Retro Exchanges

Francisco Javier Martín Cubino
Grado en Ingeniería Informática

Antoni Oller Arcas
Consultor

Santi Caballé Llobet
Profesor responsable de la asignatura

09/06/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	Retro Exchanges
Nombre del autor:	Francisco Javier Martín Cubino
Nombre del consultor:	Antoni Oller Arcas
Fecha de entrega (mm/aaaa):	06/2022
Área del Trabajo Final:	Java EE
Titulación:	*Grado en Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

El presente proyecto, tiene como objetivo la creación de una aplicación para que una comunidad de usuarios pueda comprar y vender sus videojuegos antiguos (“retro”) de manera confiable.

La idea surge al observar un aumento de la demanda de videojuegos y consolas “retro” impulsado por la nostalgia que provocan este tipo de productos entre los usuarios que los disfrutaron en su niñez.

Estos usuarios se han convertido en adultos con un mayor poder adquisitivo que desean volver a jugar y coleccionar los juegos y sistemas que tantos buenos momentos les proporcionaron o que no pudieron tener en su día.

A la hora de adquirir productos de segunda mano, siempre surge la duda de si el comprador o vendedor va a cumplir con lo acordado.

Retro Exchanges pretende ser la aplicación que facilite encontrar los productos y además añada seguridad a las transacciones mediante el empleo un sistema de reputación.

Abstract (in English, 250 words or less):

This project aims to create an application for a community of users to buy and sell their old ("retro") video games in a reliable way.

The idea arises from observing an increase in the demand for "retro" video games and consoles driven by the nostalgia that this type of products provoke among users who enjoyed them in their childhood.

These users have become adults with greater purchasing power who want to return to play and collect the games and systems that gave them so many good times or that they could not have in their day.

When acquiring second-hand products, there is always the question of whether the buyer or seller is going to comply with the agreement.

Retroexchanges aims to be the application that facilitates finding the products and also adds security to transactions through the use of a reputation system.

Palabras clave (entre 4 y 8):

Videojuegos, Consolas, Retro, Java EE, REST, Spring, MySQL

Índice

Lista de ilustraciones

1. Introducción

El presente documento pretende servir como propuesta para el Trabajo Fin de Grado (a partir de ahora TFG). En él se describe la temática, los motivos, los objetivos y alcance que se desean conseguir, así como una planificación inicial con las tareas para poder culminar el proyecto de forma correcta.

1.1 Contexto y justificación del Trabajo

El mercado de las consolas y videojuegos antiguos o también denominados “retro” está experimentando un auge impulsado en gran medida por la nostalgia que provocan este tipo de productos entre los usuarios que los disfrutaron en su niñez. Estos usuarios se han convertido en adultos con un mayor poder adquisitivo que desean volver a jugar y coleccionar los juegos y sistemas que tantos buenos momentos les proporcionaron.

Algunos de los títulos son difíciles de conseguir y están alcanzado precios de locura [1] y hay que añadir además que las empresas de videojuegos están aprovechando para distribuir ediciones coleccionista [2] de sistemas y videojuegos antiguos en ediciones más pequeñas a las originales lo que ha generado efecto llamada a este tipo de productos antiguos.

Por ese motivo, he pensado que sería interesante el crear una aplicación en Java EE en la que los usuarios puedan poner a la venta sus sistemas y videojuegos “retro” y que además puedan comprar los títulos o sistemas que aún no tienen en su poder. La aplicación incorporaría un sistema para valorar cada transacción de manera que los usuarios ganarían o perderían reputación a medida que realizan transacciones con el fin de evitar transacciones fraudulentas. También sería necesario incluir una herramienta de búsqueda que permitiera localizar de forma sencilla los productos en función al sistema, categoría, fecha de publicación, formato, etc.

1.2 Objetivos del Trabajo

El principal objetivo del trabajo es la construcción de un sistema, donde una comunidad de usuarios pueda venderse y comprarse sus videojuegos entre sí y que pueda ser utilizado desde un ordenador, tableta o móvil.

Así mismo, el sistema permitirá a los usuarios registrarse en el sistema para poder enviar ofertas de compra a los demás usuarios.

El sistema deberá de implementar mecanismos de seguridad para que la información sea modificable y visible solamente por los usuarios autorizados.

Para facilitar la búsqueda de los videojuegos y consolas, se clasificarán por categorías que serán gestionadas por un usuario con permisos de administrador.

El sistema permitirá realizar búsquedas por categoría, nombre y propietario.

El sistema permitirá marcar los videojuegos y consolas como favoritos para poder ser encontrados más fácilmente en posteriores búsquedas.

Para evitar el fraude, se implementará un sistema de valoraciones, donde comprador y vendedor de asignaran una serie de puntos al finalizar una transacción. Dicha puntuación será visible antes de enviar o aceptar una oferta.

Los usuarios con una puntuación baja podrán ser buscados y bloqueados por el usuario administrador.

La aplicación seguirá un modelo cliente servidor. La parte servidora contará con una capa de servicio basada en un API REST donde se asegurará la persistencia de la información empleando una base de datos relacional.

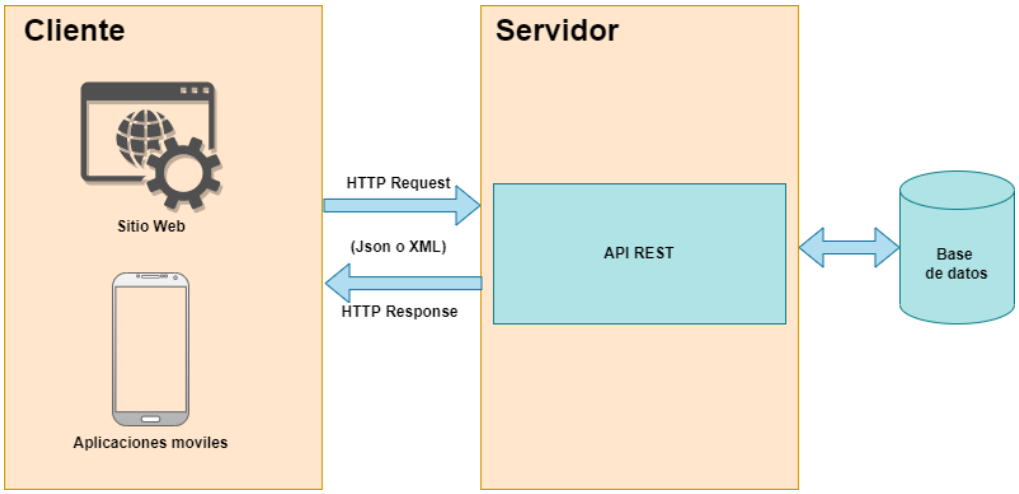


Ilustración 1: Arquitectura Cliente/Servidor empleando API REST

La parte cliente de la aplicación nos permitirá interactuar con la información contenida en la base de datos explotando el API REST mediante el uso de los métodos “get”, “post”, “put” y “delete” inherentes al protocolo HTTP.

De este modo la parte cliente puede ser una aplicación móvil o un sitio web desarrollado en java o cualquier otra tecnología.

Para el TFG se desarrollarán ambas partes siendo la parte servidora completamente en Java.

La parte cliente será un cliente web ligero para el que se utilizará un framework basado en javascript.

1.3 Enfoque y método seguido

Debido a las limitaciones de tiempo. En todo momento se ha perseguido conseguir un producto mínimo viable (MVP). Para ello, una vez analizado el problema y definido un diseño, se han ido implementando las funcionalidades en iteraciones progresivas intentando conseguir un mínimo producto viable.

1.4 Planificación del Trabajo

Para realizar la planificación se seguirá el calendario de la asignatura TFG por tanto se dividirá en cuatro fases, donde la segunda y tercera fase tendrán una duración mayor debido a la carga de trabajo que conllevan.

Fase	Tareas	Fecha Inicio	Fecha Fin
Fase 1	Plan de trabajo	16/02/22	01/03/22
Fase 2	Requerimientos, análisis y diseño	02/03/22	01/04/22
Fase 3	Implementación y pruebas	02/04/22	23/05/22
Fase 4	Memoria y presentación	24/05/22	09/06/22

Fase 1: Plan de trabajo

La primera fase tiene como objetivo la realización de una propuesta inicial consensuada con el tutor de la asignatura y la realización de un plan de trabajo de alto nivel basada en dicha propuesta. Tanto la propuesta como la planificación conformarán el documento a entregar en la PEC1. Adicionalmente, crearemos el documento que contendrá la memoria final del proyecto.

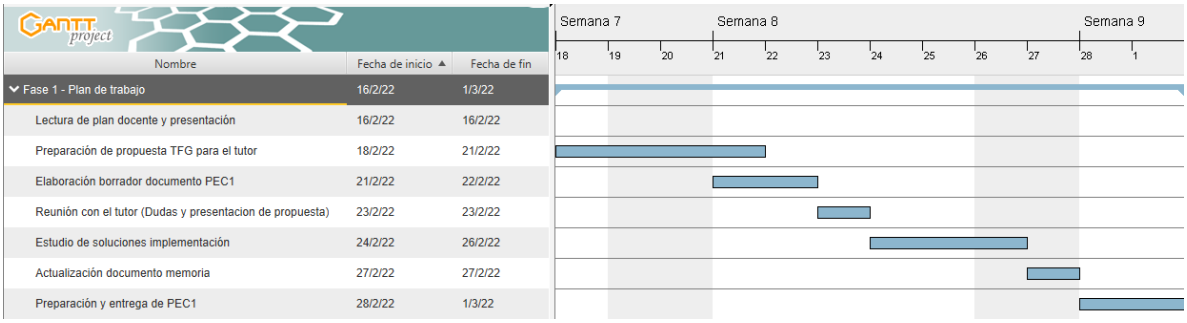


Ilustración 2: Planificación Gantt Fase 1

Fase 2: Requerimientos, análisis y diseño

Durante la segunda fase del proyecto procederemos al análisis y diseño de la solución a implementar. Para ello se crearemos las historias de usuario, así como los criterios de aceptación necesarios. Una vez realizados y validados realizaremos el diagrama de clases, modelo de datos y diseño de la arquitectura del servicio, así como un prototipado de la interfaz gráfica a implementar. Finalmente se procederemos a diseñar un plan de pruebas para la siguiente fase. Al finalizar la fase, actualizaremos el documento de la memoria con los progresos obtenidos.

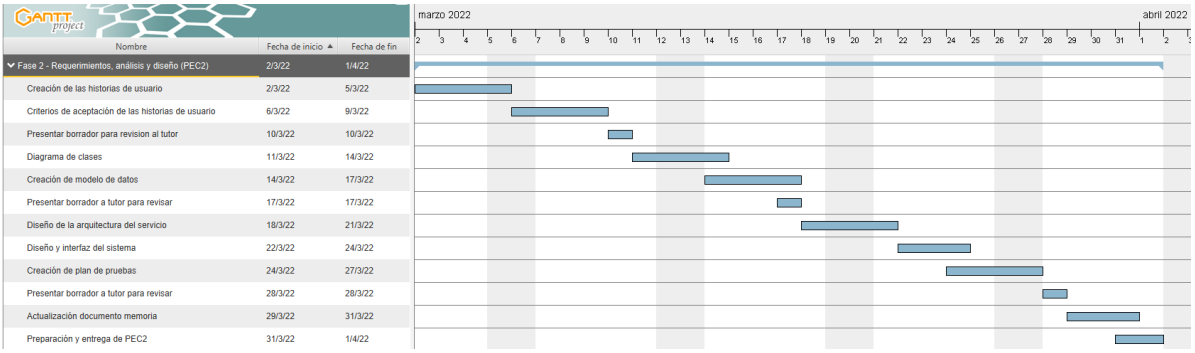


Ilustración 3: Planificación Gantt Fase 2

Fase 3: Implementación y pruebas

En la tercera fase se procederá a la puesta en marcha del entorno de trabajo. Una vez hayamos puesto en marcha el entorno de trabajo crearemos el proyecto y lo subiremos a un repositorio “git” en la nube para el respaldo y seguimiento del código fuente.
Realizaremos la implementación por iteraciones de los casos de uso y de las historias de usuario definidas en la documentación de la fase anterior, de manera que en cada una de las iteraciones vayamos obteniendo una versión más completa de la solución.
Al igual que en la fase anterior, actualizaremos el documento de la memoria con los progresos obtenidos.

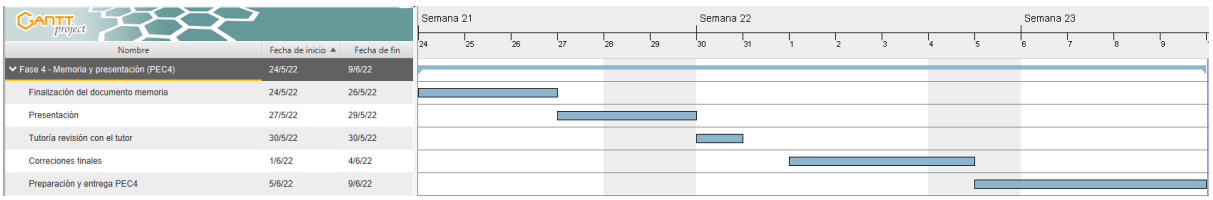


Ilustración 4: Planificación Gantt Fase 3

Fase 4: Memoria y presentación

Durante esta fase finalizaremos la documentación de la memoria que hemos ido realizando a lo largo de las cuatro fases del proyecto y elaboraremos una presentación virtual. Esta presentación virtual consistirá en un documento en video (a partir de una presentación en PowerPoint o similar) en la que sintetizaremos de forma clara y concisa el trabajo realizado y los resultados que hemos obtenido.

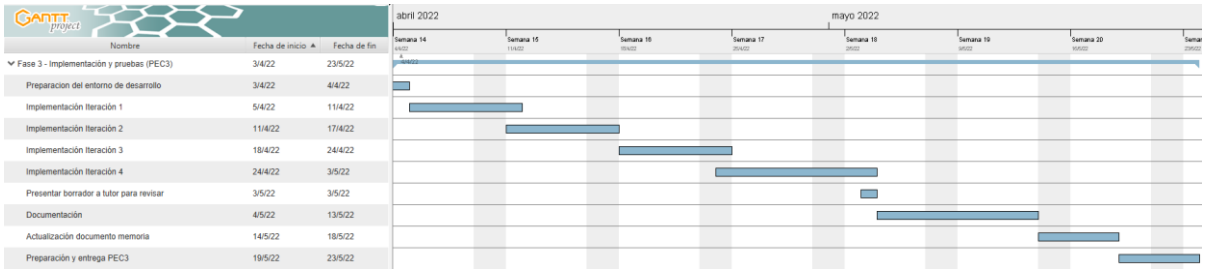


Ilustración 5: Planificación Gantt Fase 4

2. Análisis

Tras establecer los objetivos del proyecto a desarrollar en el apartado anterior, a continuación, se identifican los actores implicados, que requisitos han de cumplirse, las partes y la arquitectura que tendrá el sistema.

2.1 Requisitos

La aplicación a realizar consiste en un portal con arquitectura cliente-servidor donde los usuarios podrán poner a la venta sus videojuegos y consolas antiguos y adquirir estos productos de otros usuarios.

- Para poder poner a la venta o comprar productos en la aplicación, los usuarios se deberán de registrar en el sistema proporcionando nombre, NIF, teléfono, email y dirección postal.

- En caso de que un usuario no desee registrarse, solamente podrá realizar búsquedas para ver que productos hay a la venta, pero no podrá acceder a la compra de estos.

- El sistema contará con un panel de administración donde un usuario registrado con funciones de administrador creará las categorías donde los usuarios podrán agregar y poner a la venta los productos. Las categorías serán determinadas por el tipo de sistema de videojuegos al que pertenecen y no podrán ser modificadas por los usuarios que no sean el administrador.

- En caso de que un usuario necesite una categoría que no exista en el sistema deberá de solicitar su creación al administrador vía correo electrónico.

- Para poner a la venta un producto el usuario ha de añadirlo al sistema en una categoría indicando, el nombre, fecha de publicación, descripción, precio y aportar una serie de fotos detalladas del producto donde se pueda comprobar su estado de conservación.

- Los usuarios podrán buscar por categoría, nombre y por usuario.

- Una vez encontrado un juego o consola, podrá ser marcado como favorito para encontrarlo de forma rápida en otro momento.

- Cuando un usuario este interesado en un producto podrá indicarlo en la ficha de este, de manera que el vendedor recibirá una petición de compra.

- En caso de que el vendedor acepte la petición, este se marcara como reservado, de manera que una vez concluida la venta entre usuarios se marcará finalmente como vendido. El sistema ocultara el producto una vez reservado para que no aparezca en las búsquedas de los demás compradores.

- Al finalizar una transacción, ambos usuarios podrán valorar como se ha comportado el otro usuario, lo que ira generando en cada usuario una reputación.

- El administrador del sistema podrá bloquear el acceso de los usuarios que tengan una reputación baja. Para ello podrá consultar un listado de todos los usuarios registrados en el sistema ordenados por su reputación.

2.2 Requisitos no funcionales

El sistema estará formado por una parte cliente y una parte servidora.

La parte servidora:

- Contará con una base de datos relacional para almacenar la información necesaria para funcionar.
- Implementará un API REST desarrollado en Java EE para recuperar y manejar los recursos de la base de datos. El intercambio de información entre la parte cliente y la parte servidora se realizará empleando JSON como formato.
- El acceso al API REST debe de ser seguro para evitar accesos no deseados.

La parte cliente:

- Servirá de interfaz gráfico al usuario y realizará las peticiones necesarias al API REST con el fin de obtener o insertar recursos en la base de datos.
- Aplicación web compatible, ordenadores, tabletas, moviles.
- Se implementará una aplicación cliente web en el alcance de este TFG.

2.3 Actores

Teniendo en cuenta los requisitos del sistema se identifican los siguientes actores que emplearán la aplicación:

Ilustración 6: Actores que emplearán la aplicación

2.3.1 Usuario

- Hará uso de la plataforma de forma anónima, por lo que solamente podrá realizar búsquedas y consultar que juegos y consolas hay disponibles a la venta.
- El usuario podrá registrarse en el sistema.

2.3.2 Usuario registrado

Hará uso de la plataforma una vez se haya registrado en el sistema. Podrá realizar las siguientes acciones:

- Autenticarse
- Realizar búsquedas de consolas y videojuegos
- Poner en juegos y consolas a la venta
- Marcar como favoritos los videojuegos y consolas que considere más interesantes para su posterior compra
- Realizar una petición de compra a otro usuario.
- Atender una petición de compra que le haya realizado un usuario.
- Valorar a un usuario al finalizar una transacción de compraventa con él.

2.3.3 Administrador

El administrador es un usuario registrado (podrá realizar las mismas acciones) que, además, será el encargado de crear las categorías donde los demás usuarios registrados podrán añadir sus juegos y consolas para vender. Tendrá acceso a un panel de control de administrador donde podrá crear y consultar las categorías.
También se encargará de bloquear a los usuarios que hayan recibido malas valoraciones y tengan por tanto mala reputación.
Esto evitará que los usuarios honestos se encuentren con malos vendedores.

2.4 Historias de usuario

Dentro del marco del desarrollo ágil, encontramos las siguientes historias de usuario que procedemos a enumerar a continuación:

Búsqueda de videojuegos y consolas (HU01)

Como <*usuario*> quiero <*poder realizar búsquedas de videojuegos y consolas en la plataforma*> para <*encontrar los que más me interesan.*>

Registro de usuarios (HU02)

Como <*usuario*> quiero <*poder registrarme y tener una cuenta con mi correo electrónico y mi contraseña en el sistema*> para <*poder comprar y vender en la plataforma.*>

Gestión de información de usuarios (HU03)

Como <*usuario*> quiero <*poder modificar mis datos personales en el sistema*> para <*que se encuentren actualizados.*>

Gestión de videojuegos y consolas (HU04)

Como <*usuario registrado*> quiero <*gestionar en el sistema mis videojuegos y consolas*> para <*poder ponerlos a la venta.*>

Gestión de favoritos (HU05)

Como <*usuario registrado*> quiero <*gestionar como favoritos aquellos productos que me hayan gustado*> para <*poder encontrarlos rápidamente en una búsqueda posterior*>.

Petición de compra (HU06)

Como <*usuario registrado*> quiero <*poder realizar una petición de compra por un producto a otro usuario*> para <*realizar una compra*>

Gestión de peticiones de compra (HU07)

Como <*usuario registrado*> quiero <*poder aceptar o rechazar una petición de compra que me haya realizado otro usuario*> para <*finalizar una compraventa*>

Valoración de usuarios (HU08)

Como <*usuario registrado*> quiero <*poder valorar a un usuario al finalizar una compraventa*> para <*registrar como ha ido la transacción*>

Gestión de categorías (HU09)

Como <*administrador*> quiero <*poder gestionar las categorías que aparecerán en el sistema*> para <*que lo usuarios registrados puedan dar de alta sus productos*>

Búsqueda de usuarios (HU10)

Como <*administrador*> quiero <*poder consultar un listado de los usuarios registrados en el sistema ordenados por su reputación*> para <*poder consultar cuales de ellos tienen mala reputación*>

Bloquear usuarios (HU11)

Como <*administrador*> quiero <*poder bloquear a un usuario que tiene una reputación baja*> para <*que no pueda volver a acceder al sistema*>.

2.5 Casos de uso

Encontramos los casos de uso siguientes en función del actor que accederá a la aplicación:

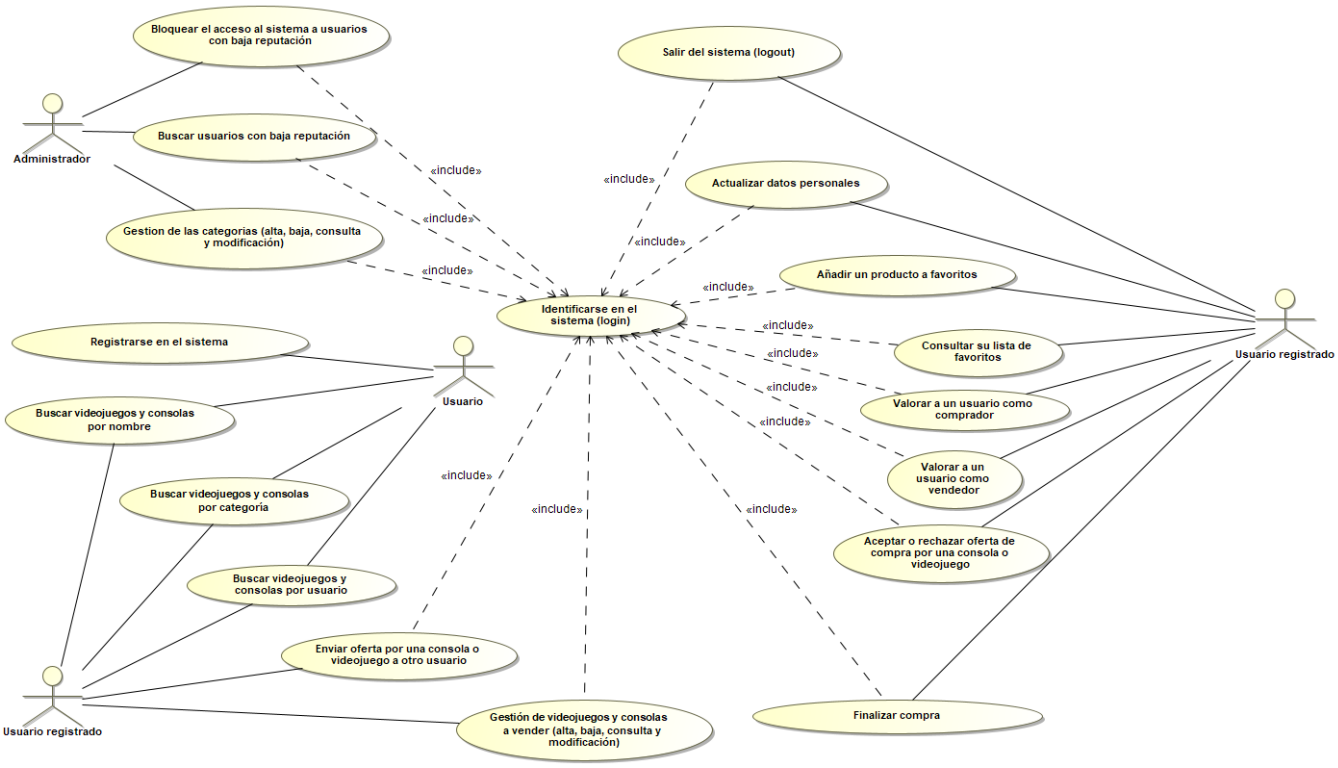


Ilustración 7: Diagrama de casos de uso

Se describen los casos de uso de las funcionalidades requeridas:

CU01	**Buscar videojuegos y consolas por nombre**
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por su nombre
Precondiciones	
Postcondición	Se muestra un listado con los elementos que contengan en el nombre el texto introducido.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario introduce en el campo de búsqueda el nombre del juego o consola a buscar 3. El usuario pulsa en el icono de la lupa para comenzar la búsqueda 4. El sistema muestra una lista con los videojuegos y consolas que contienen en texto introducido en su nombre.
Escenario alternativo	4a. El sistema no encuentra ningún producto que coincida con el texto introducido y muestra un mensaje advirtiéndolo.
CU02	**Buscar videojuegos y consolas por categoría**
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por su categoría
Precondiciones	
Postcondición	Se muestra un listado con los elementos que pertenezcan a la categoría introducido.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario hace clic en Todas las categorías 3. El sistema muestra una la lista desplegable con todas las categorías 4. El usuario selecciona la categoría que desea. 5. El sistema muestra una lista con los videojuegos y consolas que pertenecen a esa categoría.
Escenario alternativo	5a El sistema no encuentra ningún producto en la categoría seleccionada y muestra un mensaje advirtiéndolo.

