subscripción a nuestro sistema de mensajería

1.1.1 Funcionalidades generales

Para justificar el desarrollo de nuestro proyecto expondremos la siguiente funcionalidad.

Actualmente la venta de videojuegos de tiendas online es uno de los negocios más en auge a nivel mundial. Los compradores de videojuegos sienten la necesidad de unificar en una única plataforma todos los videojuegos de las distintas tiendas online. Además de poder subscribirse a determinadas categorías de videojuegos para ser notificados cuando se publica un nuevo videojuego de la categoría favorita del usuario.

Como tienda online quiero poder publicar mis videojuegos en “**My Life Is A Game**”, portal web donde los usuarios se conectan para ver el contenido de distintas tiendas de videojuegos siguiendo el estándar de autenticación Oauth para garantizar la seguridad de mis publicaciones.

Como **tienda online** quiero poder publicar, editar y eliminar videojuegos en el portal vía API.

Como **tienda online** quiero que la documentación entregada sea legible y siga los estándares de OpenAPI Specification.

Como usuario comprador de videojuegos quiero poder conectarme a “My Life Is A Game”, portal web donde puedo encontrar todos los videojuegos de las principales tiendas online de videojuegos.

Como **usuario** para acceder tendré que poder acceder con mi cuenta de gmail.

Como **usuario** tendré la posibilidad de ver todos los videojuegos publicados por las tiendas y la posibilidad de filtrar por las distintas categorías de videojuegos.

Como **usuario** podré inscribirme a una categoría concreta para ser notificado cuando se publique un nuevo juego de esa categoría por cualquiera de las tiendas.

Yo, como administrador, quiero poder dar de alta nuevas tiendas online para poder publicar vía API en el portal.

Los objetivos principales que sigue nuestra propuesta de TFG será la siguiente:

1. Creación de un API REST a la que las distintas tiendas online puedan integrarse de forma sencilla y segura.
2. Creación de una interfaz gráfica en la que puedan navegar los siguientes perfiles:
    a. Un usuario que pueda navegar por los videojuegos publicados en la plataforma y suscribirse a notificaciones de nuevas categorías

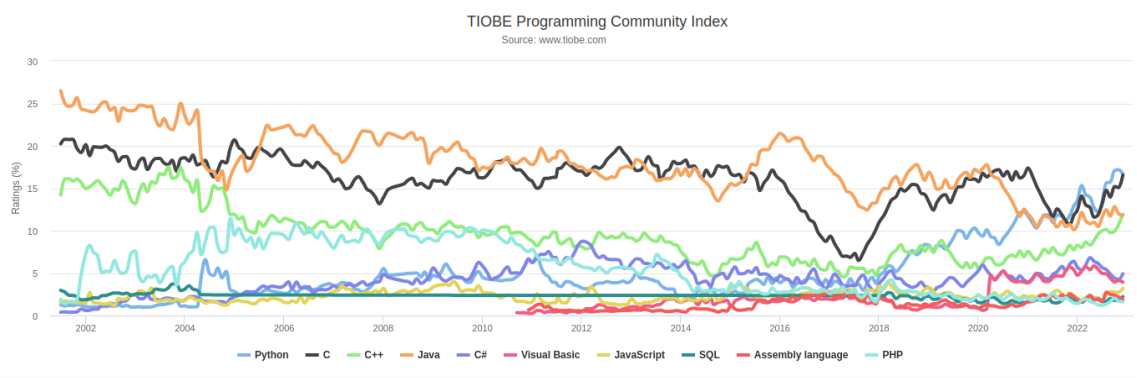
1.1.2 Stack Tecnológico

- JDK 11
- IDE: IntelliJ IDEA Ultimate
- Framework Spring-Boot (servidor tomcat embebido, spring-data, spring-security, flyway)
- Apache Maven como herramienta de software para la gestión y construcción de proyectos Java
- MySQL como gestor de base de datos
- Redis *[4]* como sistema de caché para nuestra política de peticiones

- Apache Kafka y Firebase para nuestras notificaciones PUSH
- Swagger para documentar nuestra API
- Git como software de control de versiones, junto con GitHub como gestor de repositorios.
- React *[5]* para construir nuestra interfaz gráfica
- Docker para ayudarnos a desplegar parte de nuestra infraestructura

El 90% de nuestro lenguaje de programación será en Java. Para justificar nuestra elección podemos fijarnos en índice TIOBE de 2022 donde Java sigue siendo uno de los lenguajes líderes en el sector tecnológico.

Índice TIOBE 2022 *[6]*



1.2 Enfoque y método seguido

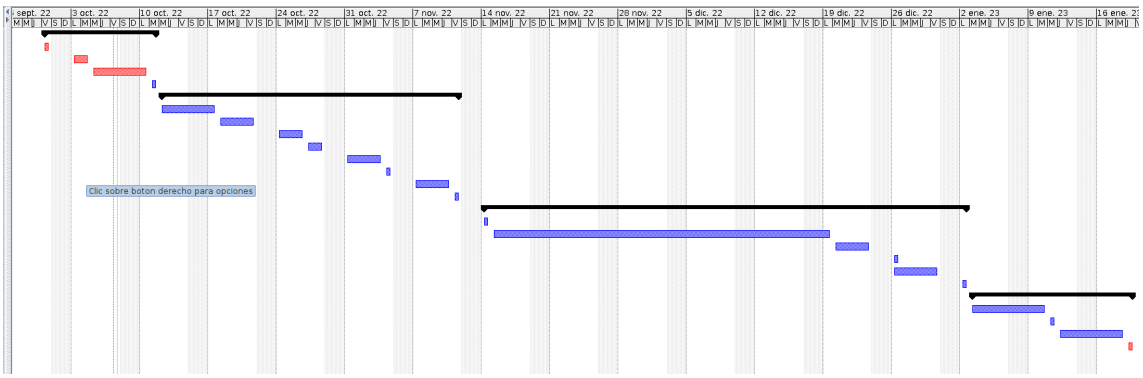
Para el desarrollo de nuestro TFG hemos querido usar todos los conocimientos adquiridos a lo largo del Grado y plasmar este conocimiento en el desarrollo de un producto nuevo usando las tecnologías y metodologías más actuales. De esta forma pondremos en práctica todo lo aprendido a lo largo de estos años y nos servirá de cara a nuestro enfoque profesional.

Hemos decidido seguir un enfoque iterativo basado en la metodología ágil SCRUM, basando nuestros entregables en pequeños sprints para marcarnos objetivos reales. El objetivo final es la entrega de un MVP (mínimo producto viable) que nos garantizara la entrega de los productos mencionados en el apartado -1.2 Objetivos del Trabajo - centrándonos principalmente en las funcionalidades de API que es donde radica la complejidad de este proyecto.

1.3 Planificación del Trabajo

Para la planificación del trabajo se ha seguido la siguiente planificación detallada en el esquema de Gant.

	Nombre	Duracion	Inicio	Terminado
1	**PEC1 - Plan de Trabajo**	8 days	**30/9/22 8:00**	**11/10/22 17:00**
2	Reunión con tutor	1 day	30/9/22 8:00	30/9/22 17:00
3	Estudio de ideas y soluciones técnicas	2 days	1/10/22 8:00	4/10/22 17:00
4	Elaboración memoria PEC1	4 days	5/10/22 8:00	10/10/22 17:00
5	Entrega PEC1	1 day	11/10/22 8:00	11/10/22 17:00
6	**PEC2 - Análisis y Diseño**	23 days	**12/10/22 8:00**	**11/11/22 17:00**
7	Elaboración historias de usuario	4 days	12/10/22 8:00	17/10/22 17:00
8	Elaboración aceptación historias de usuario	4 days	18/10/22 8:00	21/10/22 17:00
9	Diseño modelo de datos	3 days	22/10/22 8:00	26/10/22 17:00
10	Diagrama de clases	2 days	27/10/22 8:00	28/10/22 17:00
11	Diseño arquitectura	4 days	29/10/22 7:00	3/11/22 17:00
12	Envío a tutoría para revisión	1 day	4/11/22 8:00	4/11/22 17:00
13	Modificaciones y correcciones	4 days	5/11/22 8:00	10/11/22 17:00
14	Entrega PEC2	1 day	11/11/22 8:00	11/11/22 17:00
15	**PEC3 - Implementación**	36 days	**14/11/22 8:00**	**2/1/23 17:00**
16	Instalación entornos de desarrollo	1 day	14/11/22 8:00	14/11/22 17:00
17	Iteranciones sobre historias de desarrollo	25 days	15/11/22 8:00	19/12/22 17:00
18	Documentación	4 days	20/12/22 8:00	23/12/22 17:00
19	Envío para revisión	1 day	24/12/22 8:00	26/12/22 17:00
20	Modificaciones y correcciones	5 days	26/12/22 8:00	30/12/22 17:00
21	Entrega PEC3	1 day	2/1/23 8:00	2/1/23 17:00
22	**PEC4 - Memoria y presentación**	13 days	**3/1/23 8:00**	**19/1/23 17:00**
23	Documentación Memoria	6 days	3/1/23 8:00	10/1/23 17:00
24	Revisión con tutor	1 day	11/1/23 8:00	11/1/23 17:00
25	Modificaciones y correcciones	5 days	12/1/23 8:00	18/1/23 17:00
26	Entrega PEC4	1 day	19/1/23 8:00	19/1/23 17:00



1.4 Breve sumario de productos obtenidos

Plan de Trabajo
Definición de los requisitos funcionales, el stack tecnológico elegido, el objetivo del proyecto y la planificación

Análisis y Diseño del sistema
Documento de análisis y diseño del proyecto a desarrollar, con la especificación de los casos de uso y su alcance. Además, se incluyen los diagramas de arquitectura y de clases.

Implementación
Implementación completamente funcional en cuanto a los requisitos especificados para este trabajo. Además, se genera un manual de instalación y toda la documentación de las pruebas del sistema.

Presentación y Memoria
Finalmente, se obtendrá el presente documento completo junto con una presentación del trabajo realizado en diapositivas y un video como presentación virtual del trabajo desarrollado.

1.5 Breve descripción de los otros capítulos de la memoria

- Análisis: En este apartado se analizará los requisitos y funcionalidades necesarias para resolver el problema planteado.

- Diseño: Diseño elegido para implementar el trabajo a desarrollar, todos los componentes necesarios para cubrir los requerimientos, estudiaré el punto de vista de la información y la arquitectura del sistema.

- Implementación: En este capítulo concretará a más bajo nivel como se ha llevado a cabo la implementación del proyecto en las diferentes capas, detallará los inconvenientes y sensaciones a lo largo del desarrollo del trabajo y por último indicará las herramientas utilizadas.

- Pruebas: Este capítulo sirve como introducción y planteamiento del juego de pruebas llevado a cabo, que posteriormente se desarrolla con más detalle en el Anexo de Tests unitarios y pruebas del sistema.

- Mejoras y Evolución del producto: Todas las mejoras posibles y soluciones que me hubiera gustado abordar pero que por la limitación temporal de este trabajo, he tenido que dejar fuera del alcance del mismo.

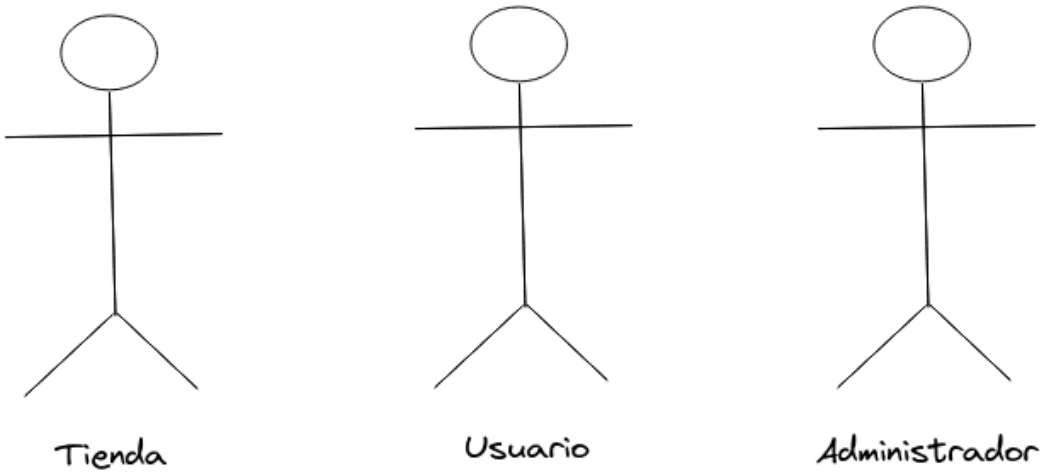
- Conclusiones: En este apartado, se comentará mi perspectiva sobre todo el proceso de elaboración de este trabajo

2. Análisis

Basándonos en los objetivos definidos en los apartados anteriores sobre el proyecto que vamos a desarrollar, pasamos a identificar los roles que van a estar implicados en nuestro proyecto, los requisitos a cumplir, los módulos implicados y la arquitectura de nuestro sistema.

2.1 Roles

Vamos a contemplar tres roles en nuestro sistema que detallamos a continuación:



→**Tienda:** Podrá acceder a los recursos públicos y asociados a su rol. Los recursos asociados serán a peticiones POST, PUT y DELETE sobre sus datos y anuncios de videojuegos.

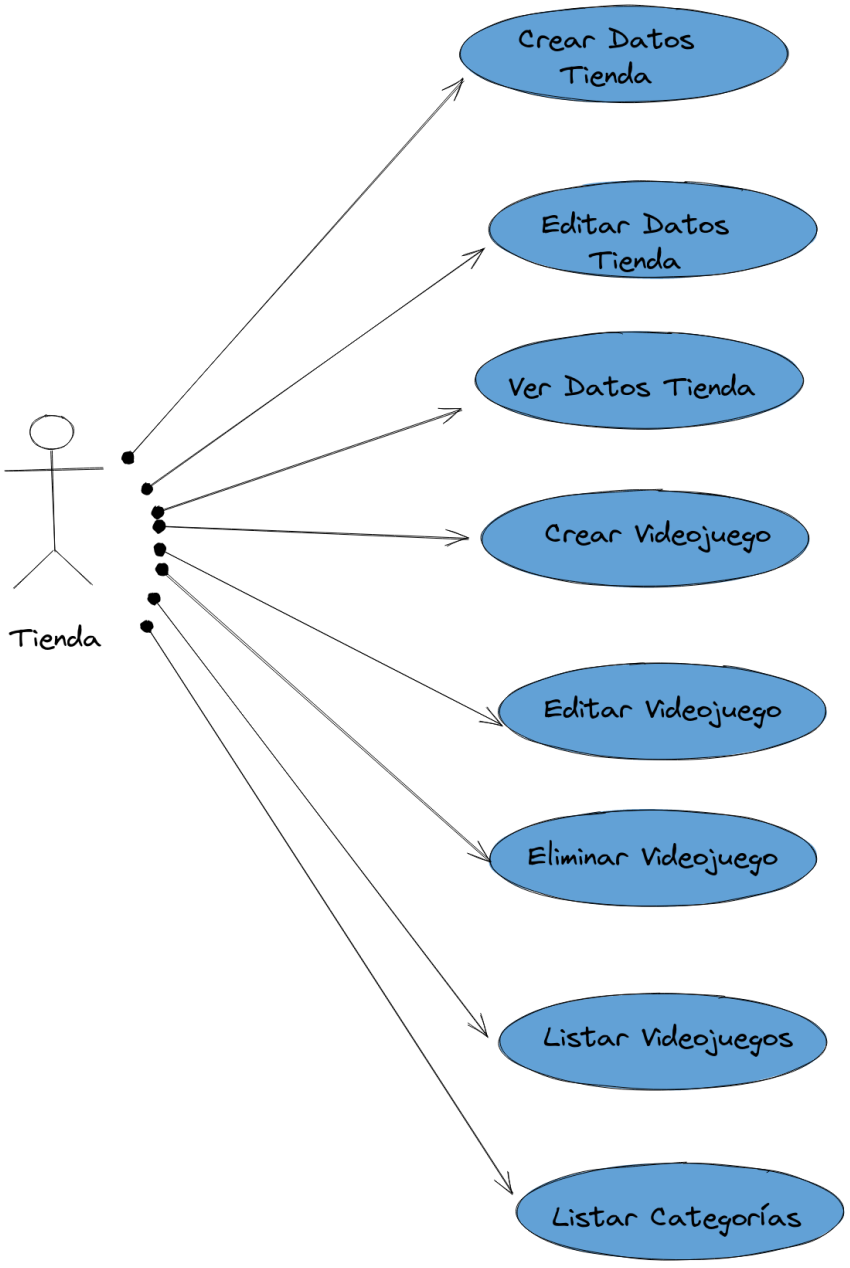
→ **Usuario:** Podrá hacer uso de los recursos públicos del API. Podrá visualizar el contenido publicado por las empresas y suscribirse a una categoría para recibir notificaciones.

→**Administrador:** Podrá acceder a todos los recursos públicos, podrá acceder a recursos protegidos asociados a su rol, tales como peticiones POST, PUT y DELETE sobre categorías y habilidades y PUT y DELETE sobre empresas y profesionales

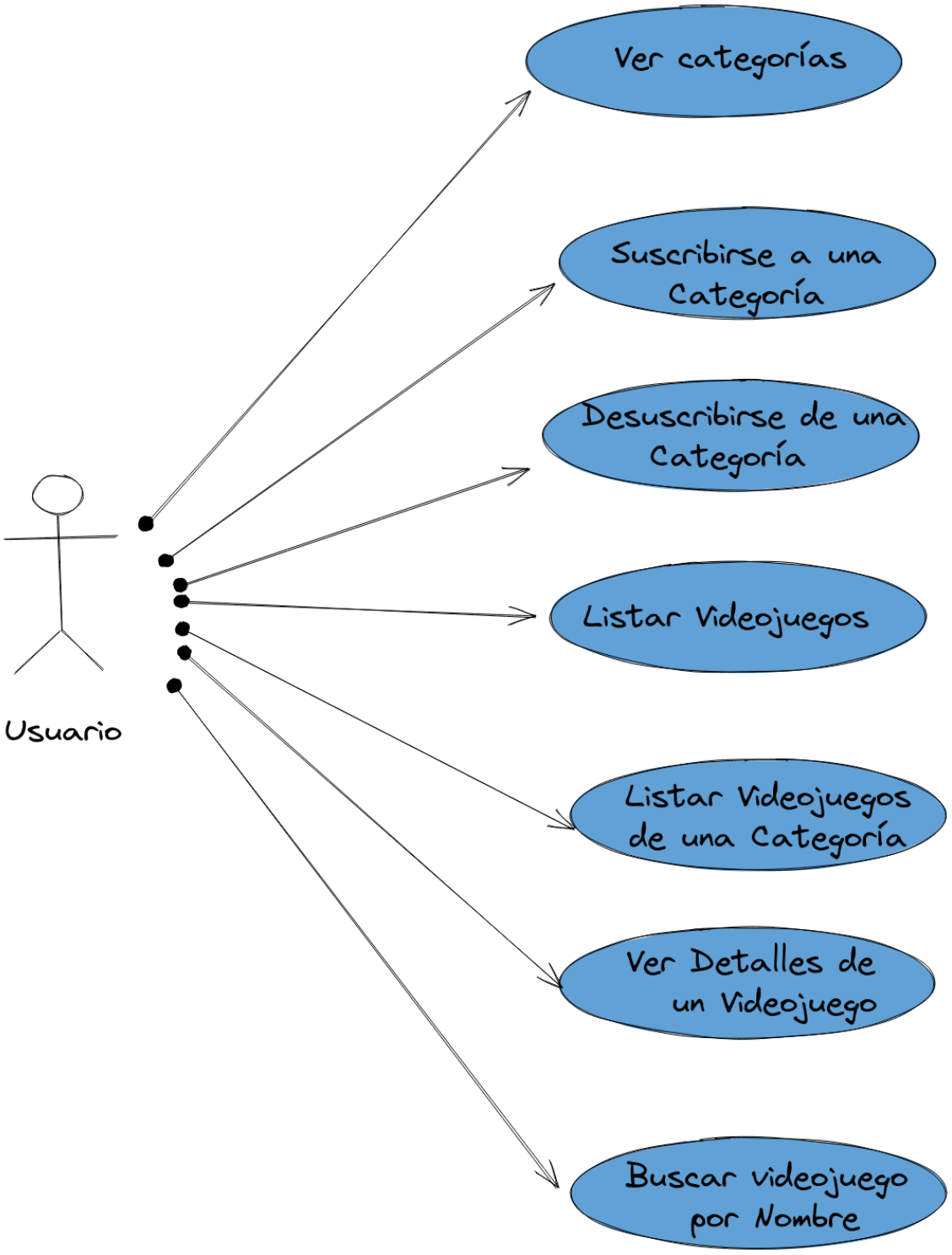
2.1.1 Diagrama UML de Casos de Uso

En el apartado 2.3 se describe en más detalle las historias de usuario

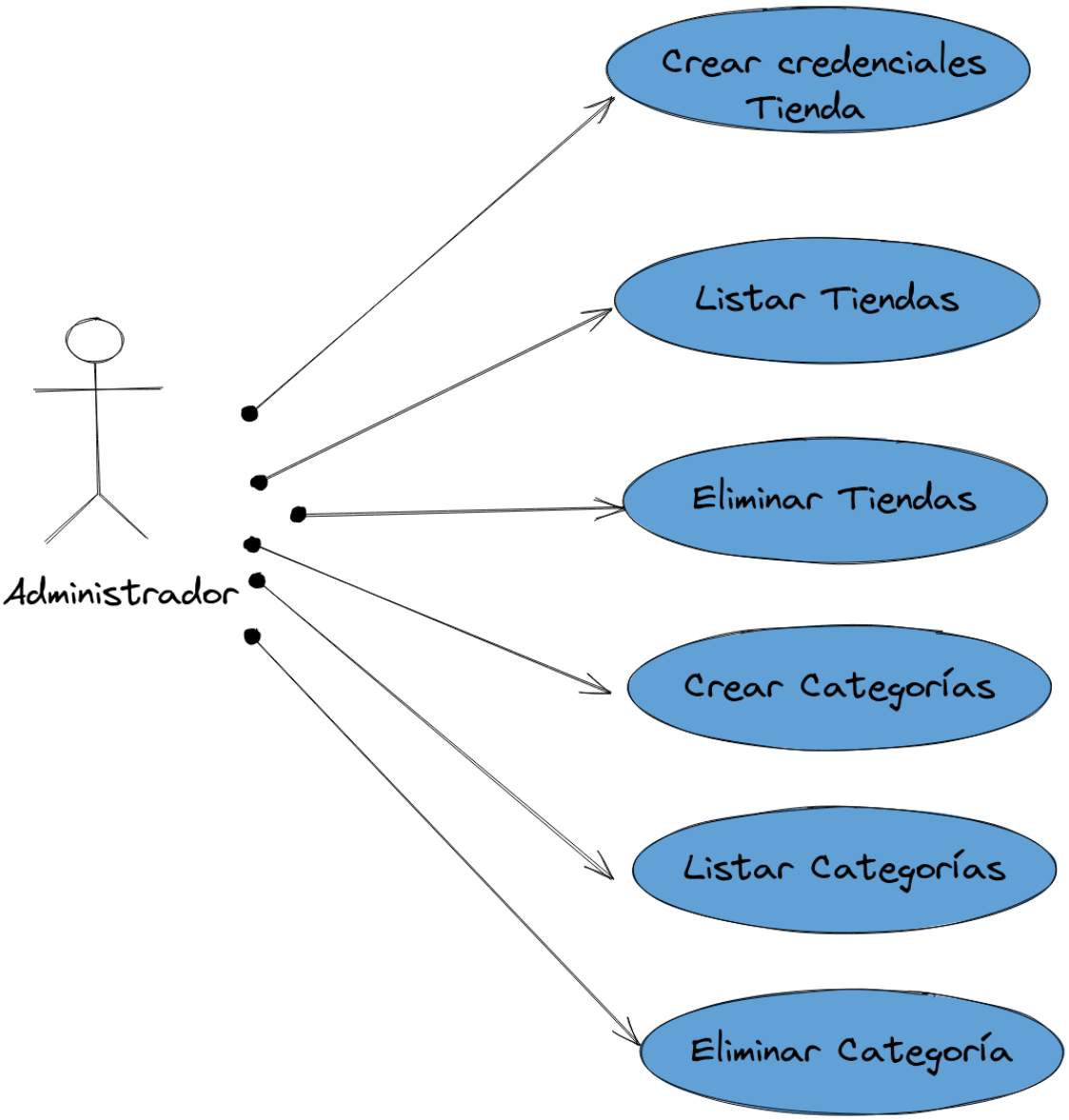
Casos de Uso de las Tiendas



Casos de Uso de los Usuarios



Casos de Uso del Administrador



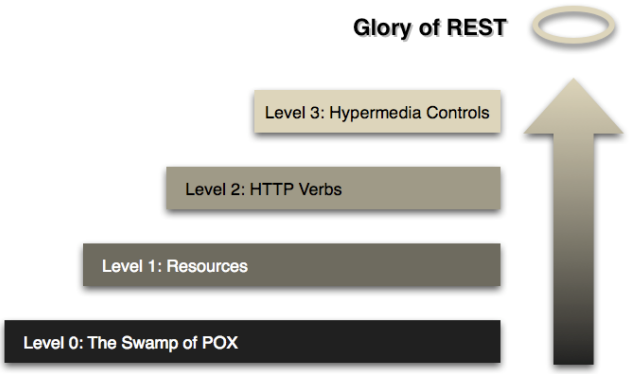
2.2 Requisitos no funcionales

El desarrollo del proyecto seguirá una metodología ágil de tipo SCRUM buscando los siguientes objetivos:

- La auto organización, porque Scrum está muy enfocado a ser capaz de autogestionarse, de saber llevar la carga de trabajo en todo momento, de que se tenga el control del tiempo, etc. Por todo eso debido al tiempo para el desarrollo de nuestro proyecto, marcaremos ritmos y el desarrollo en progresivo, ya que se va avanzando a medida que va pasando el tiempo y vamos adquiriendo madurez y se procede a un valor incremental.

