

Índice.

1.Análisis del contexto.

1.1. Necesidades que justifiquen nuestro proyecto.....	2
--	---

2.Estudio de mercado.

2.1 Tipos de cliente.....	3
2.2 Prestaciones a los clientes.....	3

3.Diseño del proyecto.

3.1. Coste de implementación del proyecto para la empresa.....	4
3.2. Plataforma y requisitos para ejecutar el proyecto.....	5
3.3 Aplicación de la ley de protección de datos.....	5
3.4 Metodología de desarrollo de proyecto.....	5
3.5 Esquema de la base de datos del proyecto.....	6

4.Organización y ejecución.

4.1 Planing de ejecución del proyecto.....	7
4.2 Recursos utilizados para su desarrollo.....	7

5.Manual del Usuario.....8

6.Manual del Programador.....23

7.Bibliografía.....52

1. Análisis del contexto.

1.2. Detección de necesidades que justifiquen nuestro proyecto.

La industria de los videojuegos ha experimentado un crecimiento exponencial en las últimas décadas, convirtiéndose en una de las formas de entretenimiento más populares a nivel mundial. Este auge ha creado una necesidad creciente de plataformas que no solo vendan videojuegos, sino que también faciliten la interacción y el intercambio de información entre los jugadores.

El desarrollo de una aplicación web centrada en la venta de videojuegos, acompañada de un foro y un chat para la interacción entre usuarios, surge de la necesidad de satisfacer diversas demandas del mercado y la comunidad de los videojuegos. La industria de los videojuegos ha experimentado un crecimiento sin precedentes en los últimos años, convirtiéndose en una parte esencial del entretenimiento mundial. Este auge ha creado un entorno altamente competitivo donde tanto jugadores como desarrolladores buscan plataformas innovadoras y eficientes para comprar, vender y discutir sobre videojuegos.

La idea de este proyecto se justifica en primer lugar por el comportamiento de los consumidores en el ámbito de los videojuegos. Los jugadores no solo desean adquirir títulos nuevos, sino que también buscan recomendaciones y evaluaciones de otros usuarios para tomar decisiones de compra más infundadas. En este sentido, una tienda de videojuegos que incluya un sistema de comentarios y valoraciones satisface esta necesidad crítica, proporcionando un espacio donde las experiencias de los usuarios pueden guiar a otros con sus recomendaciones.

Además, los videojuegos no son simplemente productos de consumo, sino que también sirven para formar nuevas comunidades entre los jugadores supliendo la necesidad de socializar. Los jugadores a menudo buscan conectarse con otros que compartan sus intereses, ya sea para discutir estrategias, compartir consejos o simplemente socializar. La introducción de un foro y un chat en la aplicación responde a esta demanda, creando un espacio donde los usuarios pueden interactuar, formar amistades y construir una comunidad en torno a sus juegos favoritos. Esta funcionalidad no solo mejora la experiencia del usuario, sino que también incrementa la fidelización y el compromiso con la plataforma.

El contexto sociocultural también apoya la pertinencia de este proyecto. La "cultura gamer" ha dejado de ser un nicho para convertirse en una tendencia dominante en la sociedad moderna. Con una base de usuarios que abarca todas las edades, géneros y lugares, la demanda de plataformas inclusivas y accesibles es más alta que nunca. Nuestra aplicación web busca ser un reflejo de esta diversidad, ofreciendo un entorno seguro y acogedor donde todos los jugadores se sientan a gusto y se diviertan.

En conclusión, este proyecto no solo soluciona a una necesidad evidente en el mercado de videojuegos, sino que también comparte las tendencias actuales de la cultura digital y las expectativas de los consumidores. Al combinar las ventas de videojuegos con un foro para la comunidad y un chat entre usuarios, la aplicación ofrece una mejora a la experiencia del usuario y fomenta una comunidad de jugadores activa y comprometida.

2.Estudio de mercado.

El mercado de los videojuegos ha experimentado un crecimiento continuo, con una amplia base de consumidores que abarca cualquier lugar y edad. Los videojuegos suponen uno de los métodos de entretenimientos más extendidos de la actualidad. Según datos recientes de la Asociación de Software de Entretenimiento (ESA, por sus siglas en inglés), el mercado global de los videojuegos generó más de 150 mil millones de dólares en ingresos en 2023, superando a cualquier año anterior y se espera que continúe creciendo a un ritmo constante en los próximos años.

Este crecimiento se debe a los avances en la tecnología, la proliferación de dispositivos móviles, consolas de juegos y plataformas de transmisión en línea, que han ampliado el acceso a los videojuegos y han creado nuevas oportunidades para los desarrolladores y distribuidores. Además, la pandemia de COVID-19 ha acelerado la demanda de entretenimiento digital, con más personas buscando formas de conectarse y entretenerse desde la comodidad de sus hogares.

Por último, los videojuegos multijugador han supuesto un nuevo método de socialización en la última década por lo que es importante la interacción de los propios usuarios entre sí.

2.1.Tipos de clientes.

Este proyecto podrá venderse y podrá ser implantado en los siguientes tipos de clientes:

Empresas de venta de productos: Aunque está basada en los videojuegos, también se podría derivar a otro tipo de tema como música, libros, etc. Existen muchas empresas en el mercado que quieren mejorar interfaz de usuarios o muchas otras que desean introducirse en este comercio y necesitan digitalizarse.

2.2. Prestaciones a los clientes.

Los clientes que adquieran este software, podrán beneficiarse de:

Mejorar la experiencia de sus clientes: Con un diseño atractivo e intuitivo, combinado con el uso de sistemas que fomentan la interacción entre usuarios, además de la socialización.

Mantenimiento: Ofrecemos 5 meses de supervisión para garantizar el perfecto funcionamiento de la aplicación así como la corrección de los posibles errores que se puedan encontrar.

Actualizaciones: Desarrollo de nuevas funciones que se deseen implementar en la página web para mejorar su funcionamiento o ampliar su impacto en los consumidores de la aplicación.

3.Diseño del Proyecto.

3.1.Coste e implementación del proyecto.

Aunque para el desarrollo de la aplicación no se han tenido costes en cuenta debido a que se utilizan tecnologías totalmente gratuitas y el almacenamiento de datos se hace en una base de datos local, si la aplicación crece exponencialmente y se viera obligada a contener una mayor cantidad de datos, se requerirá cambiar la base de datos por una que permita un mayor numero de peticiones y almacenamiento. El coste de esta dependerá de la base de datos que se desee imponer para alojar los datos.

Para el alojamiento del servidor, al principio se podría usar Hostinger que ofrece un soporte para aplicaciones pequeñas y medianas donde cuenta con diversas tarifas según nuestros intereses.

Single	Premium	Business	Cloud Startup
La solución ideal para principiantes	El paquete perfecto para webs personales	Sube de nivel con más potencia y funciones mejoradas	Disfruta de un rendimiento optimizado y recursos potentes
7,99 € AHORRA 81%	11,99 € AHORRA 78%	14,99 € AHORRA 73%	19,99 € AHORRA 50%
1,49 €/mes	2,59 €/mes	3,99 €/mes	9,99 €/mes
<small>*Con pedidos de 48 meses; IVA no incluido</small>	<small>*Con pedidos de 48 meses; IVA no incluido</small>	<small>*Con pedidos de 48 meses; IVA no incluido</small>	<small>*Con pedidos de 48 meses; IVA no incluido</small>
+3 meses gratis	+3 meses gratis	+3 meses gratis	+3 meses gratis
Añadir al carro	Añadir al carro	Añadir al carro	Añadir al carro
2,99 €/mes al renovar	5,99 €/mes al renovar	7,99 €/mes al renovar	16,99 €/mes al renovar
Funciones principales	Funciones principales	Funciones principales	Funciones principales
<ul style="list-style-type: none">✓ 1 Sitio web✓ Rendimiento estándar✓ 50 GB de SSD✓ Copias de seguridad semanales✓ 1 cuenta de email✓ SSL ilimitado gratis✓ 100 GB de ancho de banda✗ Dominio incluido (9,99 €)✗ CDN Gratis	<ul style="list-style-type: none">✓ 100 sitios web✓ Rendimiento estándar✓ 100 GB de SSD✓ Copias de seguridad semanales✓ Email gratis✓ SSL ilimitado gratis✓ Ancho de banda ilimitado✓ Dominio incluido (9,99 €)✗ CDN Gratis	<ul style="list-style-type: none">✓ 100 sitios web✓ Mayor rendimiento (hasta 5 veces)✓ 200 GB de almacenamiento NVMe✓ Copias de seguridad diarias (valor: 23,88 €)✓ Email gratis✓ SSL ilimitado gratis✓ Ancho de banda ilimitado✓ Dominio incluido (9,99 €)	<ul style="list-style-type: none">✓ 300 sitios web✓ Rendimiento máximo (hasta x10)✓ 200 GB de almacenamiento NVMe✓ Copias de seguridad diarias (valor: 23,88 €)✓ Email gratis✓ SSL ilimitado gratis✓ Ancho de banda ilimitado✓ Dominio incluido (9,99 €)
Detalles técnicos	Detalles técnicos	Detalles técnicos	Detalles técnicos
<ul style="list-style-type: none">✓ 200 000 archivos y directorios (inodos)✓ 25 PHP workers✓ ~10 000 visitas al mes✓ 2 subdominios✓ 25 conexiones en MySQL MAX✓ 2 bases de datos✓ 1 cuenta FTP✓ 2 Cronjobs✓ Acceso GIT✓ Múltiples versiones de PHP✓ Gestión de DNS✓ Administrador de caché✓ Potente panel de control✗ Acceso SSH	<ul style="list-style-type: none">✓ 400 000 archivos y directorios (inodos)✓ 40 PHP workers✓ ~25 000 visitas al mes✓ 100 subdominios✓ 50 conexiones en MySQL MAX✓ Bases de datos ilimitadas✓ Cuentas FTP ilimitadas✓ Cronjobs ilimitados✓ Acceso GIT✓ Múltiples versiones de PHP✓ Gestión de DNS✓ Administrador de caché✓ Potente panel de control✓ Acceso SSH	<ul style="list-style-type: none">✓ 600 000 archivos y directorios (inodos)✓ 60 PHP workers✓ ~100 000 visitas al mes✓ 100 subdominios✓ 75 conexiones en MySQL MAX✓ Bases de datos ilimitadas✓ Cuentas FTP ilimitadas✓ Cronjobs ilimitados✓ Acceso GIT✓ Múltiples versiones de PHP✓ Gestión de DNS✓ Administrador de caché✓ Potente panel de control✓ Acceso SSH	<ul style="list-style-type: none">✓ 2 000 000 archivos y directorios (inodos)✓ 100 PHP workers✓ ~200 000 visitas al mes✓ 300 subdominios✓ 100 conexiones en MySQL MAX✓ Bases de datos ilimitadas✓ Cuentas FTP ilimitadas✓ Cronjobs ilimitados✓ Acceso GIT✓ Múltiples versiones de PHP✓ Gestión de DNS✓ Administrador de caché✓ Potente panel de control✓ Acceso SSH
Ver menos características	Ver menos características	Ver menos características	Ver menos características

3.2. Si es un producto software, plataforma donde se ejecutará el proyecto y otros requisitos que deba tener la plataforma.

El proyecto está desarrollado para poder ser usado en cualquier plataforma, pero si que necesita tener unos requisitos mínimos del sistema para poder aguantar la aplicación. Los requisitos mínimos que se requieren son 8 Gb de memoria RAM y un espacio de almacenamiento mínimo de 5 Gb, que se verá incrementado dependiendo de la cantidad de datos que maneje la aplicación y de como se decida guardar la base de datos.

3.3. Aplicación de la ley de protección de datos en el proyecto.

Nuestra aplicación al manejar datos personales de los usuarios, tales como el nombre, los apellidos y el numero de teléfono, debemos regirnos a la normativa del RGPD (Reglamento General de Protección de Datos). Para ello, debemos cumplir una serie de normas como el consentimiento del usuario, el derecho del usuario a modificar esta información cuando lo requiera, la justificación del uso de los datos y su seguridad.

3.4. Metodología que se ha seguido para el desarrollo del proyecto.

A la hora de desarrollar la aplicación, hemos seguido los siguientes pasos:

1. Planificación general del proyecto.

Elegir el tema sobre el que trata el proyecto además del enfoque que le daría a este y y cuanto se iba a profundizar teniendo en cuenta el tiempo disponible y los recursos con los que se contaban.

2. Seleccionar las tecnologías para desarrollar el proyecto.

Elegir el entorno en el que iba a desarrollar la aplicación, en mi caso MySQL con Xampp, SpringBoot y angular. Decidí usar SpringBoot y angular puesto que son usados en muchas empresas y eso me serviría para introducirme en ellos y practicar.