CU03	Buscar videojuegos y consolas por usuario
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por el usuario al que pertenece.
Precondiciones	
Postcondición	Se muestra un listado con los elementos que pertenezcan al usuario introducido.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario introduce en el campo de búsqueda el nombre del usuario (correo electrónico) por el que desea buscar 3. El usuario pulsa en el icono de la lupa para comenzar la búsqueda 4. El sistema muestra una lista con los videojuegos y consolas que contienen pertenecen a usuario introducido.
Escenario alternativo	4a El sistema no encuentra ningún videojuego o consola asociado al usuario introducido y muestra un mensaje advirtiéndolo.

CU04	Enviar una oferta por una consola o videojuego a otro usuario
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario enviar una petición de oferta por un videojuego o consola a otro usuario
Precondiciones	El usuario debe de estar registrado en el sistema El usuario debe de estar autenticado en el sistema El videojuego o consola no ha de estar reservado
Postcondición	Se crea una petición en el sistema que deberá de ser aceptada o rechaza por el usuario al que ha sido enviada.
Escenario básico	1. El usuario realiza una búsqueda para localizar el videojuego o consola que desea. 2. El sistema le muestra el videojuego o consola que quiere el usuario. 3. El usuario hace clic en el botón enviar petición de oferta. 4. El sistema genera una petición de oferta en el sistema.
Escenario alternativo	3a El usuario regresa a la pantalla inicial 3a1 Finaliza el caso de uso

CU05	Alta de videojuegos y consolas a vender
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario dar de alta un videojuegos o consola en el sistema.
Precondiciones	El usuario debe de estar registrado previamente. El usuario debe de estar autenticado en el sistema.
Postcondición	Se crea realiza una operación en el sistema con el videojuego o consola del usuario.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario selecciona la opción “Mi cuenta” 3. El sistema muestra la pantalla con las opciones de la cuenta del usuario. 4. El usuario selecciona la opción mis juegos. 5. El sistema muestra la pantalla de gestión de videojuegos y consolas del usuario. 6. El usuario selecciona la opción de nuevo. 7. El sistema muestra un formulario para introducir la información del videojuego. 8. El usuario cumplimenta la información y pulsa guardar. 9. El sistema añade el videojuego al sistema.
Escenario alternativo	7a El usuario no introduce ninguna información y pulsa guardar 7a1 El sistema muestra un mensaje de error indicando que no se ha introducido ningún dato. 8a El usuario pulsa cancelar 8a1 El sistema regresa a la pantalla de mis juegos

CU06	Modificar videojuegos y consolas a vender
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario modificar un videojuegos o consola en el sistema.
Precondiciones	El usuario debe de estar registrado previamente. El usuario debe de estar autenticado en el sistema.
Postcondición	Se crea realiza una operación en el sistema con el videojuego o consola del usuario.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario selecciona la opción “Mi cuenta” 3. El sistema muestra la pantalla con las opciones de la cuenta del usuario. 4. El usuario selecciona la opción mis juegos. 5. El sistema muestra la pantalla de gestión de videojuegos y consolas del usuario con el listado de los productos que el usuario ha dado de alta. 6. El usuario selecciona en la lista el videojuego o consola que desea modificar. 7. El usuario pulsa en el enlace modificar. 8. El sistema muestra un formulario para introducir la información del videojuego. 9. El usuario cumplimenta la información y pulsa guardar. 10. El sistema modifica el videojuego al sistema. 11. El sistema muestra de nuevo la pantalla de gestión de videojuegos y consolas del usuario.
Escenario alternativo	9a El usuario pulsa cancelar 9a1 El sistema regresa a la pantalla de mis juegos

CU07	Registrarse en el sistema
Actores	Usuario
Objetivo	Permite al usuario registrarse en la aplicación y crear una nueva cuenta.
Precondiciones	No puede existir una cuenta con el mismo correo electrónico.
Postcondición	Se crea la cuenta de usuario y la aplicación redirigirá al usuario a la pantalla para administrar sus datos personales.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario selecciona la opción “Iniciar Sesión” 3. El sistema muestra la pantalla de inicio de sesión 4. El usuario selección la opción “Regístrate” 5. El sistema muestra la pantalla de registro. 6. El usuario cumplimenta la información de registro y pulsa “Registrarse” 7. El sistema crea la cuenta y muestra la pantalla con la cuenta del usuario.

CU08	Identificarse en el sistema
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario acceder a las funcionales privadas solo accesibles por un usuario registrado.
Precondiciones	El usuario debe de estar registrado previamente.
Postcondición	El usuario accede y tiene la visibilidad de su cuenta y puede acceder a las opciones limitadas solamente a los usuarios registrados.
Escenario básico	1. El sistema muestra la pantalla principal 2. El usuario selecciona la opción “Iniciar Sesión” 3. El sistema muestra la pantalla de inicio de sesión 4. El usuario introduce el nombre de usuario (correo electrónico) y la contraseña y pulsa “Iniciar sesión” 5. El sistema redirige al usuario a la página “Mis datos”.
Escenario alternativo	4a El usuario pulsa en cancelar 4a1 El sistema vuelve a la pantalla de inicio

CU09	Actualizar datos personales
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario actualizar sus datos personales
Precondiciones	El usuario debe de estar registrado previamente. El usuario debe de estar autenticado en el sistema.
Postcondición	Se actualizan los datos personales del usuario y la aplicación redirigirá al usuario a su pantalla inicial de la aplicación.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla “Mis datos” con el formulario para introducir los datos personales del usuario. 4. El usuario introduce los datos en el formulario y pulsa en “guardar”. 5. El sistema modifica los datos y muestra al usuario un mensaje por pantalla.
Escenario alternativo	4a El usuario pulsa en cancelar 4ª1 El sistema regresa a la pantalla Mis Datos y no modifica la información

CU10	Salir del sistema
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario finalizar la sesión del usuario en el sistema.
Precondiciones	El usuario debe de encontrarse autenticado en el sistema.
Postcondición	El usuario deba de estar autenticado en el sistema.
Escenario básico	1. Desde la pantalla de inicio del sistema estando autenticado. 2. El usuario debe seleccionar “Cerrar Sesión” 3. El sistema cierra la sesión y muestra la pantalla de inicio.

CU11	Añadir un producto a favorito
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario añadir al listado de sus favoritos un videojuego o consola.
Precondiciones	El usuario debe de encontrarse autenticado en el sistema.
Postcondición	Se añade al listado de favoritos del usuario un videojuego o consola.
Escenario básico	1. El usuario realiza una búsqueda para localizar el videojuego o consola que desea. 2. El sistema le muestra el videojuego o consola que quiere el usuario. 3. El usuario hace clic en el botón añadir a favoritos. 4. El sistema añade el videojuego o consola a la lista de favoritos del usuario.
Escenario alternativo	

CU12	Consultar lista de favoritos
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario consultar el listado de los favoritos asociados a su cuenta.
Precondiciones	El usuario debe de encontrarse autenticado en el sistema.
Postcondición	El sistema muestra un listado con los videojuegos y consolas favoritos del usuario.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Favoritos”. 5. El sistema muestra un listado con los videojuegos o consolas favoritas
Escenario alternativo	5a El usuario no tiene ningún producto favorito y el sistema muestra un mensaje advirtiéndolo.

CU13	Valorar a un usuario como comprador
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario valorar a otro usuario como comprador al finalizar una compraventa.
Precondiciones	La compraventa debe de estar finalizada.
Postcondición	Se actualiza la reputación del usuario valorado en el sistema.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Valoraciones”. 5. El sistema muestra un listado con las valoraciones pendientes de realizar. 6. El usuario selecciona la valoración que desea cumplimentar y pulsa en valorar. 7. El sistema muestra una ventana para introducir la puntuación. 8. El usuario pulsa en guardar. 9. El sistema actualiza la reputación del usuario.
Escenario alternativo	6a El usuario pulsa en cancelar 6a1 El sistema regresa a la pantalla con la lista de valoraciones pendientes de valorar.

CU14	Valorar a un usuario como vendedor
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario valorar a otro usuario como vendedor al finalizar una compraventa.
Precondiciones	La compraventa debe de estar finalizada.
Postcondición	Se actualiza la reputación del usuario valorado en el sistema.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Valoraciones”. 5. El sistema muestra un listado con las valoraciones pendientes de realizar. 6. El usuario selecciona la valoración que desea cumplimentar y pulsa en valorar. 7. El sistema muestra una ventana para introducir la puntuación. 8. El usuario pulsa en guardar. 9. El sistema actualiza la reputación del usuario.
Escenario alternativo	

CU15	Aceptar / rechazar una oferta por una consola o videojuego
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario aceptar una petición de oferta por un videojuego o consola enviada por otro usuario
Precondiciones	El videojuego o consola no ha de estar reservado
Postcondición	Se crea una petición en el sistema que deberá de ser aceptada o rechaza por el usuario al que ha sido enviada.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Solicitudes de compra”. 5. El sistema muestra un listado con las solicitudes pendientes de aceptar, rechazar o finalizar. 6. El usuario selecciona la petición que desea aceptar o rechazar. 7. El usuario pulsa en aceptar y el sistema marca el videojuego como reservado. 8. El sistema envía un correo al usuario que envió la petición con la información de contacto del vendedor.
Escenario alternativo	

CU16	Finalizar compra
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario finalizar la venta de un videojuego o consola a otro usuario
Precondiciones	El videojuego ha de encontrarse reservado.
Postcondición	Se actualiza el estado del videojuego ha vendido.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Solicitudes de compra”. 5. El sistema muestra un listado con las solicitudes pendientes de aceptar, rechazar o finalizar. 6. El usuario selecciona la petición que desea finalizar. 7. El usuario pulsa en finalizar y el sistema marca el videojuego como vendido. 8. El sistema envía un correo al usuario que envió la petición con la información de contacto del vendedor. 9. El sistema oculta el videojuego o consola para que no aparezca en las búsquedas de los demás usuarios.
Escenario alternativo	

CU17	Buscar usuarios con baja reputación
Actores	Administrador
Objetivo	Permite al administrador consultar un listado de los usuarios registrados en el sistema ordenados por reputación.
Precondiciones	
Postcondición	El sistema muestra un listado con los usuarios y su reputación.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de la cuenta del del usuario. 4. El usuario selecciona la opción en “Usuarios”. 5. El sistema muestra un listado con los usuarios y su reputación. 6. El usuario selecciona el usuario que desea bloquear. 7. El usuario pulsa en bloquear. 8. El sistema bloquea la cuenta del usuario seleccionado.
Escenario alternativo	5a No hay usuarios registrados en el sistema y se muestra un mensaje advirtiéndolo.

CU18	Gestión de las categorías (alta, consulta y modificación)
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario enviar una petición de oferta por un videojuego o consola a otro usuario
Precondiciones	El videojuego o consola no ha de estar reservado
Postcondición	Se crea una petición en el sistema que deberá de ser aceptada o rechaza por el usuario al que ha sido enviada.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El sistema muestra la pantalla con la información de su cuenta. 4. El usuario selecciona la opción categorías. 5. El sistema muestra un listado con las categorías existentes. 6. El usuario pulsa en categoría nueva. 7. El sistema muestra un formulario para cumplimentar la información de la categoría. 8. El usuario cumplimenta la información y pulsa en guardar. 9. El sistema guarda la categoría en el sistema 10. El sistema regresa a la pantalla con el listado de categorías.
Escenario alternativo	5a El sistema no tiene categorías y muestra un mensaje advirtiéndolo 8a El usuario no introduce ningún dato y pulsa guardar 8a1 El sistema muestra un mensaje de error

CU19	Enviar una oferta por un videojuego/consola a otro usuario
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario enviar una petición de oferta por un videojuego o consola a otro usuario
Precondiciones	El videojuego o consola no ha de estar reservado
Postcondición	Se crea una petición en el sistema que deberá de ser aceptada o rechaza por el usuario al que ha sido enviada.
Escenario básico	1. El sistema muestra la pantalla de inicio 2. El usuario inicia sesión. 3. El busca el producto que desea comprar 4. El usuario selecciona enviar oferta 5. El sistema enviar una oferta al propietario

3. Diseño

En este capítulo se identifican las entidades y sus relaciones mediante el modelado de un diagrama de clases, se define el modelo de la base de datos y se detalla el diseño del servicio REST

3.1 Diagrama de clases

En base a los requisitos y desde el punto de vista de la información obtenemos el siguiente diagrama de clases:

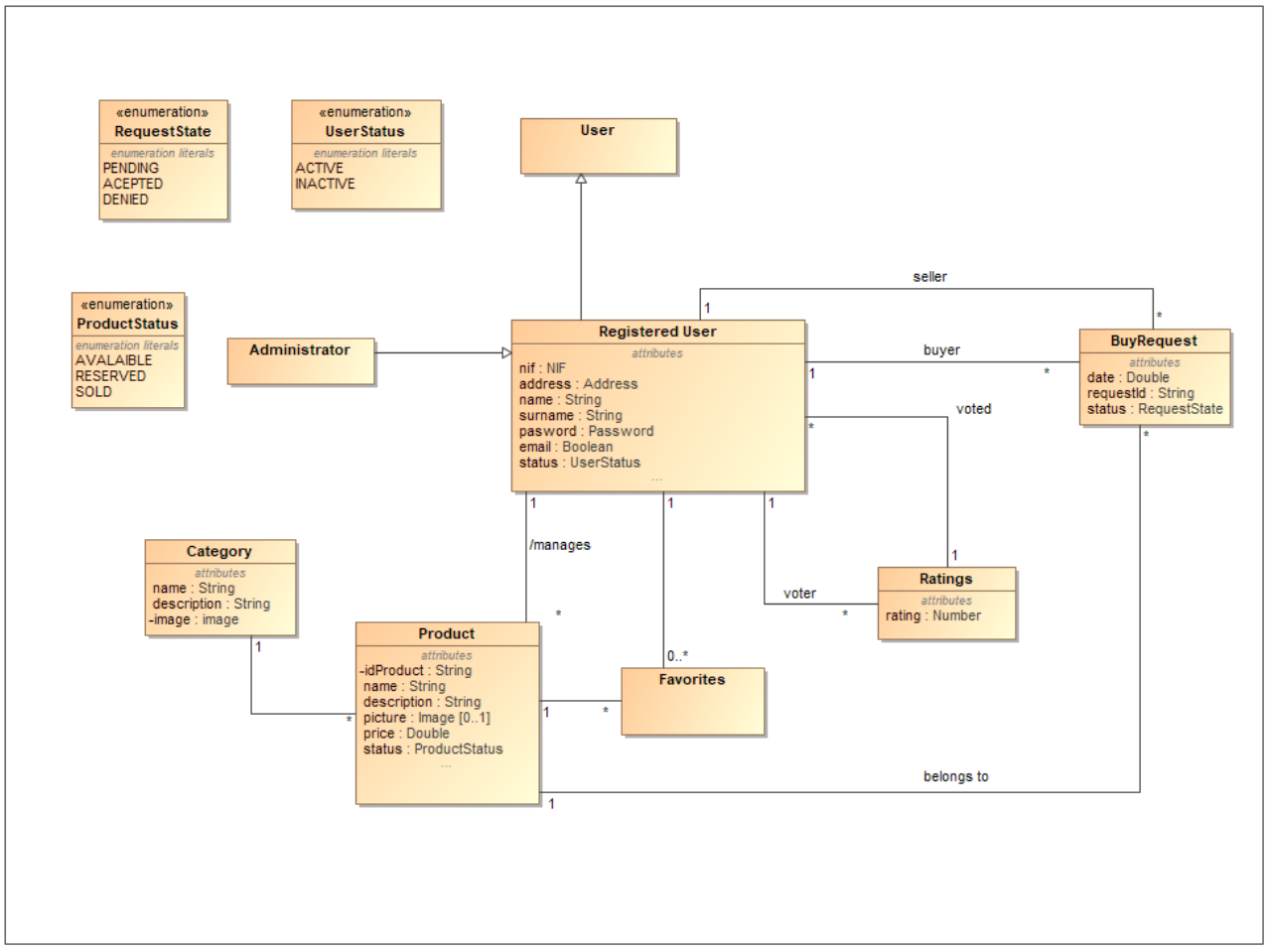


Ilustración 8: Diagrama de clases del sistema

3.2 Base de datos

A partir de las entidades determinadas en el diagrama de clases anterior, se obtienen las siguientes tablas para la base de datos:

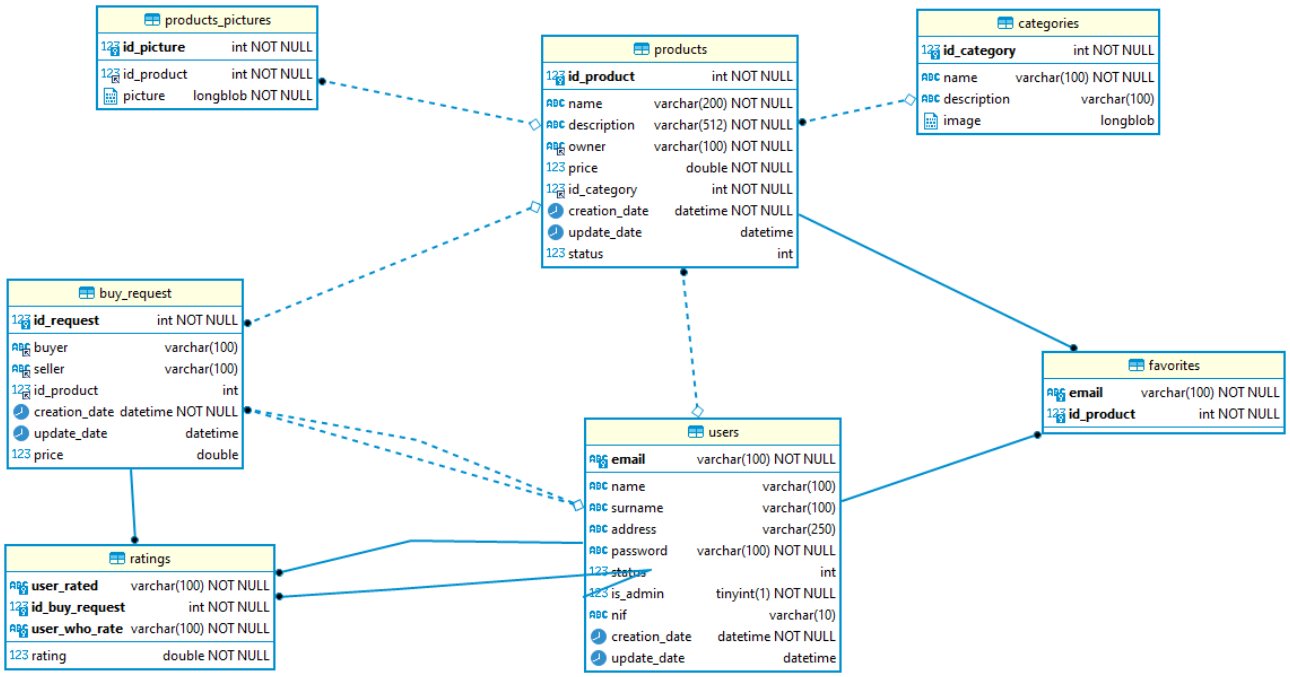


Ilustración 9: Diagrama de base de datos

categories: Almacenará las categorías creadas por el administrador en el sistema.

Campos:

- *id_categoy* (PK): Identificador de la categoría
- *name*: Nombre de la categoría
- *description*: Descripción de la categoría
- *image*: imagen representativa de la categoría (logotipo del sistema)

users: Se empleará para almacenar la información de los usuarios registrados y administradores.

Campos:

- email (PK): Correo electrónico que a su vez servirá para identificar al usuario en el sistema
- name: Nombre del usuario
- surname: Apellidos del usuario
- address: Dirección del usuario
- password: Contraseña de acceso del usuario al sistema
- status: Estado de la cuenta del usuario. Indica si esta activa o por el contrario ha sido bloqueada por el administrador por puntuaciones bajas.
- is_admin: Indica si el usuario registrado es además administrador.
- nif: Número de identificación fiscal del usuario
- creation_date: fecha en la que el usuario ser registro
- update_date: fecha en la que el usuario actualizo sus datos

products: Tabla para almacenar los videojuegos y consolas de los usuarios

Campos:

- id_product (PK): Identificador del videojuego o consola
- name: Nombre del videojuego o consola
- description: Descripción del videojuego o consola
- owner: Correo electrónico del propietario.
- id_category: Identificador de la categoría a la que pertenece.
- creation_date: Fecha en el que se da de alta el producto.
- status: estado en el que se encuentra el producto. Disponible, reservado o vendido.
- update_date: Fecha en el que el producto cambia de estado

products_pictures: Tabla para almacenar las imágenes de los videojuegos y consolas que se pondrán a la venta.

Campos:

- id_picture (PK): Identificador de la imagen.
- id_product: Identificador del producto al que pertenece.
- picture: Imagen del producto

buy_request: En ella se insertarán las peticiones de compras.

Campos:

- id_request (PK): Identificador de la petición de compra
- buyer: Correo electrónico del comprador que emite la petición
- seller: Correo electrónico del vendedor del producto que recibe la petición
- id_product: Identificador del producto que se va a comprar.
- price: Precio que el comprado propone al vendedor.
- creation_date: Fecha en el que se realiza la petición
- update_date: Fecha en el que la petición cambia de estado

favorites: Tabla donde se guardarán los productos favoritos de un usuario para poder posteriormente comenzar una compraventa.

Campos:

- email (PK): Correo electrónico del usuario
- productId(PK): Identificador del producto favorito.

ratings: Tabla donde se guardarán las valoraciones entre usuarios al finalizar una compraventa.

Campos:

- user_rated (PK): Correo electrónico del usuario valorado
- id_buy_request: Identificador de la petición de compra por la que recibe la valoración
- user_who_rate: Correo electrónico del usuario que emite la valoración
- rating: Valoración del usuario de 0 al 5.

3.3 Diseño de la API REST

La implementación del API permite la comunicación entre el cliente y el servidor usando las solicitudes HTTP descritas por el Protocolo RFC 2616. [23]

- GET: Solicita la información de uno o varios recursos al servidor.
- POST: Crea un nuevo recurso en el servidor. POST envía la información sobre el recurso a crear en el cuerpo del mensaje, en formato JSON.
- PUT: Actualizar recursos en el servidor. La nueva información del recurso se envía en el cuerpo del mensaje HTTP y el servidor informará al cliente que el recurso ha sido actualizado correctamente.
- DELETE: Se emplea para eliminar un recurso especificado.

La API REST implementada para este proyecto devolverá la representación de los recursos en formato JSON.