- La priorización, debido a que existen prioridades y se establecen criterios para saber qué trabajos son más importantes y son los primeros que hay que desarrollar. Scrum es una metodología muy abierta, flexible y que se adapta a las necesidades de los clientes en cada momento. Si aparece algo urgente, se le puede asignar una prioridad y ponerse a trabajar de forma inmediata.

- Nuestra aplicación seguirá los principios de la arquitectura REST:



Verbs Methods

GET, POST, PUT and DELETE

Recursos

- Aceptar y responder con JSON
- Usar nombres en lugar de verbos en el path de los endpoints
- Nombrar colecciones con nombres en plural
    - GET /v1/shops
    - GET /v1/shops/1234
- Recursos anidados para jerarquía de objetos
    - GET /v1/shops/1234/categories

    - Manejar errores de manera eficiente y devolver códigos de errores estándares de HTTP

- El body de las peticiones será en formato JSON para transferir la información.
- Todas las peticiones para las empresas deben estar autenticadas usando Oauth 2.0 para securizar el acceso.
- Los tiempos de respuesta deben ser razonables.
- Se debe evitar la sobrecarga del API usando un límite de peticiones para lo que se implementará un sistema de caché usando Redish.
- Se usará MySQL como gestor de base de datos.

2.3 Requisitos funcionales

Para el servicio que queremos ofrecer estableceremos los requisitos funcionales que se muestran a continuación siguiendo las siguientes categorías:

→ **Datos de las tiendas:** Las tiendas podrán dar de alta sus datos en el sistema y gestionar su perfil. También podrán gestionar la publicación de sus videojuegos dentro de una de las categorías disponibles. (Tienda)

→ **Datos de los usuarios:** los usuarios podrán acceder al listado de los videojuegos publicados, pudiendo filtrar por categorías de videojuegos y subscribirse a una categoría para ser notificado siempre que se publique un videojuego nuevo perteneciente a esa categoría. (Usuario)

→**Datos de administración:** El administrador podrá llevar a cabo el alta y desactivación de las tiendas para acceder al API de publicación. A su vez podrá gestionar las distintas categorías de videojuegos.

2.3.1 Historias de usuario

Historias de usuario	
US - 001	Como tienda quiero poder dar de alta mis datos profesionales y estar visible en la plataforma
US - 002	Como tienda quiero poder dar de alta mis videojuegos para que sean visibles para el usuario
US - 003	Como tienda quiero saber las categorías de videojuegos disponibles para poder asignar una categoría a mis videojuegos para el usuario
US - 004	Como usuario quiero poder registrarme en el sistema usando mi cuenta de

	Google
US - 005	Como usuario quiero poder ver el listado de todos los videojuegos disponibles
US - 006	Como usuario quiero poder filtrar por categoría todos los videojuegos disponibles
US - 007	Como usuario quiero poder buscar por nombre un videojuego
US - 008	Como usuario quiero poder suscribirme a una categoría concreta de videojuegos para recibir notificaciones sobre esa categoría cada vez que se publique un nuevo videojuego
US - 009	Como administrador quiero poder dar de alta credenciales para nuevas tiendas
US - 010	Como administrador quiero poder consultar todas las tiendas dadas de alta en el sistema
US - 011	Como administrador quiero poder desactivar tiendas dando de baja todos los videojuegos publicados por la tienda desactivada
US - 012	Como administrador quiero poder crear nuevas categorías para los videojuegos publicados
US - 0113	Como administrador quiero poder consultar todas las categorías disponibles

2.3.1.1 Especificación de las historias de usuario

DETALLE HISTORIA 001

US - 001	Como tienda quiero poder dar de alta mis datos profesionales y estar visible en la plataforma
Criterios de aceptación	
Dado una tienda que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al endpoint del servicio REST adecuado con una request válida que contiene los parámetros relativos a los datos de la tienda	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: shopId	
API	
POST	/shop

RESPONSE STATUS	201 Created
RESPONSE	shopId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error
PUT	/shop/{shopId}
RESPONSE STATUS	200 OK
RESPONSE	shopId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 002

US - 002	Como tienda quiero poder dar de alta mis videojuegos para que sean visibles para el usuario
Criterios de aceptación	
Dado una tienda que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a los datos de un videojuego	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: videogameId	
API	
POST	/shops/{shopId}/videogames
RESPONSE STATUS	201 Created
RESPONSE	videogameId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)

	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error
PUT	/shops/{shopId}/videogames/{videogameId}
RESPONSE STATUS	200 OK
RESPONSE	videogameId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error
DELETE	/shops/{shopId}/videogames/{videogameId}
RESPONSE STATUS	200 OK
RESPONSE	videogameId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 003

US - 003	Como tienda quiero saber las categorías de videojuegos disponibles para poder asignar una categoría a mis videojuegos para el usuario
Criterios de aceptación	
Dado una tienda que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a los datos de la tienda	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: listado de todas las categorías disponibles	
API	
GET	/shops/{shopId}/categories
RESPONSE STATUS	200 OK

RESPONSE	categories list
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 004

US - 004	Como usuario quiero poder registrarme en el sistema usando mi cuenta de Google
Criterios de aceptación	
Dado un usuario que accede al API en una aplicación cliente usando Google	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a los datos de la tienda	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: userId	
API	
POST	/user/login
RESPONSE STATUS	200 OK
RESPONSE	no response
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error
POST	/user/logout
RESPONSE STATUS	200 OK
RESPONSE	no response
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 005

US - 005	Como usuario quiero poder ver el listado de todos los videojuegos disponibles
Criterios de aceptación	
Dado un usuario que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos para poder ver todos los videojuegos	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: listado de todos los videojuegos	
API	
GET	/users/{userId}/videogames
RESPONSE STATUS	200 OK
RESPONSE	videogames list
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error
GET	/users/{userId}/videogames/{videogameId}
RESPONSE STATUS	200 OK
RESPONSE	videogame
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 006

US - 006	Como usuario quiero poder filtrar por categoría todos los videojuegos disponibles

Criterios de aceptación	
Dado un usuario que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a la categoría de los videojuegos	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: listado de todos los videojuegos por categoría	
API	
GET	/users/{userId}/videogames/categories/{categoryId}
RESPONSE STATUS	200 OK
RESPONSE	videogames list
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 007

US - 007	Como usuario quiero poder filtrar por categoría todos los videojuegos disponibles
Criterios de aceptación	
Dado un usuario que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos al nombre de los videojuegos	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: listado de todos los videojuegos por nombre	
API	
GET	/users/{userId}/videogames/categories/{search}
RESPONSE STATUS	200 OK
RESPONSE	videogames list

ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 008

US - 008	Como usuario quiero poder suscribirme a una categoría concreta de videojuegos para recibir notificaciones sobre esa categoría
Criterios de aceptación	
Dado un usuario que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a la suscripción de una categoría	
Entonces el cliente recibe un código de estado 202 (aceptado)	
Y en la respuesta se recibe la siguiente información: categoryId	
API	
POST	/users/{userId}/videogames/categories/{categoryId}/subscribe
RESPONSE STATUS	200 OK
RESPONSE	categoryId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 009

US - 009	Como administrador quiero poder dar de alta credenciales para nuevas tiendas
Criterios de aceptación	
Dado un administrador que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una	

request válida que contiene los parámetros relativos a los credenciales de una tienda	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: tokenid y clientId	
API	
POST	/admins/shops
RESPONSE STATUS	201 Created
RESPONSE	tokenId y clientId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 010

US - 010	Como administrador quiero poder consultar todas las tiendas dadas de alta en el sistema
Criterios de aceptación	
Dado un administrador que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida para obtener todas las tiendas	
Entonces el cliente recibe un código de estado 200 (ok)	
Y en la respuesta se recibe la siguiente información: shops list	
API	
POST	/admins/shops
RESPONSE STATUS	200 OK
RESPONSE	shops list
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 011

US - 011	Como administrador quiero poder desactivar tiendas dando de baja todos los videojuegos publicados por la tienda desactivada
Criterios de aceptación	
Dado un administrador que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a la desactivación de una tienda	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: shopId	
API	
DELETE	/admins/shops/{shopId}
RESPONSE STATUS	200 OK
RESPONSE	shopId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

DETALLE HISTORIA 012

US - 012	Como administrador quiero poder crear nuevas categorías para que los videojuegos publicados
Criterios de aceptación	
Dado un administrador que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a la creación de una categoría	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: categoryId	
API	
POST	/admins/categories

RESPONSE STATUS	201 Created
RESPONSE	categoryId
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

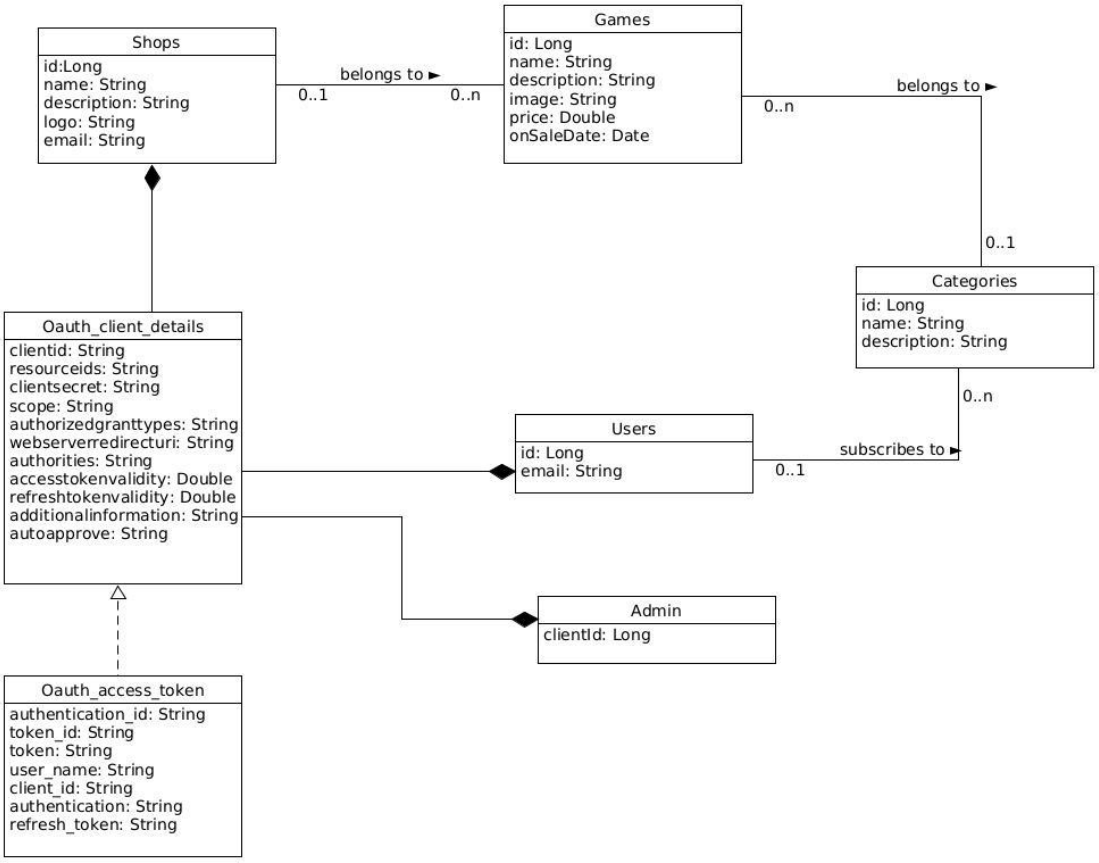
DETALLE HISTORIA 013

US - 013	Como administrador quiero poder consultar todas las categorías disponibles
Criterios de aceptación	
Dado un administrador que accede al API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a la creación de una categoría	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: category list	
API	
GET	/admins/categories
RESPONSE STATUS	201 Created
RESPONSE	category list
ALTERNATIVE	400 Bad Request (error en parámetros de entrada)
	401 Unahorized (si no está logado o no tiene el tokenId)
	500 Internal Server Error

3. Diseño

3.1 Diagrama de Clases

Siguiendo los requisitos planteados en la sección anterior, el siguiente diagrama de clases desde el punto de vista de la información con los que trabajará nuestro sistema



Detalles de las clases

→ **Shops**: Detalles de las tiendas de videojuegos, las cuales tendrán que autenticarse mediante un clientId y clientSecret de Oauth. Además guardaremos su nombre, descripción, email de contacto y logo de la tienda.

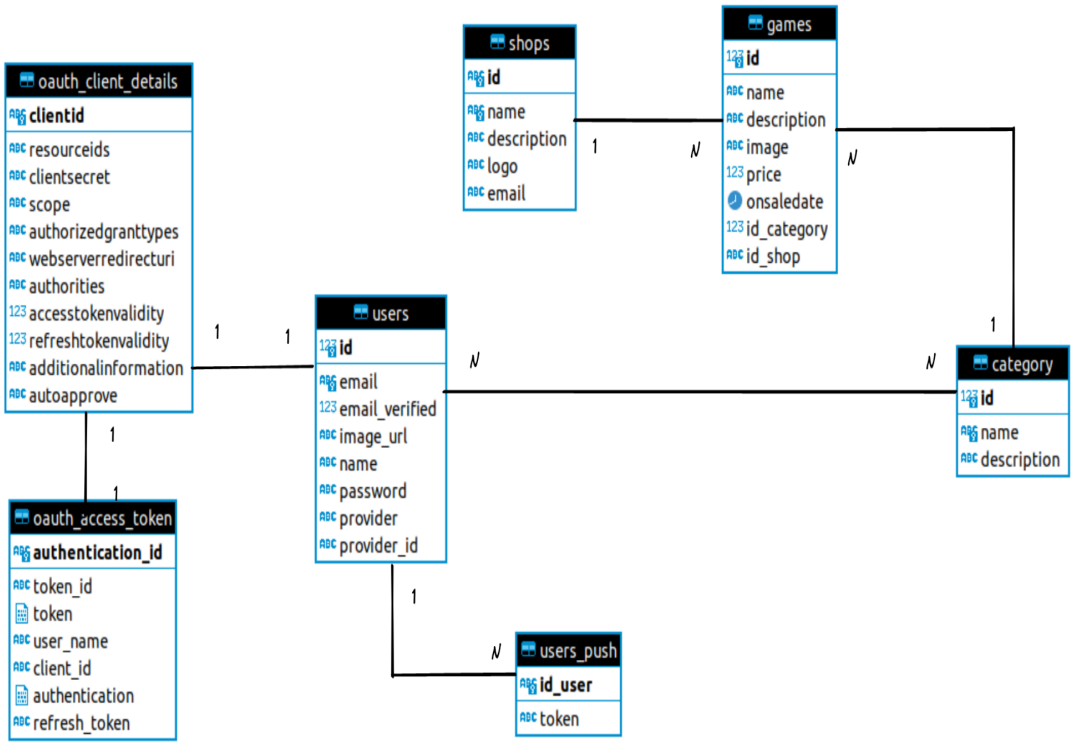
→ **User**: Usuarios que acceden a nuestro sistema, tienen que autenticar mediante un clientId y clientSecret autogestionado por Google con Oauth

→ **Admin**: El administrador del sistema, tendrá que autenticarse mediante un clientId y clientSecret de Oauth

→ **Games**: Detalles de los juegos que publican las tiendas. Se guardará el nombre, descripción, una imágen, el precio y la fecha de venta del videojuego
→ **Categories**: Las categorías a las que puede pertenecer un videojuego y a la que se pueden suscribir los usuarios. Guardaremos el nombre y descripción de la categoría.

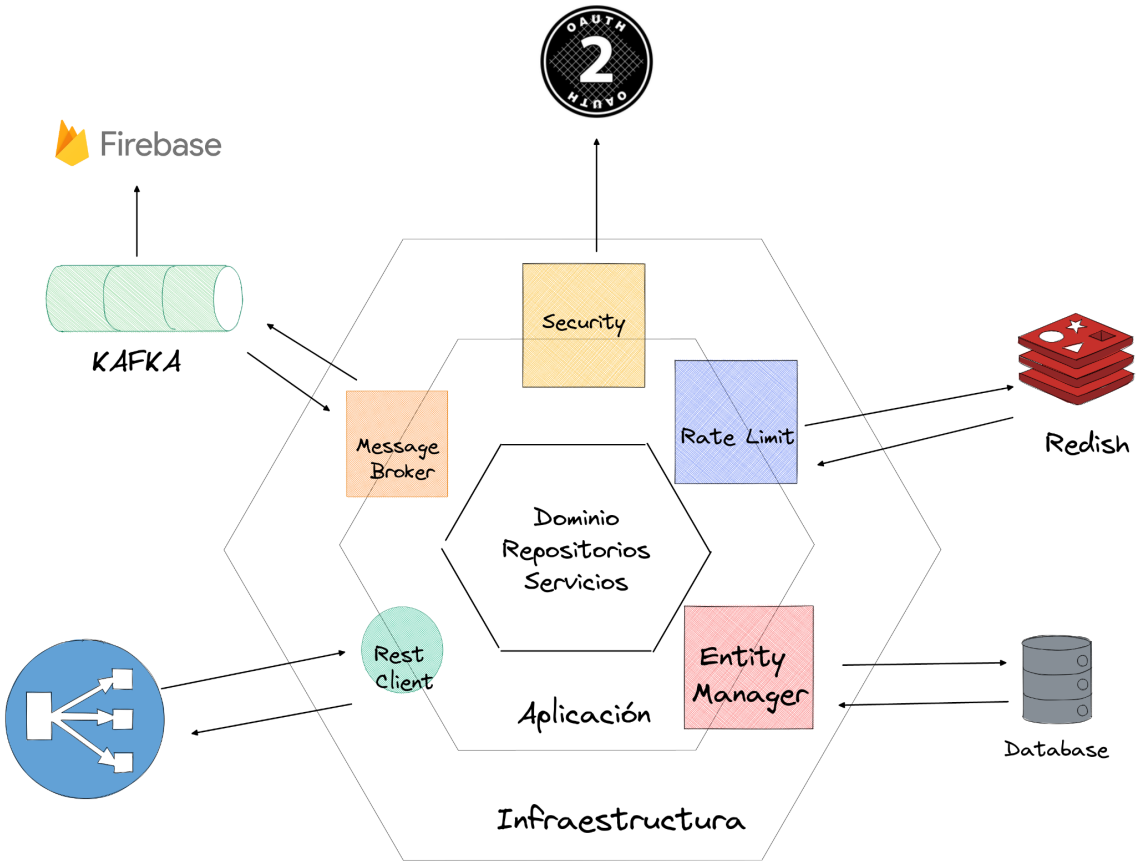
3.2 Diagrama de Entidad-Relación

A continuación estableceremos el diagrama de entidades relacionales que persistirán en nuestra base de datos, de forma orientada a nuestro SGBD elegido que en nuestro caso será MySQL.



3.3 Diagrama de Arquitectura

Nuestro sistema seguirá un enfoque de Arquitectura Hexagonal *[7]*. Tiene como principal motivación separar nuestra aplicación en distintas capas o regiones con su propia responsabilidad. De esta manera conseguimos desacoplar capas de nuestra aplicación permitiendo que evolucionen de manera aislada. Además, tener el sistema separado por responsabilidades nos facilitará la reutilización



3.3.1 Capa de Seguridad

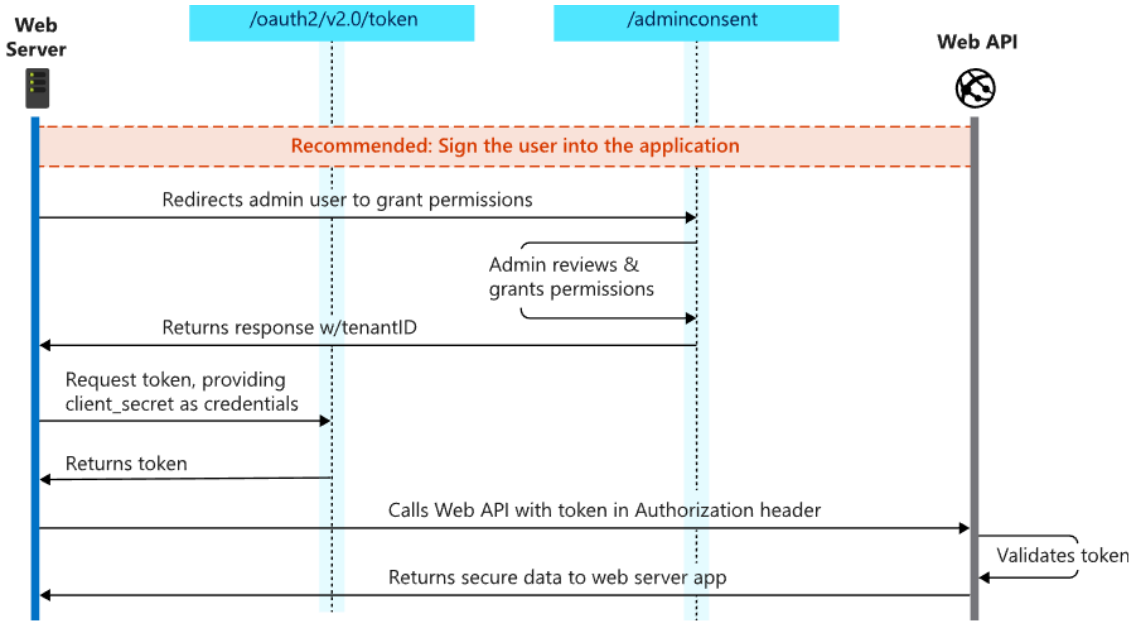
Para nuestra capa de seguridad vamos a implementar dos niveles de autenticación. El accesos a nuestra API estará securizado siguiendo Oauth client crendentials grant type y para el acceso a nuestra interfaz web se usará el mismo protocolo pero accediendo mediante una cuenta de Google.

Tipo de concesión de credenciales de cliente

La concesión de credenciales de cliente de OAuth 2.0 especificada en RFC 6749 *[8]*, a veces denominada OAuth de dos patas, para acceder a los recursos alojados en la web mediante la identidad de una aplicación. Este tipo de concesión se usa comúnmente para interacciones de servidor a servidor que deben ejecutarse en segundo plano, sin interacción inmediata con un usuario. Estos tipos de aplicaciones a menudo se denominan demonios o cuentas de servicio.

El flujo de otorgamiento de credenciales de cliente de OAuth 2.0 permite que un servicio web (cliente confidencial) use sus propias credenciales, en lugar de hacerse pasar por un usuario, para autenticarse al llamar a otro servicio web.

En el flujo de credenciales del cliente, un administrador concede los permisos directamente a la propia aplicación. Cuando la aplicación presenta un token a un recurso, el recurso exige que la propia aplicación tenga autorización para realizar una acción, ya que no hay ningún usuario involucrado en la autenticación.