3. Aprendizaje de las características generales de las tecnologías usadas.

En mi caso me llamó la atención desarrollar el FrontEnd en Angular debido a que es bastante utilizado y me tuve que iniciar en los conceptos generales y aprender como se estructura un proyecto.

4. Elaborar la base de datos y sus relaciones.

Una vez tuve claro la profundidad que le iba a dar a el proyecto y determinar las entidades necesarias junto con sus atributos, desarrollé las diferentes tablas que requería y sus respectivas relaciones.

5. Desarrollo del FrontEnd y BackEnd.

Por último, tras planificar una idea general, se procede a desarrollar la aplicación. En primer lugar se realizó la instalación de los entornos de desarrollo necesarios, en este caso Xampp con MySQL, SpringBootToolSuite y Angular.

A continuación, se creó el proyecto tanto del FrontEnd y del BackEnd y la asociación de la base de datos que se iba a usar.

Una vez todo creado, se procede a conectar las diferentes partes de la aplicación y se comprueba que funciona correctamente la transmisión de datos.

Al comprobar que todo funciona correctamente y esta todo conectado, se empieza a crear los componentes, empezando por las vistas que van a soportar los datos y posteriormente las llamadas al servidor y terminando por la lógica y la respuestas del servidor.

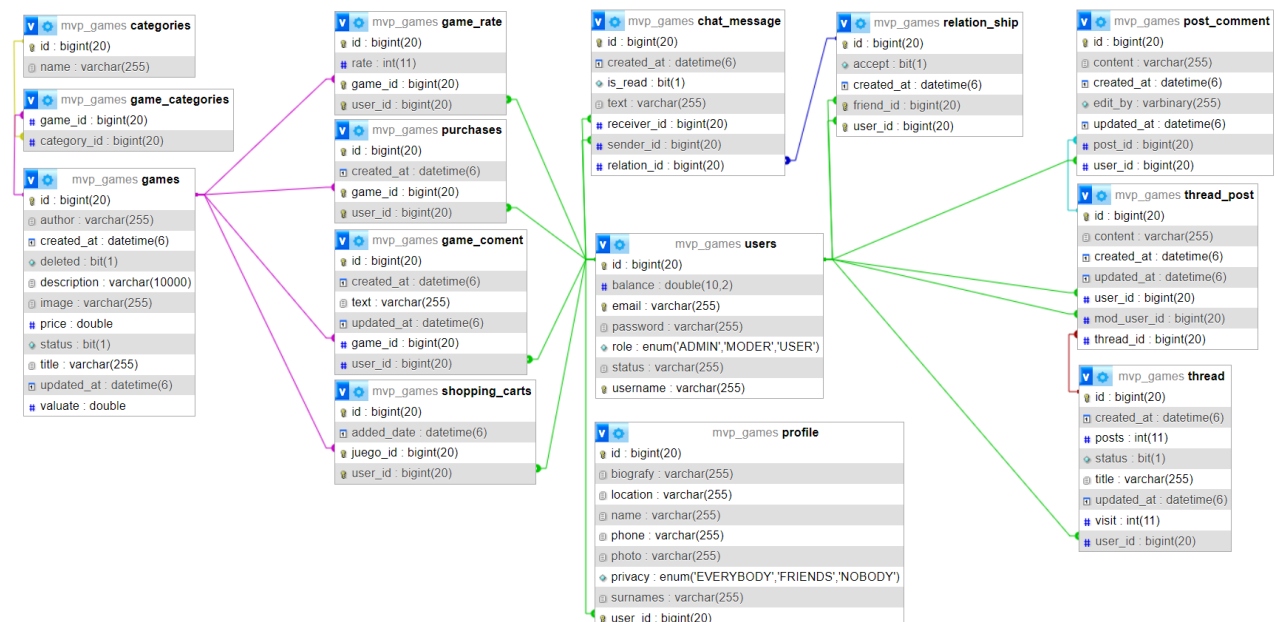
6.Pruebas y corrección de errores.

Tras terminar todos los componentes y sus funciones asociadas, se realiza una exhaustiva serie de pruebas para comprobar que todo funciona correctamente y se realizan posibles mejoras de rendimiento a la vez que se solucionan errores en la aplicación.

3.5.Eschema de Bases de datos sobre el que se va a basar el proyecto.

Para la base de datos, se ha utilizado MySQL con phpMyAdmin(Xampp).

El esquema de datos en el que está basado el proyecto es el siguiente:



- users - profile: 1 a 1
- users - chat_message: 1 a N
- users - relation_ship: 1 a N
- users - post_comment: 1 a N
- users - thread_post: 1 a N
- users - thread: 1 a N
- users - shopping_carts: 1 a N
- users - game_coment: 1 a N
- users - purchases: 1 a N
- users - game_rate: 1 a N
- games - game_categories: 1 a N
- games - game_coment: 1 a N
- games - purchases: 1 a N

4.Organización de la ejecución.

4.1.Planing de ejecución del proyecto a lo largo del periodo Marzo-Junio 2024.

El planteamiento del proyecto ha sido un poco más lento de lo normal debido a que se usan nuevas tecnologías de desarrollo en las cuales en algunos momentos surgían problemas desconocidos para mi persona.

Primera quincena de marzo: Investigación y aprendizaje sobre la implementación del sistema de seguridad de Spring Boot Security junto con la introducción de un token de autenticación. También sobre los conceptos generales de como usar angular y como se distribuyen sus componentes.

Segunda quincena de marzo: Creación de las vistas para iniciar sesión y registrarse junto con sus respectivas validaciones tanto en el cliente como en el servidor y además con la implementación del sistema de seguridad en el backend con generación del token de autenticación.

Primera quincena de abril: Creación de los componentes globales como la barra de navegación y el footer. También se realizó la parte de gestión administrativa sobre los producto y los usuarios registrados con sus vistas y su CRUD correspondiente.

Segunda quincena de abril: Implementación de la vista principal de la aplicación y la vista de un producto en específico, incorporando un sistema de valoración del producto y comentarios de los usuarios.

Primera quincena de mayo: Integración de compras de productos, añadiendo también una lista de deseados que actúa como carrito. Además se incorporó, las vistas necesarias para ingresar dinero en la cartera del usuario.

Segunda quincena de mayo: Implementación del sistema de lectura, moderación y creación de foros para los usuarios además de un chat a tiempo real entre los jugadores y corrección de errores generales de la aplicación.

4.2 Recursos utilizados para el desarrollo.

Para el desarrollo del proyecto, el único recurso que se ha utilizado es mi ordenador portátil personal y aunque se podría usar almacenamiento en la nube para almacenar los datos y las imágenes se ha optado por el almacenamiento local debido a la cantidad limitada de datos que se necesitan para mostrar el potencial de la aplicación.

MANUAL DE USUARIO

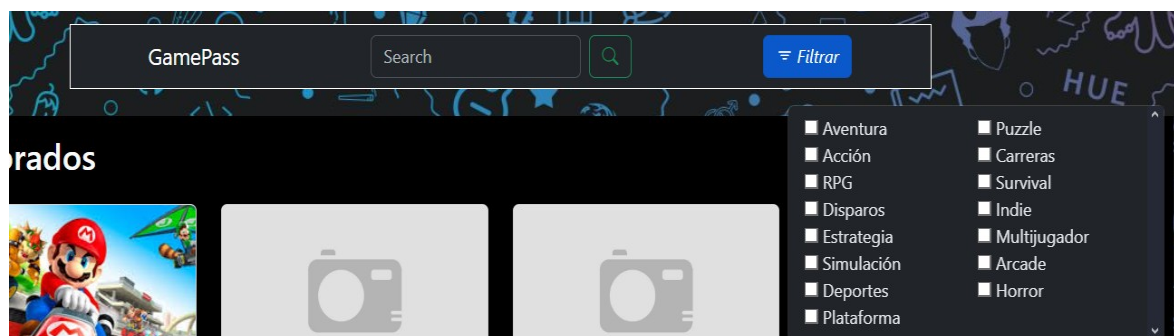
5.Manual de usuario.

Objetivo de la aplicación.

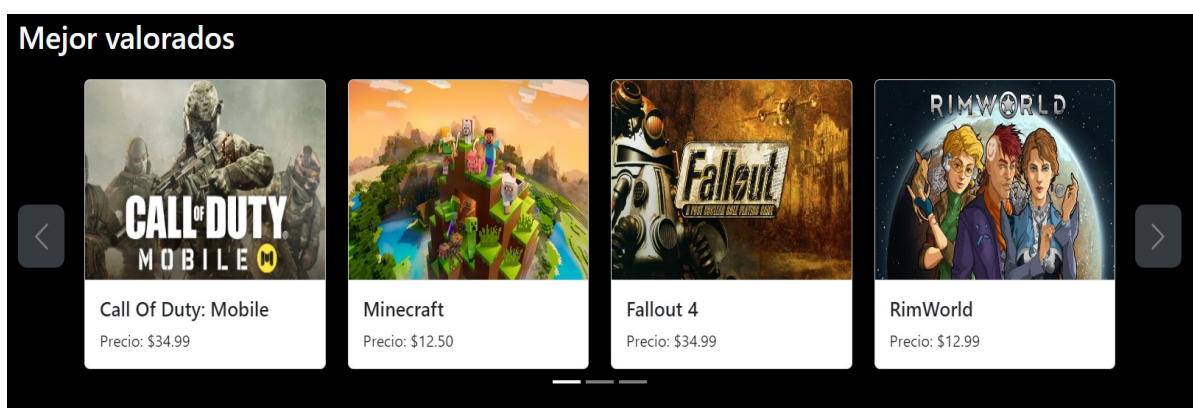
El objetivo de la aplicación es que los clientes puedan realizar las compras de los juegos que deseen basándose en las opiniones de otros usuarios a la vez que se sientan en un entorno cómodo y agradable que les permita interactuar con el resto de usuarios ya sea desde hablar por un chat a tiempo real, crear y responder en los foros de la comunidad.

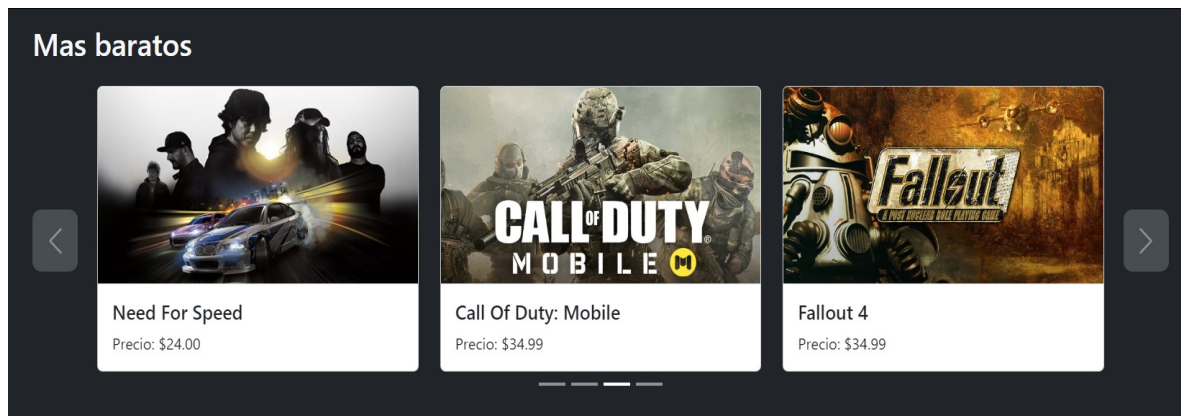
Recorrido de la aplicación.

Al iniciar la aplicación nos veremos en la pagina principal que cuenta con varias partes. Arriba del todo tenemos una barra de herramientas donde se nos permite filtrar la lista global de productos por nombre o por categoría.