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

category	/category/{id_category}	GET	HTTP 200 OK
			{
			categoryId: value,
			name: “value”,
			description: “value”,
			image: “base64file”
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST

category	/category
	
	body
	{
	name: “value”,
	description: “value”,
	image: “base64file”
	}
		POST	HTTP 200 OK
			{
			id_category: value,
			name: “value”,
			description: “value”,
			image: “base64file”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

category	/category/{id_category}
	
	body
	{
	name: “value”,
	description: “value”,
	image: “base64file”
	}
		PUT	HTTP 200 OK
			{
			name: “value”,
			description: “value”,
			image: “base64file”
			}
			
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

categories	/categories	GET	HTTP 200 OK
			{
			categories: {
			(array)
			}
			}
			
			HTTP 404 NOT FOUND
			HTTP 403 FORBIDDEN

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

user	/user/{email}	GET	HTTP 200 OK
			{
			email: “value”,
			name: “value”,
			surname: “value”,
			address: “value”,
			rating: “value”,
			nif: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

user	/register
	{
	email: “value”,
	name: “value”,
	surname: “value”,
	address: “value”,
	password: “value”,
	nif: “value”,
	status: “value”
	}
		POST	HTTP 200 OK
			{
			email: “value”,
			token: “value”,
			creation_date: “value”,
			update_date: “value”,
			expirateAt: “value”
			isAdmin: “value”
			}
			
			HTTP 400 BAD REQUEST

user	/login
	{
	email: “value”,
	password: “value”,
	}
		POST	HTTP 200 OK
			{
			email: “value”,
			token: “value”,
			creation_date: “value”,
			update_date: “value”,
			expirateAt: “value”
			isAdmin: “value”
			}
			HTTP 401 UNAUTHORIZED

user	/user/{email}
	
	{
	email: “value”
	name: “value”,
	surname: “value”,
	address: “value”,
	password: “value”,
	nif: “value”,
	status: “value”
	}
		PUT	HTTP 200 OK
			{
			email: “value”,
			name: “value”,
			surname: “value”,
			address: “value”,
			rating: “value”,
			nif: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

users	/users	GET	HTTP 200 OK
			{
			users: {
			(array)
			}
			}
			HTTP 403 FORBIDDEN

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

product	/product/{id_product}	GET	HTTP 200 OK
			{
			productId: “value”,
			name: “value”,
			description: “value”,
			owner: “value”,
			price: “value”,
			category: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”,
			pictureList: []
			
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST

product	/product/
	
	{
	name: “value”,
	description: “value”,
	owner: “value”,
	price: “value”,
	category: “value”,
	pictureList:[]
	status: “value”
	}
		POST	HTTP 200 OK
			{
			id_product: “value”,
			name: “value”,
			description: “value”,
			owner: “value”,
			price: “value”,
			id_category: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

product	/product/{id_product}
	
	{
	name: “value”,
	description: “value”,
	owner: “value”,
	price: “value”,
	category: “value”,
	pictureList:[]
	status: “value”
	}
		PUT	HTTP 200 OK
			{
			productId: “value”,
			name: “value”,
			description: “value”,
			owner: “value”,
			price: “value”,
			category: “value”,
			pictureList:[],
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

products	/products	GET	HTTP 200 OK
			{
			products: {
			(array)
			}
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

products	/products/{type}/{value}
	
	type enum
	
	user,
	category,
	name
	
	value = “valor filtro”
		GET	HTTP 200 OK
			{
			products: {
			(array)
			}
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

favorites	/favorites/{email}	GET	HTTP 200 OK
			{
			products: {
			(array)
			}
			}
			
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

favorite	/favorite/
	
	{
	id_product: “value”,
	email: “value”
	}
		POST	HTTP 200 OK
			{
			product:”value”,
			email: “value”
			}
			
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

favorite	/favorite
	{
	id_product: “value”,
	email: “value”
	}
		DELETE	HTTP 200 OK
			HTTP 404 NOT FOUND
			HTTP 403 FORBIDDEN
			HTTP 400 BAD REQUEST

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

buy_request	/request/{id_request}	GET	HTTP 200 OK
			{
			requestId: “value”,
			buyer: “value”,
			seller: “value”,
			product: “value”,
			price: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”,
			ratings:[]
			}
			
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

buy_requests	/requests/seller/{email}	GET	HTTP 200 OK
			{
			buy_requests: {
			(array)
			}
			}
			
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

buy_requests	/requests/buyer/{email}	GET	HTTP 200 OK
			{
			buy_requests: {
			(array)
			}
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

buy_requests	/request
	
	{
	id_product: “value”,
	email: “value”
	price: “value”
	}
		POST	HTTP 200 OK
			{
			id_request: “value”,
			buyer: “value”,
			seller: “value”,
			id_product: “value”,
			price: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

buy_requests	/request/{id_request}
	
	{
	update_date: “value”
	status: “value”
	}
		PUT	HTTP 200 OK
			{
			id_request: “value”,
			buyer: “value”,
			seller: “value”,
			id_product: “value”,
			price: “value”,
			creation_date: “value”,
			update_date: “value”
			status: “value”
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS

rating	/rating/{email}	GET	HTTP 200 OK
			{
			email: “value”,
			rating: “value”
			}
			HTTP 404 NOT FOUND
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

rating	/rating/{email}
	
	{
	id_buy_request: “value”,
	user_who_rate: “value”
	rating: “value”
	}
		POST	HTTP 200
			{
			email:”value”,
			rating: “value”
			}
			HTTP 400 BAD REQUEST
			HTTP 403 FORBIDDEN

4. Prototipo

A continuación, se muestran las pantallas del prototipo de la parte cliente que interactuaría con la parte servidora del API REST.

4.1 Pantalla de inicio

En la pantalla de inicio encontramos los últimos productos que han sido dados de alta en el sistema por los usuarios. Desde el menú de todas las categorías el usuario puede realizar una búsqueda por una categoría en concreto. Si el usuario emplea el cuadro de texto con la leyenda ¿Qué estas buscando?, el usuario podría comenzar una búsqueda por nombre de producto o por un usuario en concreto.
La parte superior derecha se encuentra el enlace para iniciar sesión. Si el usuario hace clic en el enlace la aplicación le redirige a la pantalla de inicio de sesión.

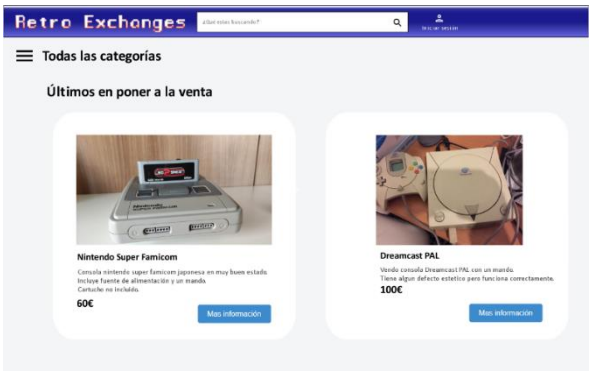


Ilustración 10: Pantalla de inicio

4.2 Inicio de sesión

Desde la ventana de inicio de sesión el usuario puede autenticarse en el sistema o lanzar el proceso de registro en caso de no encontrarse registrado.

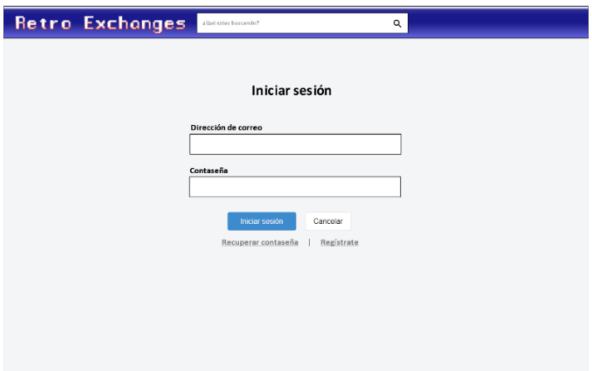


Ilustración 11: Página de inicio de sesión

4.3 Registro del usuario

Desde esta página el usuario podrá introducir sus datos y completar el registro.

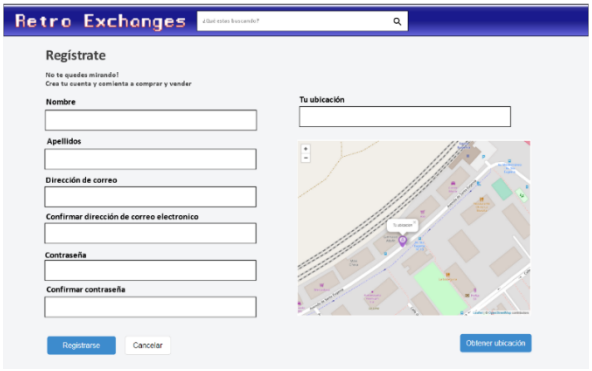


Ilustración 12: Página de registro de usuario

4.4 Cuenta del usuario

Una vez registrado y autenticado, el sistema acceder a la cuenta del usuario donde podrá elegir que hacer.

Cambiar sus datos:

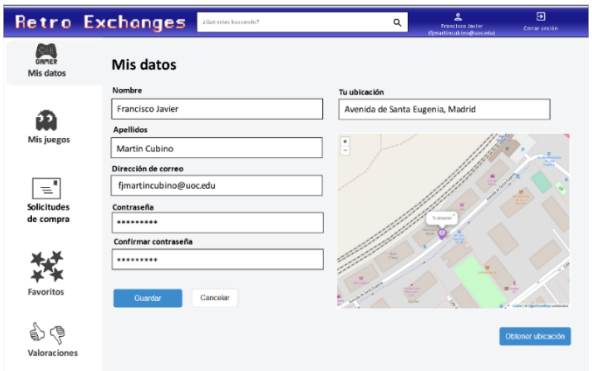


Ilustración 13: Página con los datos del usuario

Dar de alta juegos:

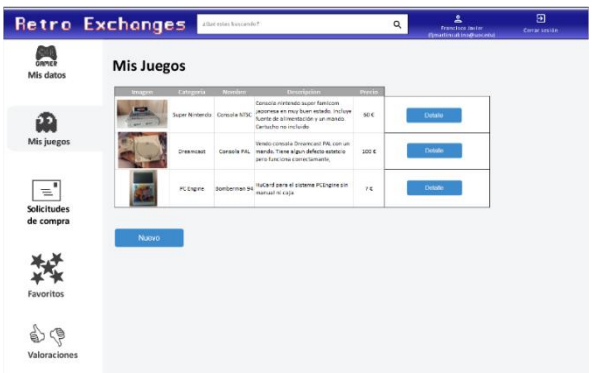


Ilustración 14: Página con los juegos del usuario

Gestionar las solicitudes de compra:

Ilustración 15: Página con las solicitudes de compra

Consultar su listado de favoritos:

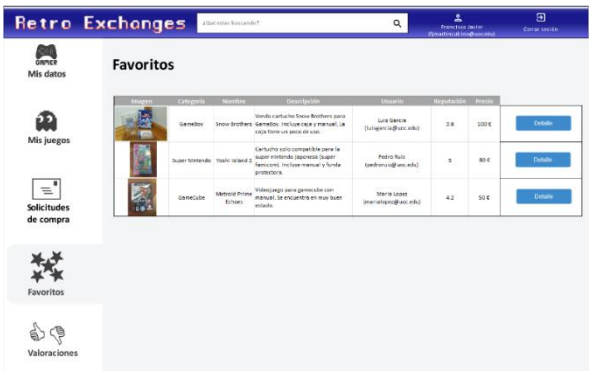


Ilustración 16: Página con los favoritos del usuario

O consultar y realizar las valoraciones pendientes:

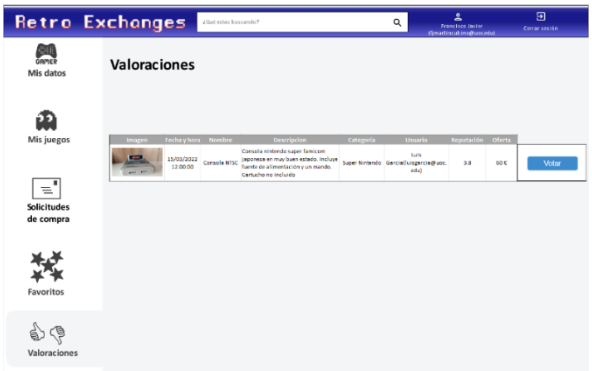


Ilustración 17: Detalle de una posible página de valoraciones

5. Implementación

Finalizado el análisis y diseño, iniciamos la implementación del proyecto con las tecnologías elegidas. Dicha implementación se divide en la implementación del servicio REST y la implementación del cliente. A lo largo del capítulo, se incluirán fragmentos del código para explicar cómo se han estructurado el código y el uso de las tecnologías empleadas.

5.1 Desarrollo del servicio REST

A continuación, se detalla la implementación del servicio REST.

5.1.1 Framework y despliegue

En un principio se valora el emplear WidlFly [26] (JBoss) como servidor de aplicaciones JavaEE, dado que es una solución ya conocida y utilizada durante el grado en diferentes asignaturas.

Dada la complejidad de su puesta en marcha (Servidor, distintos conectores base de datos, incompatibilidades entre diferentes versiones de java, jakarta EE, etc) decido acometer (después de valorar sus numerosas virtudes) el desarrollo con **SpringBoot**, concretamente en su versión 2.6.4. La versión 2.6.5 incluye un bug [8] que impedía el correcto funcionamiento del API-REST al realizas las búsquedas en base de datos.

Para el despliegue del proyecto e instalación de las dependencias se ha empleado **Maven** [19]

Maven es una herramienta open-source, que simplifica los procesos de compilación y de generación de ejecutables a partir del código fuente y de un fichero de configuración.

Mediante el sitio web https://start.spring.io/, se realizó la creación de un proyecto SpringBoot (el sitio genera el fichero pom.xlm para maven). Dicho proyecto tiene por nombre **retroexchanges-rest-service.**

retroexchanges-rest-service incluye las dependencias siguientes:

- spring-boot-starter-parent: Facilita la creación de una aplicación Java.
- spring-boot-starter-data-jpa: Permite el manejo de bases de datos relacionales.
- spring-boot-starter-web: Se emplea para la creación de un sitio web. En este caso el api-rest.
- spring-boot-starter-security: Permite dotar de seguridad al sitio web.

- spring-boot-starter-validation: Facilita el desarrollo del sitio web.
- spring-boot-devtools: Herramientas de desarrollo de spring-boot.
- spring-boot-maven-plugin: Integración de springboot con maven.
- mysql-connector-java: Conector contra la base de datos mysql.
- io.jsonwebtoken: Facilita la gestión y generación de tokens de usuario.

5.1.2 Seguridad basada en tokens JWT

Para la seguridad de API REST, se decide emplear JSON Web Token (a partir de ahora JWT) descrito en el estándar RFC 7519[24]. De esta forma el usuario tras autenticarse contra el API obtiene un token seguro que empleara para cada solicitud posterior incluyéndolo en la cabecera de la petición. También nos permite determinar a qué rutas, servicios y recursos le está permitido acceder al usuario. Para implementar esta funcionalidad se emplea la librería io.jsonwebtoken.

5.1.3 Base de datos

Se emplea el motor de base de datos mysql server. La decisión viene dada por ser una solución con una alta implantación y una amplia comunidad dando soporte.

Junto con la herramienta MySQL Workbench se ha utilizado DBeaver. Como emplear dicha herramienta viene detallado en el anexo [10.5 Creación de la base de datos]

5.1.4 Eclipse

Como editor y depurador se ha utilizado eclipse en su versión 2022-3.

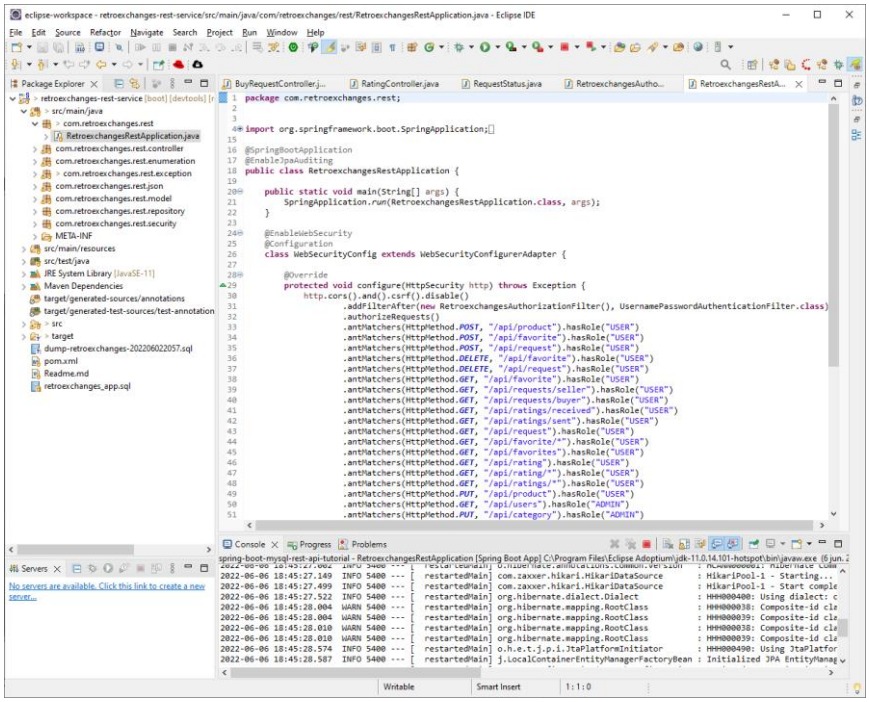


Ilustración 18: Entorno de desarrollo eclipse

5.1.5 Estructura del proyecto

El proyecto consta con una clase principal de aplicación desde donde se inicia el programa

```
Dentro de com.retroexchanges.rest
public class RetroexchangesRestApplication
```

Es también desde esa clase desde donde se define el patrón de seguridad al que tendrán acceso los usuarios en función de su rol:

ROLE_USER para el usuario registrado
ROLE_ADMIN para el usuario con permisos de administrador
PERMIT_ALL. para acceso libre a cualquier usuario

Se ha desactivado el control de acceso **cors** para poder ejecutar en el mismo sistema toda la solución

```
@Override
protected void configure(HttpSecurity http) throws Exception {
         http.cors().and().csrf().disable()
         .addFilterAfter(new RetroexchangesAuthorizationFilter(),
UsernamePasswordAuthenticationFilter.class)
         .authorizeRequests()
         .antMatchers(HttpMethod.POST, "/api/37roducto").hasRole("USER")
         .antMatchers(HttpMethod.POST, "/api/favorite").hasRole("USER")
         .antMatchers(HttpMethod.POST, "/api/request").hasRole("USER")
         .antMatchers(HttpMethod.DELETE, "/api/favorite").hasRole("USER")
         .antMatchers(HttpMethod.DELETE, "/api/request").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/favorite").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/requests/*").hasRole("USER")

         .antMatchers(HttpMethod.GET, "/api/request").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/favorite/*").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/favorites").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/rating").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/rating/*").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/ratings/*").hasRole("USER")
         .antMatchers(HttpMethod.PUT, "/api/37roducto").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/users").hasRole("ADMIN")
         .antMatchers(HttpMethod.PUT, "/api/category").hasRole("ADMIN")
         .antMatchers(HttpMethod.POST, "/api/category").hasRole("ADMIN")
         .antMatchers(HttpMethod.GET, "/api/user").hasRole("USER")
         .antMatchers(HttpMethod.GET, "/api/37roducto/*").permitAll()
         .antMatchers(HttpMethod.GET, "/api/products").permitAll()
         .antMatchers(HttpMethod.GET, "/api/products/user/*").permitAll()
         .antMatchers(HttpMethod.GET, "/api/products/category/*").permitAll()
         .antMatchers(HttpMethod.GET, "/api/products/name/*").permitAll()
         .antMatchers(HttpMethod.POST, "/api/register").permitAll()
         .antMatchers(HttpMethod.GET, "/api/categories").permitAll()
         .antMatchers(HttpMethod.GET, "/api/category/*").permitAll()
         .antMatchers(HttpMethod.POST, "/api/login").permitAll()
         .anyRequest().authenticated();
}
```

Para la implementación de la seguridad mediante la utilización de tokens de usuario se ha definido la clase

Dentro de com.retroexchanges.rest.security
`RetroexchangesAuthorizationFilter`

En esa clase se emplea `io.jsonwebtoken` para la generación de tokens de usuarios "seguros" utilizando una clave de servidor.

Una posible mejora sería la utilización de un servidor oauth2, pero por motivos de tiempo se ha decidido utilizar esta solución.

Cuando el usuario se autentica obtiene un token que caduca a las 24 horas. Empleando el token tiene autorización para realizar las operaciones que su rol le permite (en el token va implícito el rol) evitando la necesidad de emplear usuario y contraseña de nuevo.

Cada tipo de recurso que publica el servidor lleva asociado un controlador (**com.retroexchanges.rest.controller**)

Cada controlador, emplea un repositorio

`com.retroexchanges.rest.repository`

donde ser realizan las operaciones de búsqueda, creación, borrado de cada una de las entidades (`com.retroexchanges.rest.model)`. Estas son:

- Category
- User
- Product
- ProductPicture
- Favorite
- Rating

Para el intercambio de información entre el cliente y el servidor empleando ficheros “json”, se han definido en `com.retroexchanges.rest.json` algunas clases que simplifican ciertas peticiones (en algunos casos las clases son bastante complejas para ser serializadas en un json)

Por último, se han definido una serie de enumeraciones en para facilitar la implementación `com.retroexchanges.rest.enumeration`

5.1.6 Pruebas unitarias del servicio REST

Durante la implementación del servicio REST se han ido realizando pruebas unitarias para comprobar el funcionamiento de los métodos desarrollados. Para ello, se ha utilizado la herramienta PostMan [21], que permite probar todos los métodos y peticiones HTTP de la API personalizada de forma sencilla. Establece la conexión directamente mostrando cabeceras, tanto de petición como de la respuesta, así como el cuerpo de cada petición y respuesta. El resultado de las pruebas puede encontrarse en el anexo [10.1 Pruebas de validación del API REST] de este documento se detalla la batería de pruebas realizadas con esta herramienta.

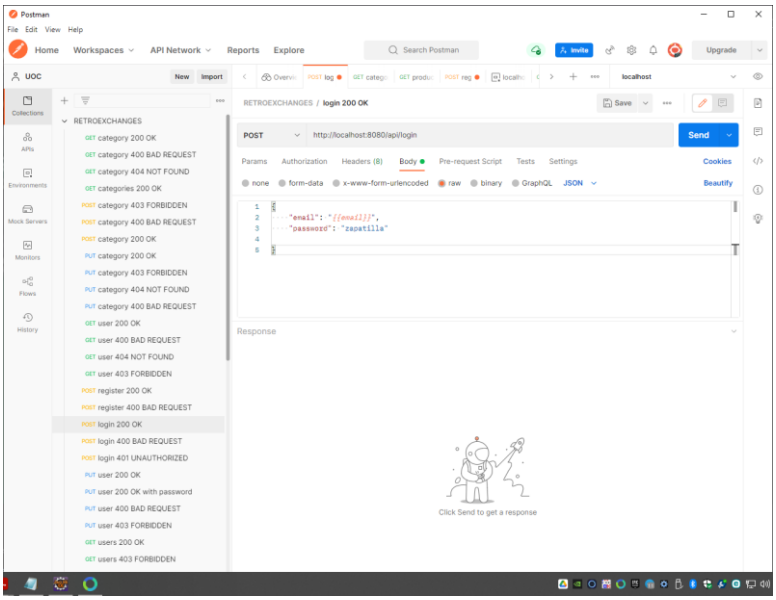


Ilustración 19: Petición de token de autenticación

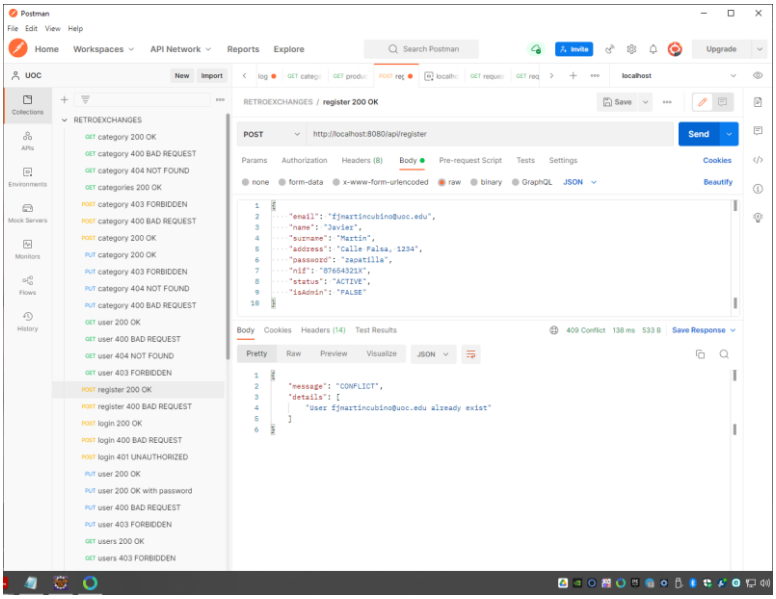


Ilustración 20: Error al registrar porque el usuario ya existe

5.2 Desarrollo del cliente REST

A continuación, se detalla la implementación del cliente.

5.2.1 Framework y despliegue

Para el desarrollo de la parte cliente se ha optado por la utilización del framework **vue.js** con la finalizad de poder testear la aplicación JavaEE con un cliente ligero, ya que la arquitectura propuesta pretende dejar en manos del servicio REST toda la carga pesada facilitando así, la implantación de clientes en cualquier plataforma (escritorio, web, móvil, etc…

Se ha empleado como motor javascript **nodejs**[18] y como framework **vue.js**[25] en su versión 2.

Para el desarrollo de las diferentes vistas he utilizado **bootstrap-vue**[11] en su versión 4.0 lo que ha facilitado ampliamente el poder realizar un cliente en tan poco tiempo.

5.2.2 Visual Studio Code

Como editor he empleado la herramienta de Microsoft Visual Studio Code que facilita la integración tanto con nodejs como con vue.