Una de las ventajas además que nos ofrece Oauth es la posibilidad de asignar roles a nuestros usuarios acreditados. En nuestro caso concreto vamos a tener tres tipos de roles para poder acceder a los recursos de nuestros endpoints:

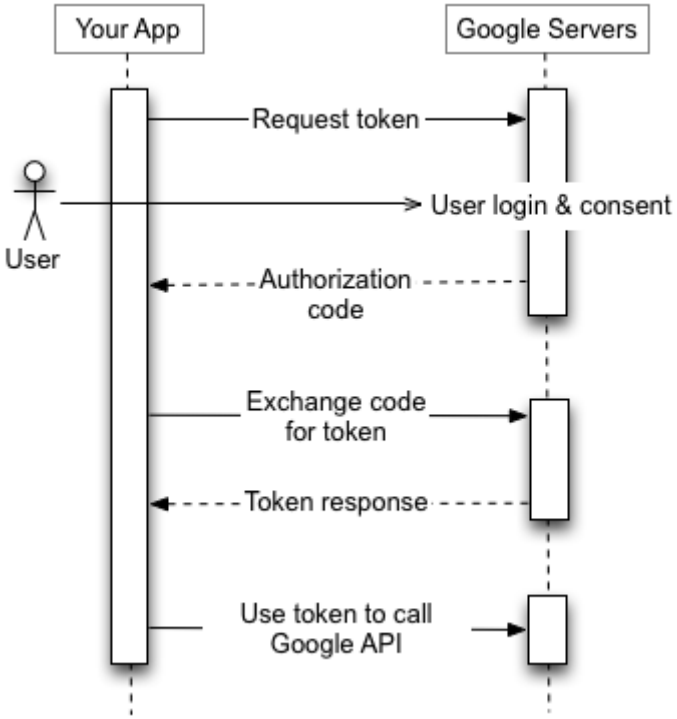
- ADMIN: Podrá acceder a todos los recursos de administración: crear credenciales, dar de alta/baja tiendas, etc.
- SHOPS: Podrá acceder a todos los recursos necesarios para las tiendas: crear, modificar y borrar videojuegos; editar su perfil de tienda.
- USERS: Podrá acceder a los recursos necesarios para los usuarios: listar videojuegos y subscribirse a una categoría.

Acceso con credenciales de Google

Los usuarios tendrán una interfaz web en la que podrán acceder con sus credenciales de Google.

La secuencia de autorización comienza cuando nuestra aplicación redirecciona un navegador a una URL de Google. La URL incluye parámetros de búsqueda que indican el tipo de acceso que se solicita. Google se encarga de la autenticación, la selección de las sesiones y el consentimiento del usuario. El resultado es un código de autorización que la aplicación puede intercambiar por un token de acceso y un token de actualización.

Nuestra aplicación debe almacenar el token de actualización para usarlo en el futuro y usarlo para acceder a una API de Google. Una vez que el token de acceso caduca, la aplicación usa el token de actualización para obtener uno nuevo.



3.3.2 Notificaciones PUSH

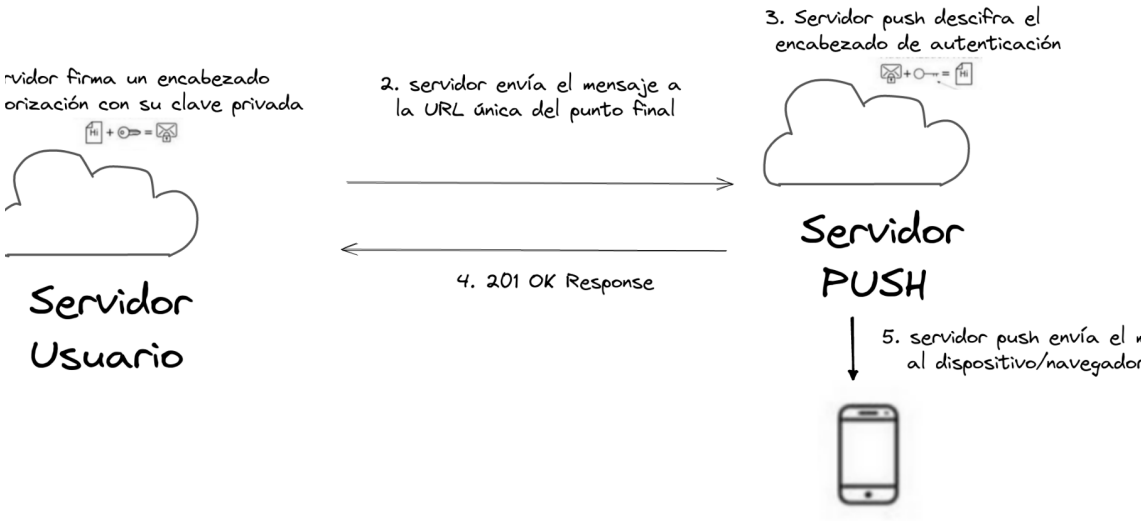
Las notificaciones web push se basan en dos estándares web: API de notificación y API push (que a su vez usa ServiceWorker).

Suscribirse a notificaciones push

- El servidor comparte su clave pública con el navegador.
- El navegador utiliza la clave pública para suscribirse a un servicio push (cada navegador tiene la suya propia)
- El servicio push devuelve una suscripción con una URL de punto final única que se puede usar para enviar mensajes push
- La suscripción se guarda en el servidor.

Envío de notificaciones automáticas

- El servidor firma un encabezado de autorización con su clave privada
- El servidor envía el mensaje a la URL única del punto final
- El servidor push descifra el encabezado de autenticación
- El servidor push envía el mensaje al dispositivo/navegador



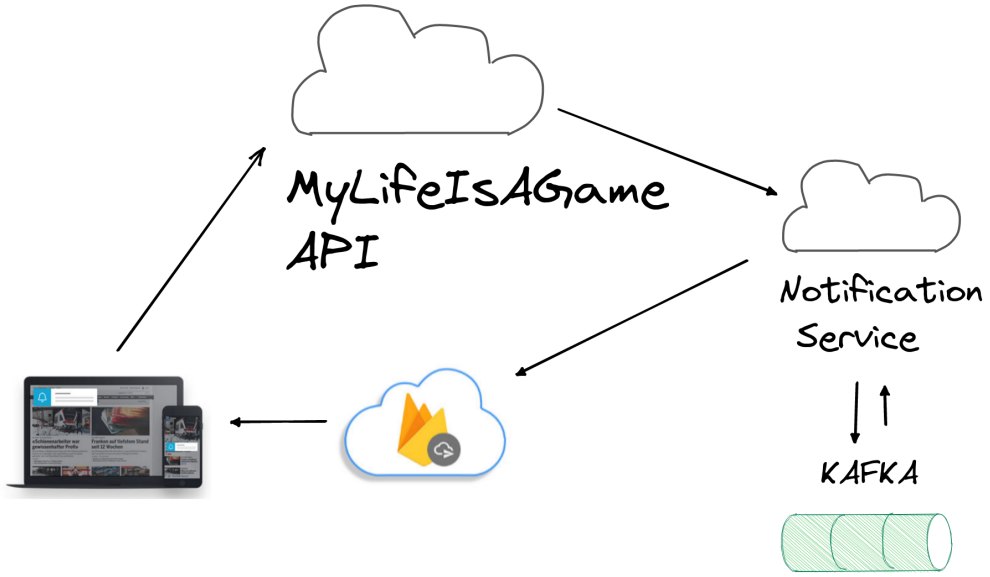
3.3.2.1 Notificaciones Web PUSH con Firebase

Es una tecnología que ofrecen los navegadores modernos, que permite enviar mensajes a los usuarios que visitaron en algún momento nuestra página web, aún si al momento del envío el usuario no está con la página abierta.

Funciona similar a las Push Notifications en las App, pero enviando los mensajes a los navegadores de nuestros clientes. En el caso de PC, el navegador debe estar abierto (no tiene porque estar en nuestra página) y en el caso de Chrome para Android, el mismo no tiene siquiera que estar abierto, brindando una herramienta muy similar a la de las Apps.

Las Web Push Notifications brindan un poderoso canal de comunicación con nuestros clientes, notificándolos en tiempo real sobre distintos asuntos, eventos, promociones, productos, etc., y derivándolos en forma sencilla a nuestra página web y/o WebApp.

Para poder enviar Web Push Notifications es necesario contar con un servidor de notificaciones, que pueda conectarse a Firebase o Google Cloud Messaging Service, quien finalmente entregará el mensaje al navegador del usuario, tal como se presenta en la siguiente figura:



3.3.3 Rate Limit

La limitación de peticiones es importante para prevenir ataques maliciosos en nuestra API.

La limitación de peticiones se aplica a la cantidad de llamadas que un usuario puede realizar a una API dentro de un marco de tiempo establecido. Esto se usa para ayudar a controlar la carga que se pone en el sistema.

La limitación de velocidad ayuda a evitar que un usuario agote los recursos del sistema. Sin limitación de peticiones, es más fácil que una parte malintencionada abrume el sistema. Esto se hace cuando el sistema se inunda con solicitudes de información, lo que consume memoria, almacenamiento y capacidad de red.

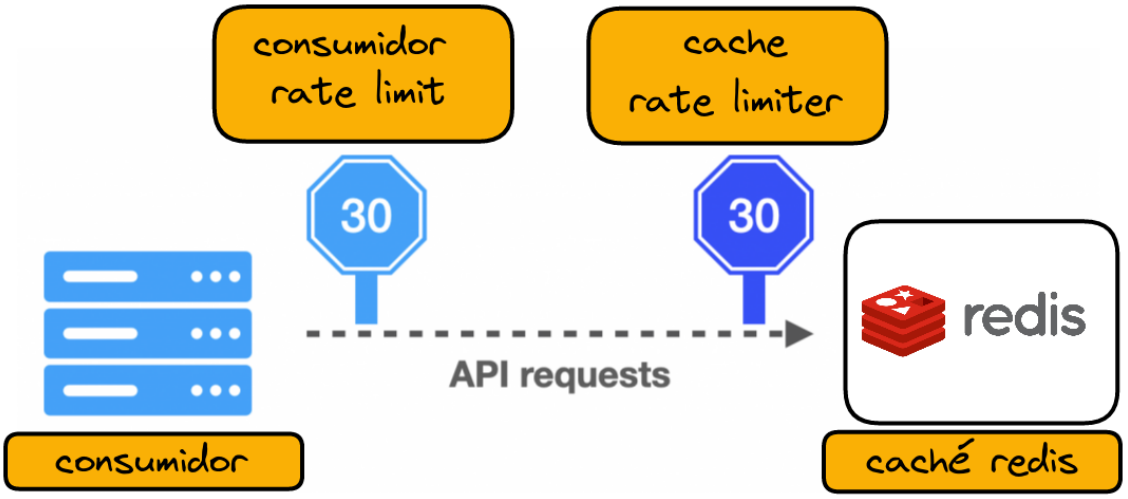
Una API que utiliza la limitación de llamadas puede limitar a los clientes que intentan realizar demasiadas llamadas o bloquearlos temporalmente por completo. Los usuarios que han sido limitados pueden ver sus solicitudes denegadas o ralentizadas durante un tiempo determinado. Esto permitirá que se cumplan las solicitudes legítimas sin ralentizar toda la aplicación.

La API responde con el código de estado HTTP 429 Too Many Requests (Demasiadas solicitudes) cuando una solicitud tiene una velocidad limitada o acelerada.

En nuestro caso concreto, usando Rate Limiter podemos hacer cumplir nuestras políticas de la siguiente manera:

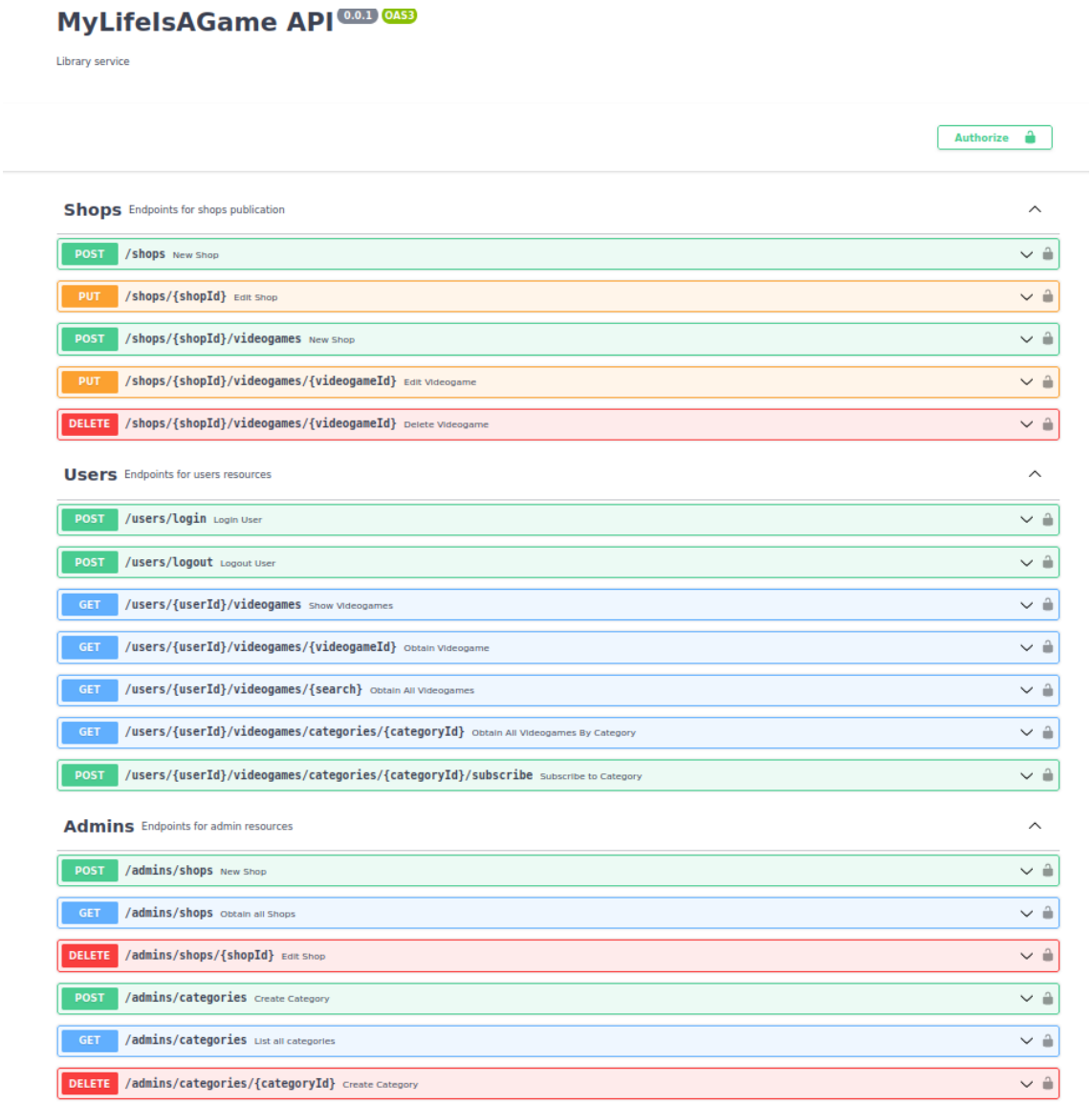
- Limitaremos las peticiones que pueden realizar las tiendas a 2 peticiones por minuto. Esto ayuda a minimizar el spam o la creación de cuentas maliciosas, y la denegación de servicio (DoS) y la denegación de servicio distribuida (DDoS)
- Limitaremos a los usuarios a 10 peticiones por segundo para garantizar una búsqueda fluida de los videojuegos, pero para garantizar la seguridad frente a un ataque

Para limitar el número de peticiones, se ha implementado un sistema de caches usando Redis en las que guardaremos la IP del consumidor de nuestra API y un contador con el número de peticiones y el TTL (time to live, o tiempo de expiración) definido para cada recurso como comentamos anteriormente. De esta forma si por ejemplo tenemos un límite de 2 peticiones por minuto para un recurso (endpoint), nuestra caché no se refrescará hasta que pase un minuto sin recibir peticiones para permitir de nuevo el acceso al recurso.



3.3.4 Interfaces de la capa Controladora de la API Rest

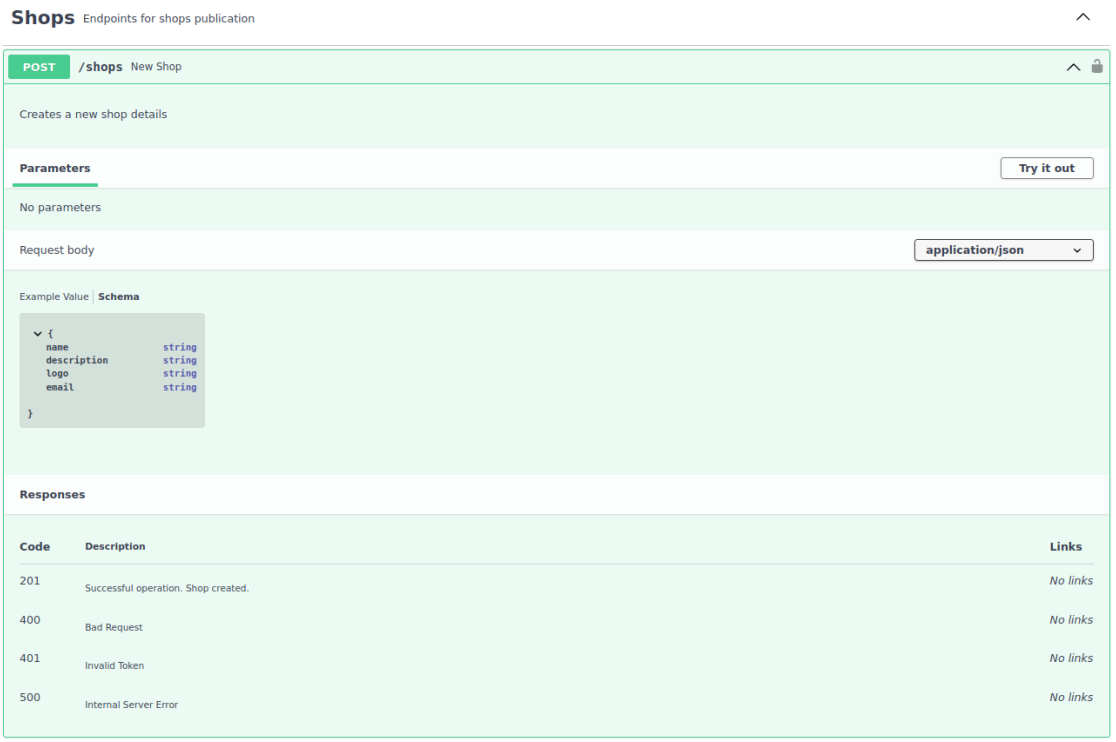
Para la definición de la capa de control de nuestra API Rest hemos seguido el estándar de OpenAPI Specification que nos ayuda a definir nuestras interfaces de una manera agnóstica del lenguaje.

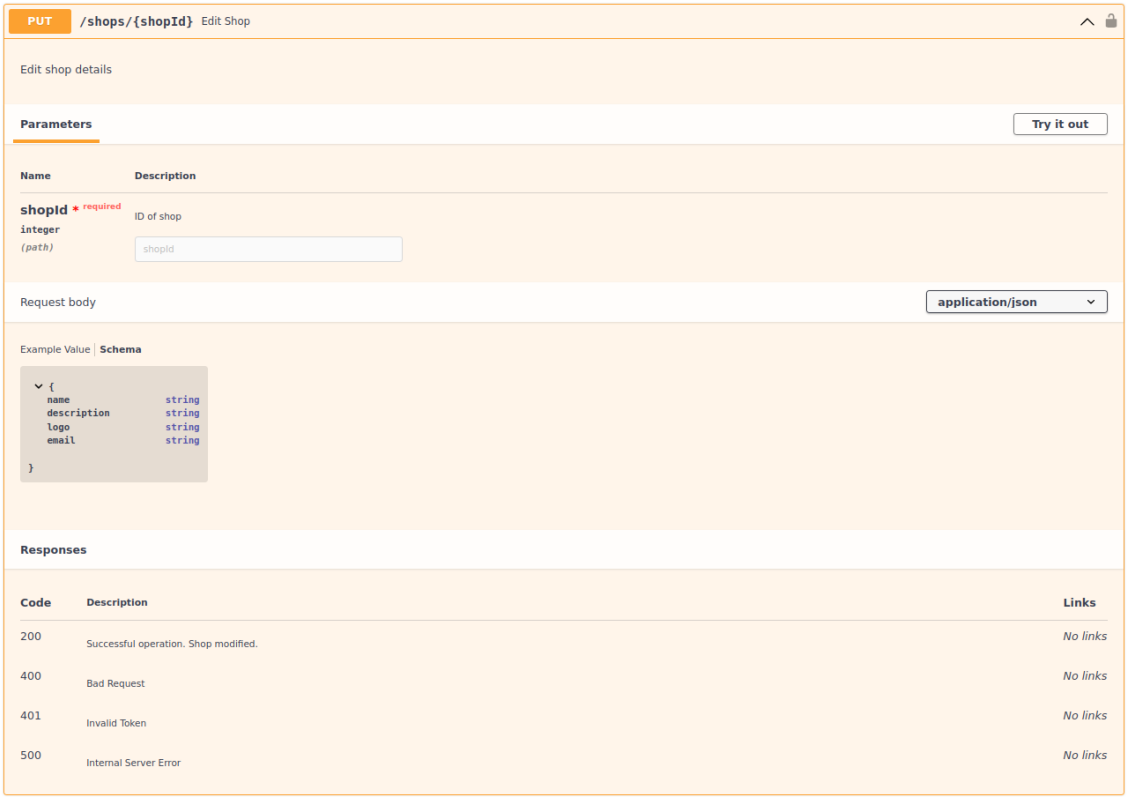


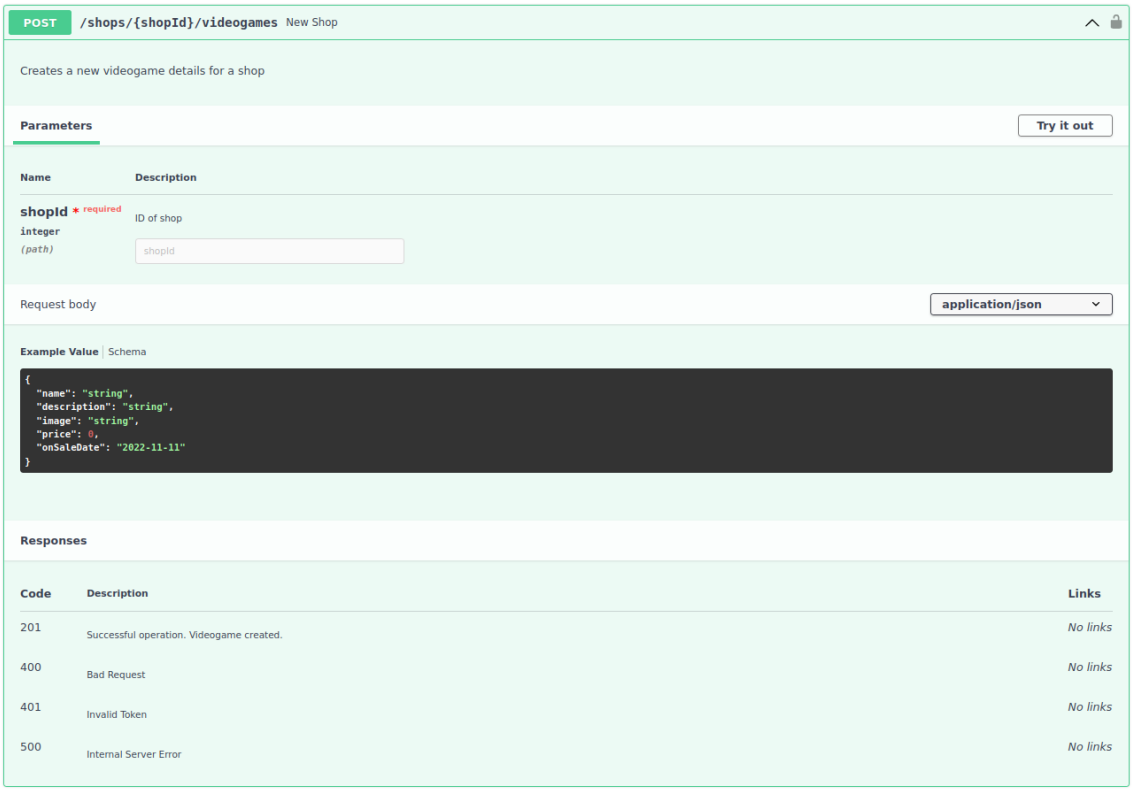
Podemos observar que nuestras interfaces quedan definidas en tres servicios cuyos detalles pasamos a mostrar a continuación