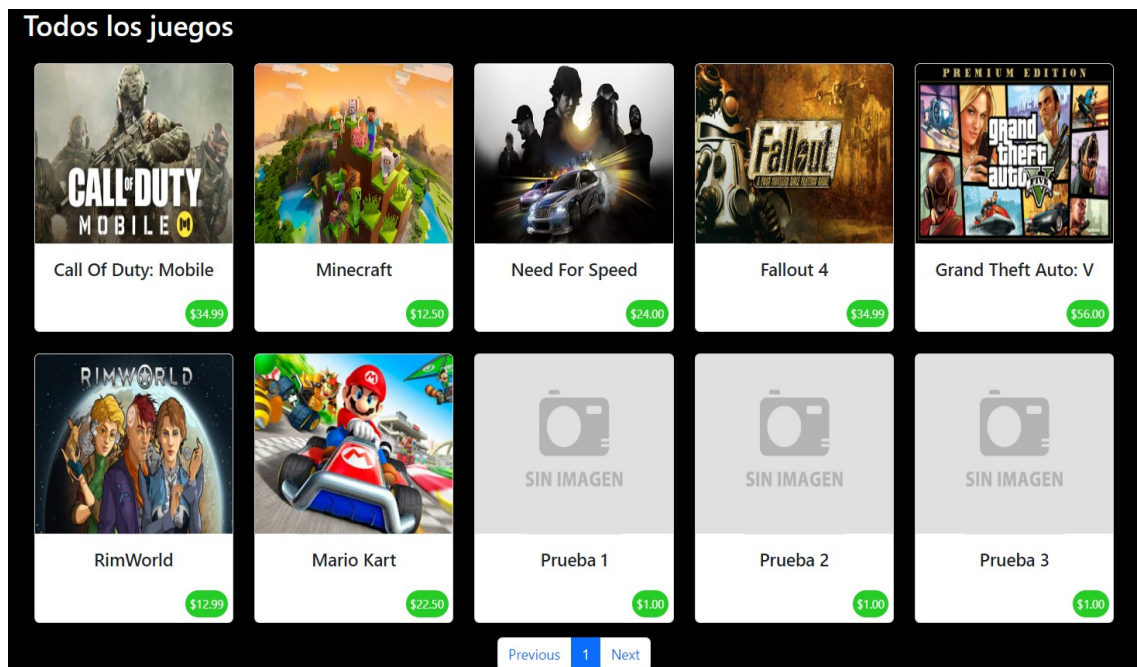


A continuación se muestran dos selecciones de juegos, una de los mejor valorados y otra de los mas baratos a modo de carrusel,mostrando los 15 juegos más característicos de este tipo.

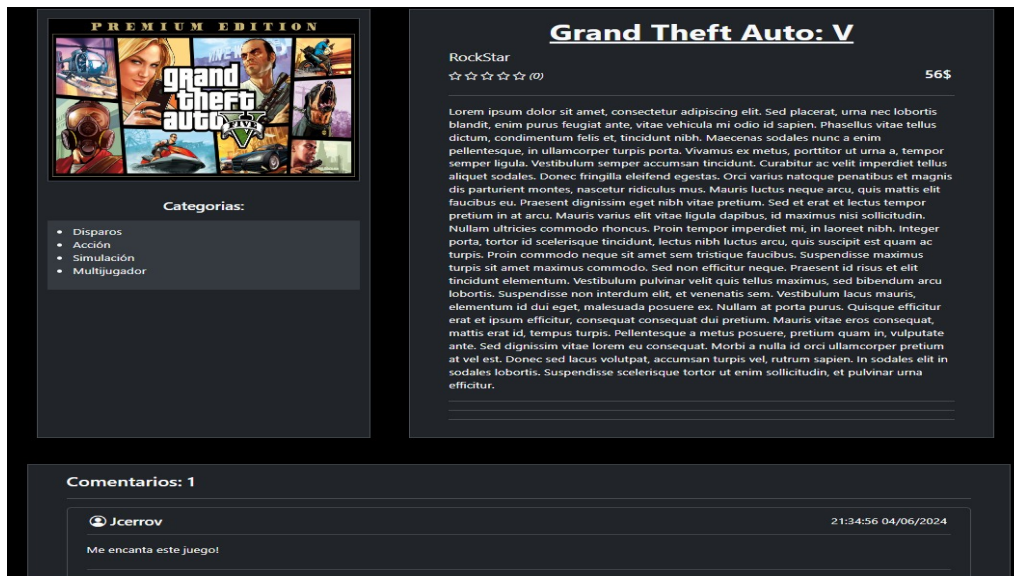




Por último tenemos una ultima lista donde se muestran todos los juegos paginados con 30 productos por pagina.



Al seleccionar un producto del catalogo, podremos ver sus características específicas junto con los comentarios que han hecho otros usuarios registrados sobre el producto. Si no se esta registrado no se permite hacer ninguna acción que no sea de lectura.



La barra de navegación cambiará una vez hayamos iniciado sesión.



Cuando seleccionamos registrar, nos mostrará un formulario para crear un nuevo usuario. El formulario cuenta con validaciones en los campos las cuales hacen muy intuitivo el registro.

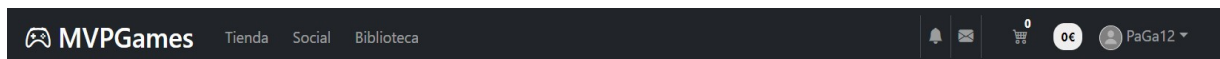
The image shows a registration form on the MVPGames website. The form is titled 'Registrarse' and features a user icon. It contains several input fields: 'Nombre' (with 'Paco' entered), 'Apellidos' (with 'Garcia' entered), 'Nombre de usuario' (with 'PaGa1d' entered), 'Correo electrónico' (with 'Paga@gmail.com' entered), 'Nº telefono (Opcional)', 'Contraseña' (with '*****' entered), 'Confirmar contraseña' (with '*****' entered), and 'Región (Opcional)' (a dropdown menu). There is a checkbox for 'He leído y acepto los términos y condiciones.' and a blue 'Registrarse' button. At the bottom, there's a link for 'Si ya tienes una cuenta, inicia sesión.'

Por otra parte, si ya tenemos un usuario creado o deseamos hacerlo mas tarde, al elegir la opción de inicio de sesión nos conducirá a otro formulario donde introducir el usuario y la contraseña y validando si es correcto.

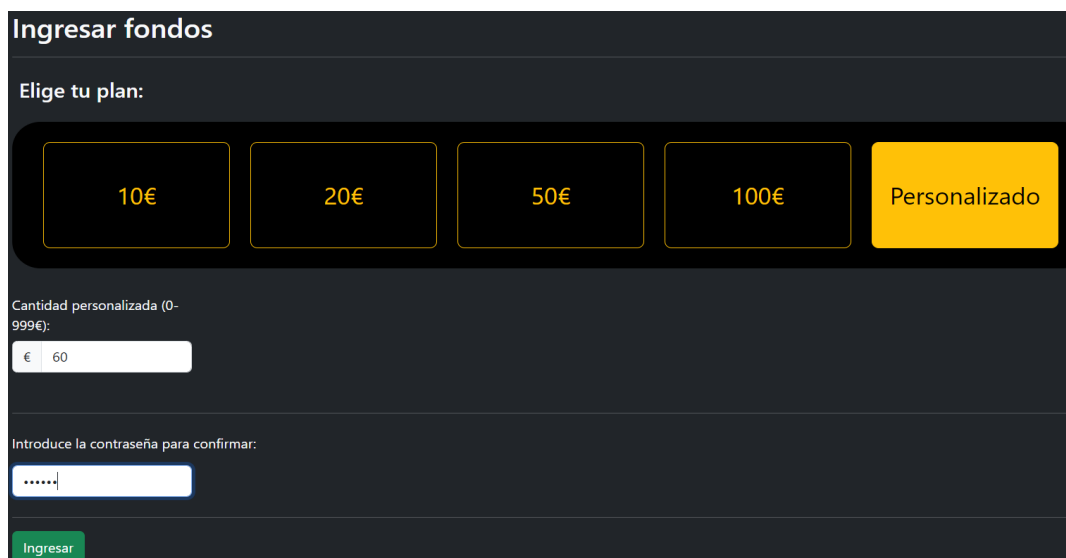


The login form is titled "Iniciar Sesión" and features a user icon at the top. It contains two input fields: "Usuario" with a placeholder "Nombre de usuario" and a red error message "Introduce el nombre de usuario"; and "Contraseña" with a placeholder "Contraseña" and a link "¿Has olvidado la contraseña?". Below the fields is a blue "Iniciar Sesión" button and a checkbox for "Mantener sesión". At the bottom, it says "Si todavía no tienes una cuenta, [regístrate](#)."

Una vez registrado, tendremos acceso a la mayor parte de las funciones de la pagina.



Antes de comprar un producto es importante añadir efectivo a la cartera del usuario. Al hacer click en el saldo del usuario nos mandará a una ventana donde se nos permite seleccionar cantidades especificas del saldo que se desea ingresar o personalizar la cantidad, ademas de introducir la contraseña para evitar usos indebidos.

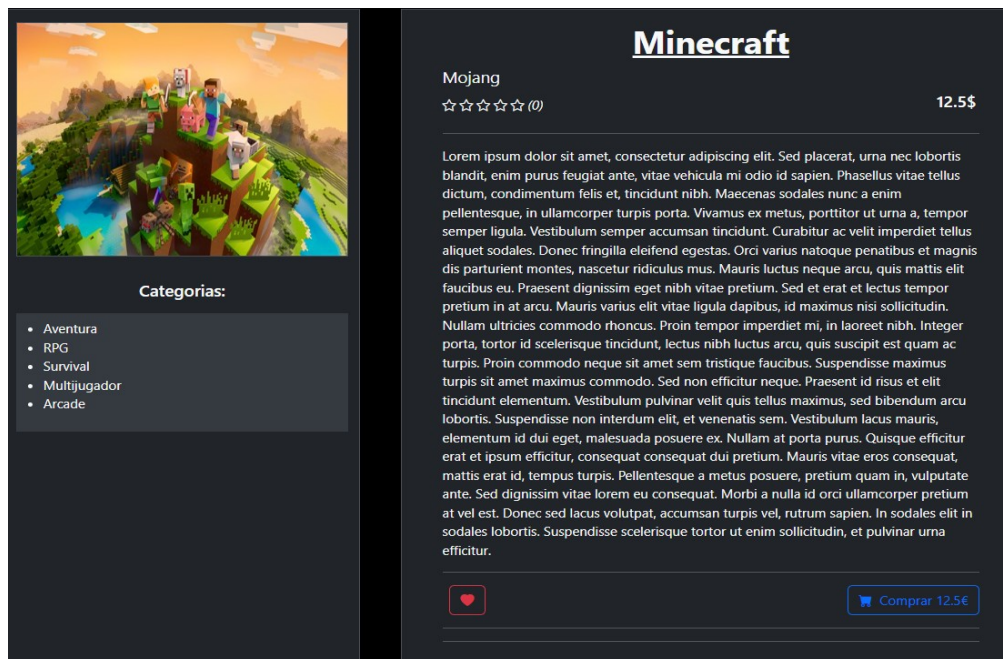


The "Ingresar fondos" form allows users to add funds. It starts with "Elige tu plan:" and offers options: 10€, 20€, 50€, 100€, and "Personalizado". Below, a "Cantidad personalizada (0-999€):" section has an input field showing "€ 60". A "Introduce la contraseña para confirmar:" field with a masked password "....." is also present. A green "Ingresar" button is at the bottom.

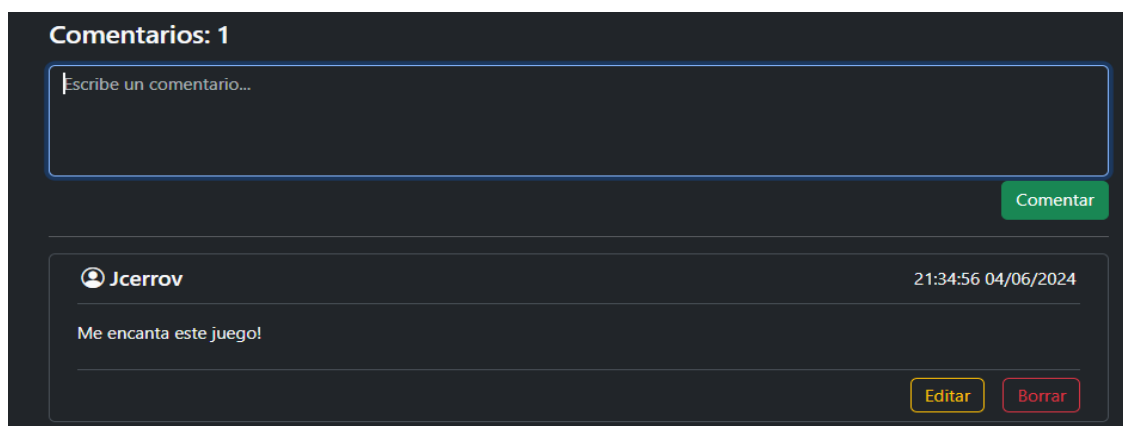
Al aceptar el ingreso se actualizará el saldo de la barra de tareas.



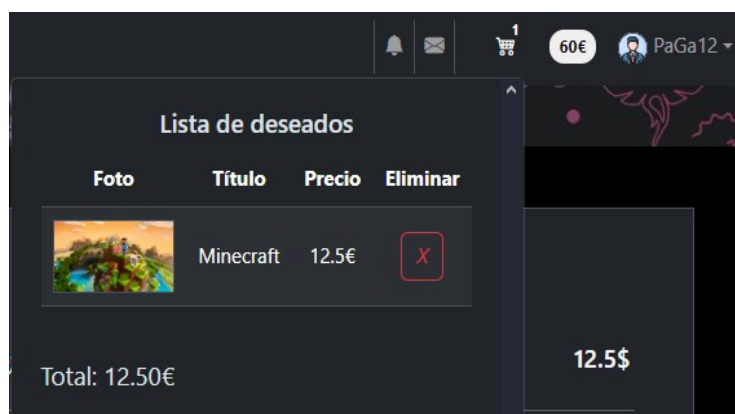
Ahora si nos dirigimos a un juego en concreto tendremos las opciones de añadirlo a la lista de deseos o comprarlo





También podremos publicar comentarios, editar o eliminar uno ya enviado.