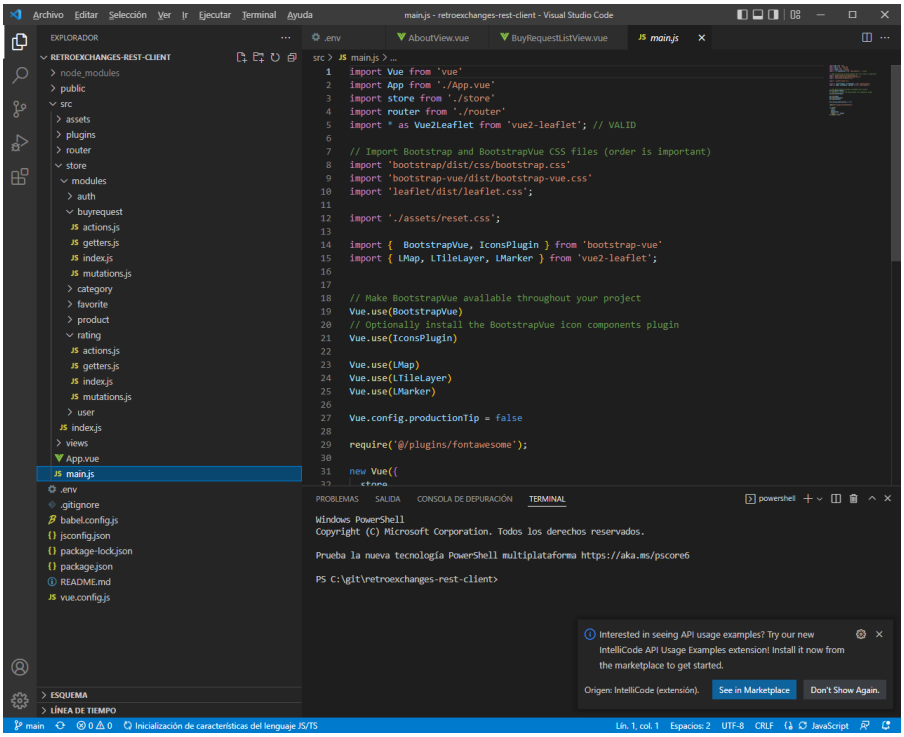


Ilustración 21: Entorno Visual Studio Code

5.2.3 Estructura del proyecto

El proyecto consta con un fichero main.js desde donde se inicia la aplicación y se importan las diferentes dependencias que se emplearan de forma global en todo el proyecto:

- Bootstrap(para la ui)
- Route (para facilitar la navegación por las vistas)
- Leaflet (para mostrar los mapas)
- Vuex (para la almacenar temporalmente los json y la info de sesión)
- Archivos .css con los estilos

Las vistas se encuentran en la carpeta views
Las peticiones que se realizan contra el api están en la carpeta store
En la carpeta route se encuentran configuradas las diferentes rutas de navegación

Finalmente, en plugins y asserts se encuentran los tipos de letra empleados y el logo del sitio web. Se ha empleado como fuente fontawesome (10)

En el fichero .env se encuentran la configuración para conectarse al servidor rest. Por defecto es localhost:8080.

A continuación, se muestran unas capturas del cliente:

Vista principal con los productos a la venta ordenados de más recientes en llegar a más antiguos:

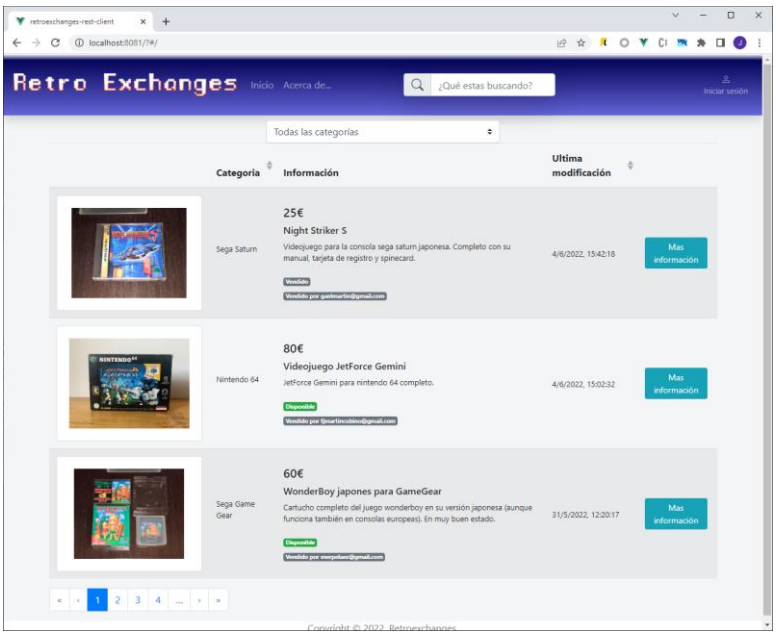


Ilustración 22: Página principal con el listado de los productos a la venta

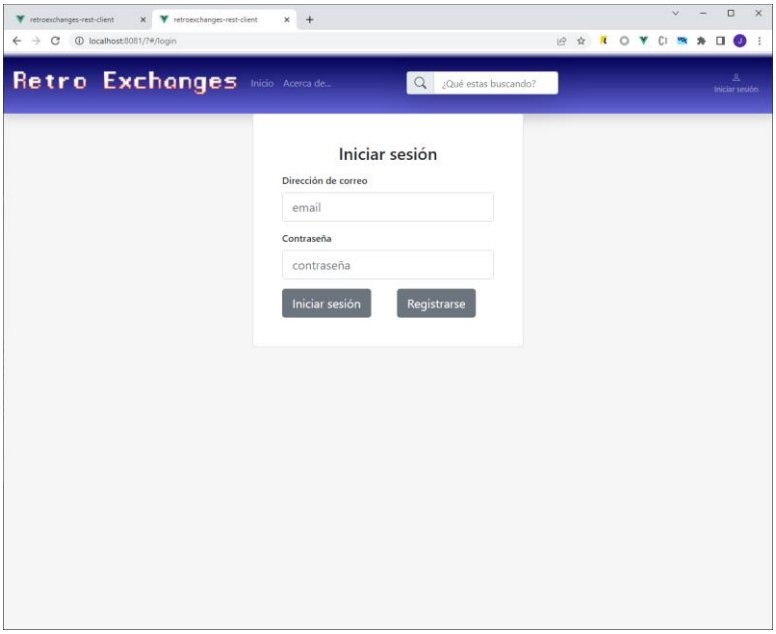


Ilustración 23: Inicio de sesión

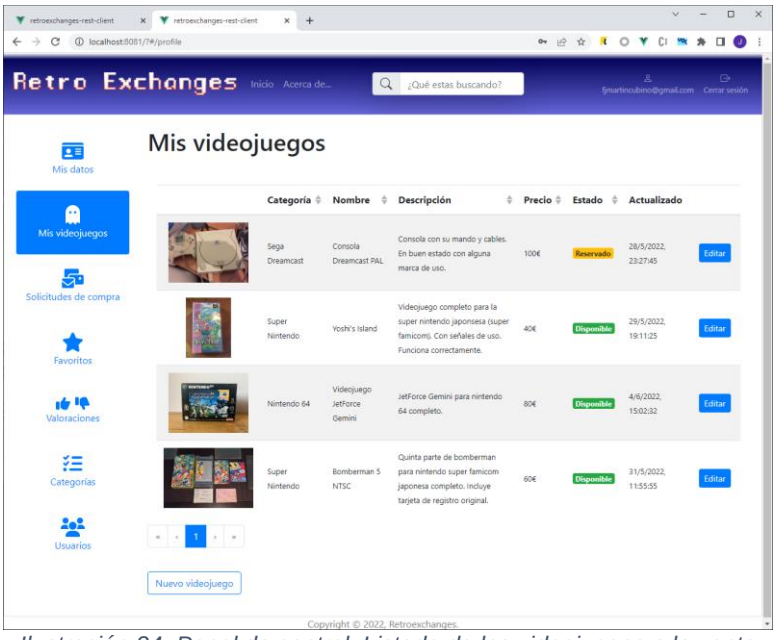


Ilustración 24: Panel de control. Listado de los videojuegos a la venta

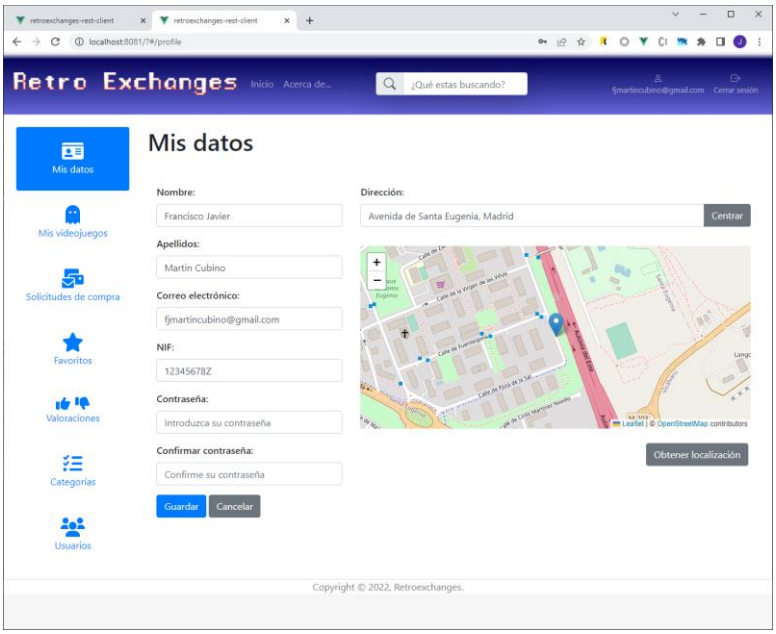


Ilustración 25: Panel de control con los datos del usuario

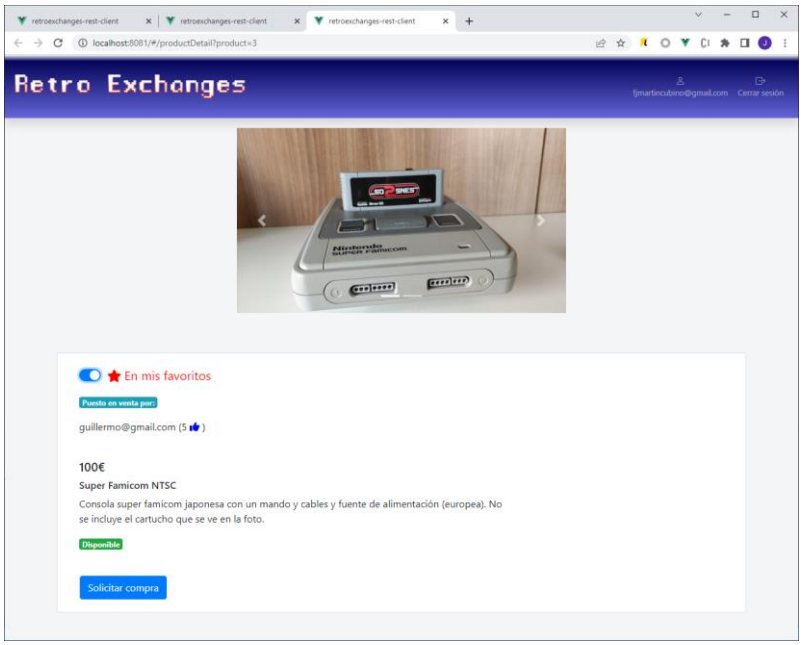


Ilustración 26: Detalle de una consola a la venta

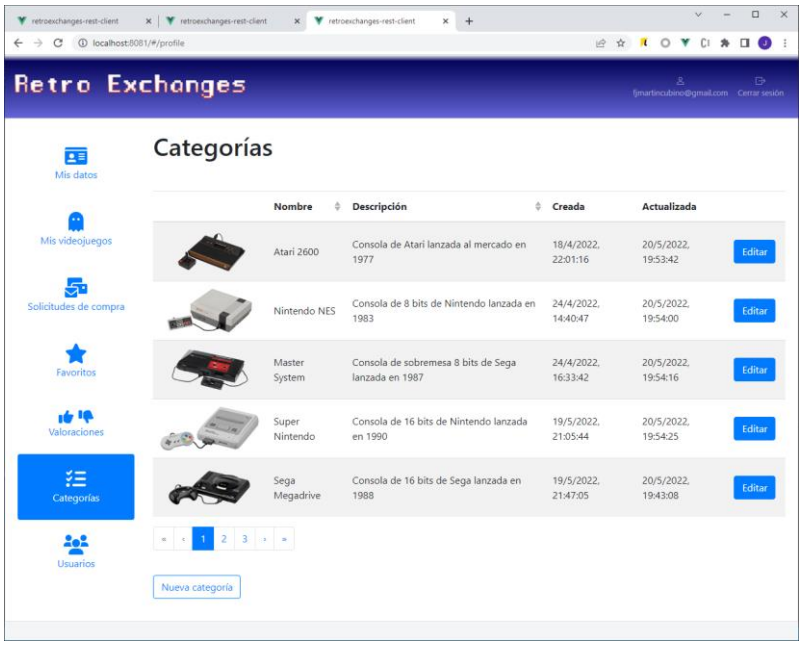


Ilustración 27: Listado de las categorías

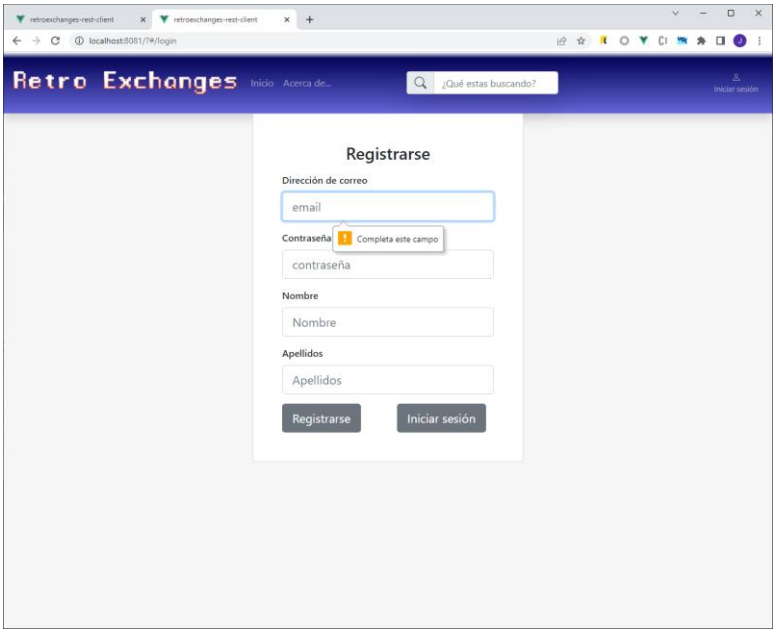


Ilustración 28: Registro de usuario

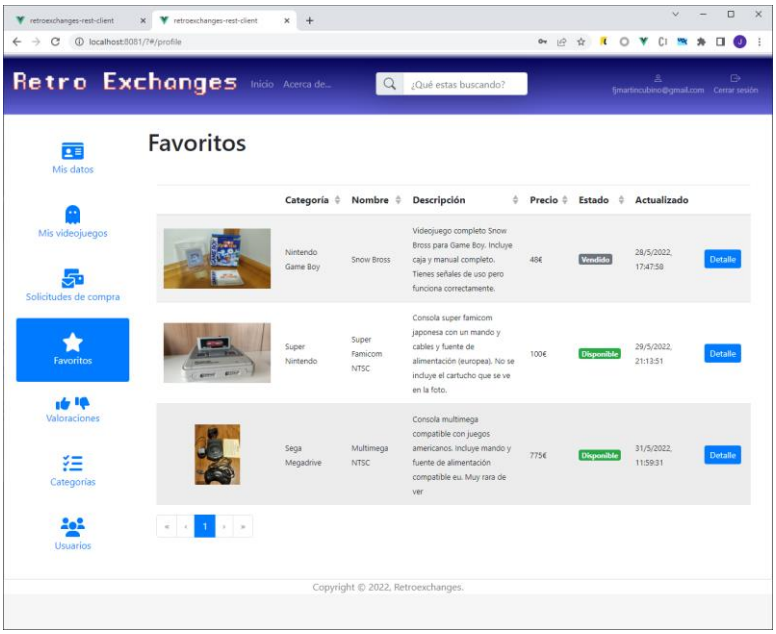


Ilustración 29: Vista de favoritos

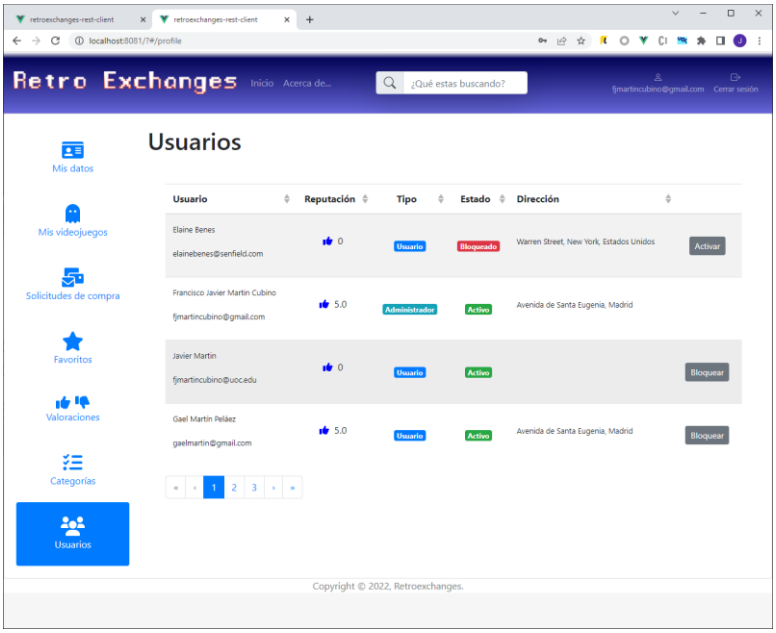


Ilustración 30: Listado de usuarios

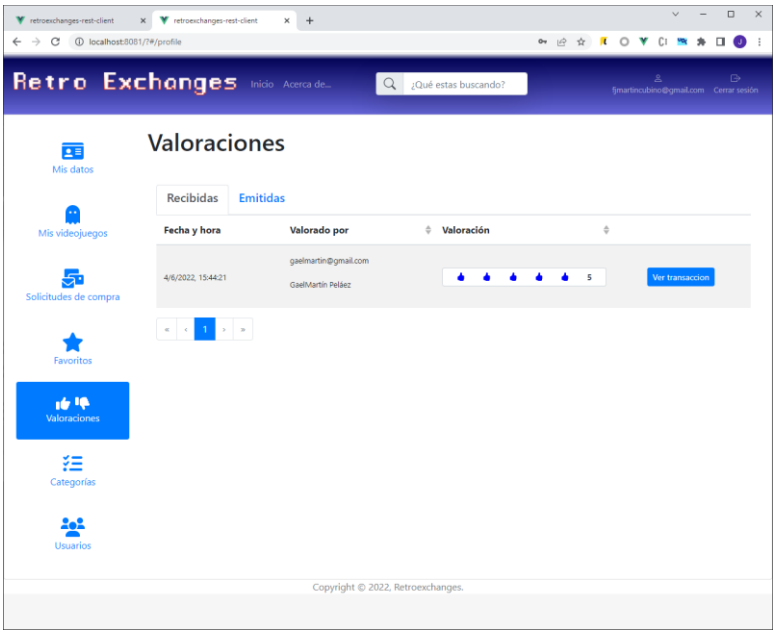


Ilustración 31: Valoraciones de usuario

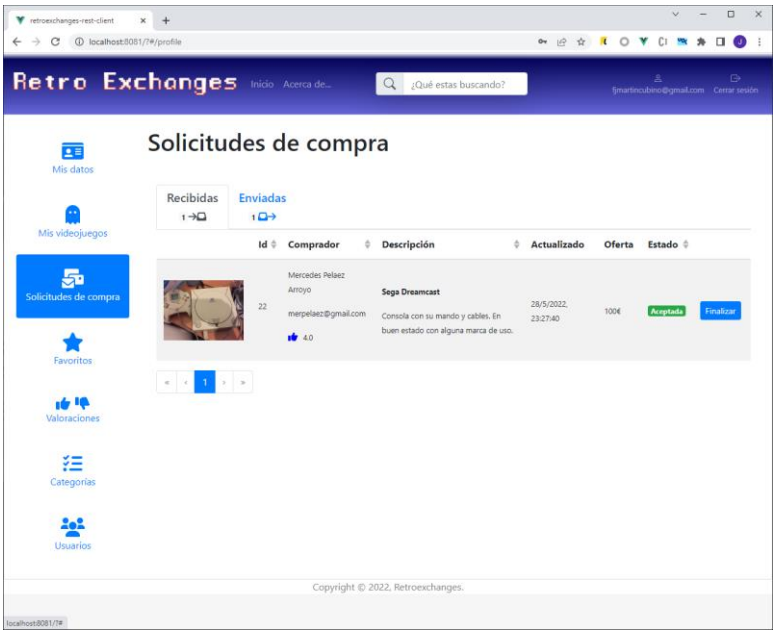


Ilustración 32: Solicitudes de compra recibidas

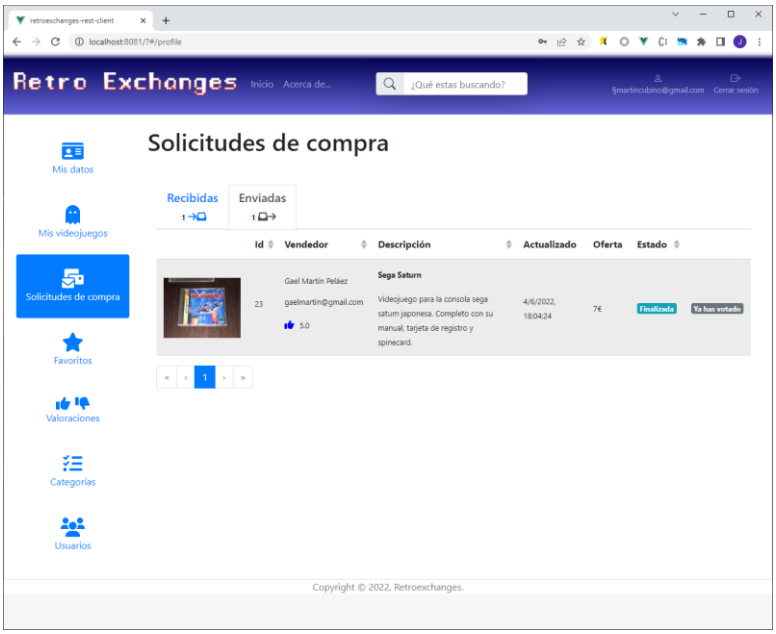


Ilustración 33: Solicitudes de compra enviadas

6. Conclusiones

Una vez finalizado el proyecto, puedo concluir, que se ha conseguido un producto viable, el cual incorpora todas las funcionalidades requeridas en los objetivos del trabajo, con la salvedad de que el cliente desarrollado solamente es compatible con ordenadores y tabletas. Debido a limitaciones de tiempo no me ha sido posible incorporar vistas adaptadas a dispositivos móviles.

La idea de este proyecto surgió de una de mis aficiones, los videojuegos. Desde pequeño me he sentido muy atraído por estas pequeñas grandes obras de arte que poco a poco se van ganando el lugar que en mi humilde opinión se merecen. Son los que jugaba en mi infancia los que más me atraen y cada vez resulta más difícil encontrarlos. Al ser un tema que me fascina sabía que iba a ayudarme en su puesta en marcha.

Solo he tenido contacto con Java EE a nivel académico durante el grado de informática, ya que nunca he tenido oportunidad de poder emplearlo a nivel profesional. Por eso me pareció una gran oportunidad acometer la creación de un proyecto empleando esta tecnología y poder poner en práctica los conocimientos adquiridos durante todos estos años.

La idea era obtener un “producto” práctico, que facilitase la tarea de comprar y vender los juegos, pero que, además fuese un API que diera soporte a múltiples clientes en distintas plataformas: Web, IOS, Android, etc.…

La elaboración del proyecto me ha permitido conocer una gran variedad de tecnologías con las que nunca había trabajado. En la parte REST, SpringBoot, Hibernate, JWT, MySQL.
En la parte cliente: Vue, Bootstrap, Leaflet, Route, Fetch, Vuex.

La curva de aprendizaje fue dura al principio, pero he encontrado muy satisfactorio lo aprendido y aplicado, no solo a nivel técnico, sino la hora de realizar el análisis y el diseño de todos los componentes.

La mayor dificultad con la que me he encontrado en el proyecto ha sido, la elección del stack tecnológico a utilizar. Al principio me decante por WildFly puesto que era lo único que había utilizado. Me encontré con muchos problemas de integración con todos los elementos que iba eligiendo. Esto desapareció en el momento que tome la decisión de utilizar SpringBoot.

En mi humilde opinión creo haber cumplido los objetivos propuestos al inicio del trabajo, aunque me hubiera gustado disponer de más tiempo para haber podido conseguir añadir alguna funcionalidad más.

En el siguiente apartado, enumero algunas de las posibles implementaciones que podrían acometerse para mejorar el producto mas completo.

7. Evolución del proyecto y posibles mejoras

Debido a las limitaciones de tiempo para poder llevar a cabo el proyecto, se han tomado decisiones que, aunque operativas pueden ser ampliamente mejoradas para dar una mejor solución. Algunas de ellas son:

7.1 Autorización OAuth2

Con OAuth2 mejoraríamos la seguridad del servicio al no tener que por qué almacenar los nombres y contraseñas de los usuarios. Mediante el uso de un framework de autorización, se podría delegar en Facebook, Google, Twitter o GitHub la autorización que permita el acceso al servicio.

7.2 Socket.io para notificaciones en tiempo real

Socket.io [27], permitirá a la página actualizar el estado de las solicitudes, valoraciones y demás acciones que pueden realizar los usuarios en tiempo real, eliminando la necesidad de recargar la página para saber si algo ha cambiado. El empleo de esta tecnología también permitiría el uso de un chat en tiempo real mediante el cual los usuarios podrían ponerse en contacto y agilizar la transacción.

7.3 Cliente para Android e IOS

En la actualidad muchos de los usuarios que realizan compras online, prefieren hacerlo desde su móvil. La implementación de un cliente adaptado a las pantallas móviles ampliaría el parque de usuarios que utilizarán la plataforma.