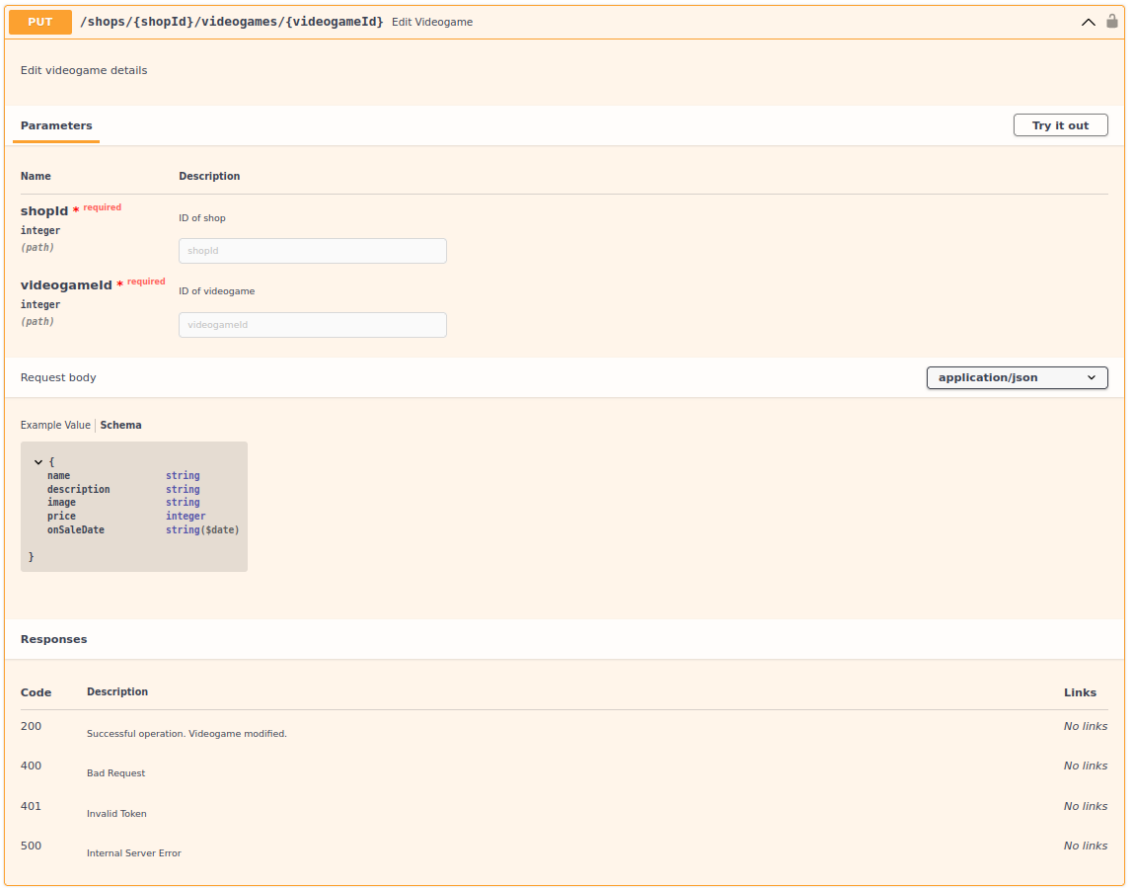
Shops

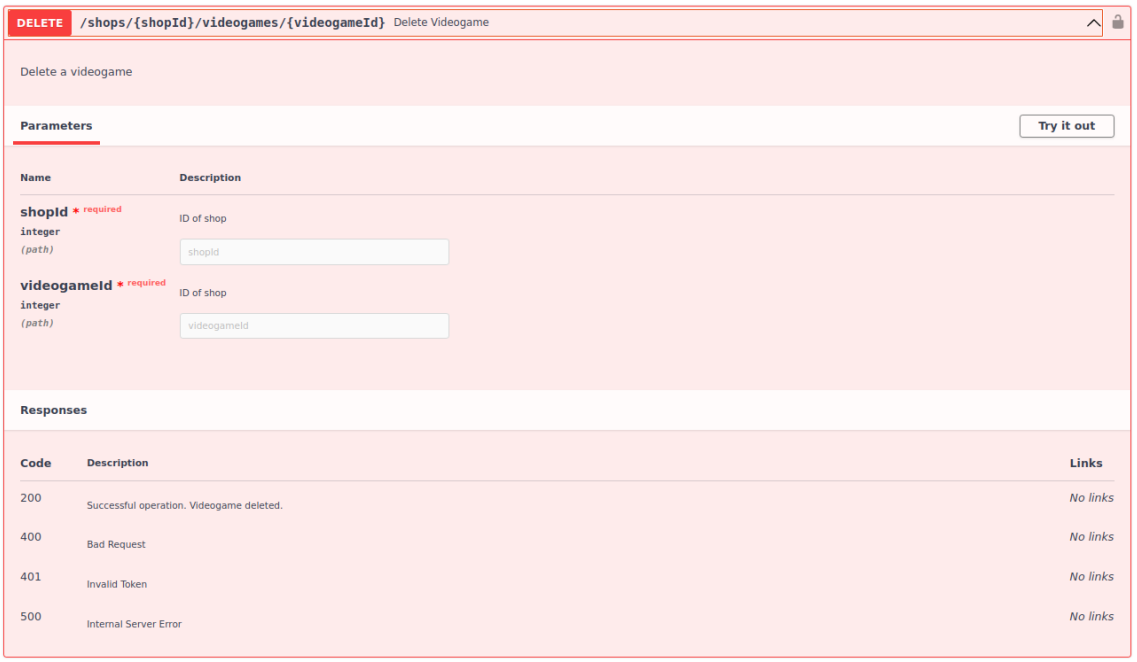
Recursos disponibles para las tiendas





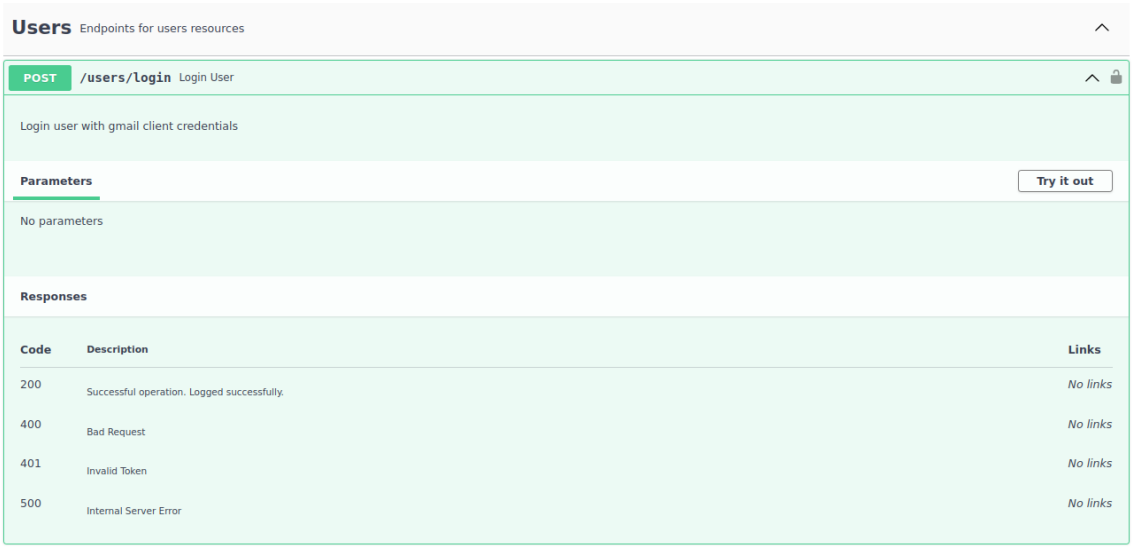

POST
/shops/{shopId}/videogames
New Shop
Creates a new videogame details for a shop
Parameters
Try it out
Name
Description
shopId * required
integer
(path)
ID of shop
shopId
Request body
application/json
Example Value | Schema
{
"name": "string",
"description": "string",
"image": "string",
"price": 0,
"onSaleDate": "2022-11-11"
}
Responses
Code
Description
Links
201
Successful operation. Videogame created.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

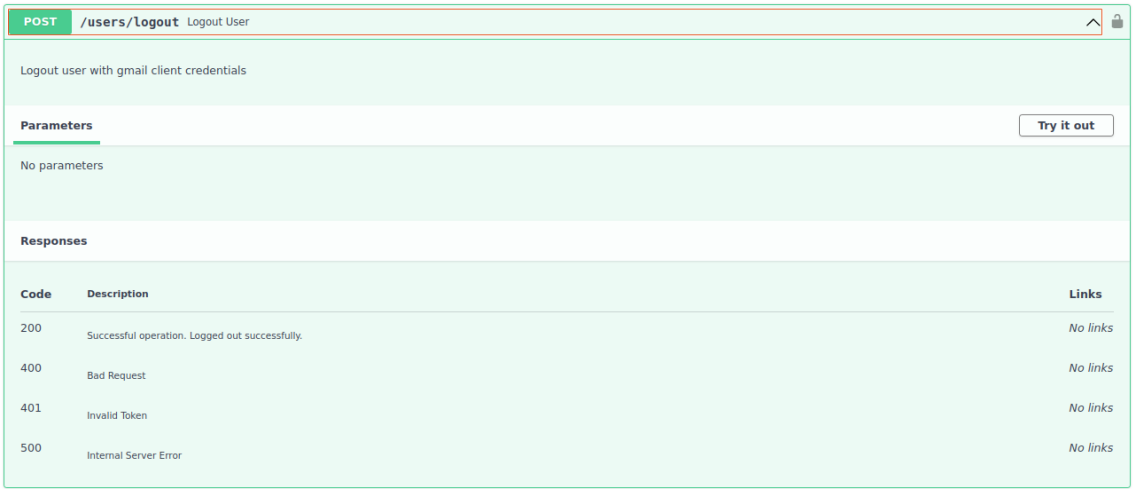

PUT
/shops/{shopId}/videogames/{videogameId}
Edit Videogame
Edit videogame details
Parameters
Try it out
Name
Description
shopId * required
integer
(path)
ID of shop
shopId
videogameId * required
integer
(path)
ID of videogame
videogameId
Request body
application/json
Example Value | Schema
{
name string
description string
image string
price integer
onSaleDate string($date)
}
Responses
Code
Description
Links
200
Successful operation. Videogame modified.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

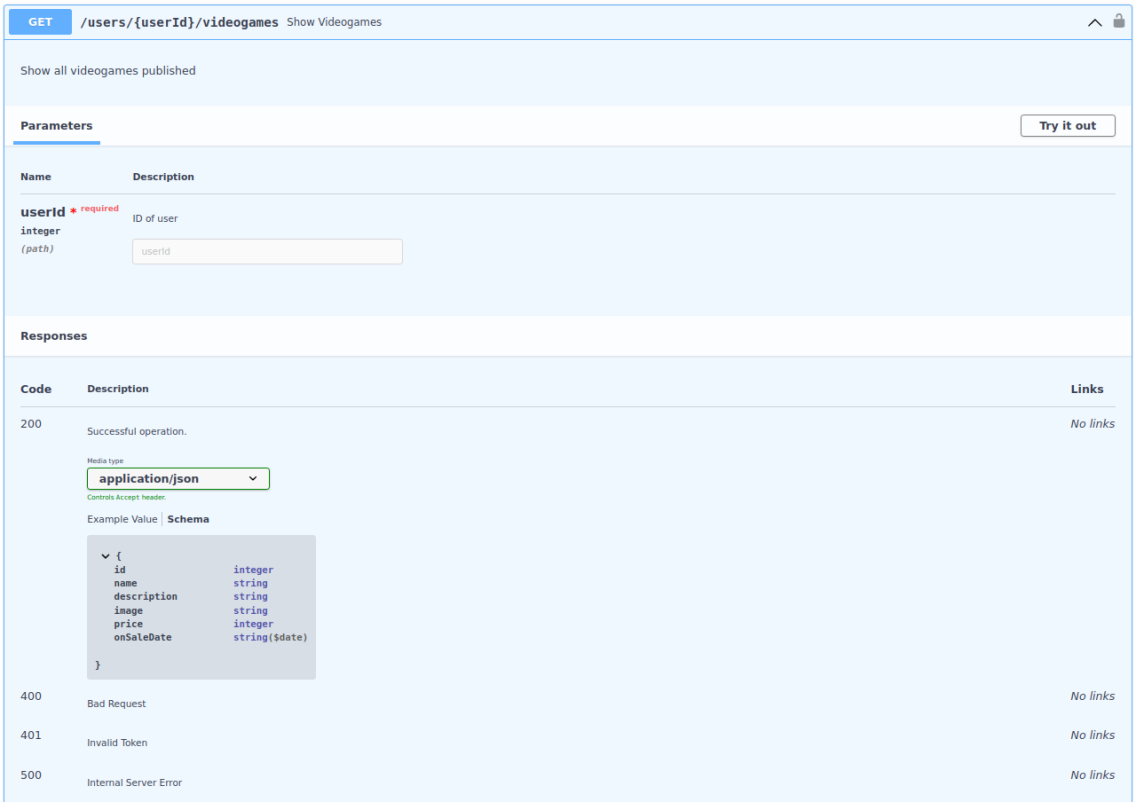


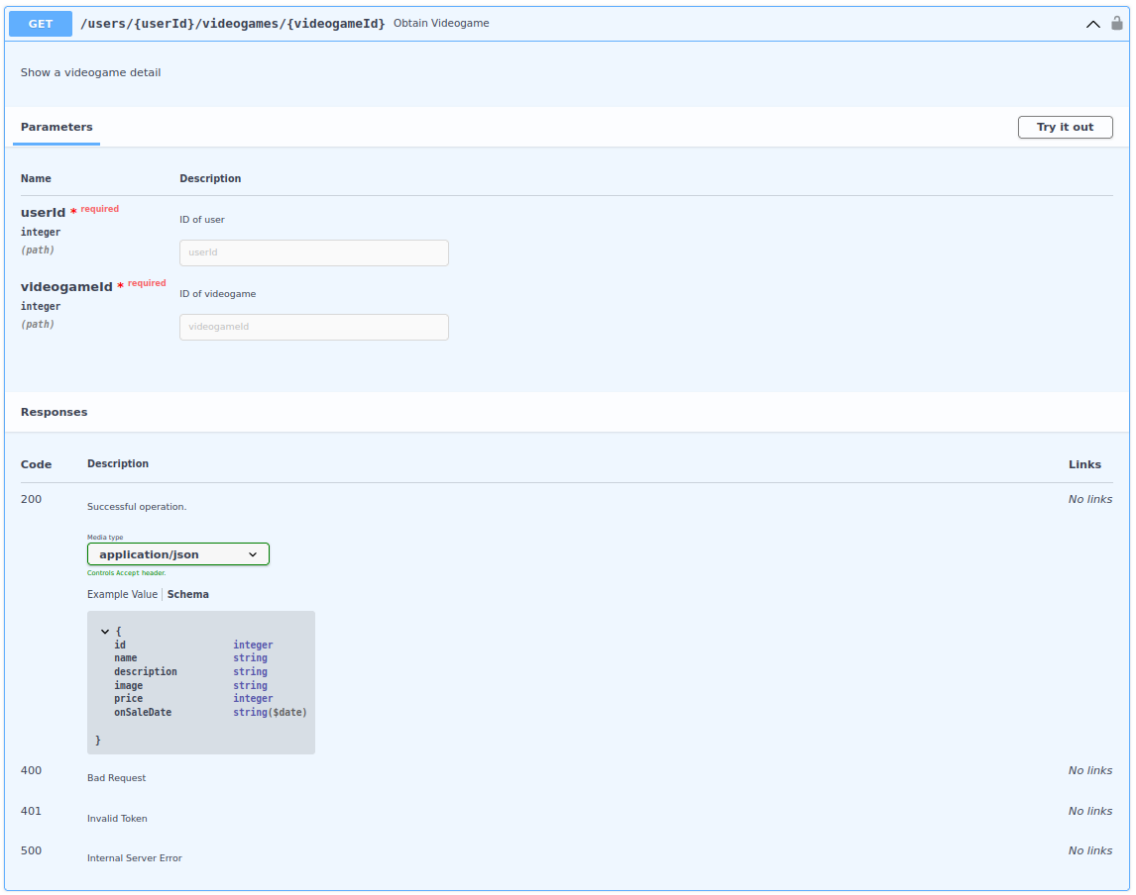
Users

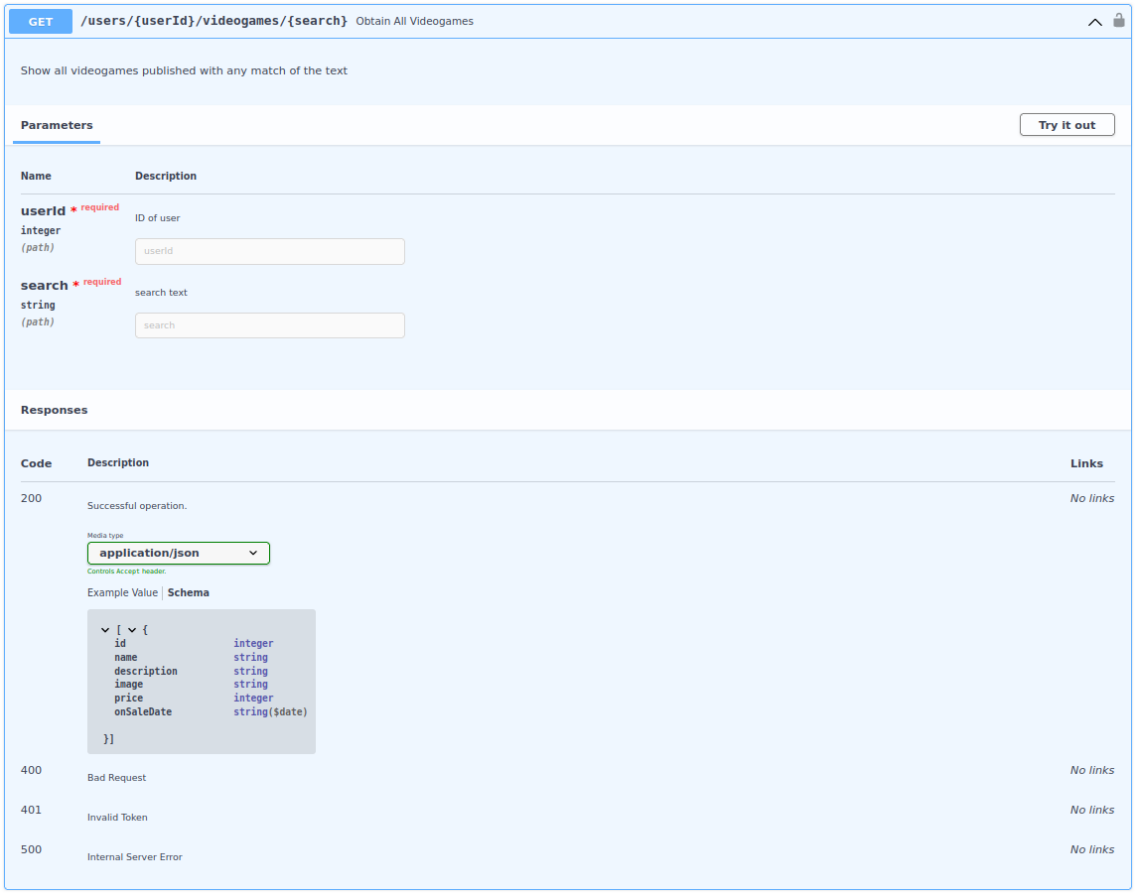
Recursos disponibles para los usuarios

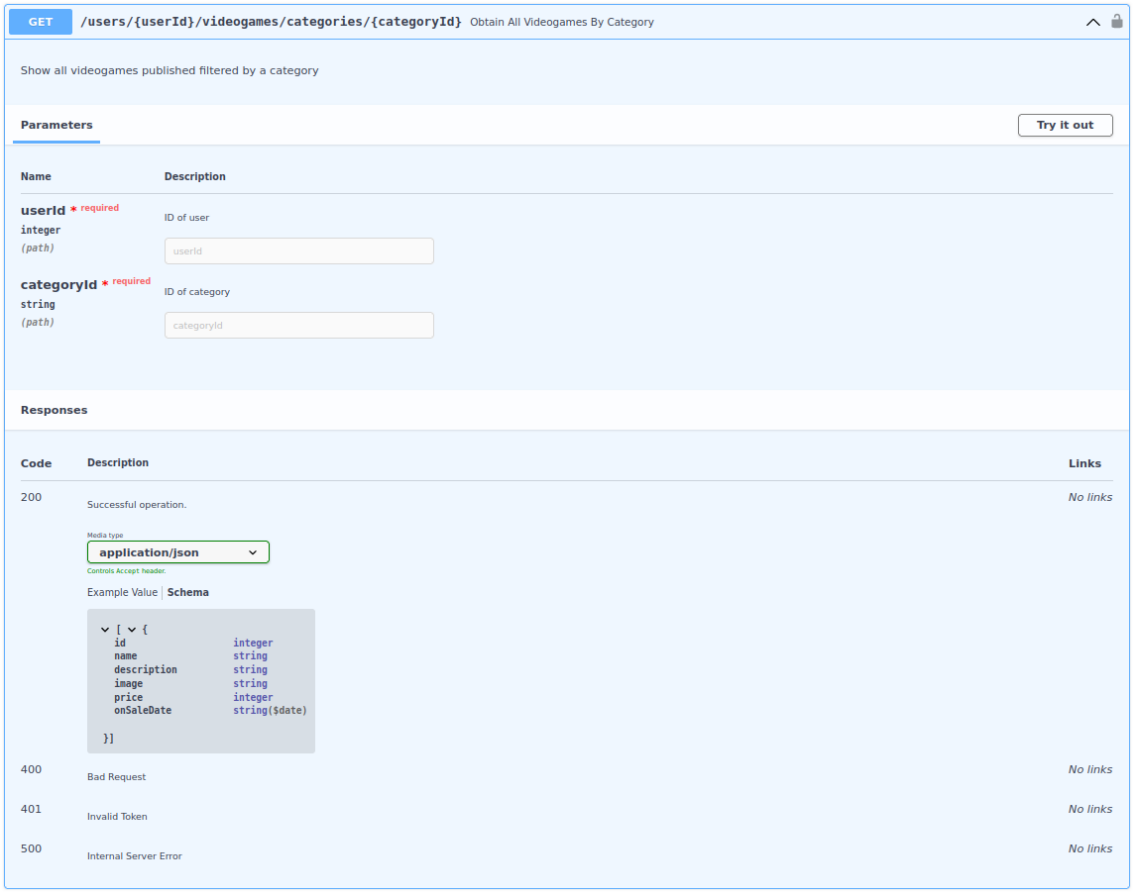


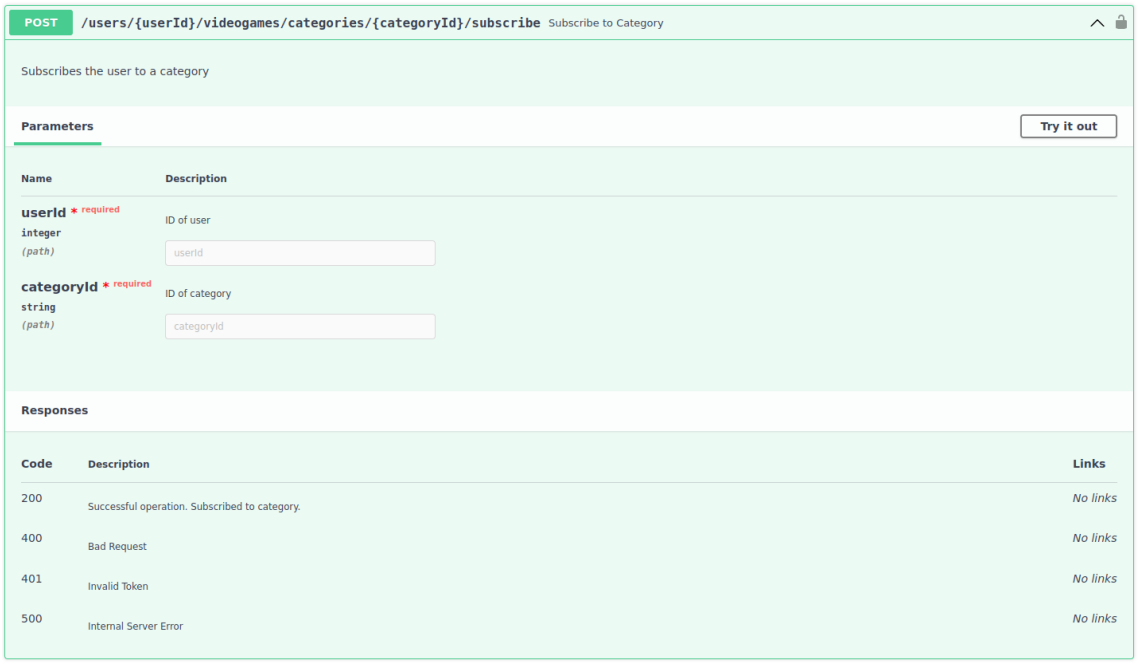

POST
/users/logout
Logout User
Logout user with gmail client credentials
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Successful operation. Logged out successfully.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links