Al añadir a la lista de deseos, se podrá visualizar en la barra de navegación y eliminarlo de la lista si se desea.



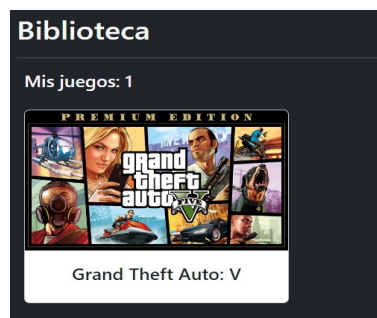
Si se desea, al hacer click sobre “Lista de deseados” nos conducirá a una vista donde se muestra mejor la lista y permite comprar todos los de la lista directamente.

Lista de favoritos					
Foto	Título	Desarrollador	Precio	Agregado	Eliminar de la lista
	Minecraft	Mojang	12.5	06/06/2024 16:47	
	Call Of Duty: Mobile	Activision	34.99	06/06/2024 16:51	
Total: 47.49€					

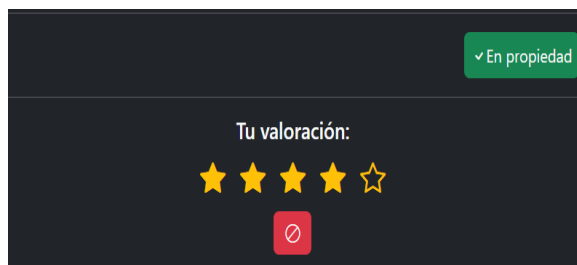
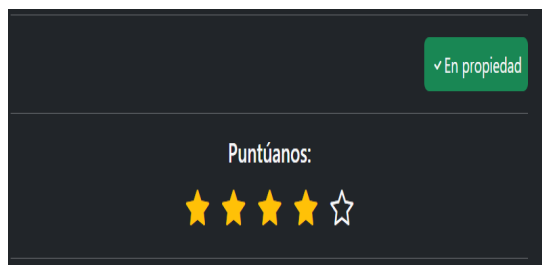
Al comprar un juego o varios, nos mostrará un resumen de la compra antes de efectuarla.

Resumen de la compra	
Usuario: PaGa12	
Importe de la cuenta:0€	
Juegos:	
Imagen	Título
	Grand Theft Auto: V
Desarrollador	Precio
RockStar	56€
Detalles:	
-56.00€	
Total:56.00€	
	

Una vez realizada la compra, veremos que el producto se encuentra en nuestra biblioteca, donde aparecerán todos los que tengamos adquiridos.

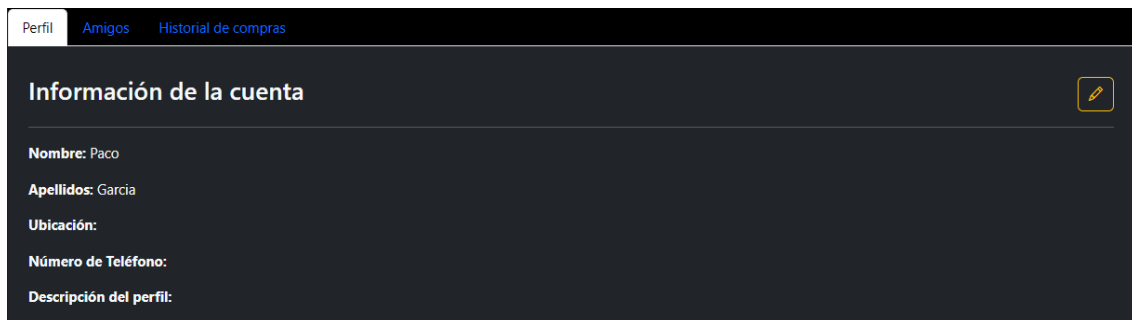


Ahora también nos permitirá realizar una valoración sobre el producto comprado o eliminarla.



Si hacemos click en el nombre de usuario en ubicado en la barra de navegación superior, tendremos la opción de cerrar la sesión o ir a nuestro perfil.

En nuestro perfil, se podrán encontrar los datos personales del usuario, los cuales pueden ser modificados.



Perfil Amigos Historial de compras

Información de la cuenta

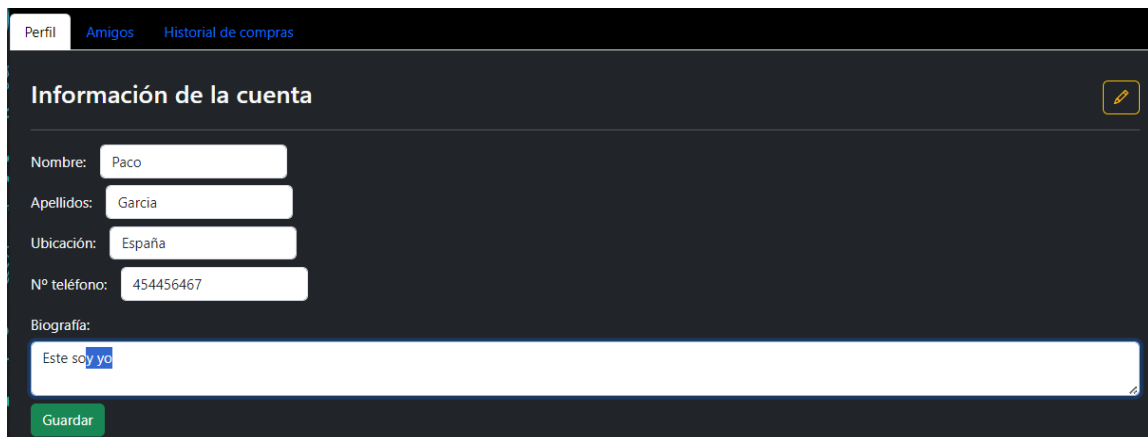
Nombre: Paco

Apellidos: García

Ubicación:

Número de Teléfono:

Descripción del perfil:



Perfil Amigos Historial de compras

Información de la cuenta

Nombre: Paco

Apellidos: García

Ubicación: España

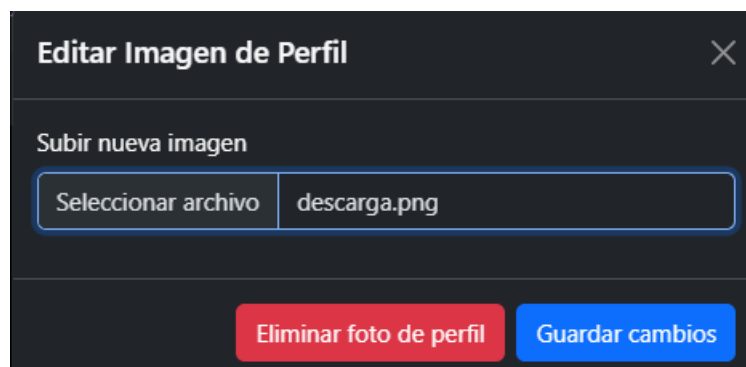
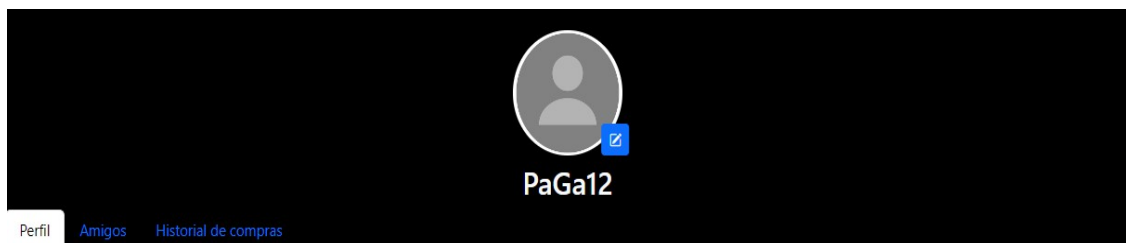
Nº teléfono: 454456467

Biografía:

Este soy yo

Guardar

También podemos cambiar nuestra foto de usuario. Clickando en el boton de la foto y se mostrará un ventana emergente.



Editar Imagen de Perfil

Subir nueva imagen

Seleccionar archivo descarga.png

Eliminar foto de perfil Guardar cambios

También podremos cambiar la contraseña y los ajustes de privacidad.

Privacidad

Quien puede ver tu perfil:

Solo amigos

Nadie

Solo amigos

Todos

Introduce la contraseña actual

Introduce la nueva contraseña

Confirma la nueva contraseña

Cambiar contraseña


También hay una pestaña que muestra las compras realizadas y otra que muestra los amigos que tiene agregados.

Perfil


Amigos

Historial de compras

Historial de Compras

Foto	Título	Desarrollador	Costo	Obtenido
	Grand Theft Auto: V	RockStar	56	04/06/2024

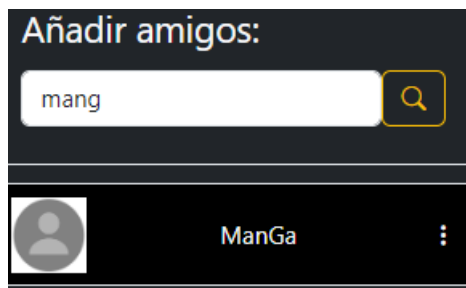
Amigos: 1



Jcerrov

Si nos vamos al apartado social tendremos la opción de buscar amigos o visitar los foros de la comunidad.

Si nos direccionamos a amigos, nos mostrara 3 listas. Una para los amigos ya agregados, otra para los nos han mandado invitación y otra a los que hemos enviado invitación. Además si deseamos agregar un nuevo usuario, podremos realizar una búsqueda por nombre de usuario y elegir el usuario que queramos y ver su perfil.

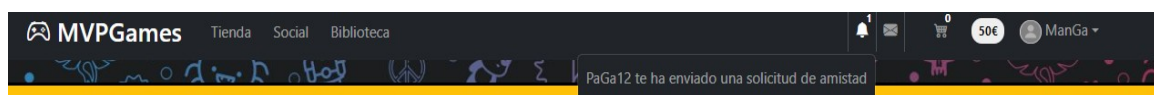


Dependiendo de la privacidad que tenga configurada el usuario, podrás ver el usuario o no.

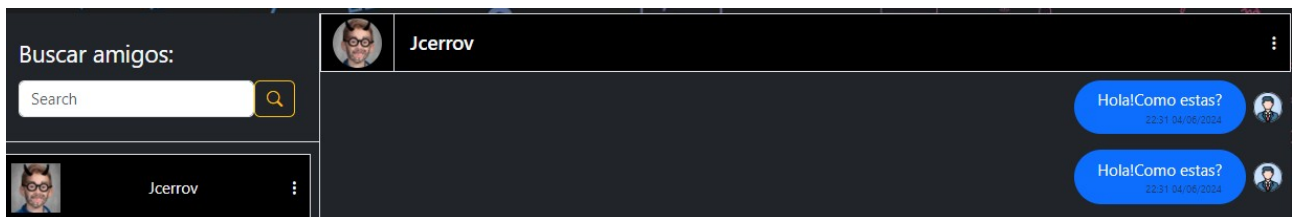


Como se puede ver, también se puede mandar petición desde ahí, eliminarlo o mandarle el mensaje.

Cuando un usuario recibe una petición de amistad, se refleja la notificación en el encabezado para que sea más facil de visualizar.



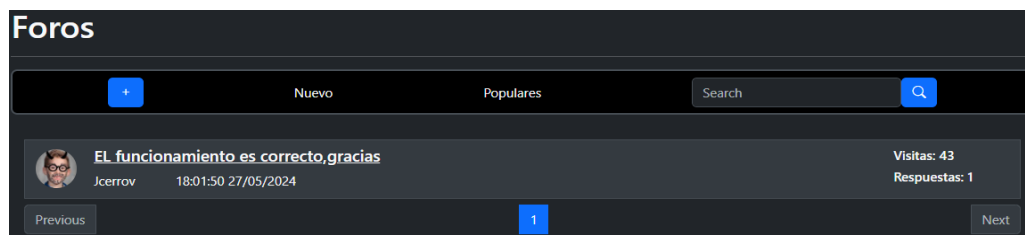
A los amigos que tengamos agregados podemos eliminarlos o mandarles un mensaje privado por chat a tiempo real.



Al mandar un mensaje tambien se reflejará en la cabecera de la página.