7.4 Notificaciones push en móviles

El cliente para móviles necesitaría el empleo de notificaciones push para alertar a los usuarios de cambios en los estados de sus peticiones de compra o para recibir información lo antes posible de la aplicación.

7.5 Pasarela de pago y envíos

El alcance actual del proyecto delega el medio de envío en los usuarios, de manera que siempre existe la posibilidad de que un usuario cometa un fraude. La implementación de un medio de pago a través de una pasarela de pago seguro aumentaría la seguridad ,y en caso de que un usuario no reciba su producto o sea víctima de un fraude, el importe de la transacción podría ser devuelto por el sistema. Por otra parte, también sería interesante implementar un sistema externo asociado a alguna empresa de paquetería nacional que facilitara el envío de las compras entre los usuarios.

8. Glosario

- API: Conjunto de definiciones y protocolos que se usa para diseñar e integrar el software de las aplicaciones.

- REST: Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

- HTTP: Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web.

- JSON: JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

- Framework: Esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software.

- JWT: JSON Web Token (JWT) es un estándar abierto (RFC-7519) basado en JSON para crear un token que sirva para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros. El caso más común de uso de los JWT es para manejar la autenticación en aplicaciones móviles o web.

- Frontend: Es la parte de un sitio web que interactúa con los usuarios, por eso decimos que está del lado del cliente.

- Backend: Es la parte que se conecta con la base de datos y el servidor que utiliza dicho sitio web, por eso decimos que el backend corre del lado del servidor.

- OAut2: Estándar abierto para la autorización de APIs, que nos permite compartir información entre sitios sin tener que compartir la identidad.

- WebSocket: Tecnología avanzada que hace posible abrir una sesión de comunicación interactiva entre el navegador del usuario y un servidor. Con esta API, puede enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar al servidor para una respuesta.

- Vue.js: Vue (pronunciado como view) es un framework progresivo para construir interfaces de usuario. La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes.

- **Node.js** es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript) creado por los desarrolladores originales de javascript.

9. Bibliografía

1. *El auge de las consolas retro en la actualidad. Disponible en https://www.retroplayingbcn.es/2021/03/el-auge-de-las-videoconsolas-retro.html*

2. *'Lo retro' bate records en la industria del videojuego. Disponible en: https://www.mundiario.com/articulo/empresas/retro-bate-records-industria-videojuego/20210731140231224869.html*

3. *Building REST services with Spring. Disponible en https://spring.io/guides/tutorials/rest/*

4. *Building a RESTful Web Service. Disponible en https://spring.io/guides/gs/rest-service/*

5. *Introduction to JSON Web Tokens. Disponible en* https://jwt.io/introduction

6. *Qué es Json Web Token y cómo funciona. Disponible en https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/*

7. *Spring Quickstart Guide. Disponible en https://spring.io/quickstart*

8. Issue with spring-data "startingWith" and hibernate 5.6.7. Disponible en https://github.com/spring-projects/spring-data-jpa/issues/2472

9. *Java JWT: JSON Web Token for Java and Android, Disponible en https://github.com/jwtk/jjwt*

10. *Welcome to the Font Awesome Docs. Disponible en https://fontawesome.com/docs*

11. *Get started with BootstrapVue. Disponible en https://bootstrap-vue.org/docs*

12. *Porcentaje de usuarios que realizaron compras a través de dispositivos móviles en España entre 2014 y 2021. Disponible en https://es.statista.com/estadisticas/872927/porcentaje-de-usuarios-que-realizaron-compras-por-dispositivos-moviles/*

13. *¿Qué es un framework? Disponible en https://www.edix.com/es/instituto/framework/*

14. *¿Qué es una API de REST? Disponible en https://www.redhat.com/es/topics/api/what-is-a-rest-api*

15. *¿Qué es una API?. Disponible en https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces*

16. *Spring Boot and OAuth2. Disponible en https://spring.io/guides/tutorials/spring-boot-oauth2/*

17. *Using WebSocket to build an interactive web application. Disponible en https://spring.io/guides/gs/messaging-stomp-websocket/*

18. *Nodejs. Disponible en https://nodejs.org/es/*

19. *Maven. Disponible en https://maven.apache.org*

20. *MySQL. Disponible en https://nodejs.org/es/*

21. *Postman REST Client. https://www.postman.com/product/rest-client/*

22. *Introducing JSON. Disponible en https://www.json.org/json-en.html*

23. *Hypertext Transfer Protocol -- HTTP/1.1. Disponible en https://datatracker.ietf.org/doc/html/rfc2616*

24. RFC 7519. JSON Web Token (JWT). Disponible en *https://datatracker.ietf.org/doc/html/rfc7519*

25. *The progressive JavaScript Framework. Disponible en https://v2.vuejs.org/*

26. *Wildfly Documentation. Disponible en https://docs.wildfly.org/17/*

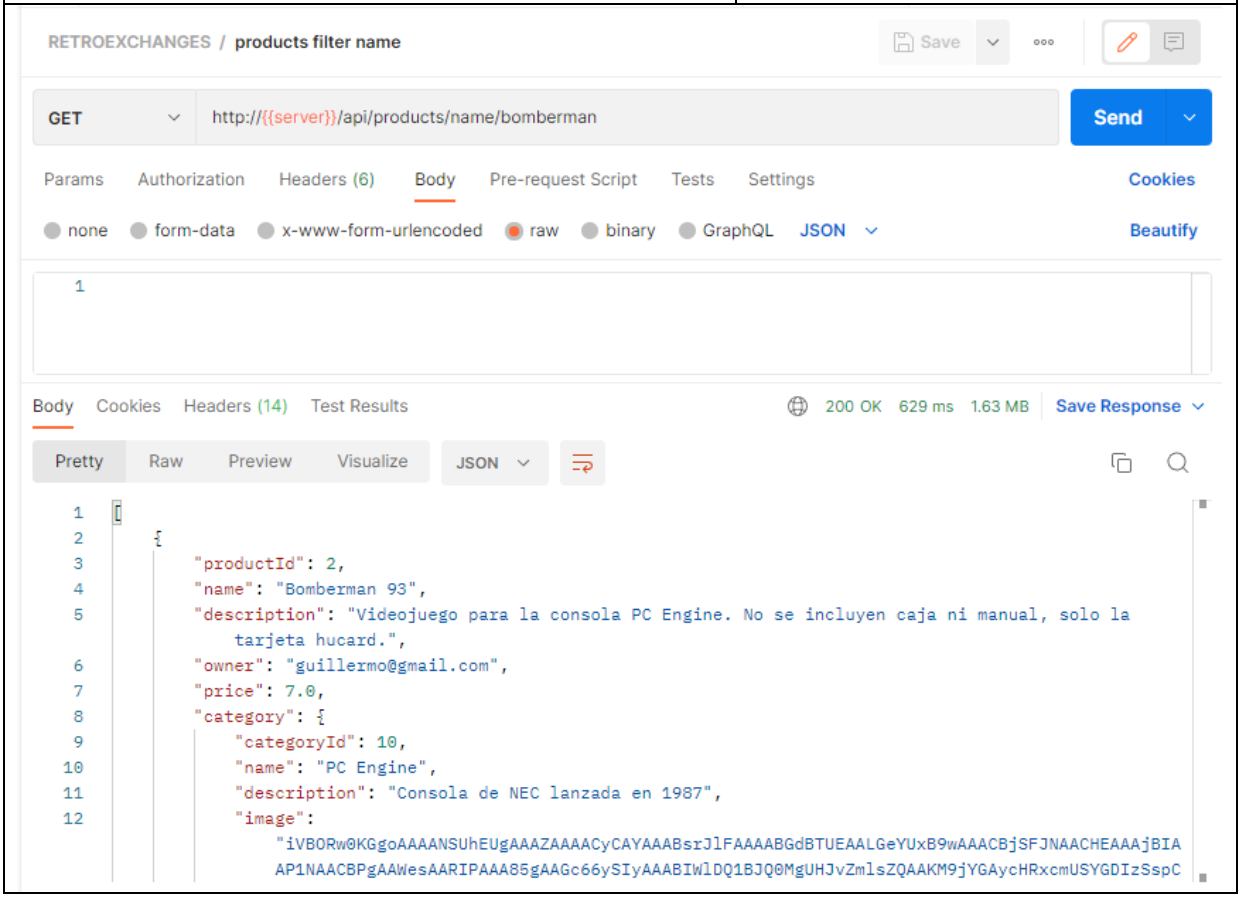
27. *What Socket.io is. Disponible en https://socket.io/docs/v4/*

10. Anexos

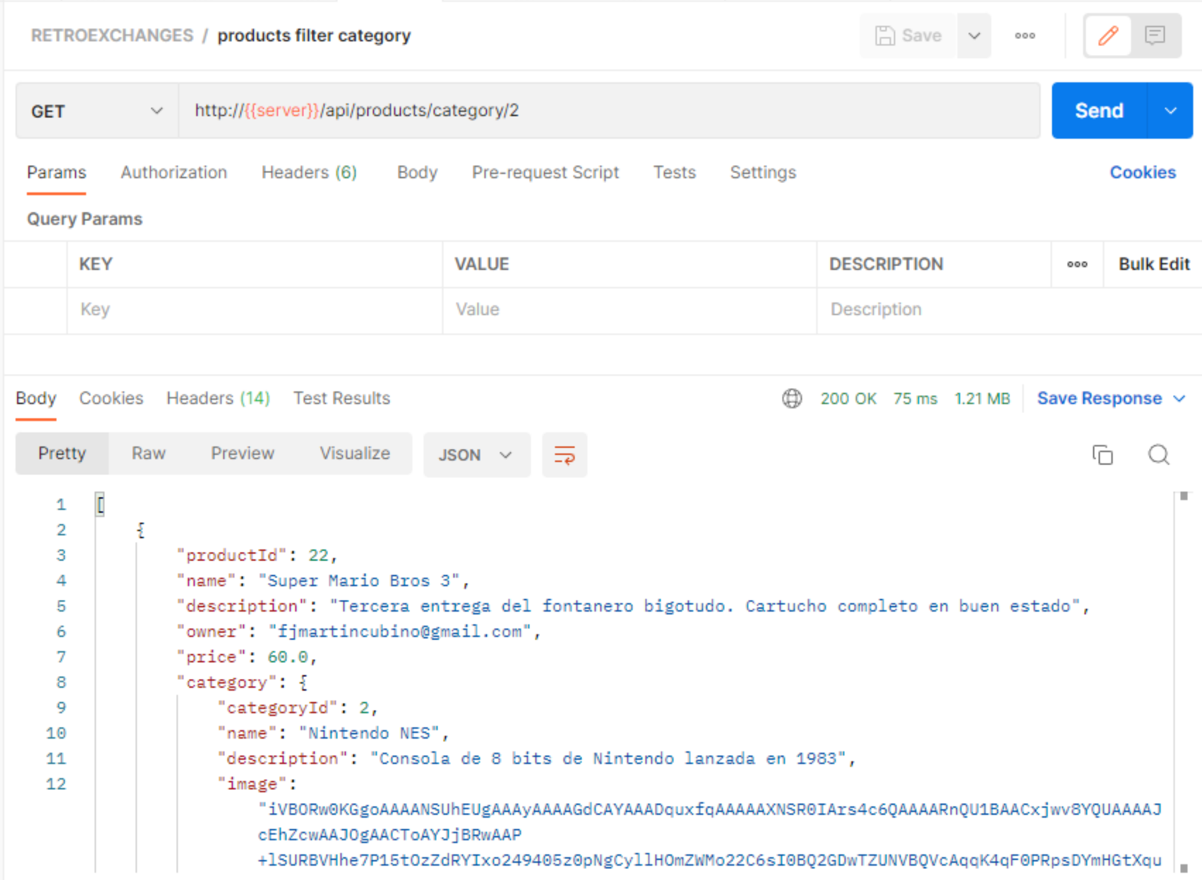
10.1 Pruebas de validación del API REST

Las pruebas unitarias del servicio REST se realizaron en base a los casos de uso definidos en el capítulo 3.5 y la especificación del api del capítulo 4.3.

CU01	Buscar videojuegos y consolas por nombre
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por su nombre
GET /api/products/name/bomberman	**200 OK - PASSED**

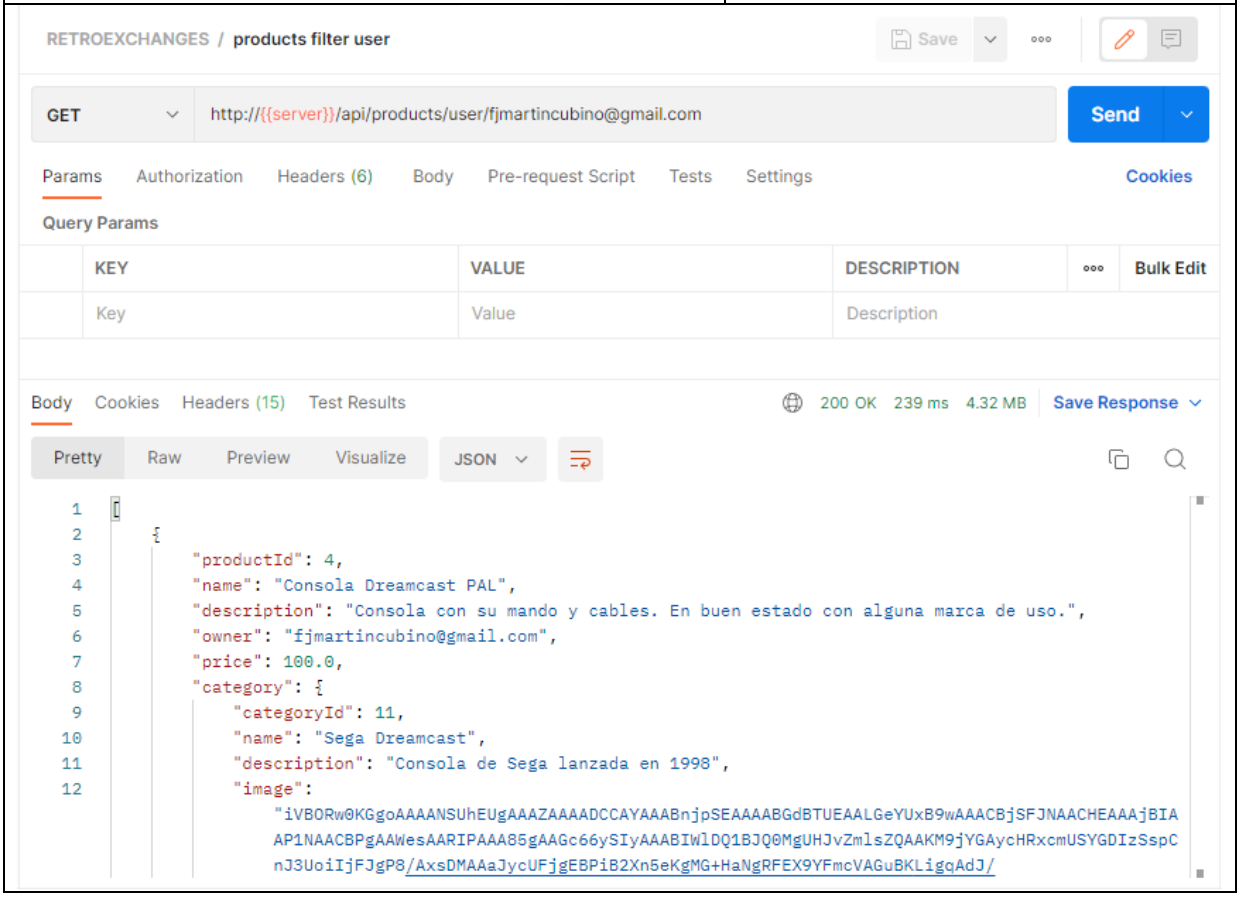


CU02	**Buscar videojuegos y consolas por categoría**
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por su categoría
GET /api/products/category/2	200 OK - PASSED



CU03	Buscar videojuegos y consolas por usuario
Actores	Todos
Objetivo	Permite al usuario buscar un videojuego o consola por el usuario al que pertenece.

GET /api/products/category/2	200 OK - PASSED



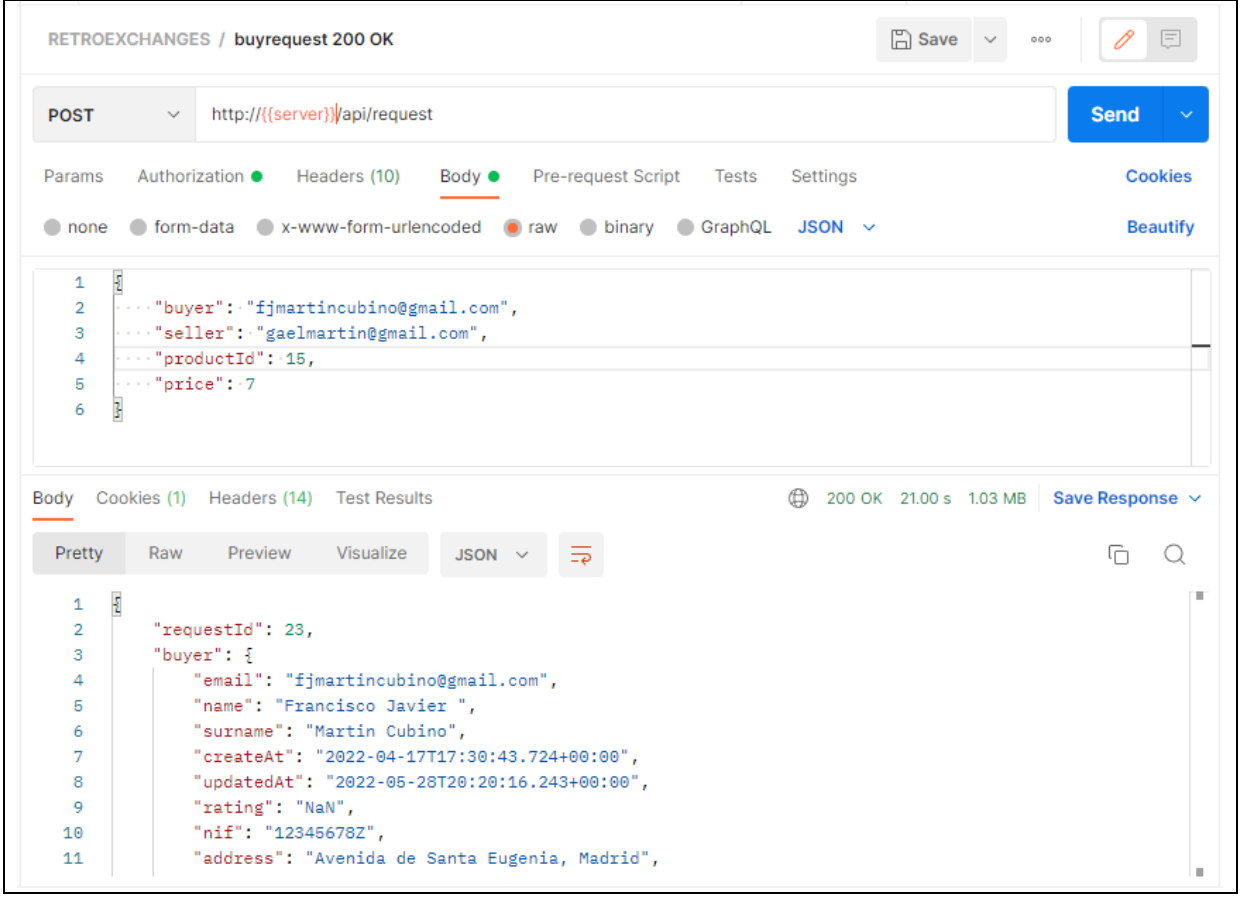
CU04	Enviar una oferta por una consola o videojuego a otro usuario
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario enviar una petición de oferta por un videojuego o consola a otro usuario

POST /api/request

```
{
    "buyer": "fjmartincubino@gmail.com",
    "seller": "gaelmartin@gmail.com",
    "productId": 15,
    "price": 7
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



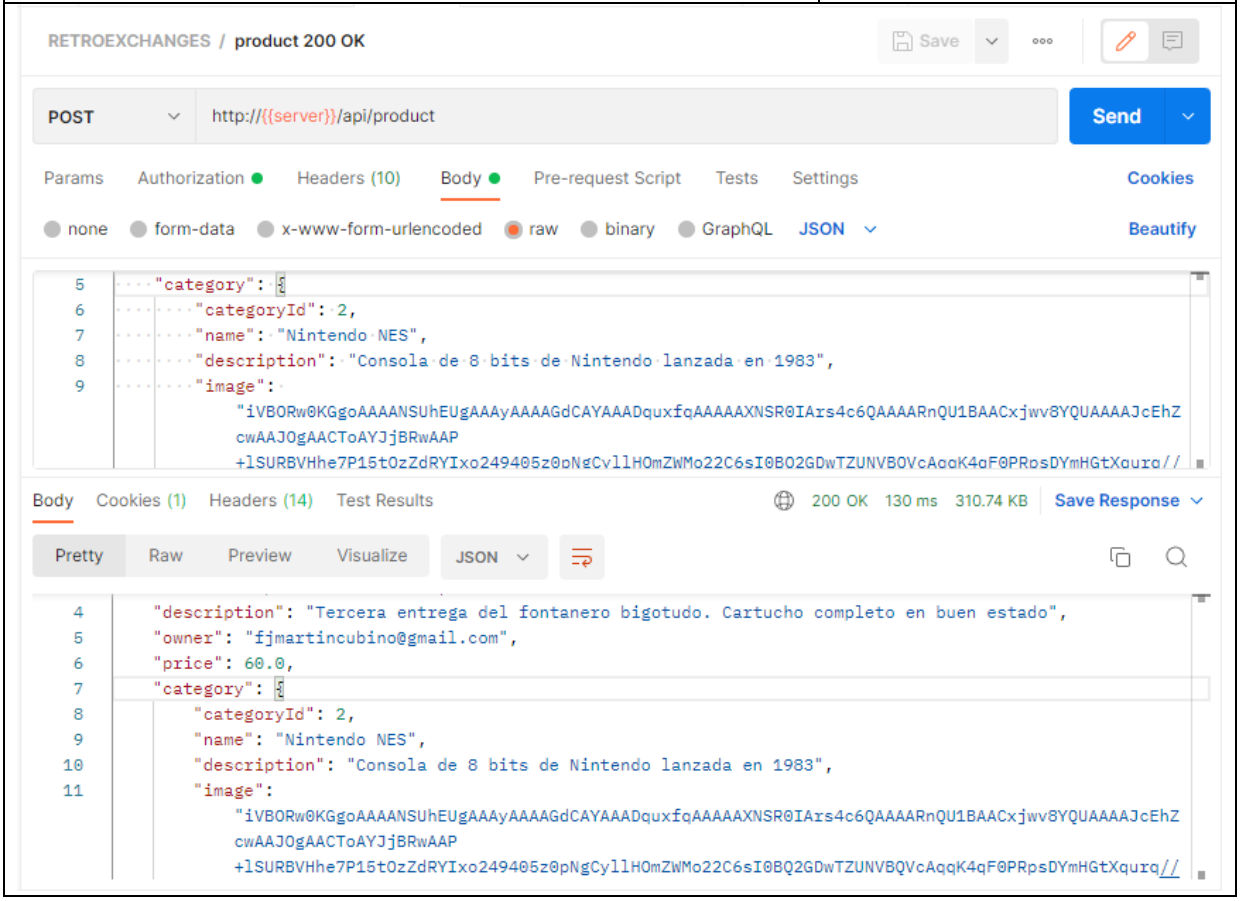
CU05	Alta de videojuegos y consolas a vender
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario dar de alta un videojuegos o consola en el sistema.

POST /api/product

```
{
    "name": "Super Mario Bros 3",
    "description": "Tercera entrega del
fontanero bigotudo.
Cartucho completo en buen estado",
    "status": "0",
    "category": {...},
    "pictureList":[],
    "owner": "fjmartincubino@gmail.com",
    "price": 60
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



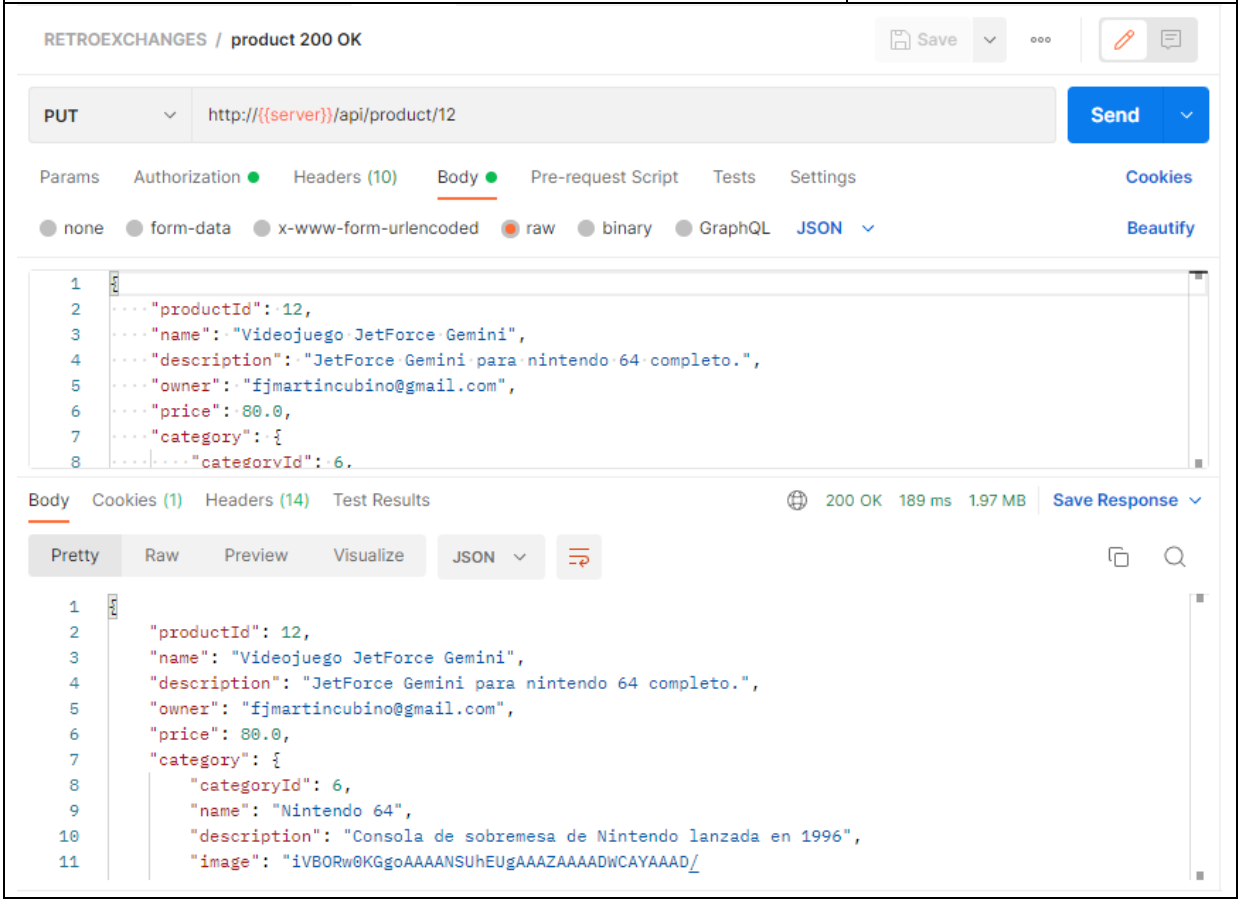
CU06	Modificar videojuegos y consolas a vender
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario modificar un videojuegos o consola en el sistema.