GET
/users/{userId}/videogames
Show Videogames
Show all videogames published
Parameters
Try it out
Name
Description
userId
required
integer
(path)
ID of user
userId
Responses
Code
Description
Links
200
Successful operation.
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
id integer
name string
description string
image string
price integer
onSaleDate string($date)
}
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links


GET
/users/{userId}/videogames/{videogameId}
Obtain Videogame
Show a videogame detail
Parameters
Try it out
Name
Description
userId * required
integer
(path)
ID of user
userId
videogameId * required
integer
(path)
ID of videogame
videogameId
Responses
Code
Description
Links
200
Successful operation.
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
id integer
name string
description string
image string
price integer
onSaleDate string($date)
}
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

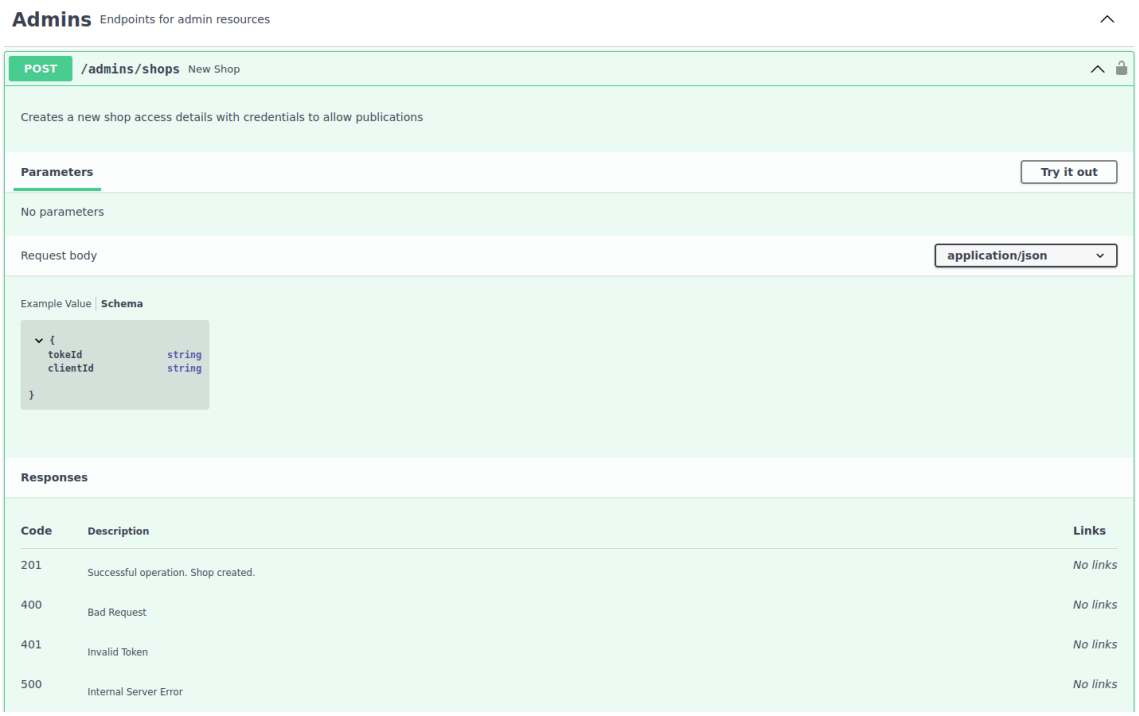

GET
/users/{userId}/videogames/{search}
Obtain All Videogames
Show all videogames published with any match of the text
Parameters
Try it out
Name
Description
userId * required
integer
(path)
ID of user
userId
search * required
string
(path)
search text
search
Responses
Code
Description
Links
200
Successful operation.
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
[{
id integer
name string
description string
image string
price integer
onSaleDate string($date)
}]
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

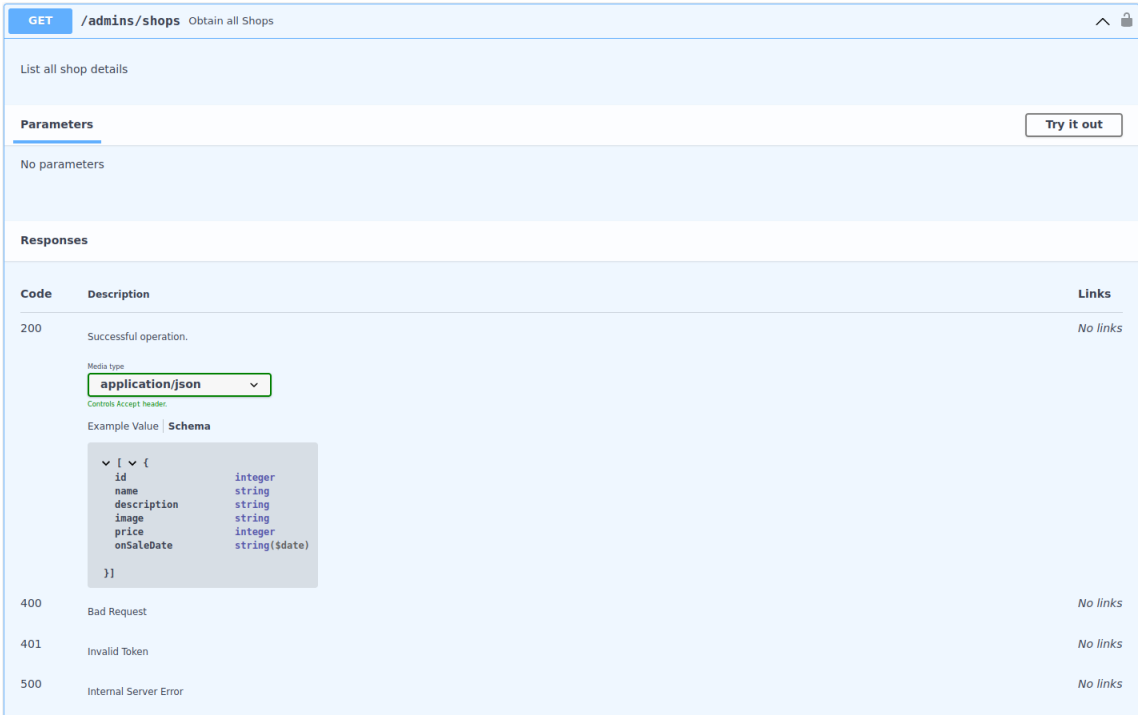

GET
/users/{userId}/videogames/categories/{categoryId}
Obtain All Videogames By Category
Show all videogames published filtered by a category
Parameters
Try it out
Name
Description
userId * required
integer
(path)
ID of user
userId
categoryId * required
string
(path)
ID of category
categoryId
Responses
Code
Description
Links
200
Successful operation.
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
[{ id integer name string description string image string price integer onSaleDate string($date) }]
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

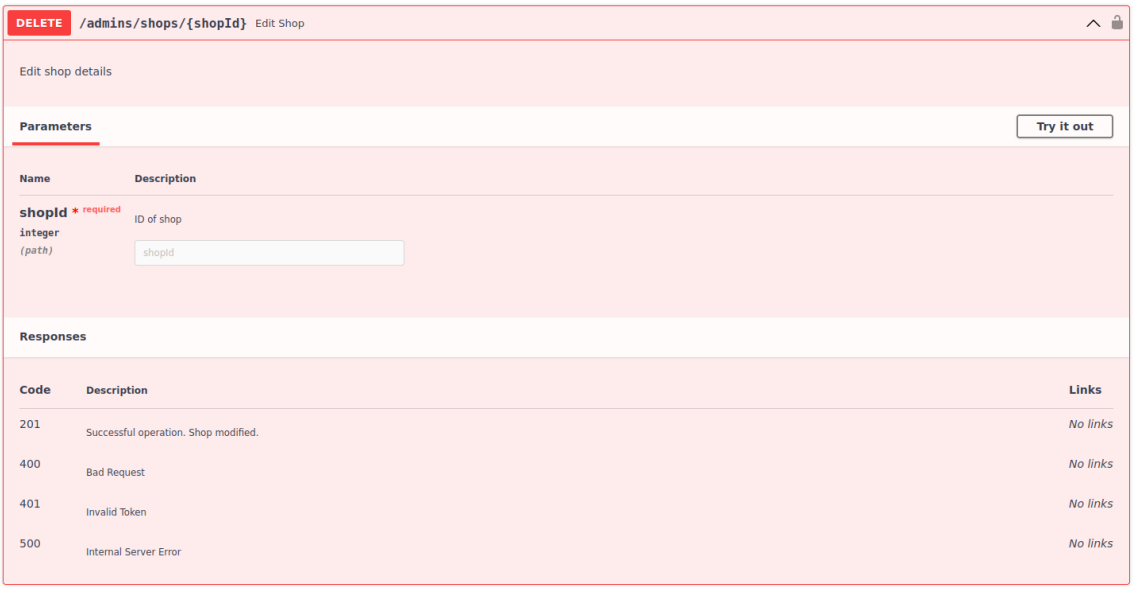

POST
/users/{userId}/videogames/categories/{categoryId}/subscribe
Subscribe to Category
Subscribes the user to a category
Parameters
Try it out
Name
Description
userId * required
integer
(path)
ID of user
userId
categoryId * required
string
(path)
ID of category
categoryId
Responses
Code
Description
Links
200
Successful operation. Subscribed to category.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

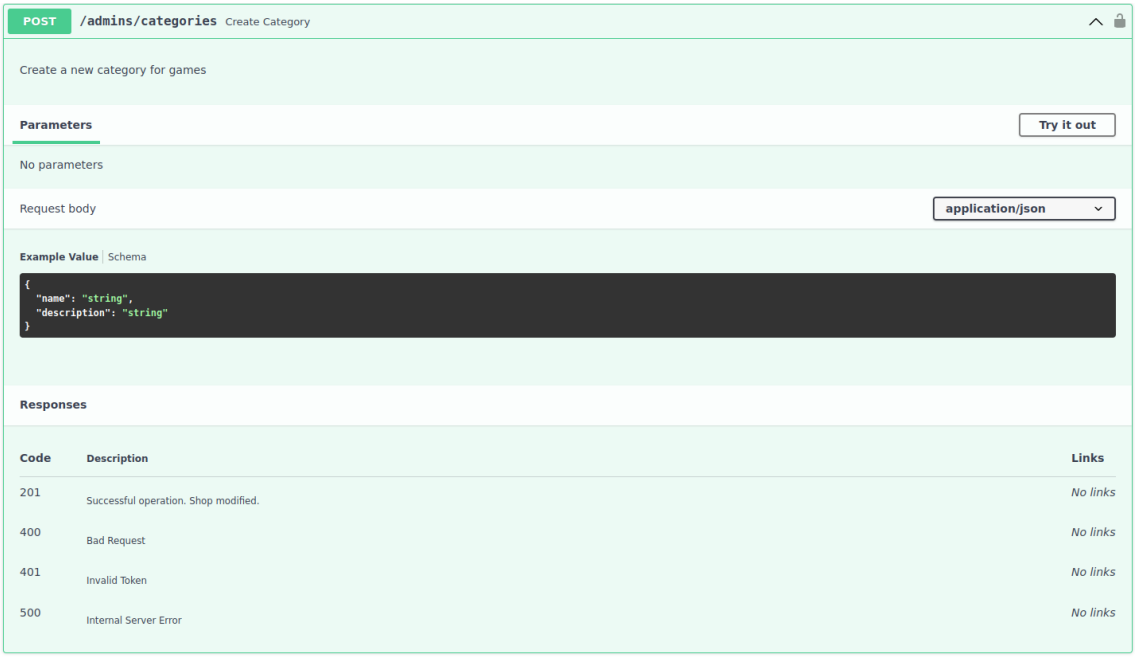
Admin

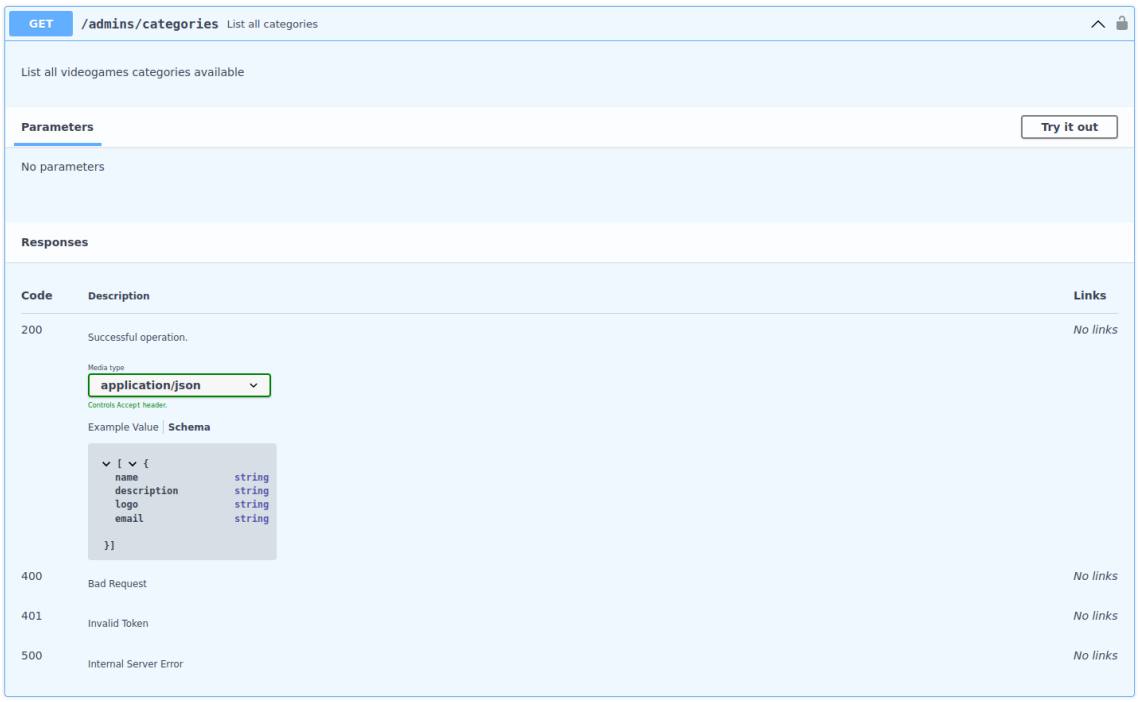
Recursos disponibles para el administrador






DELETE
/admins/shops/{shopId}
Edit Shop
Edit shop details
Parameters
Try it out
Name
Description
shopId
required
integer
(path)
ID of shop
shopId
Responses
Code
Description
Links
201
Successful operation. Shop modified.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links


POST
/admins/categories
Create Category
Create a new category for games
Parameters
Try it out
No parameters
Request body
application/json
Example Value
Schema
{
"name": "string",
"description": "string"
}
Responses
Code
Description
Links
201
Successful operation. Shop modified.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links


GET
/admins/categories
List all categories
List all videogames categories available
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Successful operation.
No links
Media type
application/json
Controls Accept header.
Example Value
Schema
[{
name string
description string
logo string
email string
}]
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

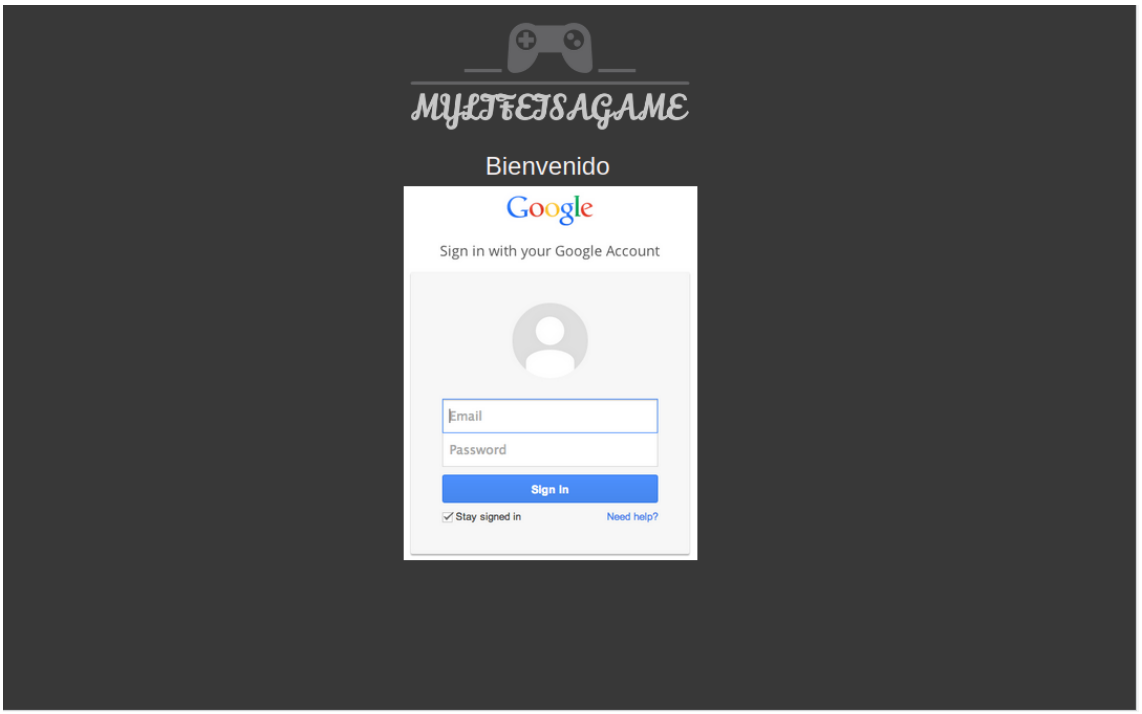

DELETE
/admins/categories/{categoryId}
Create Category
Create a new category for games
Parameters
Try it out
Name
Description
categoryId
required
integer
(path)
ID of shop
categoryId
Responses
Code
Description
Links
200
Successful operation. Category deleted.
No links
400
Bad Request
No links
401
Invalid Token
No links
500
Internal Server Error
No links

3.3.5 Interfaces de Usuario de nuestra aplicación

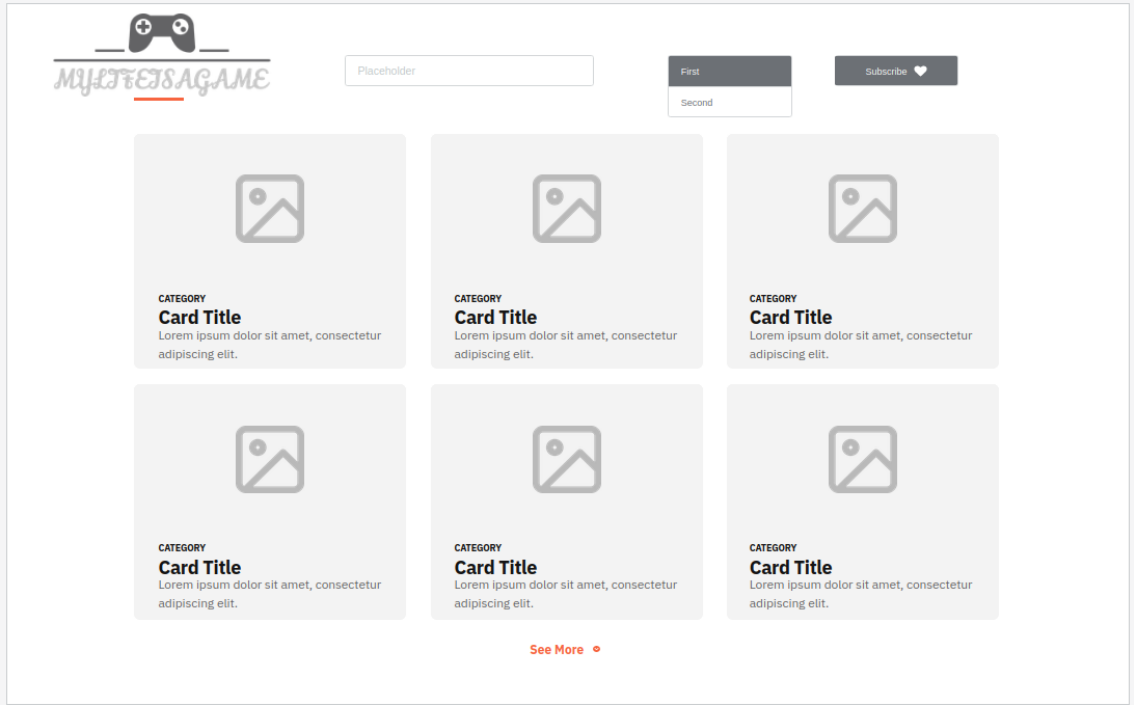
El usuario podrá acceder en la aplicación usando una cuenta de Google. Una vez logado, podrá navegar por la página para visualizar los videojuegos publicados, pudiendo usar un buscador o filtrando por categorías. Además podrá suscribirse a una categoría para que si se publica nuevo contenido sea notificado.

A continuación se detalla un prototipo de los distintos estados de navegación

1. Pantalla de acceso del usuario:



2. Pantalla de interacción del usuario: se mantendrá la navegación del usuario en una sola vista. En ella se listan por defecto los videojuegos más recientes. El usuario puede usar el buscador para buscar videojuegos que coincidan con el texto indicado, filtrar por categorías y en caso de no estar suscrito a esa categoría pulsar el botón de suscribirse.



3. Implementación

A lo largo del siguiente apartado se describirán los puntos claves de nuestro proyecto a nivel de implementación, indicando el código de los puntos más relevantes de este.

3.1 Implementación de los Endpoints

Tendremos tres clases Controllers para nuestros tres actores/consumidores principales(Administrador, Tiendas y Usuarios) , con acceso a los distintos recursos ofrecidos para cada uno de ellos.