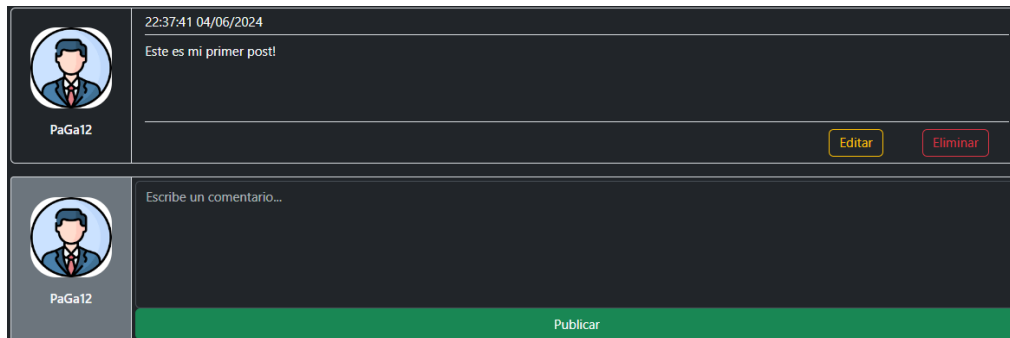
También tenemos otra opción en social donde podremos ver un listado paginado de los hilos de la comunidad y la opcion de crear uno nuevo al estar registrados.



Rellenamos los campos y podremos publicarlo.

A screenshot of a form titled 'Crear un nuevo hilo'. It has a title field labeled 'Título del hilo' and a comment field labeled 'Comentario del hilo' with a placeholder 'Escribe el contenido de tu hilo...'. At the bottom, there is a green button labeled 'Crear nuevo hilo'.

Al seleccionar un hilo en concreto, podremos ver las respuestas que este contiene. En él podrás publicar tu propia respuesta, o modificar y eliminar tus publicaciones.

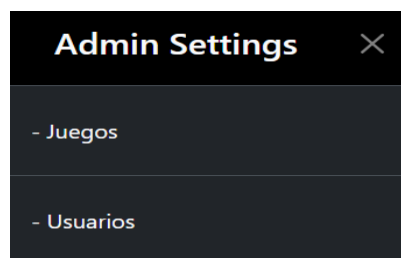


Si el usuario tiene rol de moderador podrá hacer las siguientes acciones extra.






- Modificar y eliminar los comentarios sobre los productos.
- Modificar y eliminar tanto los hilos del foro como las publicaciones de cualquier usuario.

En cambio **si el usuario posee privilegios de administrador**, podrá realizar una serie acciones para gestionar tanto los usuarios como los productos. Además incluye las funciones que tienen los moderadores.

Al seleccionar la opción de “administrar” de la barra de navegación nos saltará un menú para elegir lo que queremos gestionar.



Si elegimos “Juegos” nos redireccionará a una pagina donde se muestra un listado paginado con diferentes acciones sobre un producto en especifico como las de modificar, eliminar y hacer visible o invisible para usuarios normales. También cuenta con un filtrado por nombre de producto y un multi-seleccionador para borrar varios a la vez, ademas de un sistema de ordenación por nombre,compañía o fecha de creación.

<div> Agregar Ordenar Eliminar </div> <div> <input type="text" value="Search"/> </div>								
#	Id	Foto	Título	Compañía	Precio	Creado	Modificado	Acción
<input type="checkbox"/>	1		Call Of Duty: Mobile	Activision	34.99	14:04 27-05-2024	15:51 27-05-2024	<input type="checkbox"/> <input type="edit"/> <input type="delete"/>
<input type="checkbox"/>	2		Minecraft	Mojang	12.5	15:42 27-05-2024	15:51 27-05-2024	<input type="checkbox"/> <input type="edit"/> <input type="delete"/>
<input type="checkbox"/>	3		Need For Speed	Electronics Arts	24	15:46 27-05-2024	15:51 27-05-2024	<input type="checkbox"/> <input type="edit"/> <input type="delete"/>
<input type="checkbox"/>	4		Fallout 4	Bethesda	34.99	15:46 27-05-2024	15:51 27-05-2024	<input type="checkbox"/> <input type="edit"/> <input type="delete"/>
<input type="checkbox"/>	5		Grand Theft Auto: V	RockStar	56	15:48 27-05-2024	15:51 27-05-2024	<input type="checkbox"/> <input type="edit"/> <input type="delete"/>

Previous
1
2
5
Next

Si decidimos añadir un producto nuevo, tendremos un formulario con validación en los campos requeridos donde subiremos la imagen del juego y elegir entre las categorías que existan. Si se desea crear varios productos, seleccionamos la opción multitud para que no nos cambie de pagina

Añadir nuevo juego

[Inicio](#) / [games](#) / Nuevo

Título

Desarrollador

Descripción

Foto

Seleccionar archivo
Ningún archivo seleccionado

Precio de venta

Categorías

Todas las categorías

Aventura
Acción
RPG
Disparos

>
<
Reset

Categorías seleccionadas

Volver

Guardar
Limpiar

Multitud

Si queremos modificar un juego podremos hacerlo modificando los datos anteriores en los campos del formulario.

Actualizar juego

[Inicio](#) / [games](#) / Actualizar

Título

Fallout 4

Desarrollador

Bethesda

Descripción

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed placerat, urna nec lobortis blandit, enim purus feugiat ante, vitae vehicula mi odio id sapien. Phasellus vitae tellus dictum, condimentum felis et, tincidunt nibh. Maecenas sodales nunc a enim pellentesque, in ullamcorper turpis porta.

Foto

Seleccionar archivo

Ninguno archivo selec.

Previsualización:



Precio de venta

34.99

Categorías

Todas las categorías

Aventura
Acción
RPG
Disparos

>
<

Reset

Categorías seleccionadas

Aventura
Acción
RPG
Disparos

← Volver

Actualizar

Limpiar






Respecto a la gestión de usuarios nos mostrará un listado de los usuarios registrados

Agregar

Ordenar

Search

x

Id	Foto	Nombre de usuario	Email	Cartera	Estado	Rol	Acción
1		Jcerrov	joaquincerrovaquerizo@gmail.com	816.51€	ACTIVE	ADMIN	<div><div>\$</div><div>🚫</div><div>✎</div><div>✂</div></div>
2		ManGa	Manga@gmail.com	50€	ACTIVE	MODER	<div><div>\$</div><div>🚫</div><div>✎</div></div>
3		Oscceva	oscar@gmail.com	0€	ACTIVE	USER	<div><div>\$</div><div>🚫</div><div>✎</div><div>✂</div></div>
4		Pegadi	Pegadi@gmail.com	0€	ACTIVE	USER	<div><div>\$</div><div>🚫</div><div>✎</div><div>✂</div></div>
5		PaGa12	Paga@Gmail.com	4€	ACTIVE	USER	<div><div>\$</div><div>🚫</div><div>✎</div><div>✂</div></div>

Previous

1

Next

ADMIN

MODER

USER

En este apartado podremos bloquear o activar al usuario, cambiar el rol o consultar las compras que este ha realizado.
Al consultar el historial de compras de un usuario podremos eliminar la compra y devolver el dinero al jugador.

Historial de Compras					
Foto	Título	Desarrollador	Costo	Obtenido	Eliminar
	Need For Speed	Electronics Arts	24	27/05/2024	X
	Minecraft	Mojang	12.5	27/05/2024	X
	Call Of Duty: Mobile	Activision	34.99	04/06/2024	X

MANUAL DEL PROGRAMADOR

6.Manual del programador.

6.1.¿Qué es MVPGames?

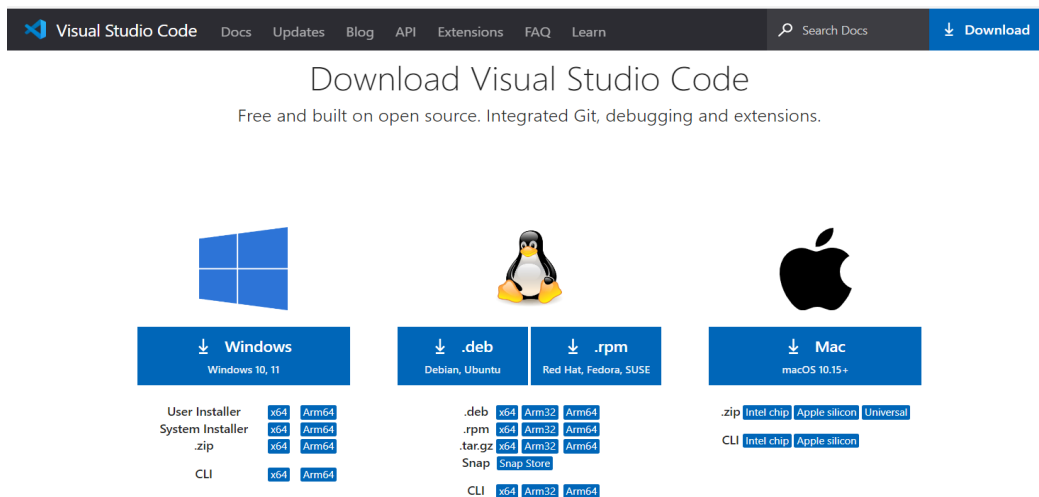
MVPGames es un proyecto que tiene como objetivo servir como herramienta digital para vender juegos de ordenadores y favorecer al consumo de los clientes estableciendo un entorno favorable donde se sientan cómodos y se les permita interactuar con otros usuarios registrados en la pagina.

6.2. Partes del proyecto.

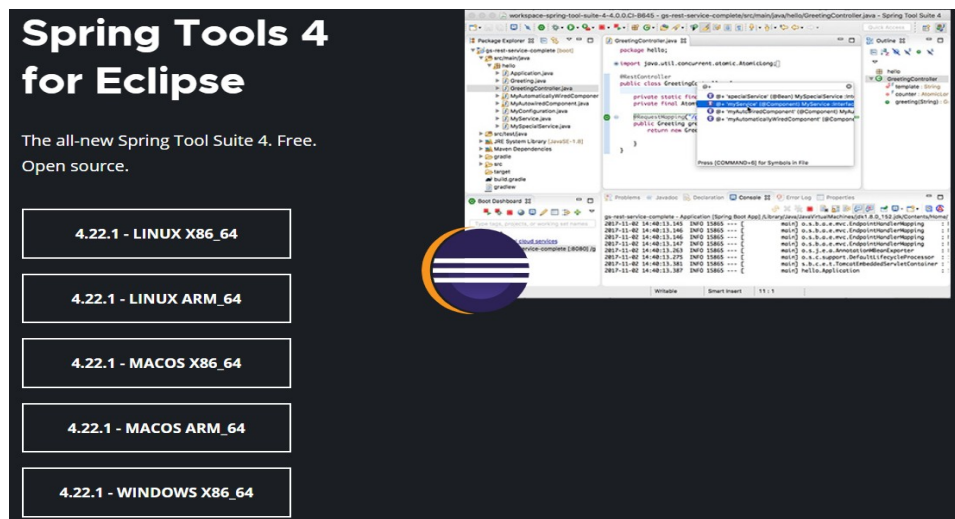
El proyecto cuenta con dos partes(BackEnd y FrontEnd) y una base de datos MySQL(Xampp) de alojamiento local.

Para el desarrollo del proyecto necesitaremos antes de nada instalar los programas necesarios,en nuestro caso instalaremos Xampp para la base de datos y utilizaremos los IDEs de Visual Studio Code y SpringBoot Tool Suite. Para ello, nos dirigimos a las webs oficiales y descargamos la ultima versión o la más estable

(<https://code.visualstudio.com/Download>)



(<https://spring.io/tools>)



Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4. Free. Open source.

- 4.22.1 - LINUX X86_64
- 4.22.1 - LINUX ARM_64
- 4.22.1 - MACOS X86_64
- 4.22.1 - MACOS ARM_64
- 4.22.1 - WINDOWS X86_64

The image shows a screenshot of the Spring Tool Suite 4 interface, which is an Eclipse IDE. It displays a project named 'workspace-spring-tool-suite-4-4.0.0-2020-08-05' with a package 'hello' containing a 'GreetingController.java' file. The code in the file is as follows:

```
package hello;

import java.util.concurrent.atomic.AtomicLong;

@Controller
public class GreetingController {
    private static final AtomicLong counter = new AtomicLong(0);
    private final Atom

    public Greeting getGreeting(String name) {
        return new Greeting(counter.incrementAndGet(), "Hello, " + name);
    }
}
```

(<https://www.apachefriends.org/es/index.html>)



¿Qué es XAMPP?

XAMPP es el entorno más popular de desarrollo con PHP

XAMPP es una distribución de Apache completamente gratuita y fácil de instalar que contiene MariaDB, PHP y Perl. El paquete de instalación de XAMPP ha sido diseñado para ser increíblemente fácil de instalar y usar.