PUT /api/producto/12

```
{
    "productId": 12,
    "name": "Videojuego JetForce Gemini",
    "description": "JetForce Gemini
para nintendo 64 completo.",
    "owner": "fjmartincubino@gmail.com",
    "price": 80.0,
    "category": {...},
    "status": "AVAILABLE",
    "createAt": "2022-05-31T09:54:09.722+00:00",
    "updatedAt": "2022-05-31T09:54:09.916+00:00",
    "pictureList": []
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED

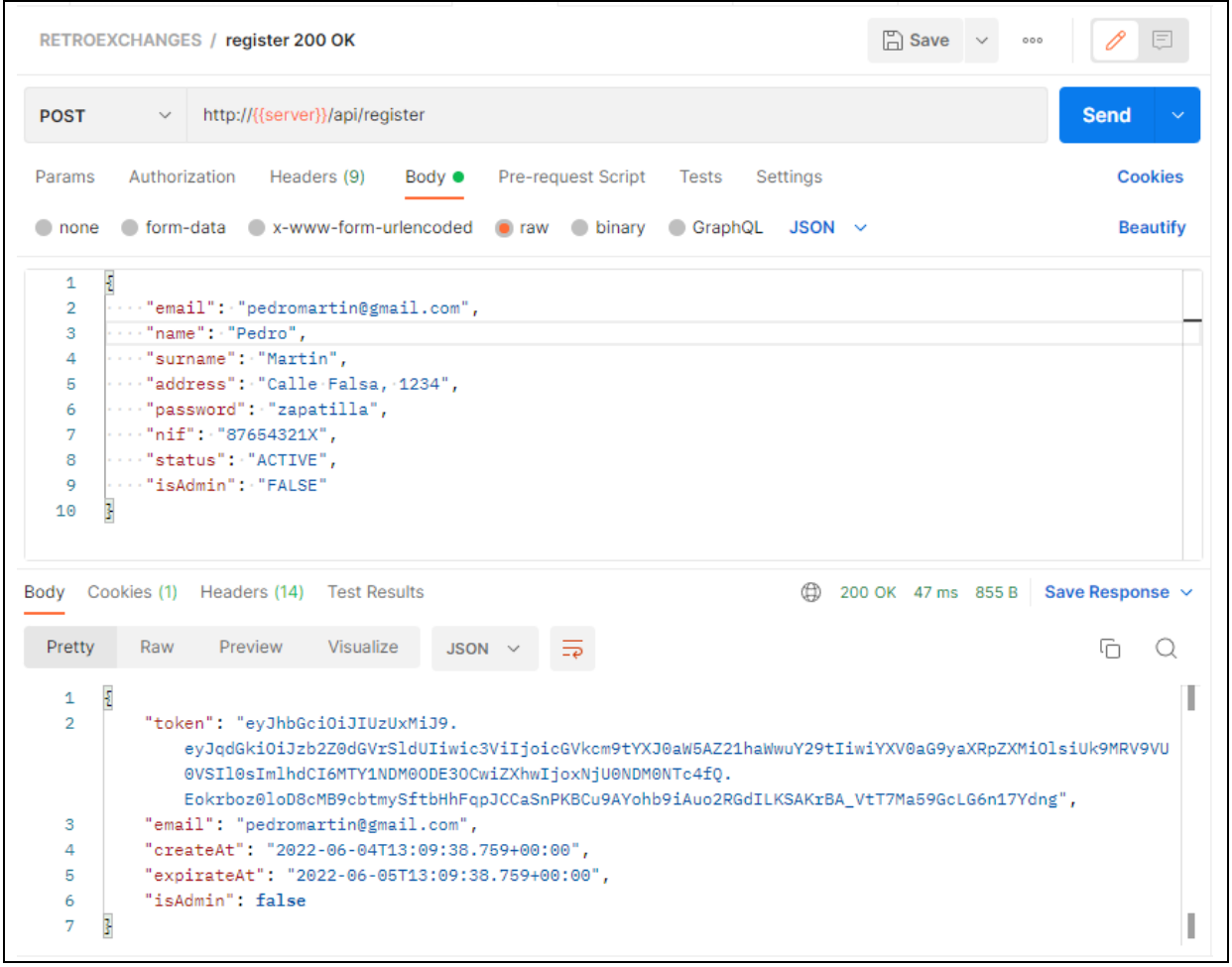


CU07	Registrarse en el sistema
Actores	Usuario
Objetivo	Permite al usuario registrarse en la aplicación y crear una nueva cuenta.

POST /api/register

```
{
    "email": "pedromartin@gmail.com",
    "name": "Pedro",
    "surname": "Martin",
    "address": "Calle Falsa, 1234",
    "password": "zapatilla",
    "nif": "87654321X",
    "status": "ACTIVE",
    "isAdmin": "FALSE"
}
```

200 OK PASSED



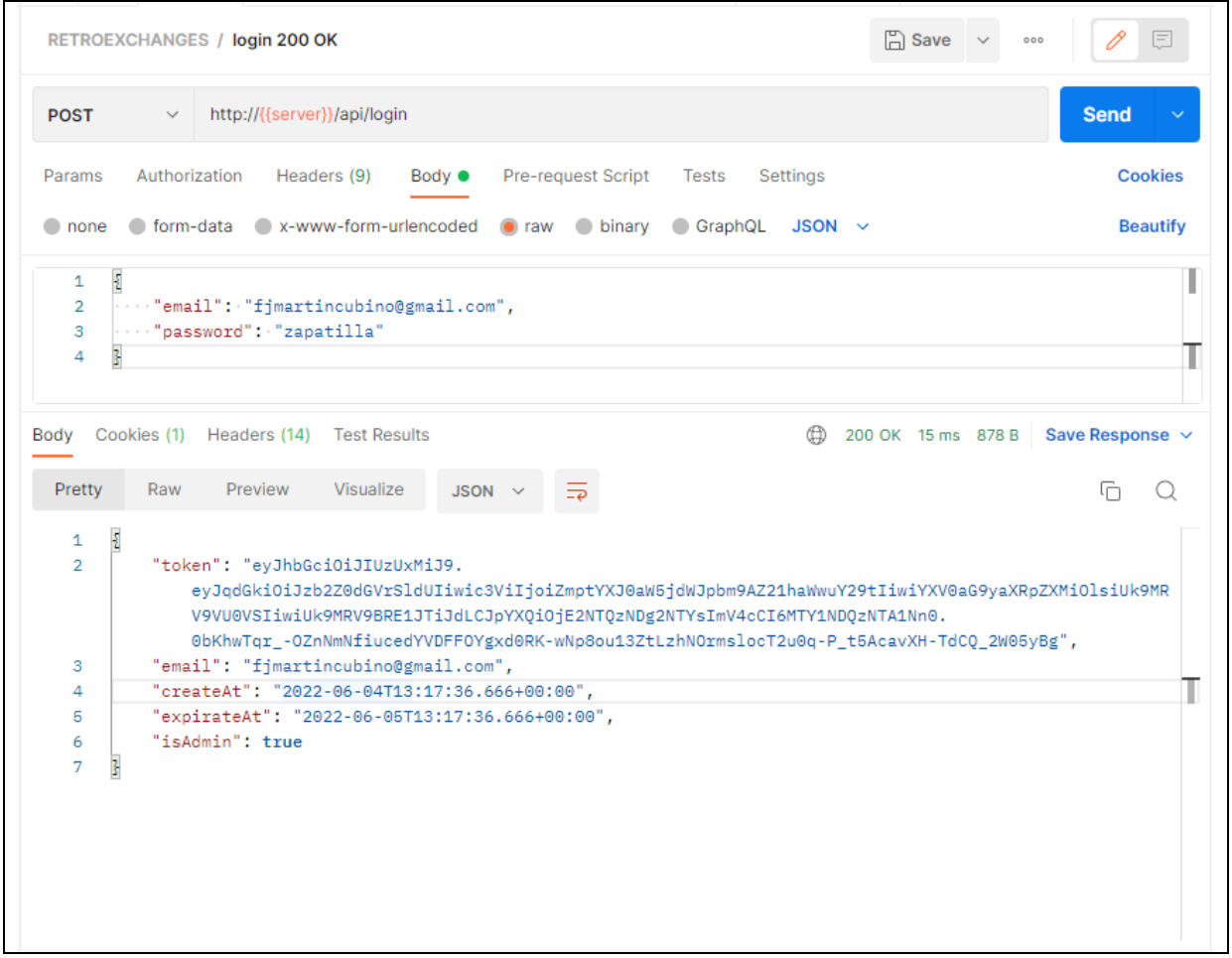
CU08	Identificarse en el sistema
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario acceder a las funcionales privadas solo accesibles por un usuario registrado.

POST /api/login

```
{
    "email": "{{email}}",
    "password": "zapatilla"

}
```

200 OK PASSED



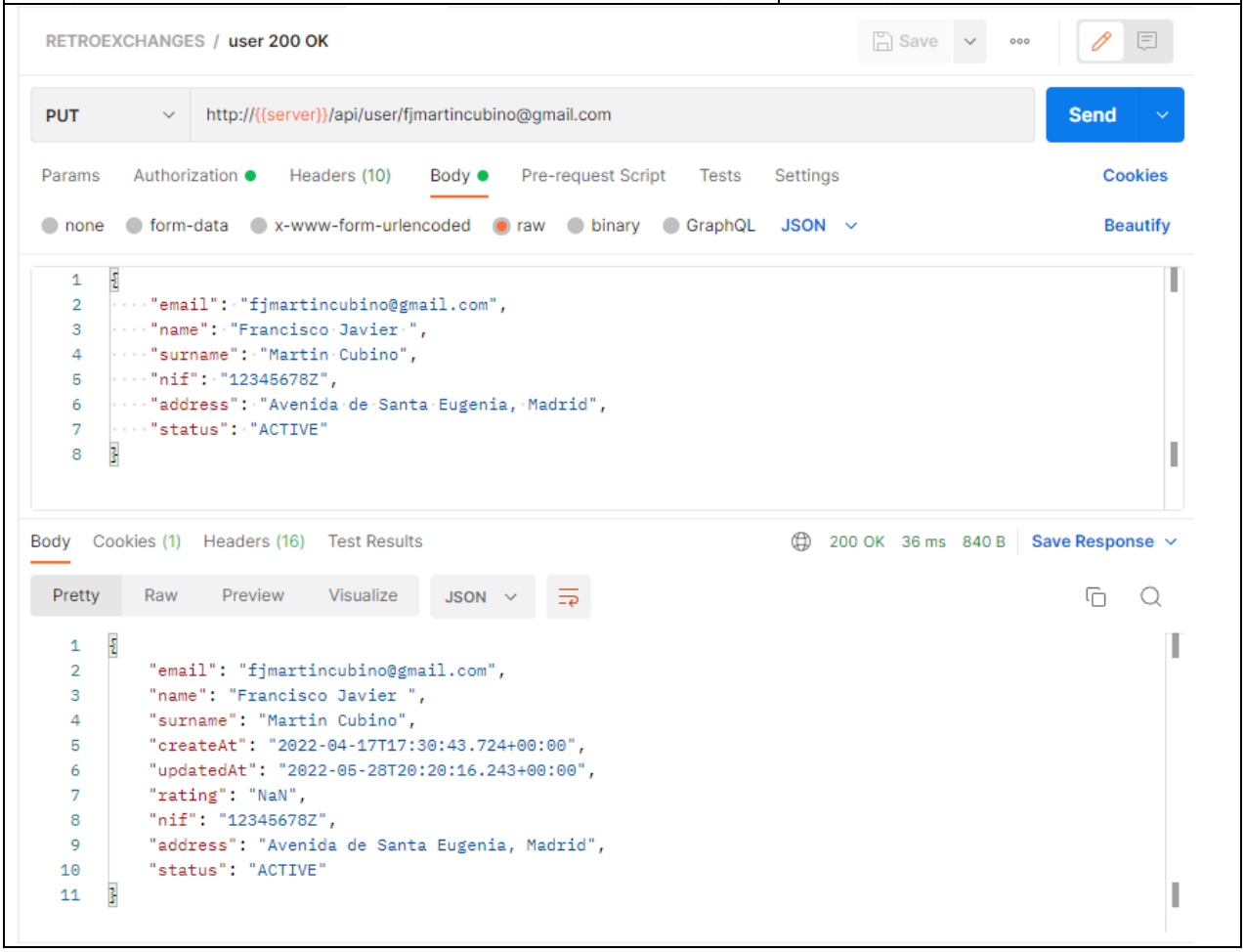
CU09	Actualizar datos personales
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario actualizar sus datos personales

PUT /api/user/fjmartincubino@gmail.com

```
{
    "email": "fjmartincubino@gmail.com",
    "name": "Francisco Javier ",
    "surname": "Martin Cubino",
    "nif": "12345678Z",
    "address": "Avenida de Santa Eugenia, Madrid",
    "status": "ACTIVE"
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



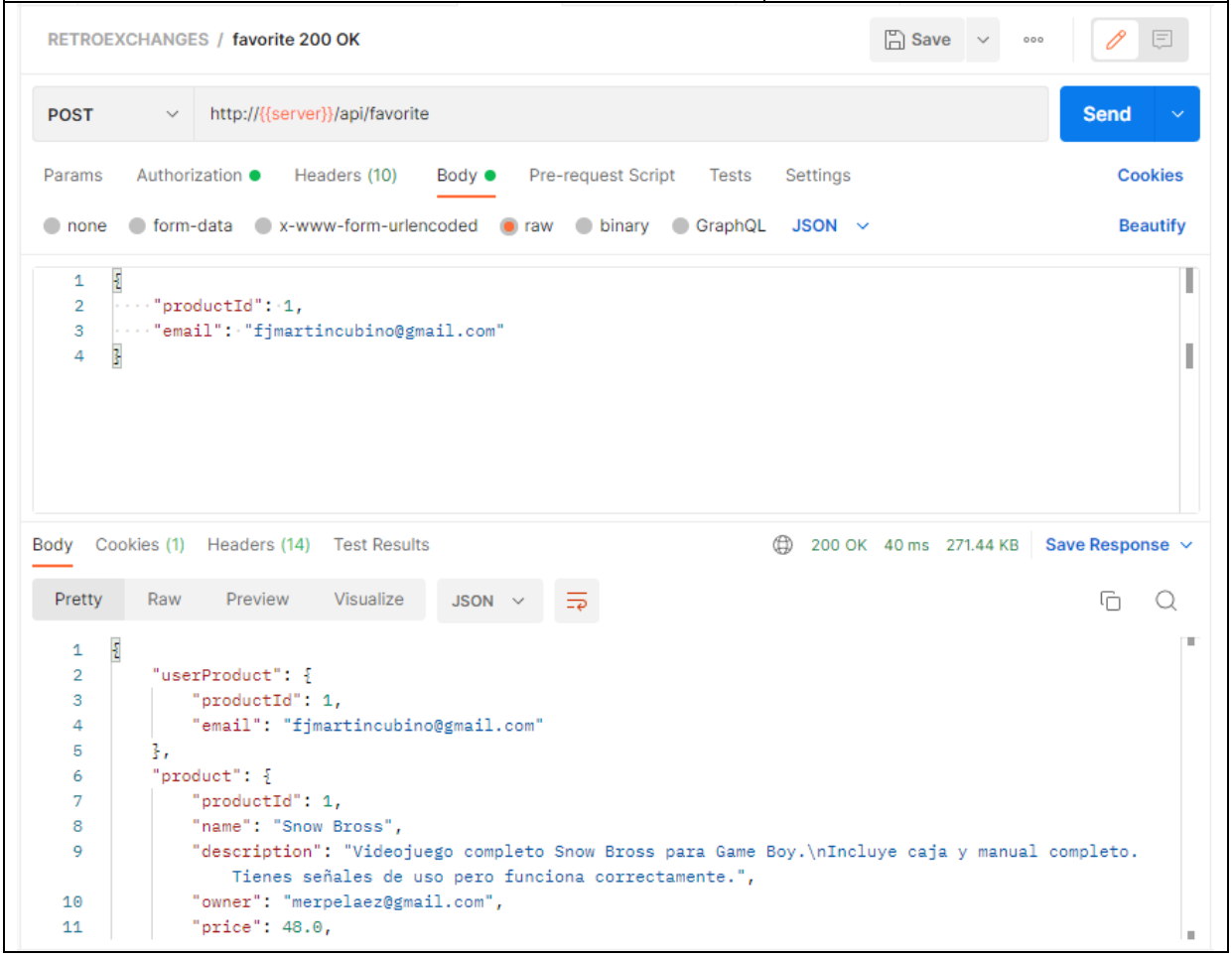
CU11	Añadir un producto a favorito
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario añadir al listado de sus favoritos un videojuego o consola.

POST /api/favorite

```
{
    "productId": 1,
    "email": "fjmartincubino@gmail.com"
}
```

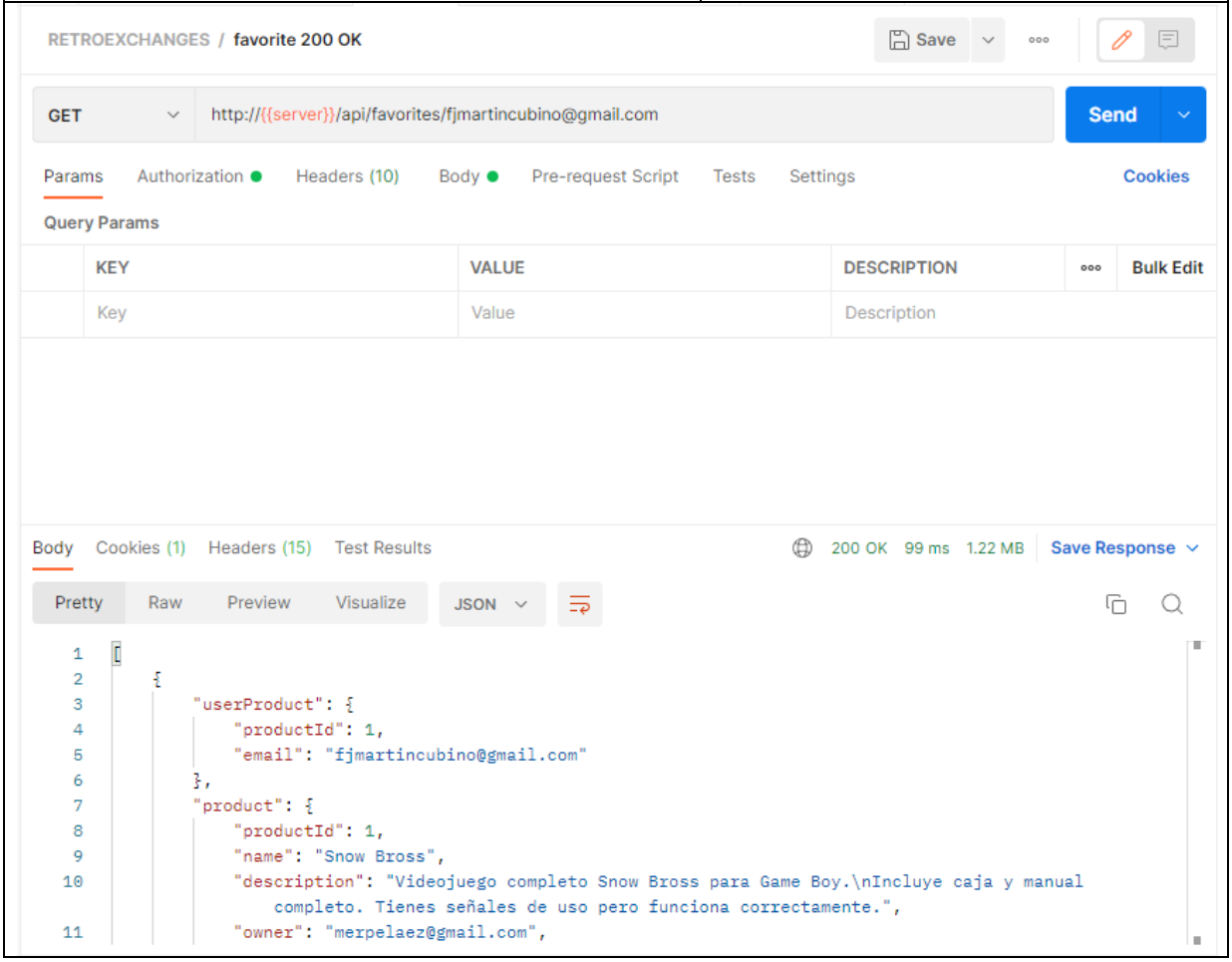
Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



CU12	Consultar lista de favoritos
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario consultar el listado de los favoritos asociados a su cuenta.

GET /api/favorites/fjmartincubino@gmail.com **Header:** “Authorization Bearer eyJh[…]”	200 OK PASSED



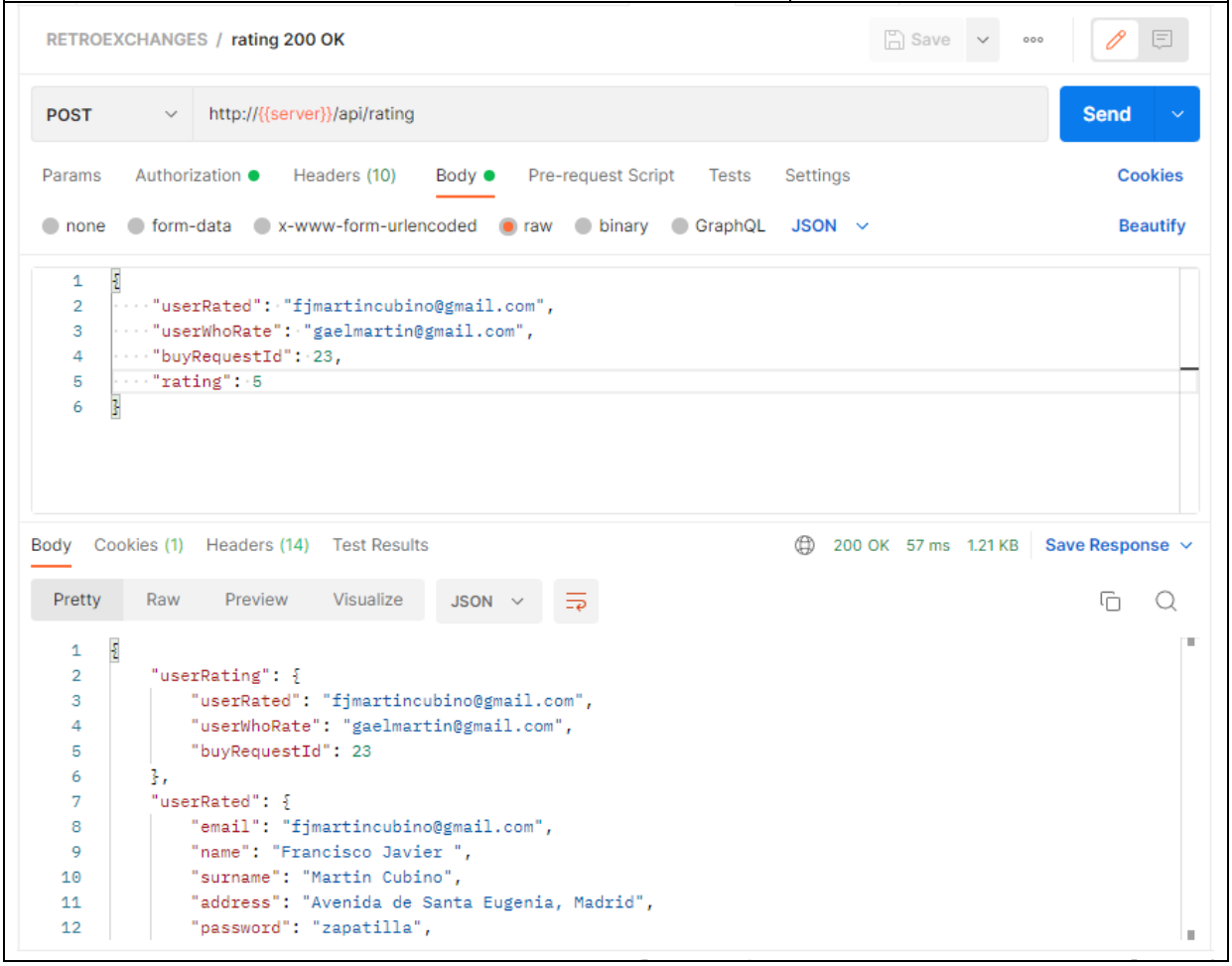
CU13	Valorar a un usuario como comprador
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario valorar a otro usuario como comprador al finalizar una compraventa.