Nuestro servicio de API estará implementado en el artefacto: **mylifeisagame-api**

AdminRESTController

```java
@Log4j2
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@PreAuthorize("hasAuthority('ADMIN')")
@RestController
public class AdminRESTController {

    private final AdminService adminService;

    @PostMapping("/admins/shops")
    public ResponseEntity<Boolean> createShopsCredentials(@RequestBody CreateShopCredentialsRequest
createShopCredentialsRequest) {
        log.trace("createShops");
        adminService.createShopsCredentials(createShopCredentialsRequest.getShopCredentials());

        return new ResponseEntity<>(true, HttpStatus.OK);
    }

    @GetMapping("/admins/shops")
    public List<Shop> findShops() {
        log.trace("findShops");

        return adminService.findAllShops();
    }

    @DeleteMapping("/admins/shops/{shopId}")
    public ResponseEntity<Boolean> deleteShops(@PathVariable String shopId) {
        log.trace("deleteShops");

        adminService.deleteShop(shopId);

        return new ResponseEntity<>(true, HttpStatus.OK);
    }

    @GetMapping("/admins/categories")
    @ResponseStatus(HttpStatus.OK)
    public List<Category> findCategories() {
        log.trace("findCategories");

        return adminService.findAllCategories();
    }

    @PostMapping("/admins/categories")
    public ResponseEntity<Long> createCategory(@RequestBody CreateCategoryRequest
createCategoryRequest) {
        log.trace("createCategory");

        log.trace("Creating category " + createCategoryRequest);
        Long categoryId = adminService.createCategory(createCategoryRequest.getCategory());
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
                .path("/{id}")
                .buildAndExpand(categoryId)
                .toUri();

        return ResponseEntity.created(uri).body(categoryId);
    }

    @DeleteMapping("/admins/categories/{categoryId}")
    public ResponseEntity<Boolean> deleteCategory(@PathVariable Long categoryId) {
        log.trace("deleteCategory");

        adminService.deleteCategory(categoryId);

        return new ResponseEntity<>(true, HttpStatus.OK);
    }
}
```

ShopsRESTController

```java
@Log4j2
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@PreAuthorize("hasAuthority('SHOP')")
@RestController
public class ShopsRESTController {

    private final ShopService shopService;
    private final VideogameAdapter videogameAdapter;


    @PostMapping("/shops")
    public ResponseEntity<Boolean> createShop(Principal principal, @RequestBody CreateShopRequest
createShopRequest) {
        log.trace("createGame: " + principal.getName());

        createShopRequest.getShop().setId(principal.getName());
        shopService.createShop(createShopRequest.getShop());

        return ResponseEntity.ok(true);
    }

    @PutMapping("/shops")
    public ResponseEntity<Boolean> updateShop(Principal principal, @RequestBody CreateShopRequest
createShopRequest) {
        log.trace("updateShop");

        createShopRequest.getShop().setId(principal.getName());
        shopService.updateShop(createShopRequest.getShop());

        return ResponseEntity.ok(true);
    }



    @GetMapping(value = "/shops/games")
    public ResponseEntity<List<VideoGame>> findGames(Principal principal) {
        log.trace("findGames");

        return ResponseEntity.ok().body(shopService.findAllGames(principal.getName()));
    }

    @GetMapping("/shops/games/{gameId}")
    public ResponseEntity<VideoGame> getGameDetails(@PathVariable Long gameId) {
        log.trace("getGameDetails");

        return shopService.findGameById(gameId).map(game -> ResponseEntity.ok().body(game))
                .orElse(ResponseEntity.notFound().build());
    }


    @PostMapping("/shops/games")
    public ResponseEntity<Long> createGame(Principal principal, @RequestBody VideoGameRequest
videoGameRequest) {
        log.trace("createGame");

        VideoGame videoGame = videogameAdapter.map(videoGameRequest,principal.getName());
        Long id = shopService.createGame(videoGame);

        URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
                .path("/{id}")
                .buildAndExpand(id)
                .toUri();

        return ResponseEntity.created(uri).body(id);
    }

    @PutMapping("/shops/games/{gameId}")
    public ResponseEntity<Long> updateGame(Principal principal, @PathVariable Long gameId, @RequestBody
VideoGameRequest videoGameRequest) {
        log.trace("updateGame");

        VideoGame videoGame = videogameAdapter.map(videoGameRequest,principal.getName());
        shopService.updateGame(gameId, videoGame);


        return ResponseEntity.ok(gameId);
    }

    @DeleteMapping("/shops/games/{gameId}")
    public ResponseEntity<Boolean> cancelGame(@PathVariable Long gameId) {
        log.trace("cancelGame");

        shopService.cancelGame(gameId);

        return ResponseEntity.ok(true);
    }

    @GetMapping("/shops/games/categories")
    @ResponseStatus(HttpStatus.OK)
    public List<Category> findCategories() {
        log.trace("findCategories");

        return shopService.findAllCategories();
    }
}
```

UsersRESTController

```java
@Log4j2
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@CrossOrigin(maxAge = 3600)
@RestController
public class UsersRESTController {

    private final UserService userService;
    private final UserCategoryRequestAdapter userCategoryRequestAdapter;

    @GetMapping("/users/games")
    public ResponseEntity<List<VideoGame>> getGames() {
        log.trace("getGames");
        List<VideoGame> videoGames = userService.findAllGames();
        if (videoGames.isEmpty()) return ResponseEntity.notFound().build();
        return ResponseEntity.ok().body(videoGames);
    }

    @GetMapping("/users/games/{gameId}")
    public ResponseEntity<VideoGame> getGameDetails(@PathVariable Long gameId) {
        log.trace("getGameDetails");

        return userService.findGameById(gameId).map(game -> ResponseEntity.ok().body(game))
                .orElse(ResponseEntity.notFound().build());
    }

    @GetMapping("/users/games/search/{name}")
    public ResponseEntity<List<VideoGame>> getGamesBySearch(@PathVariable String name) {
        log.trace("getGamesBySearch");
        List<VideoGame> videoGames = userService.findGamesByName(name);
        if (videoGames.isEmpty()) return ResponseEntity.notFound().build();
        return ResponseEntity.ok().body(videoGames);
    }

    @GetMapping("/users/games/categories/{categoryId}")
    public ResponseEntity<List<VideoGame>> getGamesByCategory(@PathVariable Long categoryId) {
        log.trace("getGamesByCategory");
        List<VideoGame> videoGames = userService.findGamesByCategory(categoryId);
        if (videoGames.isEmpty()) return ResponseEntity.notFound().build();
        return ResponseEntity.ok().body(videoGames);
    }

    @GetMapping("/users/games/categories")
    @ResponseStatus(HttpStatus.OK)
    public List<Category> findCategories() {
        log.trace("findCategories");

        return userService.findAllCategories();
    }

    @PostMapping("/users/games/categories/subscribe")
    @ResponseStatus(HttpStatus.OK)
    public void subscribeCategory(@RequestBody UserCategoryRequest request) {
        log.trace("subscribeCategory");
        userService.subscribeCategory(userCategoryRequestAdapter.map(request));
    }

    @GetMapping("/users/{userId}/games/categories/subscribe")
    public UserCategory obtainCategoriesSubscribed(@PathVariable String userId) {
        log.trace("obtainCategoriesSubscribed");

        return userService.findAllCategoriesSubscribed(userId);
    }

    @GetMapping("/users/games/categories/{categoryId}/subscribe")
    public List<UserCategory> findAllUsersSubscribedToACategory(@PathVariable Long categoryId) {
        log.trace("findAllUsersSubscribedToACategory");

        return userService.findAllUsersSubscribedToACategory(categoryId);
    }

    @PostMapping("/users/games/categories/subscribe/push")
    @ResponseStatus(HttpStatus.OK)
    public void subscribeUserPush(@RequestBody UserPush request) {
        log.trace("subscribeUserPush");
        userService.subscribeUserPush(request);
    }

    @GetMapping("/users/{userId}/games/categories/subscribe/push")
    @ResponseStatus(HttpStatus.OK)
    public UserPush obtainUserPush(@PathVariable String userId) {
        log.trace("obtainUserPush");
        return userService.obtainPush(userId);
    }


}
```

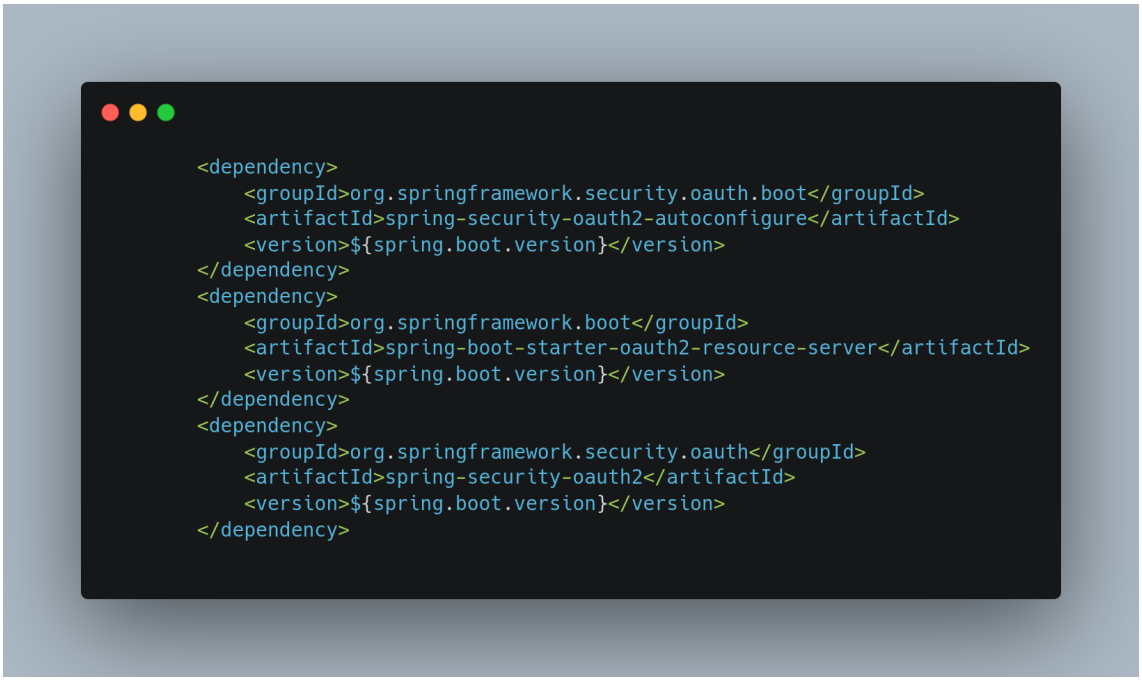
3.2 Capa de Seguridad

Como comentamos en la parte de diseño, tendremos dos capas de seguridad, una a nivel de API y otra a nivel de interfaz de usuario. Ambas capas estarán basadas en client-credentials, la única distinción será que para las tiendas y administración usarán credenciales para poder usar los endpoints de nuestra API, mientras que para los usuarios se habilitará la posibilidad de acceder usando las credenciales de una cuenta de Google.

3.2.1 Client Credentials para nuestra API

Para la capa de seguridad de nuestra api tenemos un artefacto o servicio independiente que se encargará de ello: **authserver**

Spring-boot nos facilita todas las clases necesarias para añadir nuestra capa de seguridad con Oauth 2.0 importando las siguientes dependencias

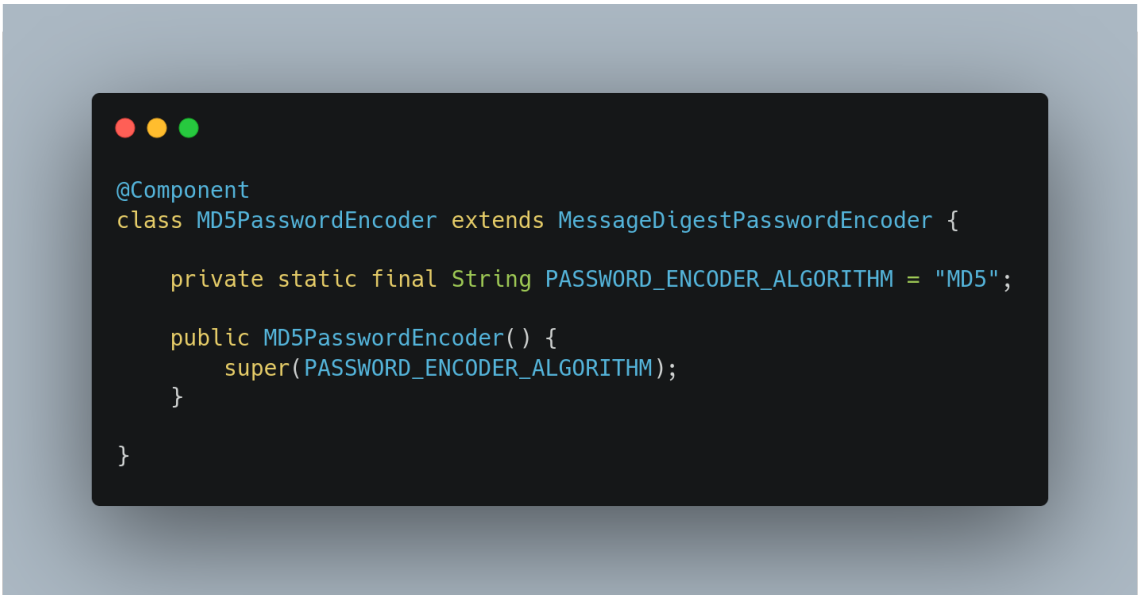


```
<dependency>
    <groupId>org.springframework.security.oauth.boot</groupId>
    <artifactId>spring-security-oauth2-autoconfigure</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Para crear una clase de seguridad personalizada, necesitamos usar @EnableWebSecurity y extender la clase con @WebSecurityConfigurerAdapter para que podamos redefinir algunos de los métodos proporcionados

```java
@Configuration
@EnableWebSecurity
class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService applicationUserDetailsService;

    @Autowired
    private PasswordEncoder md5PasswordEncoder;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .formLogin()
                .loginPage("/login").permitAll()
                .failureUrl("/login?bad_credentials")
            .and()
                .requestMatchers()
                    .antMatchers("/login",
                                 "/logout",
                                 "/oauth/authorize",
                                 "/oauth/confirm_access")
            .and()
                .authorizeRequests()
                    .anyRequest().authenticated();
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/resources/**");
    }

    @Bean
    public AuthenticationProvider applicationDaoAuthenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(applicationUserDetailsService);
        authenticationProvider.setPasswordEncoder(md5PasswordEncoder);
        return authenticationProvider;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(applicationDaoAuthenticationProvider());
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

}
```

Spring Security nos fuerza a hashear las contraseñas para que no se guarden en texto plano. Para ello vamos a usar la clase MD5PasswordEncoder para guardar todas nuestras credenciales en MD5*[9]*



```
@Component
class MD5PasswordEncoder extends MessageDigestPasswordEncoder {

    private static final String PASSWORD_ENCODER_ALGORITHM = "MD5";

    public MD5PasswordEncoder() {
        super(PASSWORD_ENCODER_ALGORITHM);
    }

}
```

Para crear nuestro token de acceso y refrescar tendremos la clase *CustomTokenService* que extiende de DefaultTokenServices, que nos garantizará por defecto que el token de acceso seguirá vigente durante 12 horas (para un caso real es mejor bajar la tasa de refresco a cada 5 minutos por ejemplo para garantizar mayor seguridad, pero para nuestro caso de estudio por facilitar las cosas hemos decidido usar el de defecto)

```java
@Transactional
public class CustomTokenService extends DefaultTokenServices {

    private static final Logger LOGGER = getLogger(CustomTokenService.class);

    /**
     * This method fixes when exists duplicate rows were being generated when multiple threads coming
into JdbcTokenStore.storeAccessToken at the same time.
     */
    @Override
    public synchronized OAuth2AccessToken createAccessToken(OAuth2Authentication authentication) throws
AuthenticationException {
        OAuth2AccessToken existingAccessToken = null;
        try {
            existingAccessToken = super.getAccessToken(authentication);

            if (existingAccessToken == null || existingAccessToken.isExpired()){
                return super.createAccessToken(authentication);
            }
        } catch (DuplicateKeyException dke) {
            LOGGER.info(format("Duplicate key found:  %s", dke));
        } catch (Exception ex) {
            LOGGER.info(format("Exception while creating access token %s", ex));
        }

        return existingAccessToken;
    }

    @Override
    public synchronized OAuth2AccessToken refreshAccessToken(String refreshTokenValue, TokenRequest
tokenRequest) throws AuthenticationException {
        synchronized (refreshTokenValue.intern()) {
            return super.refreshAccessToken(refreshTokenValue, tokenRequest);
        }
    }
}
```

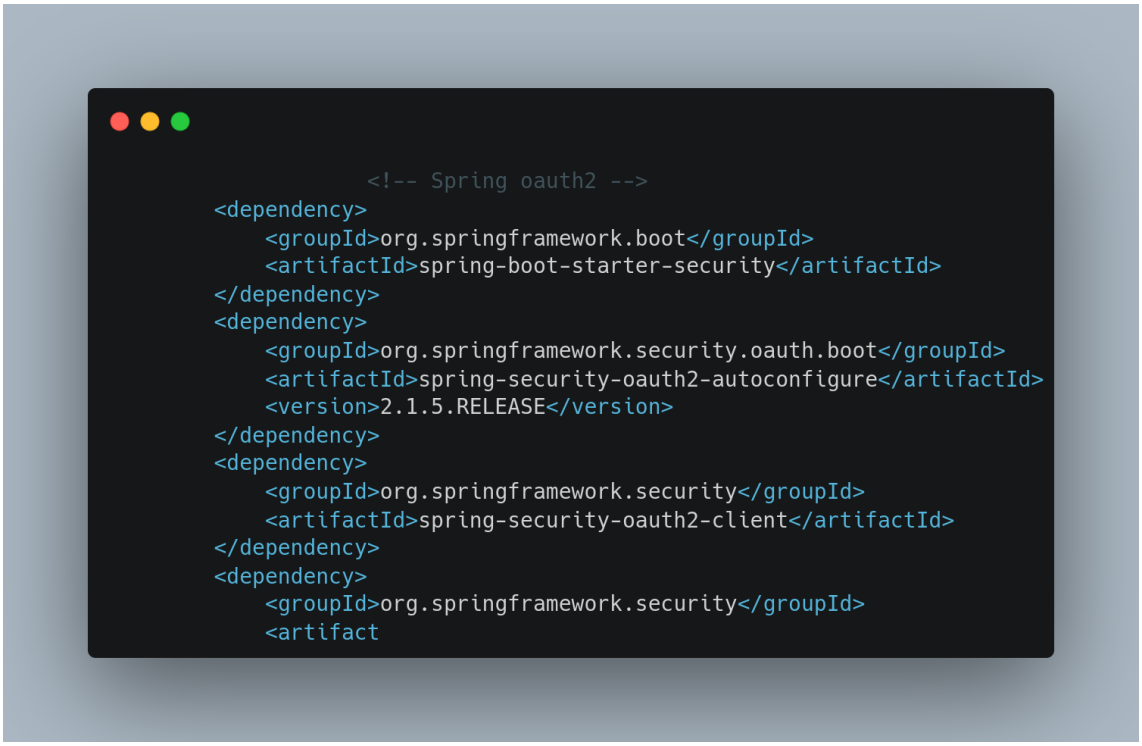
Por último, Oauth nos obliga crearnos una clase JPA con unos campos por defecto donde guardará todos los datos necesarios para su correcto funcionamiento, entre ellos el clientId, el token de refresco y el scope (rol) del perfil .

```java
public interface OAuthClientsRepository extends JpaRepository<OAuthClient, String> {

}
```

```java
package com.ilozano.oauth.clients.persistence;

import javax.persistence.Entity;
import javax.persistence.Id;

import org.hibernate.annotations.Immutable;

@Immutable
@Entity(name="oauth_client_details")
public class OAuthClient {

    @Id
    public String clientId;

    public String resourceIds;

    public String clientSecret;

    public String scope;

    public String authorizedGrantTypes;

    public String webserverRedirectUri;

    public String authorities;

    public Integer accessTokenValidity;

    public Integer refreshTokenValidity;

    public String additionalInformation;

    public String autoapprove;

}
```

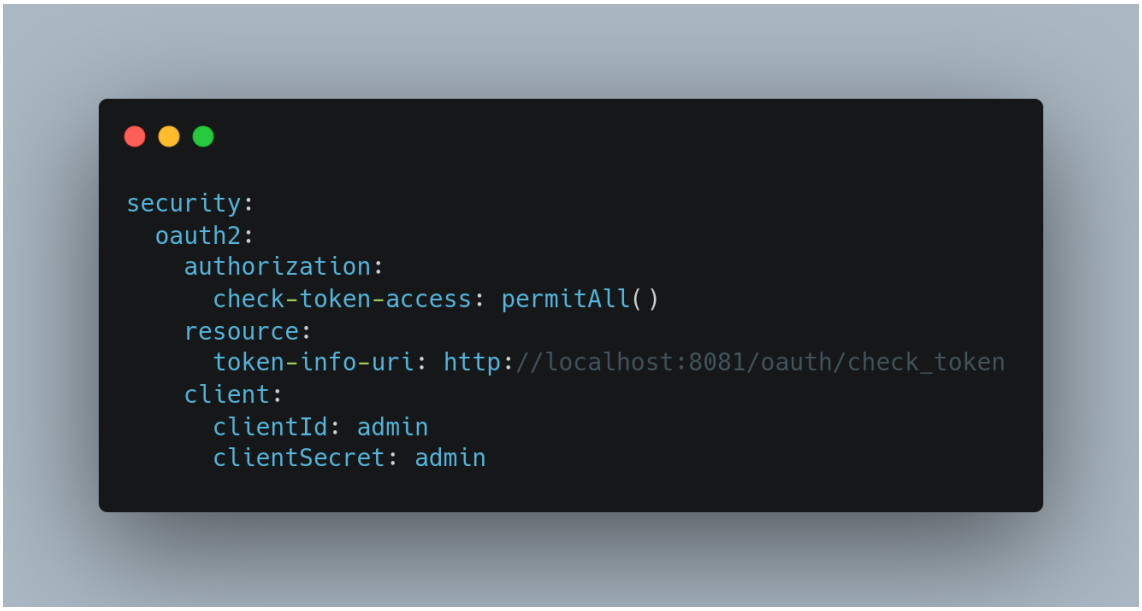
Uso de *auth-server* dentro de nuestra API (*mylifeisagame-api*):

Spring-boot nos facilita también todas las clases necesarias para añadir nuestra capa de seguridad con Oauth 2.0 en el cliente.



```
<!-- Spring oauth2 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security.oauth.boot</groupId>
    <artifactId>spring-security-oauth2-autoconfigure</artifactId>
    <version>2.1.5.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifact
```

Además deberemos añadir la siguiente configuración en nuestro fichero properties o yml de configuración para añadir la url del servidor de autenticación, pudiendo además indicar un usuario por defecto para acceder



```
security:
  oauth2:
    authorization:
      check-token-access: permitAll()
    resource:
      token-info-uri: http://localhost:8081/oauth/check_token
    client:
      clientId: admin
      clientSecret: admin
```

Las clases más relevantes son las siguientes:

MyLifeIsAGameApplication

En nuestra clase main de spring-boot usaremos las anotaciones “@EnableResourceServer” para habilitar nuestra capa de seguridad con oauth y “@EnableGlobalMethodSecurity(prePostEnabled = true)” para facilitarnos la gestiona de accesos a determinados recursos en función de los roles de acceso.

```java
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true)
@SpringBootApplication
public class MyLifeIsAGameApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyLifeIsAGameApplication.class, args);
    }

}
```

Si observamos en el apartado donde describimos los endpoints de nuestra API Rest, observaremos que se usa la anotación “@PreAuthorize("hasAuthority(XXX)")” donde “XXX” es el rol creado en Oauth para permitir acceder a recursos concretos para ese ROL

```
@Log4j2
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@PreAuthorize("hasAuthority('ADMIN')")
@RestController
public class AdminRESTController {

    private final AdminService adminService;
```

3.2.2 Google Credentials en la interfaz de usuario

Para securizar la parte de nuestra interfaz de usuario tendremos un artefacto ad hoc para ello: **authserver-login**

Para permitir usar las credenciales de google en nuestro proyecto deberemos crear un proyecto nuevo en la Consola de Desarrolladores de Google*[10]* y meter las siguientes configuraciones generadas en nuestro fichero de configuración asociadas a nuestro proyecto

```
security:
  oauth2:
    client:
      registration:
        google:
          clientId: 972684121149-60pfqvcavqqdsslcapj522kittghtptp.apps.googleusercontent.com
          clientSecret: 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
          redirectUri: "{baseUrl}/oauth2/callback/{registrationId}"
          scope:
            - email
            - profile
```

Necesitaremos además de las dependencias usadas en los anteriores proyectos de spring-security las siguientes dependencias para gestionar los tokens de Google

```xml
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.2</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.2</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.2</version>
    <scope>runtime</scope>
</dependency>
```

Para poder usar las credenciales obtenidas de google usaremos la siguiente clase de configuración

```java
@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private final Auth auth = new Auth();
    private final OAuth2 oauth2 = new OAuth2();

    public static class Auth {
        private String tokenSecret;
        private long tokenExpirationMsec;

        public String getTokenSecret() {
            return tokenSecret;
        }

        public void setTokenSecret(String tokenSecret) {
            this.tokenSecret = tokenSecret;
        }

        public long getTokenExpirationMsec() {
            return tokenExpirationMsec;
        }

        public void setTokenExpirationMsec(long tokenExpirationMsec) {
            this.tokenExpirationMsec = tokenExpirationMsec;
        }
    }

    public static final class OAuth2 {
        private List<String> authorizedRedirectUris = new ArrayList<>();

        public List<String> getAuthorizedRedirectUris() {
            return authorizedRedirectUris;
        }

        public OAuth2 authorizedRedirectUris(List<String> authorizedRedirectUris) {
            this.authorizedRedirectUris = authorizedRedirectUris;
            return this;
        }
    }

    public Auth getAuth() {
        return auth;
    }

    public OAuth2 getOauth2() {
        return oauth2;
    }
}
```

Almacenaremos el estado y la uri de redirección de Google en una cookie de corta duración. La siguiente clase proporciona funcionalidad para almacenar la solicitud de autorización en cookies y recuperarla.

```java
@Component
public class HttpCookieOAuth2AuthorizationRequestRepository implements
AuthorizationRequestRepository<OAuth2AuthorizationRequest> {
    public static final String OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME = "oauth2_auth_request";
    public static final String REDIRECT_URI_PARAM_COOKIE_NAME = "redirect_uri";
    private static final int cookieExpireSeconds = 180;

    @Override
    public OAuth2AuthorizationRequest loadAuthorizationRequest(HttpServletRequest request) {
        return CookieUtils.getCookie(request, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME)
                .map(cookie -> CookieUtils.deserialize(cookie, OAuth2AuthorizationRequest.class))
                .orElse(null);
    }

    @Override
    public void saveAuthorizationRequest(OAuth2AuthorizationRequest authorizationRequest,
HttpServletRequest request, HttpServletResponse response) {
        if (authorizationRequest == null) {
            CookieUtils.deleteCookie(request, response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME);
            CookieUtils.deleteCookie(request, response, REDIRECT_URI_PARAM_COOKIE_NAME);
            return;
        }

        CookieUtils.addCookie(response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME,
CookieUtils.serialize(authorizationRequest), cookieExpireSeconds);
        String redirectUriAfterLogin = request.getParameter(REDIRECT_URI_PARAM_COOKIE_NAME);
        if (StringUtils.isNotBlank(redirectUriAfterLogin)) {
            CookieUtils.addCookie(response, REDIRECT_URI_PARAM_COOKIE_NAME, redirectUriAfterLogin,
cookieExpireSeconds);
        }
    }

    @Override
    public OAuth2AuthorizationRequest removeAuthorizationRequest(HttpServletRequest request) {
        return this.loadAuthorizationRequest(request);
    }

    public void removeAuthorizationRequestCookies(HttpServletRequest request, HttpServletResponse
response) {
        CookieUtils.deleteCookie(request, response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME);
        CookieUtils.deleteCookie(request, response, REDIRECT_URI_PARAM_COOKIE_NAME);
    }
}
```

3.3 Rate Limit

Para implementar nuestro control de peticiones que puede recibir nuestra API vamos a usar Redis como caché para implementar un algoritmo de control. Para ello spring-boot nos facilita también un conector de redis importando las siguientes dependencias

```
            <!--REDIS -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.session</groupId>
            <artifactId>spring-session-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <type>jar</type>
        </dependency>
```

Para nuestro sistema de caché vamos a guardar para nuestras peticiones la ip y un contador de llamadas para cada recurso. Para ello tendremos una clase interceptadora que se llamará antes de llegar a nuestros Controllers para verificar si puede realizarse la petición o ha superado el limite, en cuyo caso devolverá un error http 409 indicando que se han realizado demasiadas llamadas.