Descargar
Pulsa aquí para otras versiones

XAMPP para Windows
8.2.12 (PHP 8.2.12)

XAMPP para Linux
8.2.12 (PHP 8.2.12)

XAMPP para OS X
8.2.4 (PHP 8.2.4)

The image shows the XAMPP logo, which is an orange square with a white play button icon in the center. Below the logo, the word 'XAMPP' is written in bold black letters.

-Instalación de angular.

Para instalar Angular seguiremos una serie de pasos:

Instalar nodeJS.

(<https://nodejs.org/en>)



Una vez instalado comprobamos que esta correctamente instalado además de npm package manager.

```
C:\Users\JZ1CY7VX>node -v
v20.11.1

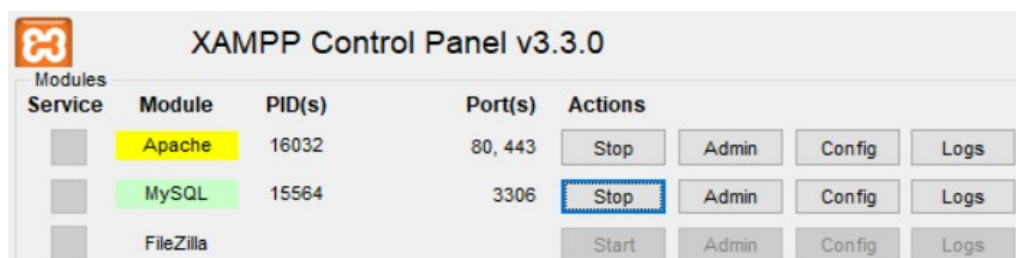
C:\Users\JZ1CY7VX>npm -v
10.2.4
```

Por ultimo, instalamos Angular desde la consola de comandos, ejecutando el comando:
- *npm install -g @angular/cli*

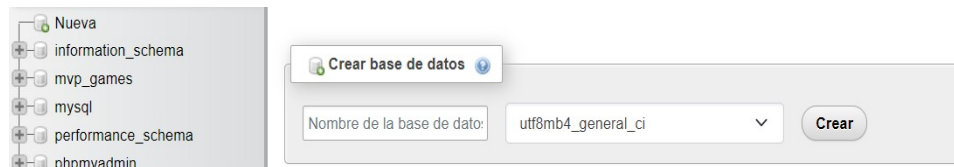
Creación del proyecto.

-Base de datos.

Abrimos Xampp y activamos los servicios de Apache y MySQL

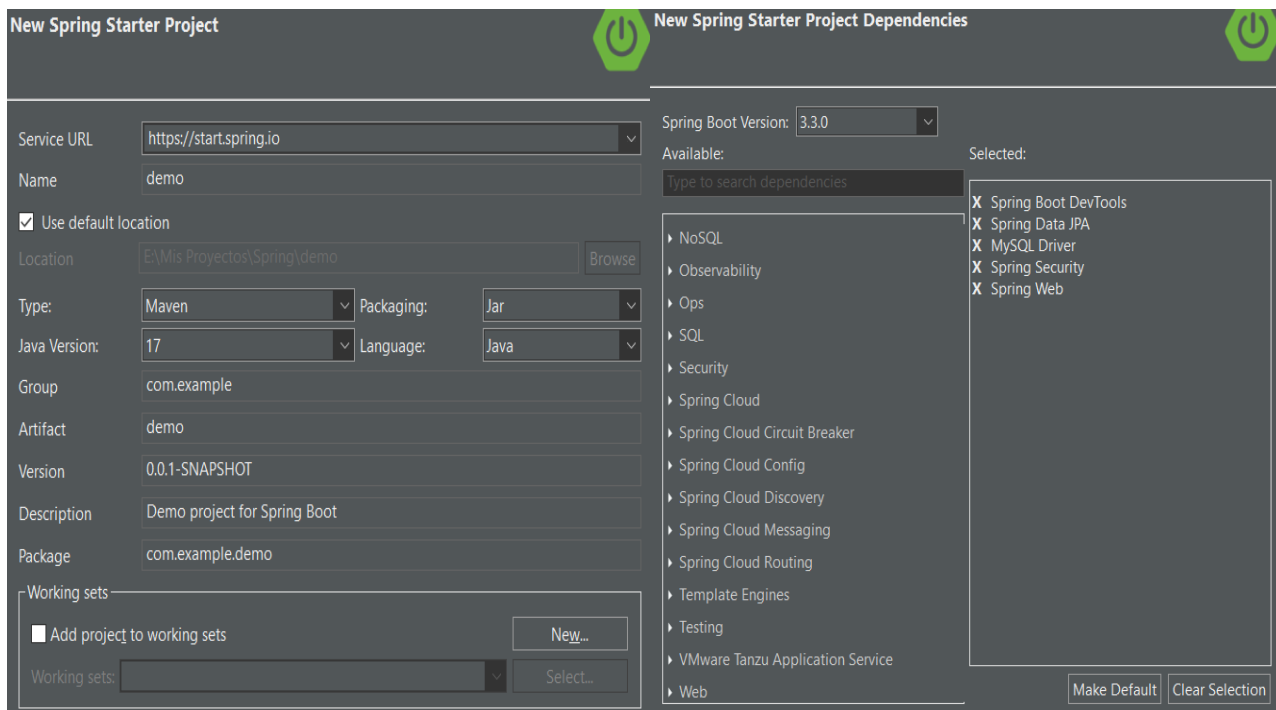


Ahora abrimos el navegador y nos dirigimos a <https://localhost/phpmyadmin/index.php>
Una vez dentro creamos la nueva base de datos.



-SpringBoot.

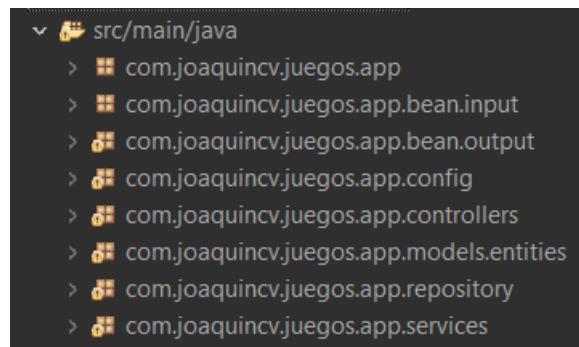
Al ser portable, no es necesario instalar nada adicional. Abrimos SpringToolSuite y creamos un nuevo proyecto SpringBoot con maven e instalamos las dependencias necesarias.



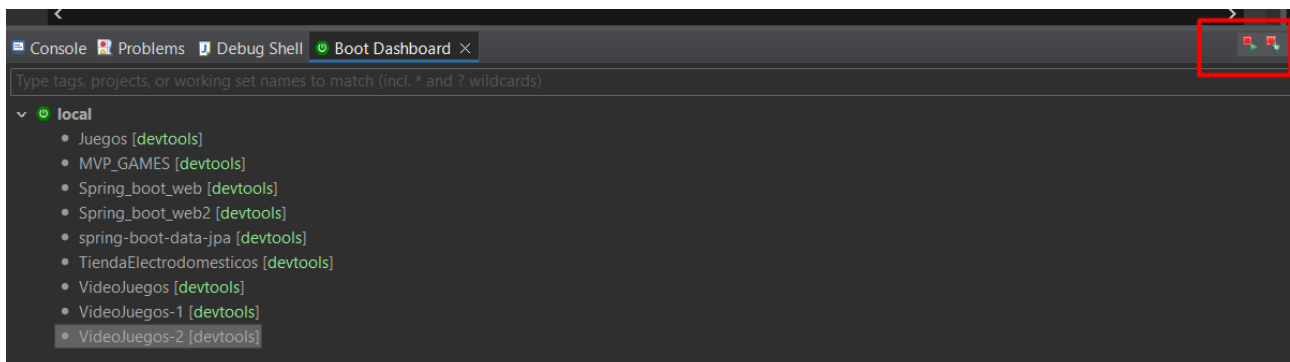
Conectamos el proyecto a la base de datos añadiendo al application.properties la siguientes lineas:

```
1 spring.datasource.url= jdbc:mysql://localhost:3306/mvp_games
2 spring.datasource.username=root
3 spring.datasource.password=
```

También podemos ver como se estructura el proyecto y su organización.



Una vez comprendido todo, podremos levantar el servidor

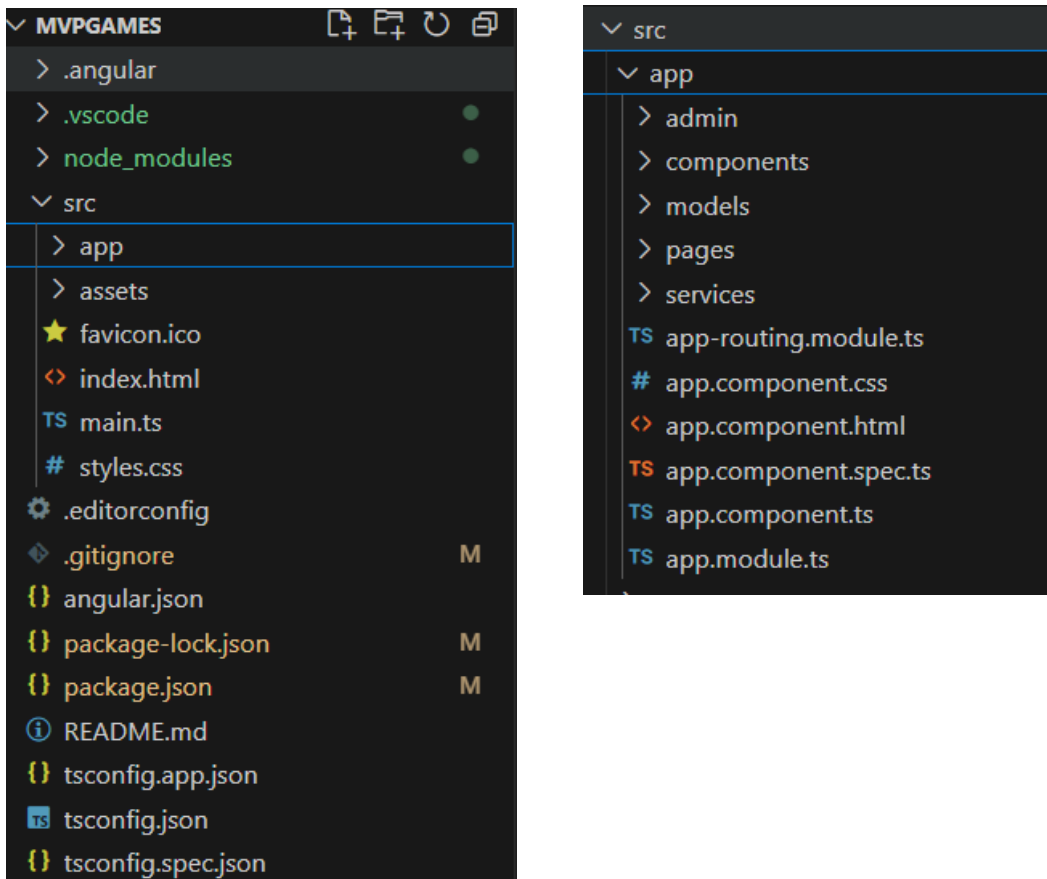


-Angular.

Para crear un nuevo proyecto, simplemente ejecutamos el comando:

ng new MVPGames.

Una vez creado, veremos la estructura del proyecto:



De todos estos componentes, solo tenemos que quedarnos con los que son realmente importantes:

-node_modules: Es donde se encuentran las librerías de angular.

-src: Donde se guardan los componentes de la aplicación, en “app” los que creamos.

-app-routing-module: Aquí se definen las rutas de navegación de la aplicación.

-app-modules: Donde se definen los componentes que creamos.

-styles.css: Los estilos CSS que se aplicaran a todos los componentes.

-index.html: La vista principal de la aplicación. Aquí llamaremos a los componentes los cuales se mostrarán y ocultarán a conveniencia.

Es importante importar aquí las dependencias de Bootstrap que se utilizan para el desarrollo de la aplicación.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhJY6hW+ALEwIH" crossorigin="anonymous">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
</head>
<body>
<app-root class="d-flex flex-column min-vh-100"></app-root>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDz0xhy9GkcIdS1K1eN7N6jIeHz" crossorigin="anonymous"></script>
```

Como se puede ver en la imagen, en el componente principal tenemos un elemento llamado “app-root” que hace referencia a los componentes que creemos, que se cargarán cuando sea debido.

Una vez aprendida la estructura del proyecto, necesitaremos iniciar nuestro proyecto, para ello necesitaremos ejecutar uno de los siguientes comandos:

-ng serve

Si queremos abrirlo directamente en el navegador predeterminado:

-ng serve -o

Y también podremos abrirlo para cualquier dispositivo que este conectado a la misma red usando:

-ng serve -host 0.0.0.0

La ruta de acceso a la aplicación es localhost:4200 pudiendo **cambiar el puerto** en el archivo de configuración de **angular.json**.

Existen tres tipos de componentes que necesitaremos crear para construir nuestra aplicación. Estos son los componentes simples, los servicios y los modelos de datos.

-Servicios: Sirven para establecer la conexión con el servidor y hacer las respectivas peticiones. Es recomendable también crear una constante que contenga la url principal del servidor para simplificar.

Para crearlo simplemente ejecutamos el comando: **-ng g s game**

```
TS game.service.spec.ts
TS game.service.ts
```

```
1 let baseURL = 'http://localhost:8080'
2 export default baseURL;
```

-Modelos: Sirven para definir los datos de los objetos que se manejan en el FrontEnd, se pueden crear a modo de interfaz o de clase, siendo este último mas utilizado debido a su simplicidad.