POST /api/rating

```
{
    "userRated": "fjmartincubino@gmail.com",
    "userWhoRate": "gaelmartin@gmail.com",
    "buyRequestId": 23,
    "rating": 5
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



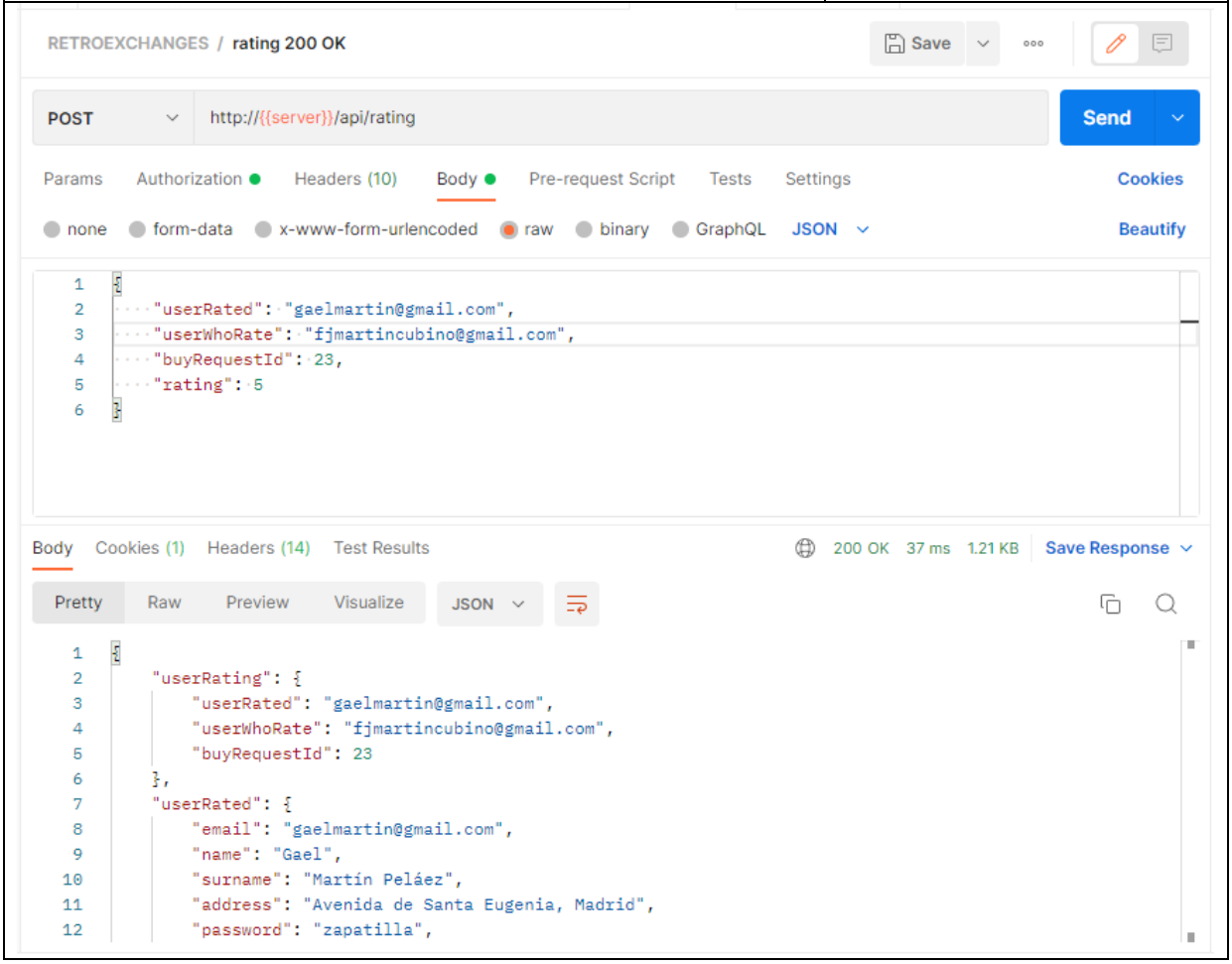
CU14	Valorar a un usuario como vendedor
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario valorar a otro usuario como vendedor al finalizar una compraventa.

POST /api/rating

```
{
    "userRated": "gaelmartin@gmail.com",
    "userWhoRate": "fjmartincubino@gmail.com",
    "buyRequestId": 23,
    "rating": 5
}
```

Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



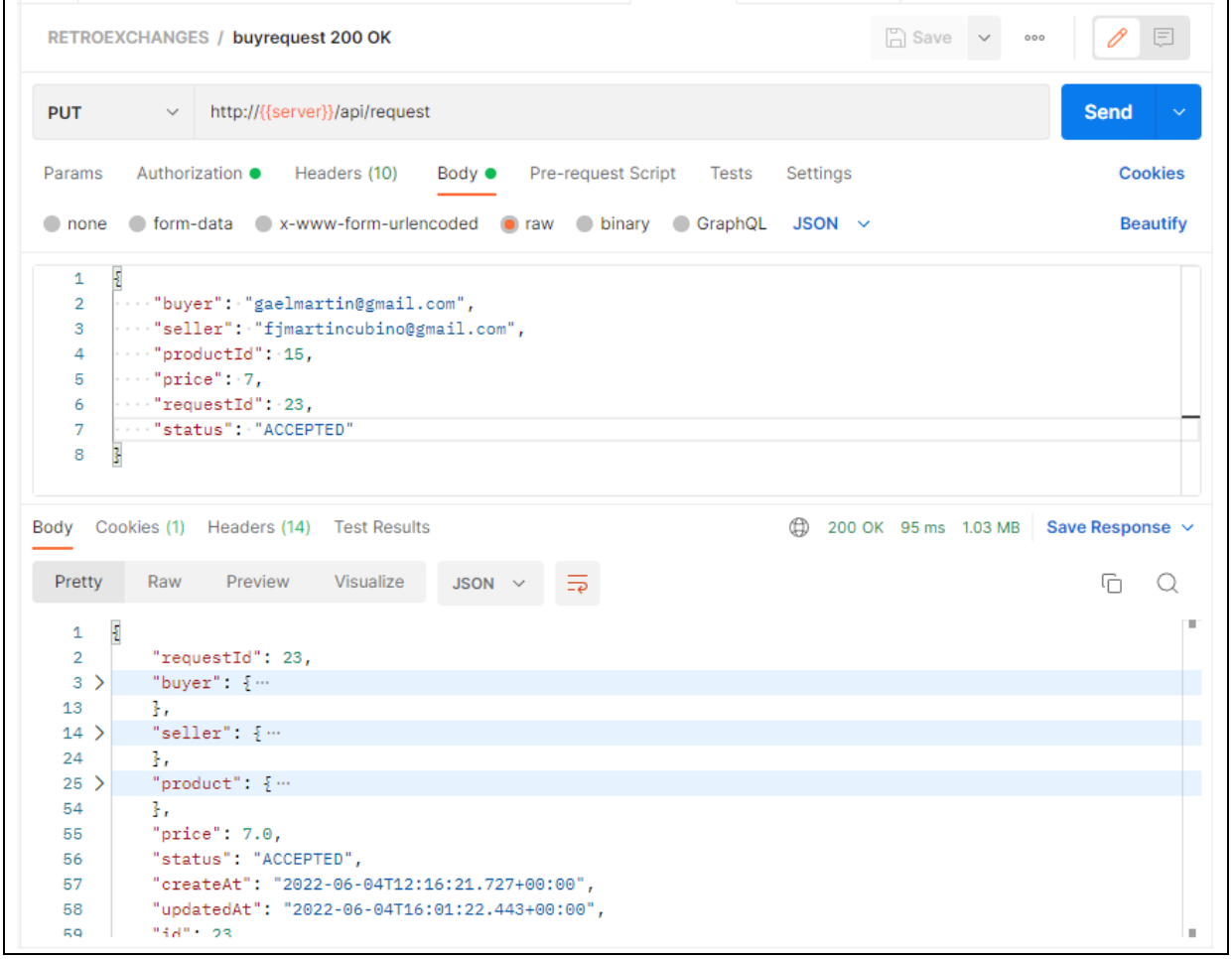
CU15	Aceptar/rechazar una oferta por una consola/videojuego
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario aceptar una petición de oferta por un videojuego o consola enviada por otro usuario

PUT /api/request