```java
@Slf4j
@Component
@RequiredArgsConstructor
public class RateLimitInterceptor extends HandlerInterceptorAdapter {

    private final RateLimitRepository rateLimitRepository;
    private final RateLimitConfiguration rateLimitConfiguration;
    private final CustomResponseCreator customResponseCreator;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
{
        try {
            boolean hasReachedRateLimit = tooManyRequestInShortTime(request);
            if (!hasReachedRateLimit) return true;
            customResponseCreator.createResponse(response, HttpStatus.TOO_MANY_REQUESTS, "Ratelimit:
Too Many Requests");
            return false;
        } catch (Exception e) {
            log.error("Error during rate limit calculation: ", e);
            return true;
        }
    }

    private boolean tooManyRequestInShortTime(HttpServletRequest request) {
        String requestIp = RequestUtils.getUserIp(request);
        log.info("ip: " + requestIp + ", url: " + request.getRequestURI());
        String requestDomain = RequestUtils.obtainRequestDomain(request);
        if ("redoc".equals(requestDomain)) return false;
        long calls = rateLimitRepository.incrementAndCount(requestIp, requestDomain);
        return calls > rateLimitConfiguration.getMaxRequestPerTime(requestDomain);
    }
}
```

```java
@Slf4j
@Component
public class RateLimitRepository {

    private final RedisTemplate<String, Object> template;
    private RateLimitConfiguration rateLimitConfiguration;

    @Autowired
    public RateLimitRepository(RedisTemplate<String, Object> template, RateLimitConfiguration
rateLimitConfiguration) {
        this.template = template;
        this.rateLimitConfiguration = rateLimitConfiguration;
    }

    public long incrementAndCount(String ip, String requestDomain) {
        ValueOperations<String, Object> valueOps = template.opsForValue();

        long requestCount = valueOps.increment((ip + "_" + requestDomain), 1);
        if (requestCount == 1 || moreCallsThanAllowed(requestCount, requestDomain)) {
            setExpirationTimeForKey(ip, requestDomain);
        }

        return requestCount;
    }

    private void setExpirationTimeForKey(String ip, String requestDomain) {
        try {
            RedisTTL ttl = rateLimitConfiguration.getTTL(requestDomain);
            boolean success = template.expire((ip + "_" + requestDomain), ttl.getValue(),
ttl.getTimeUnit());
            if (!success) template.delete(ip);
        } catch (Exception e) {
            log.error("Error setting expiration for key " + ip + "_" + requestDomain + ": ", e);
            throw new RuntimeException(e);
        }
    }

    private boolean moreCallsThanAllowed(long callsNumber, String requestDomain) {
        return callsNumber > rateLimitConfiguration.getMaxRequestPerTime(requestDomain);
    }
}
```

El tiempo definido y el número de peticiones se indicará en nuestro fichero de configuración

```yaml
ratelimit:
  categories:
    requests: 5
    unit: SECONDS
  shops:
    requests: 2
    unit: MINUTES
  games:
    requests: 10
    unit: SECONDS
```

```java
@Configuration
public class RateLimitConfig {

    @Value("${ratelimit.categories.requests}")
    private Integer categoriesRequests;
    @Value("${ratelimit.categories.unit}")
    private TimeUnit categoriesUnit;

    @Value("${ratelimit.shops.requests}")
    private Integer shopsRequests;
    @Value("${ratelimit.shops.unit}")
    private TimeUnit shopsUnit;

    @Value("${ratelimit.games.requests}")
    private Integer gamesRequests;
    @Value("${ratelimit.games.unit}")
    private TimeUnit gamesUnit;

    @Bean
    public RateLimitConfiguration rateLimitConfiguration() {

        RedisTTL categoriesTTL = new RedisTTL(categoriesRequests, categoriesUnit);
        RedisTTL shopsTTL = new RedisTTL(shopsRequests, shopsUnit);
        RedisTTL gamesTTL = new RedisTTL(gamesRequests, gamesUnit);

        return new RateLimitConfiguration(
                categoriesTTL, shopsTTL, gamesTTL);
    }

}
```

3.4 Push Notifications

Para la publicación de notificaciones PUSH tendremos un artefacto que se encargará de consumir nuestros eventos y notificarlos: notification

Para consumir nuestros eventos vamos a usar Apache Kafka como plataforma distribuida para la transmisión de datos que permite no solo publicar, almacenar y procesar flujos de eventos de forma inmediata, sino también suscribirse a ellos.

Spring-boot nos permite la comunicación con Kafka usando las siguientes dependencias

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

Para consumir los eventos tendremos la siguiente clase consumidora

```java
@EnableKafka
@Configuration
class KafkaConsumerConfig {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    @Bean
    public ConsumerFactory<String, VideoGame> showConsumerFactory() {

        JsonDeserializer<VideoGame> deserializer = new JsonDeserializer<>(VideoGame.class);
        deserializer.setRemoveTypeHeaders(false);
        deserializer.addTrustedPackages("*");
        deserializer.setUseTypeMapperForKey(true);

        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                bootstrapServers);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                deserializer);

        return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(), deserializer);
    }

    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, VideoGame>>
kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, VideoGame> factory =
                new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(showConsumerFactory());
        return factory;
    }
}
```

Que recibirá todos los videojuegos nuevos publicados en desde nuestra API

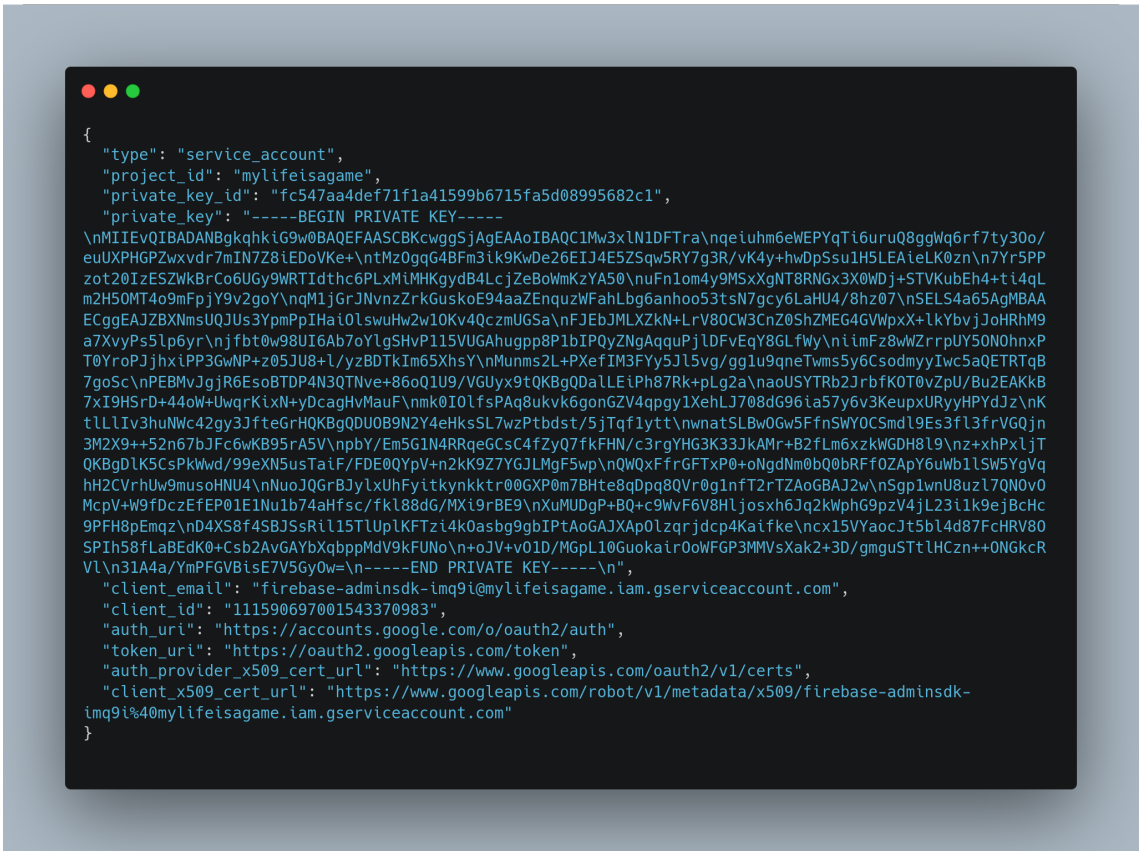
```java
 @Override
    public Long createGame(VideoGame show) {

        Category category = categoryRepository.findCategoryById(show.getCategory().getId()).get();

        show.setCategory(category);

        Long id = gameRepository.createGame(show);

        log.trace("Sending " + show + " to " + KafkaConstants.SHOW_TOPIC + KafkaConstants.SEPARATOR +
KafkaConstants.COMMAND_ADD + " topic.");
        kafkaTemplate.send(KafkaConstants.SHOW_TOPIC + KafkaConstants.SEPARATOR +
KafkaConstants.COMMAND_ADD, gameRepository.findGameById(id).get());

        return id;
    }
```

Y una clase Listener que se encargará de emitir todos los eventos nuevos

```java
@Log4j2
@Component
public class KafkaClassListener {

    private final NotificationService notificationService;

    @Autowired
    public KafkaClassListener(NotificationService notificationService) {
        this.notificationService = notificationService;
    }

    @KafkaListener(topics = KafkaConstants.SHOW_TOPIC + KafkaConstants.SEPARATOR +
KafkaConstants.COMMAND_ADD, groupId = "group-1")
    void showAdded(VideoGame videoGame) {
        log.trace("gameAdded");

        notificationService.notifyGameCreation(videoGame);
    }
}
```

Para la emisión de notificaciones PUSH usaremos Firebase para lo cual tendremos que configurar un conector que registrando nuestra aplicación y registrarlo en las configuraciones de nuestro artefacto

```java
@Configuration
public class NotificationConfiguration {

    private static final String FILE_CREDENTIALS = "firebase/serviceAccountKey.json";

    @Bean
    public FirebaseApp fireBaseConfiguration() throws IOException {
        URL res = getClass().getClassLoader().getResource(FILE_CREDENTIALS);
        FileInputStream serviceAccount = new FileInputStream(res.getPath());


        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                .build();

        FirebaseApp firebaseApp = FirebaseApp.initializeApp(options);

        return firebaseApp;
    }
}
```



```
{
  "type": "service_account",
  "project_id": "mylifeisagame",
  "private_key_id": "fc547aa4def71f1a41599b6715fa5d08995682c1",
  "private_key": "-----BEGIN PRIVATE KEY-----
\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQC1Mw3xlN1DFTra\nqeiuhm6eWEPYqTi6uruQ8ggWq6rf7ty3Oo/
euUXPHGPZwxvdr7mIN7Z8iEDoVKe+\ntMzOgqG4BFm3ik9KwDe26EIJ4E5ZSqw5RY7g3R/vK4y+hwDpSsu1H5LEAieLK0zn\n7Yr5PP
zot20IzESZWkBrCo6UGy9WRTIdthc6PLxMiMHKgydB4LcjZeBoWmKzYA50\nuFn1om4y9MSxXgNT8RNGx3X0WDj+STVKubEh4+ti4qL
m2H5OMT4o9mFpjY9v2goY\nqM1jGrJNvnzZrkGuskoE94aaZEnquzWFahLbg6anhoo53tsN7gcy6LaHU4/8hz07\nSELS4a65AgMBAA
ECggEAJZBXNmsUQJUs3YpmPpIHaiOlswuHw2w1OKv4QczmUGSa\nFJEbJMLXZkN+LrV8OCW3CnZ0ShZMEG4GVWpxX+lkYbvjJoHRhM9
a7XvyPs5lp6yr\njfbt0w98UI6Ab7oYlgSHvP115VUGAhugpp8P1bIPQyZNgAqquPjlDFvEqY8GLfWy\niimFz8wWZrrpUY5ONOhnxP
T0YroPJjhxiPP3GwNP+z05JU8+l/yzBDTkIm65XhsY\nMunms2L+PXefIM3FYy5Jl5vg/gg1u9qneTwms5y6CsodmyyIwc5aQETRTqB
7goSc\nPEBMvJgjR6EsoBTDP4N3QTNve+86oQ1U9/VGUyx9tQKBgQDalLEiPh87Rk+pLg2a\naoUSYTRb2JrbfKOT0vZpU/Bu2EAKkB
7xI9HSrD+44oW+UwqrKixN+yDcagHvMauF\nmk0IOlfsPAq8ukvk6gonGZV4qpgy1XehLJ708dG96ia57y6v3KeupxURyyHPYdJz\nK
tlLlIv3huNWc42gy3JfteGrHQKBgQDUOB9N2Y4eHksSL7wzPtbdst/5jTqf1ytt\nwnatSLBwOGw5FfnSWYOCSmdl9Es3fl3frVGQjn
3M2X9++52n67bJFc6wKB95rA5V\npbY/Em5G1N4RRqeGCsC4fZyQ7fkFHN/c3rgYHG3K33JkAMr+B2fLm6xzkWGDH8l9\nz+xhPxljT
QKBgDlK5CsPkWwd/99eXN5usTaiF/FDE0QYpV+n2kK9Z7YGJLMgF5wp\nQWQxFfrGFTxP0+oNgdNm0bQ0bRFfOZApY6uWb1lSW5YgVq
hH2CVrhUw9musoHNU4\nNuoJQGrBJylxUhFyitkynkktr00GXP0m7BHte8qDpq8QVr0g1nfT2rTZAoGBAJ2w\nSgp1wnU8uzl7QNOvO
McpV+W9fDczEfEP01E1Nu1b74aHfsc/fkl88dG/MXi9rBE9\nXuMUDgP+BQ+c9WvF6V8Hljosxh6Jq2kWphG9pzV4jL23i1k9ejBcHc
9PFH8pEmqz\nD4XS8f4SBJSsRil15TlUplKFTzi4kOasbg9gbIPtAoGAJXApOlzqrjdcp4Kaifke\ncx15VYaocJt5bl4d87FcHRV8O
SPIh58fLaBEdK0+Csb2AvGAYbXqbppMdV9kFUNo\n+oJV+vO1D/MGpL10GuokairOoWFGP3MMVsXak2+3D/gmguSTtlHCzn++ONGkcR
Vl\n31A4a/YmPFGVBisE7V5GyOw=\n-----END PRIVATE KEY-----\n",
  "client_email": "firebase-adminsdk-imq9i@mylifeisagame.iam.gserviceaccount.com",
  "client_id": "111590697001543370983",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-
imq9i%40mylifeisagame.iam.gserviceaccount.com"
}
```

Por su lado, si el cliente tiene habilitado en su navegador los permisos para recibir notificaciones se enviarán

```java
@Log4j2
@Service
public class PushService {


    @Autowired
    private FirebaseApp fireBaseConfiguration;

    public void push(String token, Notification notification) {

        try {
            sendPushViaFirebase(token, notification);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }


    boolean sendPushViaFirebase(String token, Notification notification) {
        Map<String, String> map = new HashMap<>();
        map.put("llave", "valor");
        Message message = Message.builder()
                .putAllData(map)
                .putData("priority", "high")
                .setToken(token) // deviceId
                .setNotification(notification)
                .build();
        try {


            String response = FirebaseMessaging.getInstance(fireBaseConfiguration).send(message);
            log.info("Push notification Message sended succesfully. Response {}", response);
            return true;
        } catch (FirebaseMessagingException e) {
            log.error("Error trying send push notification via firebase to token {}, check details:
{}", token, e.getMessage());
        }
        return false;
    }
```

3.5 Interfaz de Usuario

Para la interfaz de usuario se ha decido usar React, que es una librería de JavaScript que nos ayuda a crear interfaces de usuario interactivas de forma sencilla (**react-app**).

Nuestra interfaz de usuario no es el objetivo de este TFG, con lo que se ha realizado de una forma sencilla, y los puntos que nos interesa centrarnos únicamente es el uso para autenticarnos con los credenciales de Google y la integración con Firebase para recibir notificaciones push que pasamos a detallar.

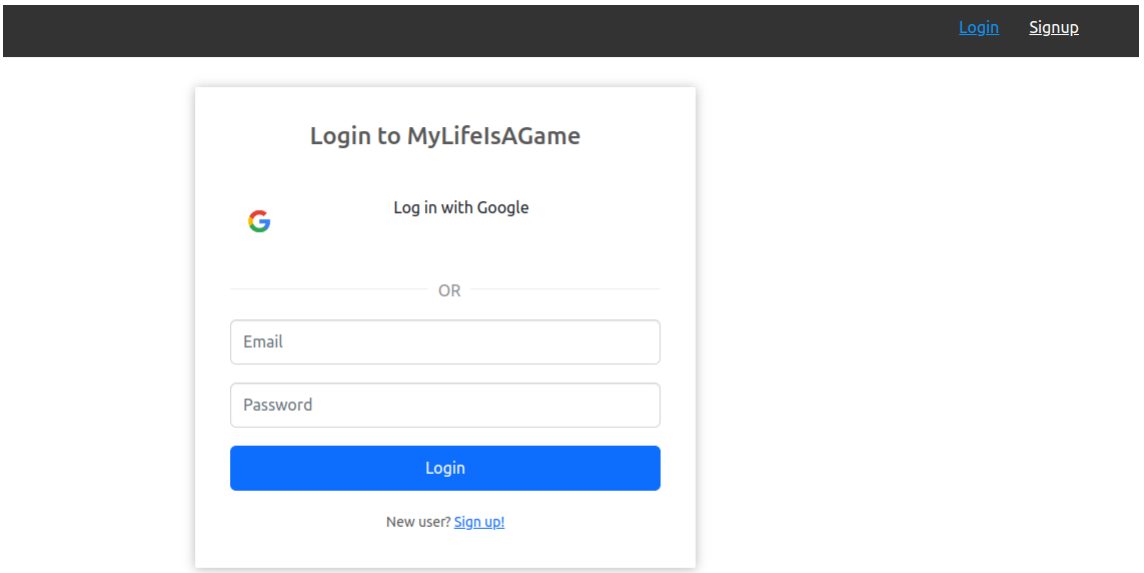
Integración con Google

Para la integración con Google usaremos un componente que nos redirigirá a la página de google para mostrar la página de login y posteriormente contra nuestro servicio authserver-login para registrar el usuario en nuestra aplicación.

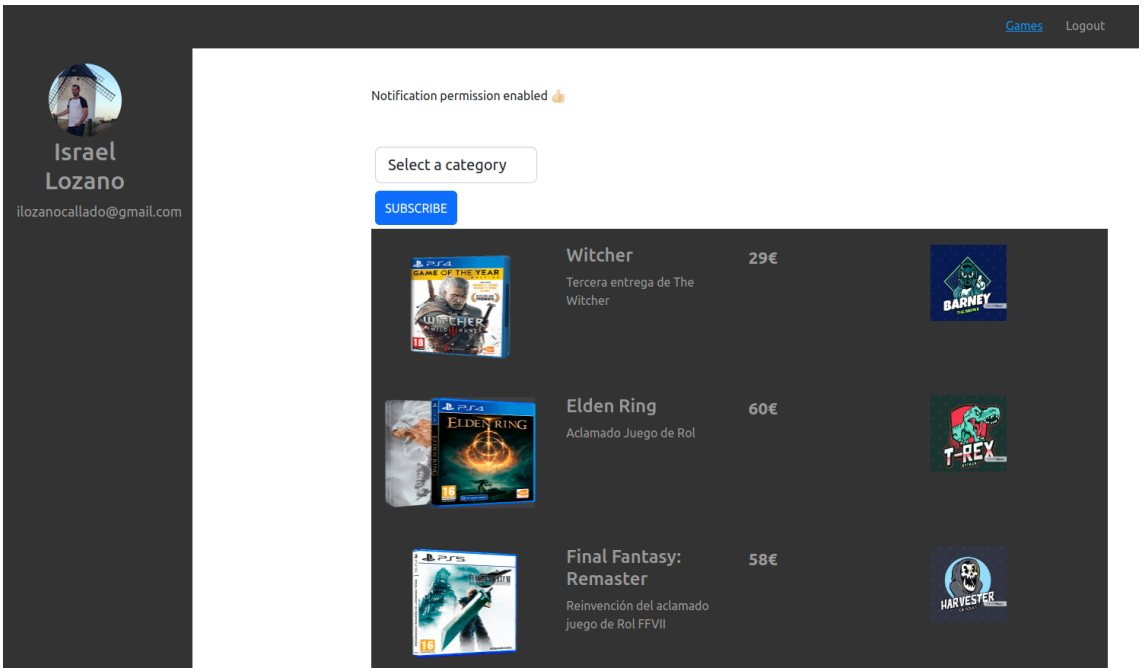
```jsx
import React, { Component } from 'react';
import './Login.css';
import { GOOGLE_AUTH_URL, ACCESS_TOKEN } from '../../constants';
import { login } from '../../util/APIUtils';
import { Link, Redirect } from 'react-router-dom'
import fbLogo from '../../img/fb-logo.png';
import googleLogo from '../../img/google-logo.png';
import githubLogo from '../../img/github-logo.png';
import Alert from 'react-s-alert';

class Login extends Component {
    componentDidMount() {
        // If the OAuth2 login encounters an error, the user is redirected to the /login page with an
error.
        // Here we display the error and then remove the error query parameter from the location.
        if(this.props.location.state && this.props.location.state.error) {
            setTimeout(() => {
                Alert.error(this.props.location.state.error, {
                    timeout: 5000
                });
                this.props.history.replace({
                    pathname: this.props.location.pathname,
                    state: {}
                });
            }, 100);
        }
    }

    render() {
        if(this.props.authenticated) {
            return <Redirect
                to={{
                pathname: "/",
                state: { from: this.props.location }
            }}/>;
        }

        return (
            <div className="login-container">
                <div className="login-content">
                    <h1 className="login-title">Login to MyLifeIsAGame</h1>
                    <SocialLogin />
                    <div className="or-separator">
                        <span className="or-text">OR</span>
                    </div>
                    <LoginForm {...this.props} />
                    <span className="signup-link">New user? <Link to="/signup">Sign up!</Link></span>
                </div>
            </div>
        );
    }
}

class SocialLogin extends Component {
    render() {
        return (
            <div className="social-login">
                <a className="btn btn-block social-btn google" href={GOOGLE_AUTH_URL}>
                    <img src={googleLogo} alt="Google" /> Log in with Google</a>
            </div>
        );
    }
}


class LoginForm extends Component {
    constructor(props) {
        super(props);
        this.state = {
            email: '',
            password: ''
        };
        this.handleInputChange = this.handleInputChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleInputChange(event) {
        const target = event.target;
        const inputName = target.name;
        const inputValue = target.value;

        this.setState({
            [inputName] : inputValue
        });
    }

    handleSubmit(event) {
        event.preventDefault();

        const loginRequest = Object.assign({}, this.state);

        login(loginRequest)
        .then(response => {
            localStorage.setItem(ACCESS_TOKEN, response.accessToken);
            Alert.success("You're successfully logged in!");
            this.props.history.push("/");
        }).catch(error => {
            Alert.error((error && error.message) || 'Oops! Something went wrong. Please try again!');
        });
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <div className="form-item">
                    <input type="email" name="email"
                        className="form-control" placeholder="Email"
                        value={this.state.email} onChange={this.handleInputChange} required/>
                </div>
                <div className="form-item">
                    <input type="password" name="password"
                        className="form-control" placeholder="Password"
                        value={this.state.password} onChange={this.handleInputChange} required/>
                </div>
                <div className="form-item">
                    <button type="submit" className="btn btn-block btn-primary">Login</button>
                </div>
            </form>
        );
    }
}

export default Login
```

```
import React, { Component } from 'react';
import { ACCESS_TOKEN } from '../../constants';
import { Redirect } from 'react-router-dom'

class OAuth2RedirectHandler extends Component {
    getUrlParameter(name) {
        name = name.replace(/[\[]/, '\\[').replace(/[\]]/, '\\]');
        var regex = new RegExp('[\\?&]' + name + '=([^&#]*)');

        var results = regex.exec(this.props.location.search);
        return results === null ? '' : decodeURIComponent(results[1].replace(/\+/g, ' '));
    };

    render() {
        const token = this.getUrlParameter('token');
        const error = this.getUrlParameter('error');

        if(token) {
            localStorage.setItem(ACCESS_TOKEN, token);
            return <Redirect to={{
                pathname: "/profile",
                state: { from: this.props.location }
            }}/>;
        } else {
            return <Redirect to={{
                pathname: "/login",
                state: {
                    from: this.props.location,
                    error: error
                }
            }}/>;
        }
    }
}

export default OAuth2RedirectHandler;
```

```
import React, {Component} from 'react';
import './Signup.css';
import {Link, Redirect} from 'react-router-dom'
import {GOOGLE_AUTH_URL} from '../../constants';
import {signup} from '../../util/APIUtils';
import googleLogo from '../../img/google-logo.png';
import Alert from 'react-s-alert';

class Signup extends Component {
    render() {
        if(this.props.authenticated) {
            return <Redirect
                to={{
                pathname: "/",
                state: { from: this.props.location }
            }}/>;
        }

        return (
            <div className="signup-container">
                <div className="signup-content">
                    <h1 className="signup-title">Signup with SpringSocial</h1>
                    <SocialSignup />
                    <div className="or-separator">
                        <span className="or-text">OR</span>
                    </div>
                    <SignupForm {...this.props} />
                    <span className="login-link">Already have an account? <Link to="/login">Login!
</Link></span>
                </div>
            </div>
        );
    }
}


class SocialSignup extends Component {
    render() {
        return (
            <div className="social-signup">
                <a className="btn btn-block social-btn google" href={GOOGLE_AUTH_URL}>
                    <img src={googleLogo} alt="Google" /> Sign up with Google</a>
            </div>
        );
    }
}

class SignupForm extends Component {
    constructor(props) {
        super(props);
        this.state = {
            name: '',
            email: '',
            password: ''
        }
        this.handleInputChange = this.handleInputChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleInputChange(event) {
        const target = event.target;
        const inputName = target.name;
        const inputValue = target.value;

        this.setState({
            [inputName] : inputValue
        });
    }

    handleSubmit(event) {
        event.preventDefault();

        const signUpRequest = Object.assign({}, this.state);

        signup(signUpRequest)
        .then(response => {
            Alert.success("You're successfully registered. Please login to continue!");
            this.props.history.push("/login");
        }).catch(error => {
            Alert.error((error && error.message) || 'Oops! Something went wrong. Please try again!');