```
export class Game{
  id?:number;
  title?:string;
  author?:string;
  description?:string;
  price?:number;
  image?:string;
  categories?:Category[];
  status?:boolean;
  valuate?:number;
  createdAt?:string;
  formatCreateAt?:string;
  updatedAt?:string;
  formatUpdatedAt?:string;

  constructor(){
    this.categories = [];
  }
}
```

```
export interface User{
  id?:number;
  username:string;
  email:string;
  role:string;
  balance:number;
  status:number;
  profile:Profile;
  shoppingCart:ShoppingCart[];
  library:Game[];
  friends:User[];
}
```

No existe un comando específico para crear modelos así que debemos crearlos a mano.

Es conveniente crear una carpeta para almacenar los distintos modelos que se creen para mantener el código limpio.

Componentes: Son las propias vistas de la pagina. Se trata de un bloque de código que vienen compuestos de tres elementos uno para aplicar el CSS del componente, otro para desarrollar la pagina html y otro para aplicar la lógica TypeScript que necesite. También trae otro adicional para realizar pruebas pero en nuestro caso no lo usaremos.

Para crearlo simplemente ejecutamos el comando:

-ng g c shop

```
# shop.component.css
<> shop.component.html
TS shop.component.spec.ts
TS shop.component.ts
```

Funcionalidades.

Ya explicado todo esto, comenzaremos a explicar como se comporta el código y como realiza las operaciones de lectura y escritura.

Autenticación.

Antes de nada la mayor parte de la aplicación necesita que el usuario este registrado para poder acceder además de poder diferenciar los distintos roles de usuario. Para ello se explicara como funciona el sistema de seguridad implantado en el backend.

Para el desarrollo se ha usado SpringBoot Security junto con la autenticación por JWTToken. Para su uso es necesario instalar dos dependencias.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.2</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.2</version>
  <scope>runtime</scope>
</dependency>
```

A continuación se procede a crear una clase de configuración para establecer los ajustes de seguridad que tendrá la aplicación.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private UserRepository repository;

    @Autowired
    @Lazy
    private JWTAuthenticationFilter jwtAuthFilter;

    @Bean
    protected SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        return http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(authRequest -> authRequest.requestMatchers("/**").permitAll()
            .anyRequest().authenticated())
            .sessionManagement(sessionManager -> sessionManager.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }
}
```

Aquí desactivamos el protocolo de seguridad que viene por defecto y activamos el nuestro que solo permite acceder sin estar registrado a las rutas que deseemos y según el rol del usuario. También establecemos que la seguridad se va a llevar a cabo mediante jwtAuthToken.


```

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {
    return configuration.getAuthenticationManager();
}

@Bean
public AuthenticationProvider authenticationProvider () {
    DaoAuthenticationProvider authenticationProvider=new DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(userDetailsService());
    authenticationProvider.setPasswordEncoder(passwordEncoder());
    return authenticationProvider;
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public UserDetailsService userDetailsService() {
    return username -> repository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado"));
}

```

También establecemos los métodos para obtener los detalles del usuario y otro para encriptar la contraseña cuando se registre el usuario.

Ahora estableceremos los parámetros de creación del token de autenticación que nos servirá para iniciar sesión.

```

@Service
public class JwtService {

    private static final String SECRET_KEY="194730275639475937507365836493874020574686868";

    public String getToken(UserDetails user) {
        return getToken(new HashMap<>(),user);
    }

    private String getToken(Map<String, Object> claims, UserDetails user) {
        return Jwts.builder().setClaims(claims)
            .setSubject(user.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis()+1000 * 60 * 60 * 24 * 7))
            .signWith(getKey(), SignatureAlgorithm.HS256)
            .compact();
    }

    private Key getKey() {
        byte[] keybytes=Decoders.BASE64.decode(SECRET_KEY);
        return Keys.hmacShaKeyFor(keybytes);
    }

    private Claims getAllClaims(String token) {
        return Jwts.parserBuilder().setSigningKey(getKey()).build().parseClaimsJws(token).getBody();
    }
}

```

Como vemos, tenemos que establecer una clave secreta que sirva para codificar el token y así dificultar su falsificación. En los claims (por así decirlo cabecera del token que contiene cierta información) metemos información importante como el nombre de usuario, la fecha de creación y el tiempo de duración que tendrá antes de que expire.

```

private Claims getAllClaims(String token) {
    return Jwts.parserBuilder().setSigningKey(getKey()).build().parseClaimsJws(token).getBody();
}

public <T> T getClaim(String token, Function<Claims, T> claimsResolver) {
    Claims claims = getAllClaims(token);

    return claimsResolver.apply(claims);
}

public String getUsernameFromToken(String token) {
    return getClaim(token, Claims::getSubject);
}

private Date getExpiration(String token) {
    return getClaim(token, Claims::getExpiration);
}

private boolean isTokenExpired(String token) {
    return getExpiration(token).before(new Date());
}

public boolean isValidToken(String token, UserDetails details) {
    String username = getUsernameFromToken(token);
    return username.equals(details.getUsername()) && !isTokenExpired(token);
}

```

También tendremos algunos métodos mas para obtener los datos de ese token, así como el tiempo de expiración, y si esta ya caducado.

Una vez establecida la configuración, pasaremos a ver como funciona el ciclo de vida del registro de un usuario.

Empezaremos por la vista donde el usuario se registra.

```

export class SignupComponent implements OnInit {

    registerForm! : FormGroup;
    existe:boolean=true;
    userExist:boolean=false;
    mailExist=false;

    constructor(private fb:FormBuilder, private authService:AuthService, private route:Router){
        this.registerForm=this.fb.group({
            name: ['', [Validators.required, Validators.maxLength(20)]],
            lastname: ['', [Validators.required, Validators.maxLength(25)]],
            username: ['', [Validators.required, Validators.minLength(5), Validators.maxLength(15)]],
            email: ['', [Validators.required, Validators.email]],
            phone: ['', [Validators.pattern("^\+[0-9]{1,3}[0-9]{6,14}$")]],
            country: ['',],
            password: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(16)]],
            password2: ['', [Validators.required]],
            policy: ['', Validators.requiredTrue]
        }),
        { validators: this.passwordConfirm });
    }
}

```

Para el formulario de registro, se ha usado un formGroup que controla todos los elementos html con la ayuda de FormBuilder. De este modo podremos darle un valor de entrada si queremos y lo más importante establecer validaciones en los campos como si es requerido y el tamaño del texto. Por supuesto también se nos permite realizar nuestras propias validaciones como en la contraseña que tienen que coincidir o el usuario que tiene que comprobar que no exista uno igual en la base de datos.

```

verifyUser(){
  const username=this.registerForm.get('username')?.value;
  debounceTime(3000)
  this.authService.checkUser(username).subscribe(data => {
    this.userExist=data;
  });
}

verifyEmail(){
  const email=this.registerForm.get('email')?.value;
  this.authService.checkEmail(email).subscribe(data=>this.mailExist=data);
}

passwordConfirm(form:FormGroup) {
  const password = form.get('password')?.value;
  const confirmPassword = form.get('password2')?.value;
  if (password!==confirmPassword) {
    form.get('password2')?.setErrors({passwordError:true});
  } else {
    form.get('password2')?.setErrors(null);
  }
}

```

También podemos ver los métodos para comprobar si existe el usuario o el email.

```

<label for="username" class="form-label">Nombre de usuario</label>
<input type="text" class="form-control" id="username" required formControlName="username" (input)="verifyUser()" />
<div *ngIf="registerForm.get('username')?.invalid && registerForm.get('username')?.touched || userExist" class="alert alert-danger">
  <span *ngIf="registerForm.get('username')?.errors?.['required']">Campo obligatorio</span>
  <span *ngIf="registerForm.get('username')?.errors?.['minlength'] || registerForm.get('username')?.errors?.['maxlength']">Debe tener
  <span *ngIf="userExist" >Este nombre de usuario ya está en uso</span>

```

Si no existe ningún error, mandará los datos al servicio y el servicio se comunicará y enviará los datos al controlador en el backend.

```

register():void{
  if(this.registerForm.valid && !this.userExist && !this.mailExist ){
    this.authService.register(this.registerForm.value).subscribe(
      ()=>{
        console.log('Registrado correctamente');
        this.route.navigateByUrl('#');
      },
      error=>console.error('Error al guardar el usuario')
    )
  }
}

```

```

public register(request:registerRequest): Observable<registerRequest>{
    return this.http.post<any>(`${baseUrl}/auth/register`,request).pipe(
        tap( (userData)=> {
            localStorage.setItem('token',userData.token);
            this.currentUserLog.next(true);
            this.currentUser.next(userData)
        }),
        map((userData)=>userData.token),
    );
}

public checkUser(username: string): Observable<boolean> {
    return this.http.get<boolean>(`${baseUrl}/auth/check-username?username=${username}`);
}

public checkEmail(email:String):Observable<boolean>{
    return this.http.get<boolean>(`${baseUrl}/auth/check-email?email=${email}`)
}

```

```

@PostMapping("/auth/register")
public ResponseEntity<AuthResponse> register(@RequestBody RegisterRequest registerRequest) {
    userService.saveUser(registerRequest);
    return ResponseEntity.ok(authservice.register(registerRequest));
}

```

```

@Transactional
@Override
public void saveUser(RegisterRequest registerRequest) {

    User user=new User(registerRequest.getUsername(),
        encoder.encode(registerRequest.getPassword()),registerRequest.getEmail());
    Profile profile=new Profile(registerRequest.getName(),registerRequest.getLastname(),registerRequest.getPhone(),
        registerRequest.getCountry());
    user.setProfile(profile);
    profile.setUser(user);
    user=userRepository.save(user);
}

```

En el backend, guardará los datos del usuario y lo registrará en la base de datos, junto a su perfil (información personal del usuario).

A cambio el backend generará un token con la información del usuario y lo mandará al frontend para completar el registro y guardar el token de acceso en almacenamiento local del navegador.

El inicio de sesión es igual que el registro pero en vez de crear un nuevo usuario recuperas uno ya creado si el usuario y la contraseña coinciden y genera el token que devolverá al cliente. Existe la posibilidad de guardar el token en el localStorage o SessionStorage según se desee para mantener mas o menos tiempo la sesión

```

public login(credentials:LoginRequest):Observable<any>{
  return this.http.post<any>(`${baseUrl}/auth/login`,credentials).pipe(
    tap( (userData) => {
      if(credentials.keepSession){
        localStorage.setItem('token',userData.token);
      }else{
        sessionStorage.setItem('token',userData.token);
      }
      this.currentUserLog.next(true);

      this.getCurrentUser().subscribe(data=>this.currentUser.next(data))
    }),
    map((userData)=>userData.token),
    catchError(this.handlerError))
}

```

Para cerrar sesion,simplemente borramos el token del almacenamiento del navegador.

```

public logout(){
  localStorage.removeItem('token');
  sessionStorage.removeItem('token')
  this.currentUserLog.next(false);
}

```

El token se necesitará más tarde para hacer peticiones al servidor pero para ello debemos mandar el token en la cabecera de la petición para así comprobar que esta registrado siempre.

Para que esto funcione desarrollaremos un interceptor:

```
export class JwtInterceptorService implements HttpInterceptor{

  constructor() { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const token = sessionStorage.getItem('token')||localStorage.getItem('token');

    let request = req;

    if (token) {
      request = req.clone({
        setHeaders: {
          authorization: `Bearer ${ token }`
        }
      });
    }

    return next.handle(request);
  }
}
```

```
intercept(req:HttpRequest<any>, next:HttpHandler):Observable<HttpEvent<any>>{
  return next.handle(req).pipe(
    catchError(error=>{
      console.error(error);
      return throwError(()=>error)
    })
  )
}
```

Y actualizamos el app.Module para modificar la cabecera de todas las peticiones de todos los componentes.