```
{
    "buyer": "gaelmartin@gmail.com",
    "seller": "fjmartincubino@gmail.com",
    "productId": 15,
    "price": 7,
    "requestId": 23,
    "status": "ACCEPTED"
}
```

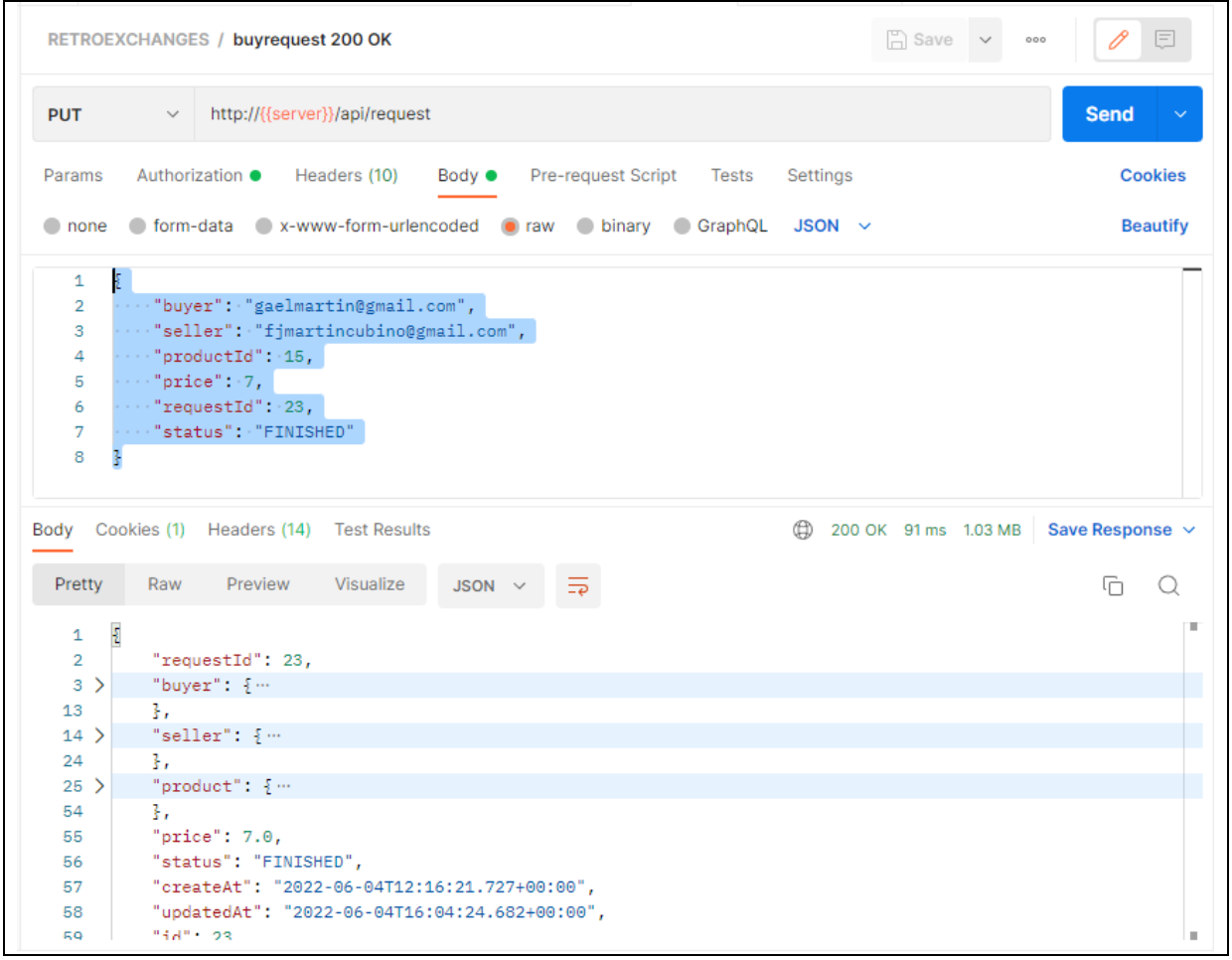
Header: “Authorization Bearer eyJh[…]”

200 OK PASSED



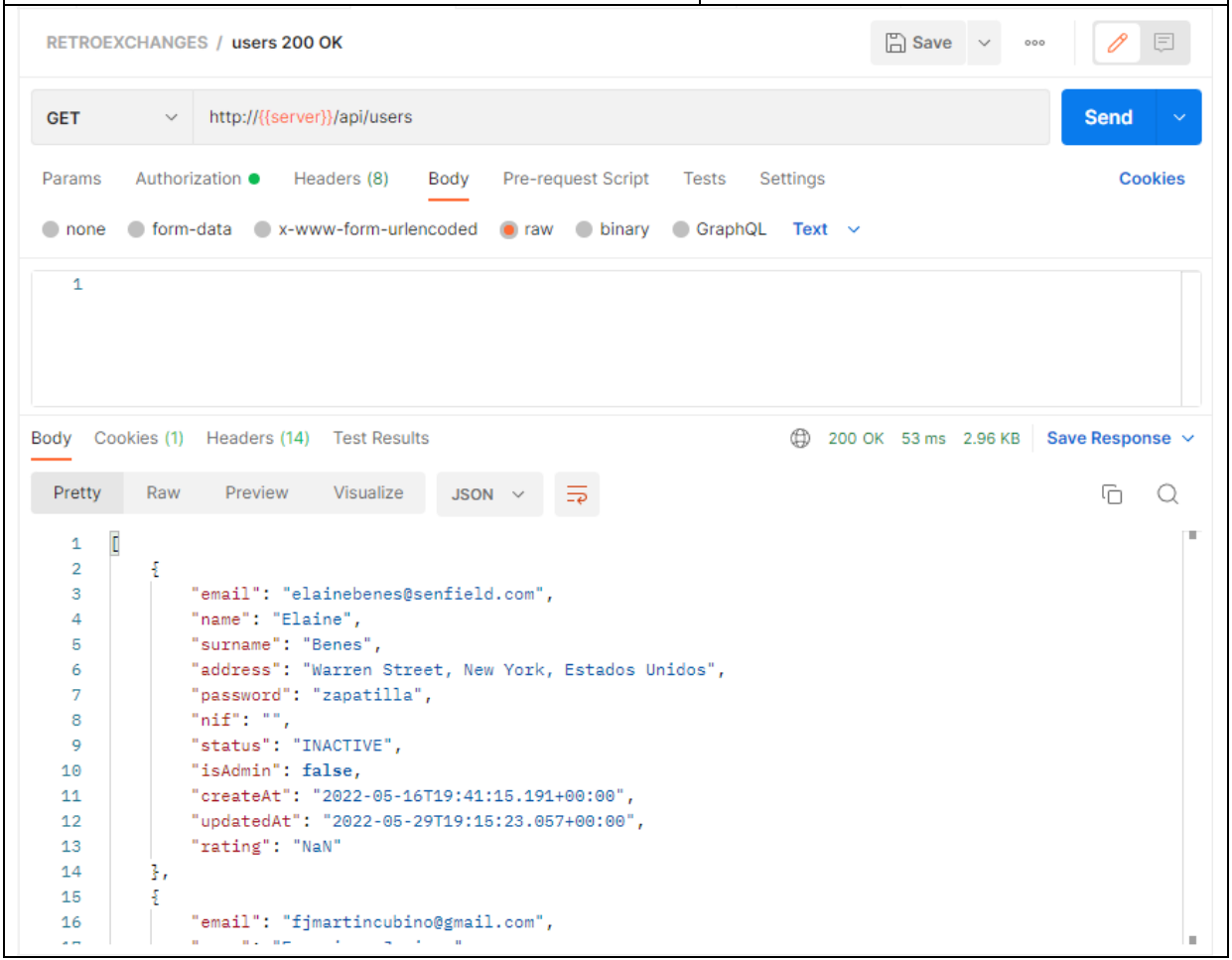
CU16	Finalizar compra
Actores	Usuario registrado, Administrador
Objetivo	Permite al usuario finalizar la venta de un videojuego o consola a otro usuario

PUT /api/request
{
"buyer": "gaelmartin@gmail.com",
"seller": "fjmartincubino@gmail.com",
"productId": 15,
"price": 7,
"requestId": 23,
"status": "FINISHED"
}
Header: "Authorization Bearer eyJh[…]"
	200 OK PASSED



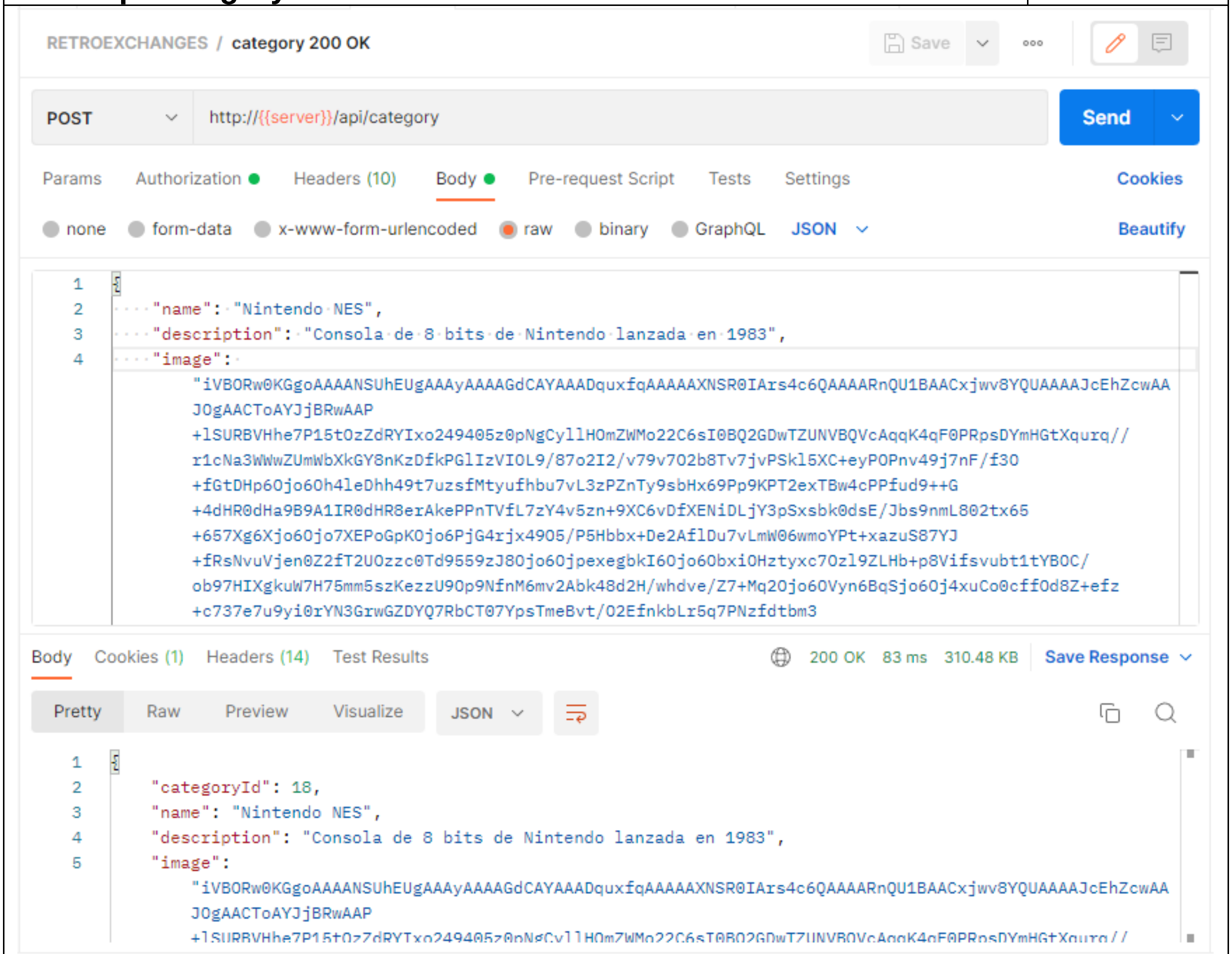
CU17	Buscar usuarios con baja reputación
Actores	Administrador
Objetivo	Permite al administrador consultar un listado de los usuarios registrados en el sistema ordenados por reputación.

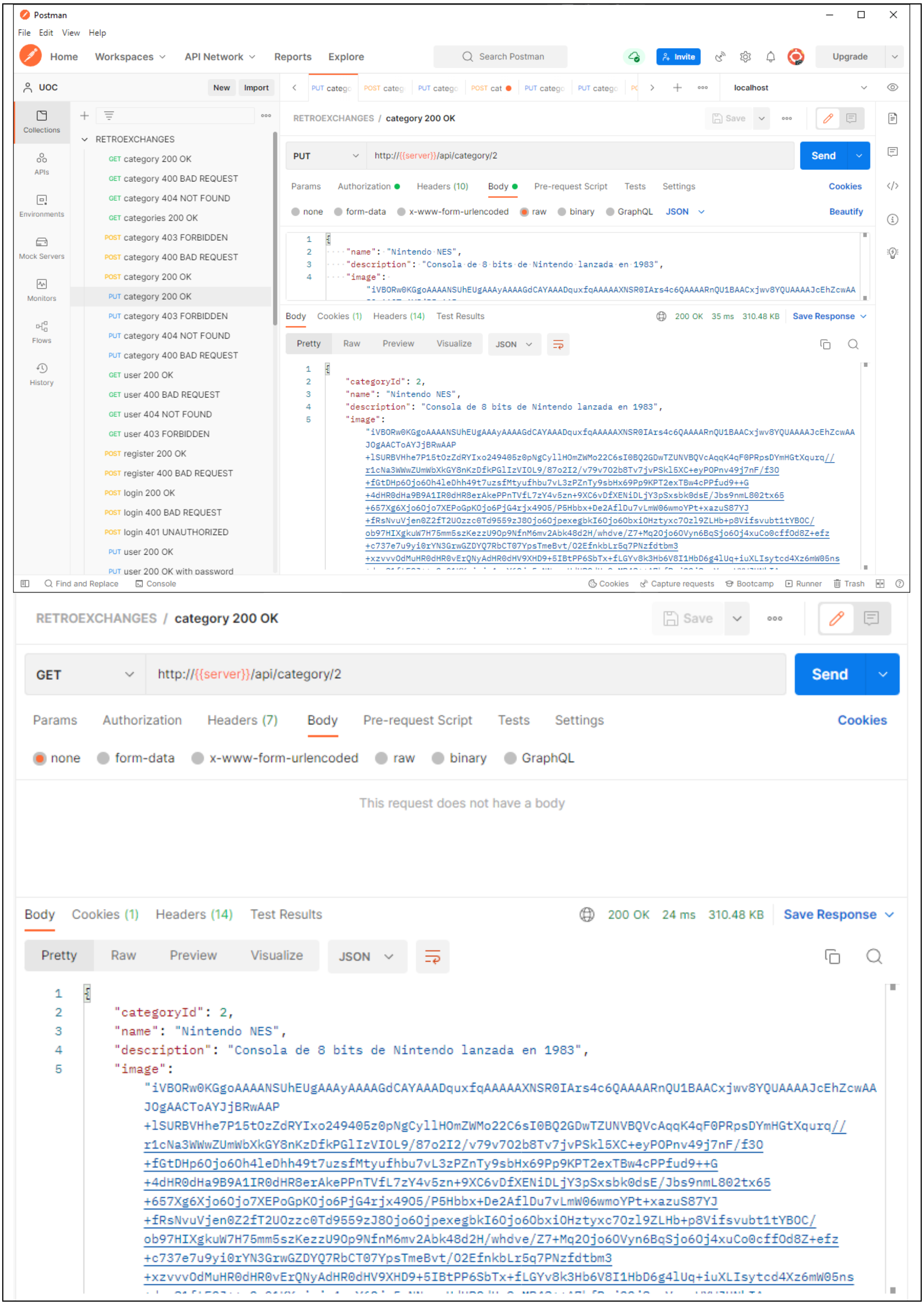
GET /api/users **Header:** “Authorization Bearer eyJh[…]”	200 OK PASSED



<table>
<tr><th>CU18</th><th colspan="2">Gestión de las categorías (alta, baja, consulta y modificación)</th></tr>
<tr><td>Actores</td><td colspan="2">Usuario registrado, Administrador</td></tr>
<tr><td>Objetivo</td><td colspan="2">Permite al usuario enviar una petición de oferta por un videojuego o consola a otro usuario</td></tr>
<tr><td colspan="2">POST /api/category
{
"name": "Nintendo NES",
"description": "Consola de 8 bits de Nintendo lanzada en 1983",
"image": "iVBORw0KGgoAAAANSUhEUgAAA[...]"
}
Header: “Authorization Bearer eyJh[…]”

PUT /api/category/2

GET /api/category/2</td><td>200 OK
PASSED</td></tr>
<tr><td colspan="3">  </td></tr>
</table>


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
UOC
New Import
localhost
RETROEXCHANGES
GET category 200 OK
GET category 400 BAD REQUEST
GET category 404 NOT FOUND
GET categories 200 OK
POST category 403 FORBIDDEN
POST category 400 BAD REQUEST
POST category 200 OK
PUT category 200 OK
PUT category 403 FORBIDDEN
PUT category 404 NOT FOUND
PUT category 400 BAD REQUEST
GET user 200 OK
GET user 400 BAD REQUEST
GET user 404 NOT FOUND
GET user 403 FORBIDDEN
POST register 200 OK
POST register 400 BAD REQUEST
POST login 200 OK
POST login 400 BAD REQUEST
POST login 401 UNAUTHORIZED
PUT user 200 OK
PUT user 200 OK with password
RETROEXCHANGES / category 200 OK
PUT http://{{server}}/api/category/2
Send
Params Authorization Headers (10) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies (1) Headers (14) Test Results
200 OK 35 ms 310.48 KB
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
RETROEXCHANGES / category 200 OK
Save
GET http://{{server}}/api/category/2
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
This request does not have a body
Body Cookies (1) Headers (14) Test Results
200 OK 24 ms 310.48 KB
Save Response
Pretty Raw Preview Visualize JSON
{
"categoryId": 2,
"name": "Nintendo NES",
"description": "Consola de 8 bits de Nintendo lanzada en 1983",
"image":
"iVBORw0KGgoAAAANSUhEUgAAAyAAAAGdCAYAAADquxfqAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAA
JOgAACToAYJjBRwAAP
+lSURBVHhe7P15tOzZdRYIxo249405z0pNgCyllHOmZWMo22C6sI0BQ2GDwTZUNVBQVcAqqK4qF0PRpsDYMHGtXqurq//
r1cNa3WWwZUmWbXkGY8nKzDfkPGlIzVIOL9/87o2I/v79v7vjvPSk15XXC+eyP99PNXC5P2Z/
+fGtDHp60jo60h4leDhh49t7uzsfMtyufhbu7vL3zPZnTy9sbHx69Pp9KPT2exTBw4cPPfud9++e
+4dHR0dHa9B9A1IR0dHR8erAkePPnTVfL7zY4v5zn+++9XC6vDfXENiDLjY3SsSbsk0dsE/Jbs9nnmL802tx65
+657Xg6Xjo6o6o7XEPoGpK0jo6PjG4rjx4905
+fRsNvuVjen0Z2fT2U0zzc0Td9559zJ0LH3zz0BtXg6u07Jg73Iw03BGnx
+fGtDHp6Oj060h4leDhh49t7uzsfMtyufhbu7vL3zPZnTy9sbHx69Pp9KPT2exTBw4cPPfud9++G
+4dHR0dHa9B9A1IR0dHR8erAkePPnTVfL7zY4v5zn+9XC6vDfXENiDLjY3SsSbsk0dsE/Jbs9nmL802tx65
+657Xg6Xjo6o6o7XEPoGpK0jo6PjG4rjx4905/P5Hbbbx+De2AflDu7vLmW06wmoYPt+xazuS87YJ
+fRsNvuVjen0Z2fT2U0zzc0Td9559zJ80jo60jpexegbkI60jo60jy2v7HuHMuB+aN+AIDd
+cd7u7yi0rYN3GrwGZDYQ7RbCT07YpsTmeBvt/02EfnkbLr5q7PNzfdtbm3
ob97HIXgkuW7H75mm5szKezzU9Op9NfnM6mv2Abk48d2H/whde/Z7+Mq2oj60Vyn6BqSj060j4xuCo0cfzf0d8Z+efz
+c737e7u9yi0rYN3GrwGZDYQ7RbCT07YpsTmeBvt/02EfnkbLr5q7PNzfdtbm3
+xzvvvOdMuHR0dHR0vErQyAdHR0dHV9XHD9+5BtPP6SbTx+fLGYv8k3Hb6V8I1HbD6g4lUq+iuXLIsytcd4Xz6mW05ns
+xzvvvOdMuHR0dHR0vErQyAdHR0dHV9XHD9+5BtPP6SbTx+fLGYv8k3Hb6V8I1HbD6g4lUq+iuXLIsytcd4Xz6mW05ns

10.2 Repositorios

El código se encuentra en dos repositorios ubicados en github de forma pública. La versión estable más reciente se encuentra en la rama “main”.

El código fuente JAVA con el servidor REST puede descargarse de:

https://github.com/martincubino/retroexchanges-rest-service.git

El código fuente del cliente se encuentra en:

https://github.com/martincubino/retroexchanges-rest-client.git

10.3 Java

Como versión de java se ha empleado OpenJDK11, concretamente se ha instalado la versión disponible en el siguiente enlace:

https://github.com/adoptium/temurin11-binaries/releases/download/jdk-11.0.14.1%2B1/OpenJDK11U-jdk_x64_windows_hotspot_11.0.14.1_1.msi

Una vez instalado hemos de asegurarnos que la variable de entorno JAVA_HOME está establecida en el sistema. El instalador nos propone hacerlo automáticamente durante la instalación.

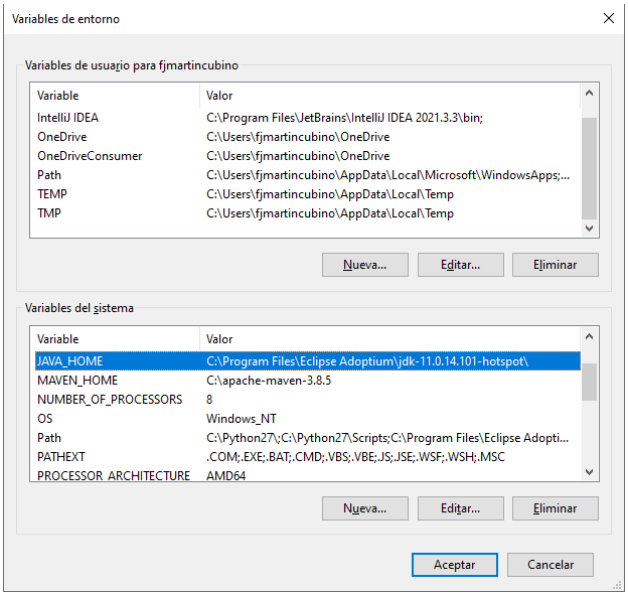


Ilustración 34: Variable de entorno JAVA_HOME

10.4 MySQL

Se ha elegido motor de base de datos para la persistencia de la información MySQL en su versión mysql-installer-community-8.0.28.0.msi.

https://cdn.mysql.com//Downloads/MySQLInstaller/mysql-installer-community-8.0.29.0.msi

Una vez instalado el motor de base de datos es necesario crear el usuario: **USER** y con la contraseña: **PASSWORD.** El tipo de autenticación debe ser **Standard.**

El usuario y contraseña puede ser cambiado en el proyecto en el fichero **application.properties**, ubicado en *retroexchanges-rest-service\src\main\resource* en la carpeta del proyecto.

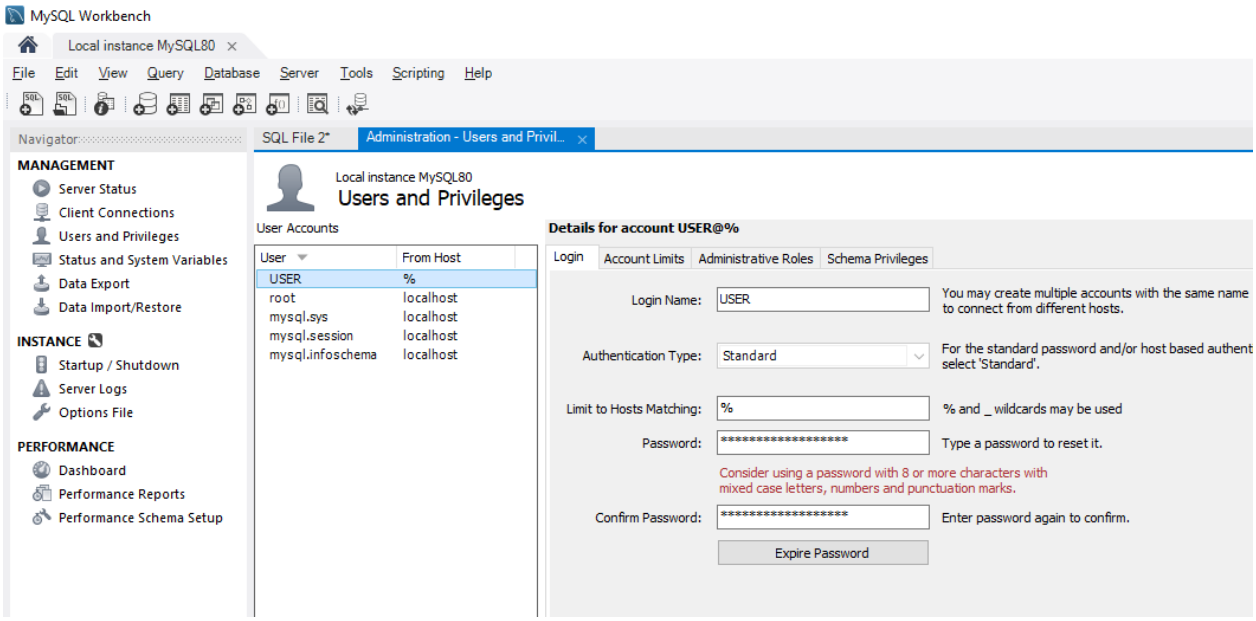


Ilustración 35: Detalle de la creación de usuario en MySQL

10.5 Creación de la base de datos

En la carpeta de la raíz del proyecto retroexchanges-rest-service, se encuentra un volcado con el contenido de una base de datos de pruebas (incluye usuario admin, las categorías y algunos videojuegos ya dados de alta). El fichero tiene por nombre **'retroexchanges_app.sql'.** Para su generación y posterior importación se ha empleado la herramienta Dbeaver.io

https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe

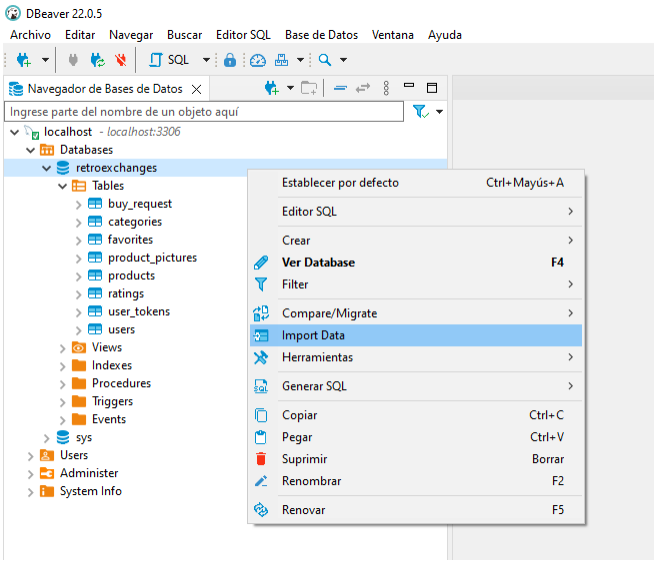


Ilustración 36: Importar fichero para crear la base de datos

10.6 Puesta en marcha del servicio API-REST

Para poder obtener todas las dependencias necesarias para el correcto funcionamiento del servidor emplearemos MAVEN.
Está disponible en la url https://dlcdn.apache.org/maven/maven-3/3.8.5/binaries/apache-maven-3.8.5-bin.zip

Descomprimiremos en c:\apache-maven-3.8.5 el contenido del zip descargado y crearemos la variable de entorno **MAVEN_HOME** en el sistema indicando donde hemos instalado Maven.

Será necesario actualizar la variable de entorno PATH, para incluir la ruta c:\apache-maven-3.8.5\bin de manera que podamos invocar los comandos de Maven desde la consola de Windows.

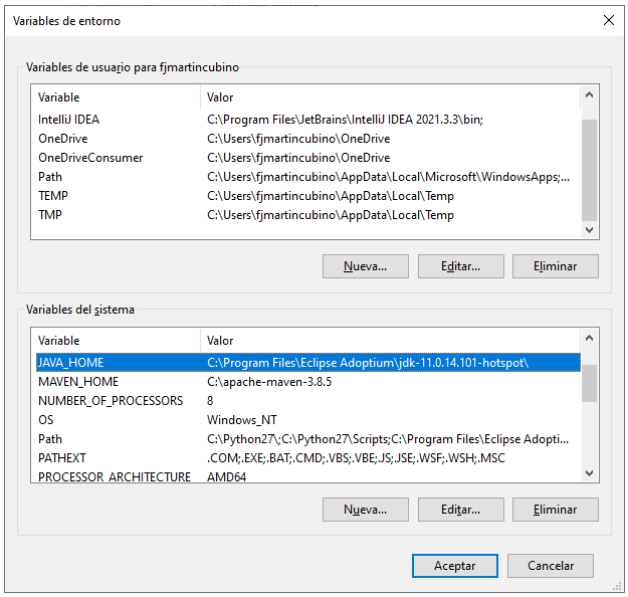


Ilustración 37: Variable de entorno MAVEN_HOME

Una vez configurado Maven, desde una consola Windows, accederemos a la ruta donde hemos descargado el código fuente de la aplicación. (en c:\git\retroexchanges-rest-service, por ejemplo)

Desde esa ubicación teclearemos **mvn package**

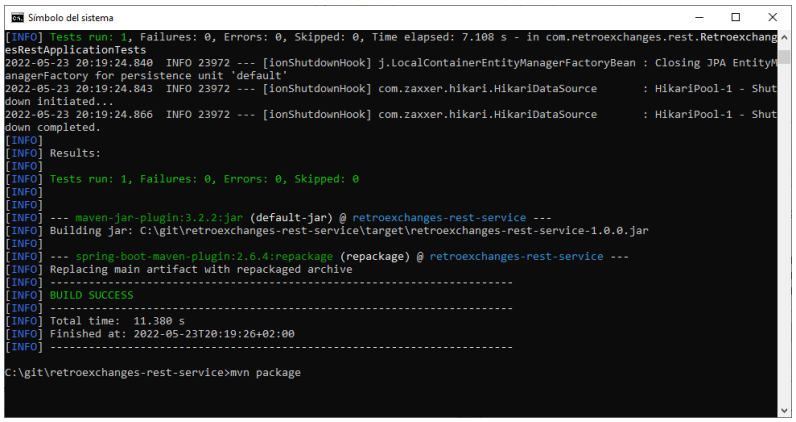


Ilustración 38: Descarga de dependencias del proyecto con maven

A continuación, ejecutaremos **mvn spring-boot:run.** De esta manera el servidor rest se iniciará escuchado en el puerto 8080.

10.7 Puesta en marcha del cliente

El cliente desarrollado (retroexchanges-rest-client) emplea el framework **vue.js** (https://vuejs.org/) en su versión 2 (https://v2.vuejs.org/).
Para la puesta en marcha del proyecto, es necesario descargar e instalar **nodejs** en su versión 16.14.2 https://nodejs.org/dist/v16.14.2/node-v16.14.2-x64.msi

Una vez instalada, ejecutaremos en la consola de Windows y en la ruta donde hemos descargado el código el comando **npm install**

(es necesario estar conectado a internet para que se descarguen todas las dependencias)

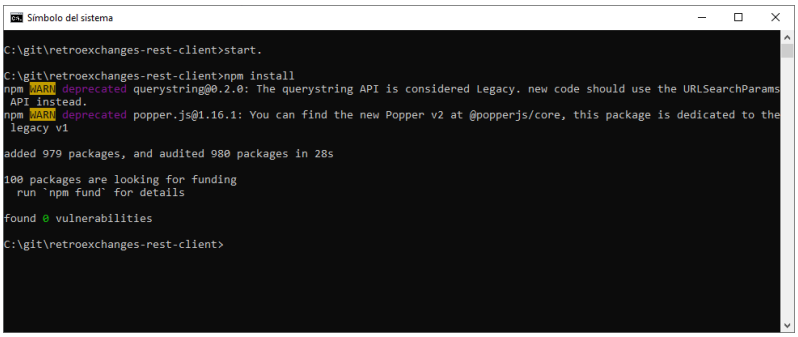


Ilustración 39: Instalación de las dependencias de vue.js

Una vez finalizada la descarga de dependencias, ejecutaremos en la consola el comando

```
npm run serve -- --port 8081
```

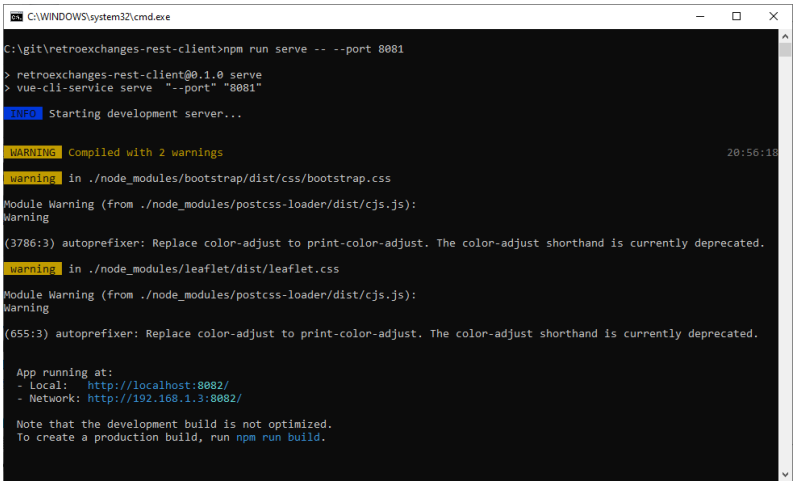


Ilustración 40: Detalle del cliente ejecutándose

10.8 Acceso a la aplicación

Una vez iniciada el servicio y el cliente accederemos a la aplicación empleando el navegador web (Chrome, Firefox,etc…) en la url http://localhost:8081.

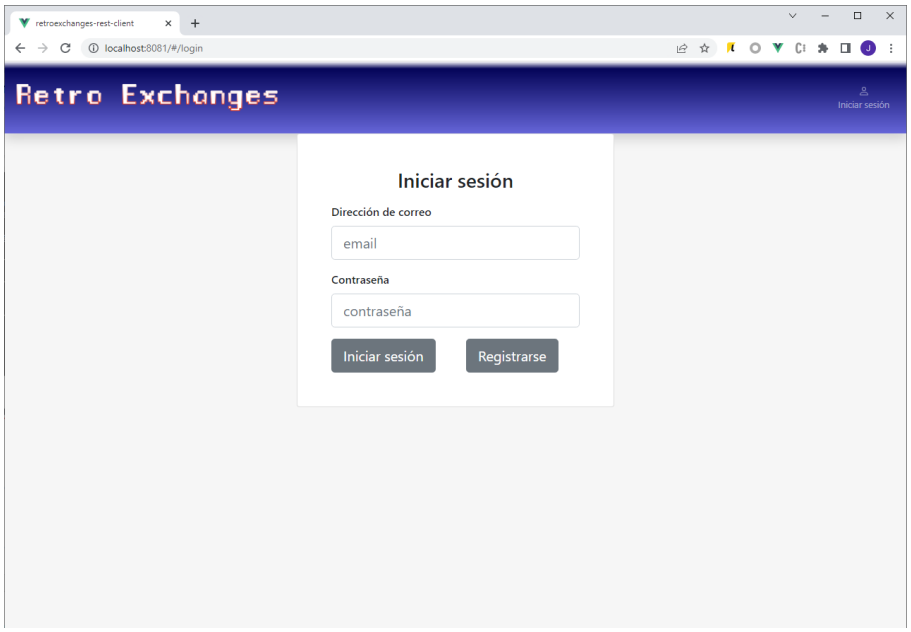


Ilustración 41: Ventana de login y acceso al sistema

El usuario administrador, es fjmartincubino@gmail.com y la contraseña es **zapatilla**

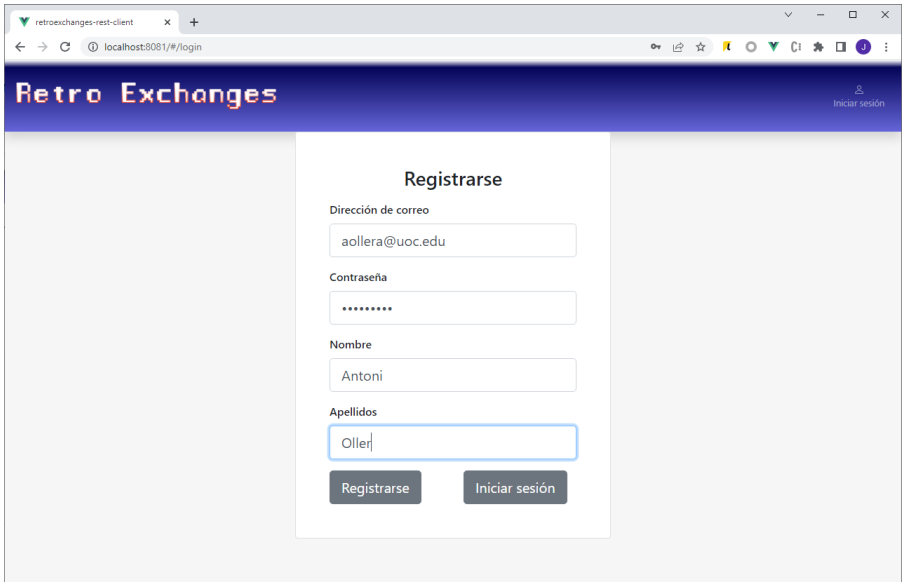


Ilustración 42: Ventana de registro

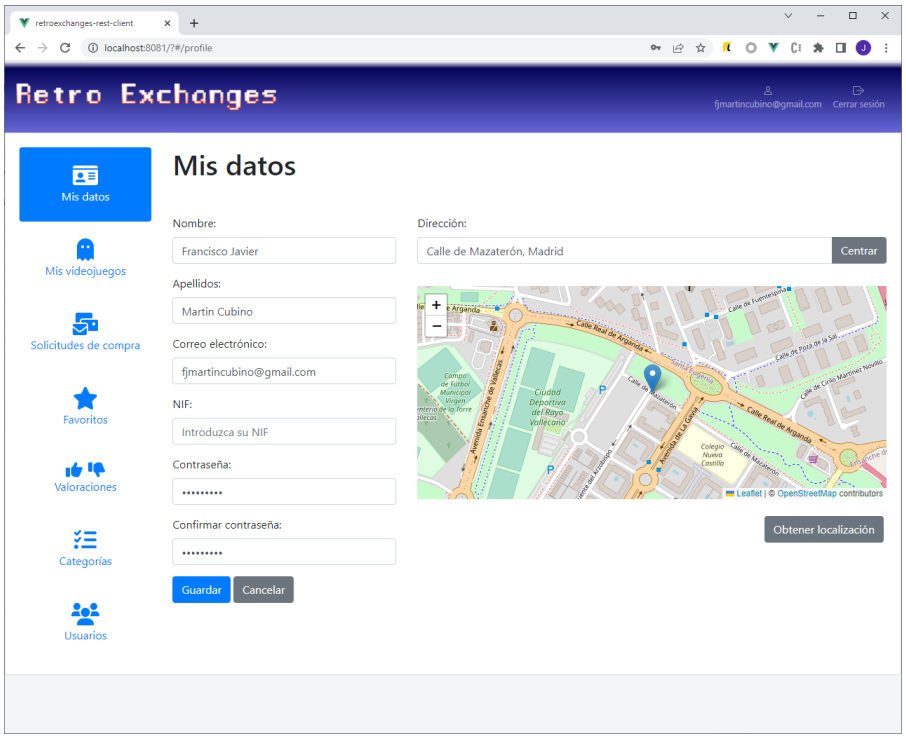


Ilustración 43: Panel de control del usuario administrador