        });
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <div className="form-item">
                    <input type="text" name="name"
                        className="form-control" placeholder="Name"
                        value={this.state.name} onChange={this.handleInputChange} required/>
                </div>
                <div className="form-item">
                    <input type="email" name="email"
                        className="form-control" placeholder="Email"
                        value={this.state.email} onChange={this.handleInputChange} required/>
                </div>
                <div className="form-item">
                    <input type="password" name="password"
                        className="form-control" placeholder="Password"
                        value={this.state.password} onChange={this.handleInputChange} required/>
                </div>
                <div className="form-item">
                    <button type="submit" className="btn btn-block btn-primary" >Sign Up</button>
                </div>
            </form>

        );
    }
}

export default Signup
```



Una vez logado nos redirigirá a la página principal con el listado de videojuegos y en el lateral izquierdo los detalles de nuestra cuenta en Google con nuestro avatar, nombre y email.



Para la integración con firebase para las notificaciones push una vez suscrito a una categoría, se necesita registrar la configuración de Firebase generada anteriormente

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getMessaging, getToken, onMessage } from "firebase/messaging";
import {subscribePushNotifications} from "../../util/APIUtils";
import Alert from "react-s-alert";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
    apiKey: "AIzaSyByXLxlHLqPzkS1I3lkp6VNsdWTa8EA00A",
    authDomain: "mylifeisagame.firebaseapp.com",
    projectId: "mylifeisagame",
    storageBucket: "mylifeisagame.appspot.com",
    messagingSenderId: "972684121149",
    appId: "1:972684121149:web:d1bc6611cca77e28488654"
};

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
const messaging = getMessaging(firebaseApp);

export const fetchToken = (setTokenFound,email, setTokenUnRegistered, isTokenUnRegistered) => {
    return getToken(messaging, {vapidKey:
'BA56MswFJC8RR4HB1YPglUY8whKDJs8KIF0sASWBM3Yjz4UUoS6MAUGq4f90ppNgozdJI9Skrtd6RwCvWDIc4xQ'}).then((curre
ntToken) => {
        if (currentToken) {
            console.log('current token for client: ', currentToken);
            setTokenFound(true);
            // Track the token -> client mapping, by sending to backend server
            // show on the UI that permission is secured

            if(isTokenUnRegistered) {
                const userToken = {
                    userId: email,
                    token: currentToken
                };
                subscribePushNotifications(userToken).then(response => {
                    Alert.success("You're successfully registered for push notifications.");
                    setTokenUnRegistered(false)
                }).catch(error => {
                    setTokenUnRegistered(true)
                   // Alert.error((error && error.message) || 'Oops! Something went wrong. Please try
again!');
                });
            }

        } else {
            console.log('No registration token available. Request permission to generate one.');
            setTokenFound(false);
            setTokenUnRegistered(true)
            // shows on the UI that permission is required
        }
    }).catch((err) => {
        console.log('An error occurred while retrieving token. ', err);
        // catch error while creating client token
    });
}

export const onMessageListener = () =>
    new Promise((resolve) => {
        onMessage(messaging, (payload) => {
            resolve(payload);
        });
    });
```

Y tendremos una función que será encargada de mostrarnos las notificaciones cada vez que se cree un nuevo videojuego perteneciente a nuestra categoría registrada

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getMessaging, getToken, onMessage } from "firebase/messaging";
import {subscribePushNotifications} from "../../util/APIUtils";
import Alert from "react-s-alert";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
    apiKey: "AIzaSyByXLxlHLqPzkS1I3lkp6VNsdWTa8EA00A",
    authDomain: "mylifeisagame.firebaseapp.com",
    projectId: "mylifeisagame",
    storageBucket: "mylifeisagame.appspot.com",
    messagingSenderId: "972684121149",
    appId: "1:972684121149:web:d1bc6611cca77e28488654"
};

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
const messaging = getMessaging(firebaseApp);

export const fetchToken = (setTokenFound,email, setTokenUnRegistered, isTokenUnRegistered) => {
    return getToken(messaging, {vapidKey:
'BA56MswFJC8RR4HB1YPglUY8whKDJs8KIF0sASWBM3Yjz4UUoS6MAUGq4f90ppNgozdJI9Skrtd6RwCvWDIc4xQ'}).then((curre
ntToken) => {
        if (currentToken) {
            console.log('current token for client: ', currentToken);
            setTokenFound(true);
            // Track the token -> client mapping, by sending to backend server
            // show on the UI that permission is secured

            if(isTokenUnRegistered) {
                const userToken = {
                    userId: email,
                    token: currentToken
                };
                subscribePushNotifications(userToken).then(response => {
                    Alert.success("You're successfully registered for push notifications.");
                    setTokenUnRegistered(false)
                }).catch(error => {
                    setTokenUnRegistered(true)
                   // Alert.error((error && error.message) || 'Oops! Something went wrong. Please try
again!');
                });
            }

        } else {
            console.log('No registration token available. Request permission to generate one.');
            setTokenFound(false);
            setTokenUnRegistered(true)
            // shows on the UI that permission is required
        }
    }).catch((err) => {
        console.log('An error occurred while retrieving token. ', err);
        // catch error while creating client token
    });
}

export const onMessageListener = () =>
    new Promise((resolve) => {
        onMessage(messaging, (payload) => {
            resolve(payload);
        });
    });
```


Notification permission enabled 👍
NUEVO VIDEOJUEGO just now ×Close
Nuevo Juego añadido para tu categoría suscrita:
Cyber Punk
Select a category
SUBSCRIBE
Witcher
Tercera entrega de The Witcher
29€
Elden Ring
60€

4. Pruebas

Para el desarrollo de nuestras pruebas hemos centrado nuestros esfuerzos principalmente en nuestro servicio de API *(mylifeisagame-api*). Se han realizado tres niveles de test que pasamos a describir.

Test Unitarios

Se prueba toda la comunicación con desde nuestros servicios hasta el repositorio para verificar los valores esperados

UsersServiceUnitTest		56 ms
obtainPush()	passed	42 ms
findGameById()	passed	2 ms
findAllCategories()	passed	1 ms
subscribeUserPush()	passed	2 ms
findAllUsersSubscribedToACategory()	passed	2 ms
subscribeCategory()	passed	1 ms
findGamesByName()	passed	3 ms
findGamesByCategory()	passed	2 ms
findAllCategoriesSubscribed()		passed
findAllGames()	passed	1 ms

ShopServiceUnitTest		8 ms
deleteCategory()		passed
findGameById()		passed
findAllCategories()		passed
updateGame()		passed
updateShop()	passed	1 ms
findCategoryById()	passed	1 ms
findGamesByName()	passed	1 ms
createCategory()	passed	1 ms
createGame()	passed	1 ms
createShop()		passed
findGamesByCategory()		passed
cancelGame()	passed	3 ms
findAllGames()		passed

ShopServiceUnitTest		8 ms
deleteCategory()		passed
findGameById()		passed
findAllCategories()		passed
updateGame()		passed
updateShop()	passed	1 ms
findCategoryById()	passed	1 ms
findGamesByName()	passed	1 ms
createCategory()	passed	1 ms
createGame()	passed	1 ms
createShop()		passed
findGamesByCategory()		passed
cancelGame()	passed	3 ms
findAllGames()		passed

Test de Integración

Para testear nuestros endpoint usaremos la @WebMvcTest que nos facilita junit para testear y mockear las peticiones a nuestros endpoints.

AdminRESTControllerIntegrationTest		58 ms
deleteCategory()	passed	25 ms
createShopsCredentials()	passed	11 ms
deleteShops()	passed	6 ms
findCategories()	passed	5 ms
findShops()	passed	6 ms
createCategory()	passed	5 ms

ShopsRESTControllerIntegrationTest		62 ms
updateGame()	passed	27 ms
updateShop()	passed	5 ms
findCategories()	passed	5 ms
findGames()	passed	6 ms
getGameDetails()	passed	5 ms
createGame()	passed	4 ms
createShop()	passed	6 ms
cancelGame()	passed	4 ms

AdminRESTControllerIntegrationTest		58 ms
deleteCategory()	passed	25 ms
createShopsCredentials()	passed	11 ms
deleteShops()	passed	6 ms
findCategories()	passed	5 ms
findShops()	passed	6 ms
createCategory()	passed	5 ms

Test de arquitectura

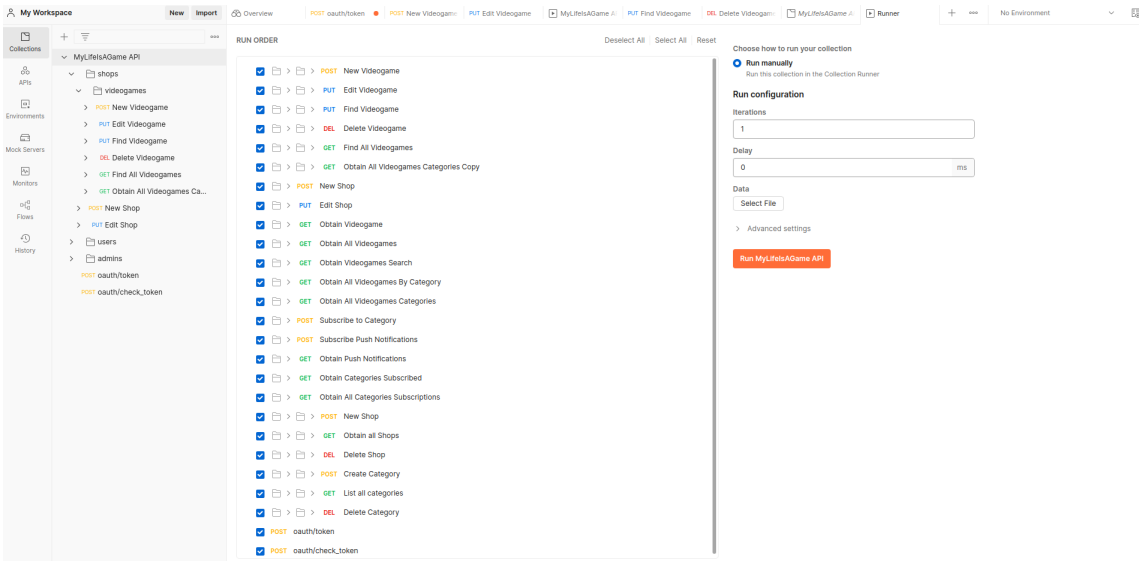
Para validar que nuestro código sigue una arquitectura hexagonal hemos usado ArchUnitTest para verificarlo.

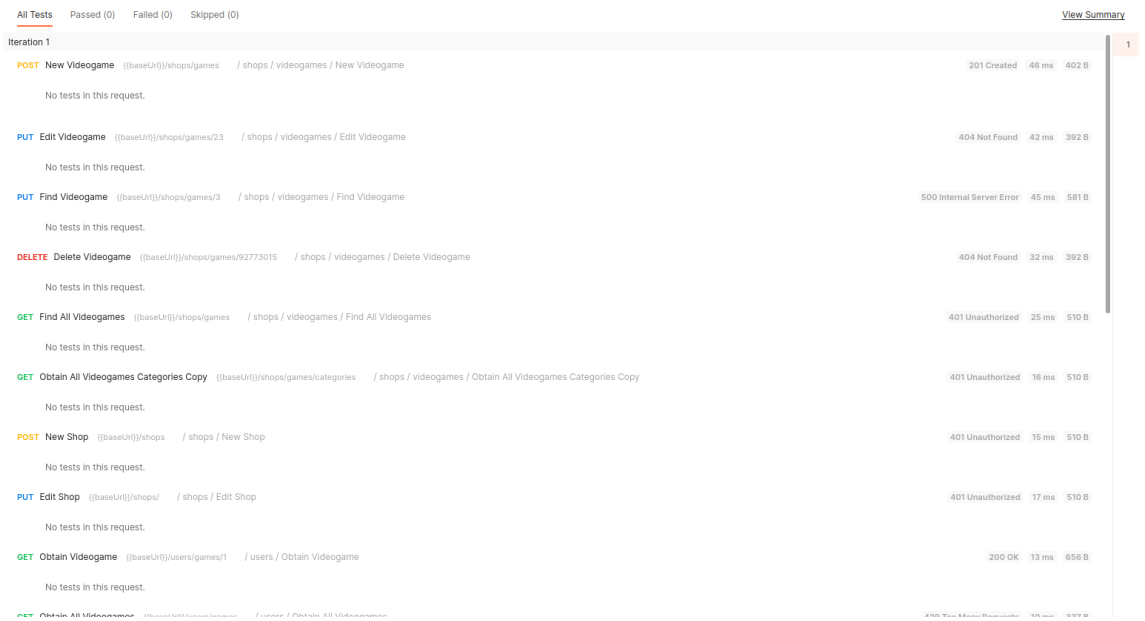
```java
@AnalyzeClasses(packages = "com.ilozano.mylifeisagame", importOptions = {
ImportOption.DoNotIncludeTests.class })
public class OnionArchitectureTest {

    @ArchTest
    static final ArchRule onion_architecture_is_respected = onionArchitecture()
            .domainModels("..domain..")
            .domainServices("..domain.service..")
            .applicationServices("..application..")
            .adapter("infrastructure", "..infrastructure..");

    @ArchTest
    static ArchRule services_should_end_with_ServiceImpl =
            classes()
                    .that().areAnnotatedWith(Service.class)
                    .and().resideInAPackage("..domain.service")
                    .should().haveSimpleNameEndingWith("ServiceImpl");
```

OnionArchitectureTest		2.36 s
onion_architecture_is_respected	passed	2.36 s
services_should_end_with_ServiceImpl	passed	4 ms

Test End To End

Se han realizado todos los test de casos reales desde que se llama al endpoint hasta que se recibe el valor esperado usando Postman*[11]*





5. Mejoras y evolución

El objetivo principal del proyecto se ha cumplido al 100 por ciento, que era generar la infraestructura y módulos necesarios para integrar una API en tiempo real con todos los elementos necesarios para la interacción entre servicios externos y los usuarios.

No obstante debido al escaso tiempo necesario sigue habiendo cabida a muchas mejoras aunque el esqueleto esté entero, entre ellas podemos destacar las siguientes:

- Autenticación:
  - Cambiar la generación del token para que caduque con un menor intervalo de tiempo haciendo necesario que el cliente tenga que refrescar el token de acceso para mayor seguridad
  - Para la generación de credenciales para nuestro caso de estudio hemos facilitado un endpoint que crea las credenciales directamente. Hay herramientas más adecuadas para esto, como puede ser Keycloak *[12]* que se integra perfectamente con Spring y nos facilita una consola de administración donde poder gestionar las credenciales y roles de acceso de nuestros usuarios

- Rate Limit:
  - Mejorar el sistema de caché para evitar mejor posibles ataques DDos. Una de las posibilidades que investigué fue usar SpiKe Interceptors para garantizar mejor los tiempos de ventana.

- Mejorar la interfaz de usuario: Al no ser el objetivo de nuestro proyecto más allá de poder visualizar la integración de nuestros componentes, visualmente es poco agradable. Mejorar tanto a nivel de estilos (hacer responsive), cómo añadir más funcionalidades que ya nos provee el API pero no se han integrado.

- Añadir integración continua:
  - Integrar nuestros artefactos a Jenkins y usar git-flow

- Monitorización: Una de las partes que creo que es imprescindible es poder analizar las peticiones recibidas, para ello necesitamos alguna sistema de monitorización como Grafana *[13]*
- Escalar en distintas máquinas gestionadas por algún balanceador de carga como Nginx *[14]*

6. Conclusiones

Llevo varios años dedicándome al sector de la Ingeniería Informática, pero los últimos seis años he pasado a ser responsable de producto, alejándome un poco del sector. Una de mis grandes metas a la hora de abordar este proyecto, era abordar un proyecto desde cero y trabajar en todas las fases, desde el diseño hasta la implementación, y hacerlo con tecnologías principalmente backend, que es lo que siempre me ha gustado, y que abordará necesidades actuales en el ámbito tecnológico.

Con este TFG he conseguido no solo ejercer roles como arquitecto, diseñador y analista, si no que además he tenido que llegar hasta las tripas del código para poder entender todas las funcionalidades y necesidades, algo que me ha recordado que es necesario hacer siempre para ser un buen Ingeniero del Software.

Puedo decir con satisfacción, que independientemente del resultado de este TFG, a nivel personal estoy completamente satisfecho y orgulloso de haber logrado todas las metas y objetivos propuestos.

Para poder realizar este proyecto con hemos seguido rigurosamente la planificación propuesta en el apartado: *1.3 Enfoque y método seguido*

Hemos trabajado siguiendo SCRUM como metodología ágil, y realmente ha sido la más adecuada para nuestro caso. Al planificar Sprints cortos nos ponemos objetivos realistas que después del sprint volvíamos a analizar los cambios necesarios de forma que no hacíamos un gran desarrollo que luego hubiera que descartar, si no pequeños hitos.

Una parte que me hubiera gustado explorar más hubiera sido a nivel de infraestructura, para integrar monitorización para el uso del API y haber podido investigar más sobre Integración Contínua.

7. Glosario

- Caché: memoria que se sitúa entre la unidad central de procesamiento (CPU) y la memoria de acceso aleatorio (RAM) para acelerar el intercambio de datos
- IDE: Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- API: La interfaz de programación de aplicaciones, conocida también por la sigla API, en inglés, application programming interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.
- REST: Acrónimo de Representational State Transfer, es un estilo de arquitectura software stateless que utiliza el protocolo HTTP para el intercambio de información entre distintos sistemas en formatos muy específicos (json o xml)
- SCRUM: Es una de las llamadas metodologías ágiles más conocidas y usadas. La principal característica de esta metodología es que pretende obtener prototipos cuanto antes adaptándose a los cambios en los requisitos del producto.
- Evento: un evento es una acción que es detectada por un programa; éste, a su vez, puede hacer uso del mismo o ignorarlo. Por lo general, una aplicación cuenta con uno o más hilos de ejecución dedicados a atender los distintos eventos que se le presenten.
- Framework: Un framework es un marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de software. Utilizar un framework permite agilizar los procesos de desarrollo ya que evita tener que escribir código de forma repetitiva, asegura unas buenas prácticas y la consistencia del código.

8. Bibliografía

- [1] https://spec.openapis.org/oas/latest.html, OpenAPI Specification
- [2] https://oauth.net/2/, Oauth 2.0
- [3] https://firebase.google.com/, FireBase
- [4] https://redis.io/, Redis
- [5] https://es.reactjs.org/, React
- [6] https://www.tiobe.com/tiobe-index/, Índice TIOBE
- [7] Evans, Eric, Domain-Driven Design: Tackling Complexity in the Heart of Software, 4ª Edición (2004)
- [8] https://tools.ietf.org/html/rfc6749#section-4.4, RFC 6794
- [9] https://es.wikipedia.org/wiki/MD5, MD5
- [10] https://console.cloud.google.com/, Consola de Desarrolladores de Google
- [11] https://www.postman.com/, Postman
- [12] https://www.keycloak.org/, Keycloak
- [13] https://grafana.com/, Grafana
- [14] https://www.nginx.com/, Nginx

- Leonard Richardson, RESTful Web APIs: Services for a Changing World, O'Reilly Media; N.º 1 edición (30 septiembre 2013)

- Charlotte & Peter Fiell, REST IN PRACTISE: Hypermedia and Systems Architecture, O'REILLY MEDIA; N.º 1 edición (12 octubre 2010)

- Charlotte & Peter Fiell, SERVICE DESIGN PATTERNS: Fundamental Design Solutions for RESTful Web Services, ADDISON-WESLEY EDUCATIONAL PUBLISHERS INC; N.º 1 edición (9 noviembre 2011)

- Michael Bohner, RESTful Web API Handbook , CreateSpace Independent Publishing Platform (17 diciembre 2015)
- Arun Chinnachamy, Redis Applied Design Patterns , Packt Publishing (23 septiembre 2014)
- George Coulouris , Jean Dollimore, Tim Kindberg, Gordon Blair (Autor) ; Distributed Systems: International Edition, Pearson; N.º: 5 edición (6 noviembre 2013)

9. Anexos

El código del repositorio se encuentra subido en el siguiente enlace con toda la explicación:
https://github.com/ilozanocallado/mylifeisagame.git

Requisitos previos

Entorno Linux
Java 11.0
Npm
Docker
Versión Chrome 108.0.5359.124 o superior
MySQL

- Crear esquema de base de datos con nombre: mylifeisagame y puerto: 3306

Despliegue

Para implementar la infraestructura necesaria para el proyecto (imágenes de Kafka y redish), primero ejecute:

```
sudo docker-compose
```

desde la raíz donde se encuentra el archivo docker-compose.yml.

Los módulos integrados en nuestros artefactos tienen un script de ruta que carga la base de datos con todos los datos requeridos y alguna información completa para las tiendas y videojuegos por defecto.

Inicie los siguientes módulos de arranque desde su IDE (los artefactos se desarrollaron usando IntelliJ o genere los archivos jar con el plugin de maven).

Es importante seguir el siguiente orden:

- auth-server
- mylifeisagame-api (`http://localhost:18081`)
- auth-login
- notification

Dentro del directorio raíz de la aplicación react-app ejecute:

```
npm start
```

Para probar la API, se proporciona un script de Postman (MyLifeIsAGame API.postman_collection.json):

- [/ouath/token] punto final para generar las credenciales de Bearer. Tenemos administradores y usuarios predeterminados para probar:
    - Administrador: admin / admin
    - Tiendas: trex / pruebas1

- [shops] endpoints disponibles para tiendas. Se necesitan credenciales de tiendas
- [admins] endpoints disponibles para el administrador. Se necesitan credenciales de administrador
- [users] endpoints para la interacción de los usuarios. Solo se puede probar en la aplicación react después de iniciar sesión con las credenciales de Google

Para iniciar sesión en la aplicación de react: (http://localhost:3000/) debe acceder utilizando una cuenta de Google.

Para permitir las notificaciones automáticas en el navegador, se debe habilitar las notificaciones en él.