```
providers: [DatePipe ,
  {
    provide: HTTP_INTERCEPTORS,
    useClass: JwtInterceptorService,
    multi: true
  }
]
```

Ahora ya tendríamos el sistema de autenticación por token completo y podríamos obtener los datos del usuario al iniciar sesión a través de ese token.

Para simplificar las peticiones al servidor y mantener los datos actualizados en todos los componentes, se hace el uso de BehaviourSubject. Un BehaviorSubject es una variante de un observable, es decir, una variable a la que los componentes se pueden suscribir y al detectar cambios, actualizan sus valores. Entonces para mantener la información del usuario la cual la requieren muchos componentes, es necesario usarlo y cuando alguno de ellos modifique esta información, se comunique al resto de componentes.

Deben de tener un valor inicial.

```
user?:User

currentUser:BehaviorSubject<User>=new BehaviorSubject<User>(this.user!);
currentUserLog: BehaviorSubject<boolean>=new BehaviorSubject<boolean>(false);

constructor(private http:HttpClient) {

  this.currentUserLog=new BehaviorSubject<boolean>(localStorage.getItem("token")!=null || sessionStorage.getItem('token')!=null);
  this.currentUserLog.subscribe(data=>{const login=data
    if(login){
      this.getCurrentUser().subscribe(data=>this.currentUser.next(data))
    }
  })
}
```

En este caso usamos dos. Una para almacenar si el usuario está registrado y otro para su información.

Para evitar errores, en el constructor obtenemos los datos del usuario suscribiendonos al que indica si se ha iniciado sesión o no.

Rutas

Para navegar por la aplicación se utiliza Angular Router que viene por defecto y nos permite tener muchas paginas o vistas sin necesidad de crear varios documentos html para mostrar.

Lo que angular nos muestra al iniciar la aplicación es el index.html que se irá actualizando según el contenido que queramos mostrar en el **app.component**.

```
<app-navbar></app-navbar>
<router-outlet class="flex-shrink-0"></router-outlet>
<app-footer></app-footer>
```

En nuestro caso mostraremos tres componentes siempre, la cabecera o barra de navegación que muestre los datos del usuario y los botones para navegar por la pagina web el pie de pagina y entre ellos el contenido principal.

Para que se muestre un componente u otro, debemos establecer unas rutas asociadas a cada componente y esto tendrá lugar en el archivo **app-routing-module**.

```

{
  path: 'admin/users',
  component: UserListComponent,
  pathMatch: 'full'
},
{
  path: 'admin/users/:id/update',
  component: ModifyUserComponent,
},
{
  path: 'users/:username',
  component: ProfileComponent
},
{
  path: 'profile',
  component: ProfileComponent
},
},

```

Como se puede ver en la imagen, a cada ruta se le asigna un componente el cual se mostrará al estar en dicha ruta.

También es posible asignar diferentes rutas a un mismo componente si tiene una estructura parecida pero hay que realizar unos pequeños cambios en su comportamiento.

Entidades y JPA

Volviendo al funcionamiento de la aplicación he de detallar que para el manejo de las diferentes entidades y tablas se ha utilizado JPA(Jakarta persistence) que es una API propia de java que te permite crear automáticamente la base de datos a partir de las entidades y las relaciones que definamos con sus anotaciones.

```

@Entity
@Table(name = "games")
public class Game {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    private String author;

    @Column(length = 10000)
    private String description;

    @Column(scale = 2)
    private double price;

    private boolean status;

    private String image;

    private double valuate;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.MERGE)
    @JoinTable(name = "game_categories", joinColumns = @JoinColumn(name="game_id", referencedColumnName = "id"),
    inverseJoinColumns = @JoinColumn(name="category_id", referencedColumnName = "id"))
    private List<Category> categories;

    private boolean deleted;

    private Date createdAt;

    private Date updatedAt;
}

```



```

@Entity
@Table(name = "users")
public class User implements UserDetails{

    private static final long serialVersionUID = 6435400157055099675L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;

    private String status;

    @Column(columnDefinition = "DOUBLE(10,2)")
    private double balance;

    @Enumerated(EnumType.STRING)
    private Role role;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private Profile profile;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private List<ShoppingCart> shoppingCart;
}

```

Al ser muchas clases, mostraré las principales solamente.

Además de eso, también se hace uso de la interfaz de JPA Repository, propia de SpringBoot, que nos sirve realizar las consultas básicas de un CRUD donde además podemos hacer nuestras propias consultas.

```

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);

    @Query("SELECT u FROM User u WHERE u.username LIKE %:filter%")
    Page<User> findAll(Pageable pageable, @Param("filter") String filter);

    @Query("SELECT u FROM User u WHERE u.username LIKE %:filter%")
    List<User> findAll(@Param("filter") String filter);

    boolean existsByUsername(String username);

    Optional<User> findByEmail(String email);

    boolean existsByEmail(String email);
}

```

```

@Query("SELECT g FROM Game g WHERE g.title LIKE %:filter% AND g.deleted = false")
Page<Game> findAll(Pageable pageable, @Param("filter") String filter);

@Query("SELECT g FROM Game g WHERE g.title LIKE %:filter% AND g.deleted = false AND g.status=true")
Page<Game> findAll1(Pageable pageable, @Param("filter") String filter);

@Query("SELECT g FROM Game g JOIN g.categories c WHERE "
+ "c.id IN :categoryIds AND g.deleted = false AND g.status = true GROUP BY g HAVING COUNT(c.id) = :categoryCount")
Page<Game> findByCategory(Pageable pageable, @Param("categoryIds") List<Long> categoryIds, @Param("categoryCount") Long categoryCount);

@Query("SELECT g FROM Game g WHERE g.deleted = false")
Page<Game> findAll(Pageable pageable);

@Query("SELECT g FROM Game g WHERE g.deleted = false AND g.status=true")
Page<Game> findAll1(Pageable pageable);

List<Game> findByTitle(String title);

```

Games CRUD

Haremos un recorrido por las funciones de los juegos en la aplicación.

En primer lugar, obtenemos el catalogo de los juegos los cuales gracias a la paginación y al parámetro por el cual se van a ordenar, podremos hacer tanto la lista de todos los juegos como las secciones de mejor valorados y la de mas baratos las cuales podemos utilizar dentro del componente principal.

```
ngOnInit(): void {
  this.loadGames();
  this.loadCategories()
}

loadCategories(){...
}

loadGames() {
  this.gameService.games1(this.page,15,'id',true,this.inputValue).subscribe(
    data=>{
      this.games=data.content,
      this.isFirst=data.first,
      this.isLast=data.last,

      this.totalPages=data.totalPages
    },
    error=>{ console.error("Error al recibir los datos de los juegos:",error)});
}
```

Como vemos al iniciar el componente se hace una petición al servidor para traer los juegos y las categorías.

```
//Para usuarios normales
games1(page:number,size:number,order:string,asc:boolean,filterChain:string):Observable<any>{
  return this.http.get<any>(`${baseUrl}/games?`+'page=${page}&size=${size}&order=${order}&asc=${asc}&filter=${filterChain}`);
}

games(page:number,size:number,order:string,asc:boolean,filterChain:string):Observable<any>{
  return this.http.get<any>(`${baseUrl}/admin/games?`+'page=${page}&size=${size}&order=${order}&asc=${asc}&filter=${filterChain}`);
}
```

Al hacer las peticiones, debemos completar una serie de parámetros para establecer el número de la pagina que vamos a traer, el campo por el que se va a ordenar y el número de elementos que traerá. También podemos establecer un parámetro opcional para filtrar por nombre.

Debemos diferenciar entre si se realiza desde la administración o desde la pagina principal donde solo se mostrarán los que sean visibles para usuarios normales.

```
getGamesByCategories(params:HttpParams):Observable<any>{
  return this.http.get<any>(`${baseUrl}/games/categoryfilter`, { params })
}
```

También podremos obtener los juegos filtrado por categorías, donde básicamente se devuelve al servidor una lista de identificadores de categorías para buscar los que coincidan.

```

@GetMapping("/admin/games")
public ResponseEntity<Page<Game>>getAll(
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "10")int size,
    @RequestParam(defaultValue = "id")String order,
    @RequestParam(defaultValue = "true") boolean asc,
    @Param("filter")String filter)
{
    Page<Game> games=gameService.getAllGames(PageRequest.of(page, size,Sort.by(order)),filter);

    if (!asc) {
        games=gameService.getAllGames(
            PageRequest.of(page, size,Sort.by(order).descending()),filter);
    }
    return new ResponseEntity<Page<Game>>(games,HttpStatus.OK);
}

```

El controlador le pasa los datos al registro y el registro se encarga de acceder al repositorio y devolver la lista de juegos según nos interese

```

public Page<Game> getAllGames(Pageable pageable,String filter) {
    if (filter!=""&& filter!=null) {
        return gameRepository.findAll(pageable ,filter);
    }
    return gameRepository.findAll(pageable);
}

@Override
public Page<Game> getGames(Pageable pageable,String filter) {
    if (filter!=""&& filter!=null) {
        return gameRepository.findAll1(pageable ,filter);
    }
    return gameRepository.findAll1(pageable);
}

```

```

@Query("SELECT g FROM Game g WHERE g.title LIKE %:filter% AND g.deleted = false")
Page<Game> findAll(Pageable pageable,@Param("filter")String filter);

@Query("SELECT g FROM Game g WHERE g.title LIKE %:filter% AND g.deleted = false AND g.status=true")
Page<Game> findAll1(Pageable pageable,@Param("filter")String filter);

@Query("SELECT g FROM Game g JOIN g.categories c WHERE c.id IN "
+ ":categoryIds AND g.deleted = false AND g.status = true GROUP BY g HAVING COUNT(c.id) = :categoryCount")
Page<Game> findByCategory(Pageable pageable, @Param("categoryIds") List<Long> categoryIds, @Param("categoryCount") Long categoryCount);

@Query("SELECT g FROM Game g WHERE g.deleted = false")
Page<Game> findAll(Pageable pageable);

@Query("SELECT g FROM Game g WHERE g.deleted = false AND g.status=true")
Page<Game> findAll1(Pageable pageable);

```

Para obtener la información de un juego se hace a través del id.

```
getGame(id:number):Observable<Game>{  
    return this.http.get<Game>(`${baseUrl}/admin/games/${id}`)  
}
```

```
@GetMapping("/admin/games/{id}")  
public ResponseEntity<Game>getGame(@PathVariable Long id){  
    Game game=gameService.findById(id);  
    return new ResponseEntity<Game>(game,HttpStatus.OK);  
}
```

Si se desea borrar un juego, se ha implementado un borrado virtual en el que si el campo esta marcado no se mostrarán para nadie. Esto se hace para garantizar el optimo funcionamiento y que al borrar un juego, no le desaparezca al usuario que ya lo ha comprado al estar relacionadas las tablas.

```
deleteGame(id:number){  
    return this.http.delete(`${baseUrl}/admin/games/delete/${id}`);  
}
```

```
@DeleteMapping("/admin/games/delete/{id}")  
public void deleteGame(@PathVariable Long id) {  
    gameService.deleteGame(id);  
}
```

```
@Override  
public void deleteGame(Long id) {  
    Game game=gameRepository.findById(id).get();  
    game.setDeleted(true);  
    gameRepository.save(game);  
}
```

Como se ve, simplemente se marca como actualizado y se actualiza en la base de datos.

Para modificar un juego, tiene un comportamiento similar.

Primero, el cliente recibe el objeto y el formulario toma los valores de sus respectivos campos.

```
ngOnInit(): void {  
  this.loadGame();  
  this.loadCategories();  
  this.initGameForm();  
}
```

Cuando termine, si todos los campos son correctos, hace la petición para actualizar y devuelve los nuevos datos del objeto.

En caso de que algún campo del formulario no cumpla los requisitos, el sistema de validación marcará el campo que contenga el error.

```
update(): void {  
  if(this.gameForm.valid){  
    swal.fire({  
      title: "¿Deseas actualizar este juego?",  
      showCancelButton: true,  
      confirmButtonText: "Actualizar",  
    }).then((result) => {  
      if (result.isConfirmed) {  
        this.updatedGame.title=this.gameForm.get('title')?.value  
        this.updatedGame.author=this.gameForm.get('author')?.value  
        this.updatedGame.description=this.gameForm.get('description')?.value  
        this.updatedGame.price=this.gameForm.get('price')?.value  
        this.updatedGame.image=this.gameForm.get('photo')?.value  
        this.gameService.updateGame(this.actualGame.id!,this.updatedGame).subscribe(  
          () => {  
            swal.fire("Actualizado correctamente!", "", "success");  
            this.route.navigateByUrl("/admin/games")  
          },  
          error => {  
            swal.fire("Ha ocurrido un error!", "", "error");  
            console.error('Error al guardar el juego:', error);  
          });  
      }  
    });  
  }  
  else{  
    this.gameForm.markAllAsTouched();  
  }  
}
```

```
@PostMapping("/admin/games/update/{id}")
public ResponseEntity<Game> updateGame(@PathVariable Long id,@RequestBody Game game){
    gameService.updateGame(id,game);
    return new ResponseEntity<Game>(HttpStatus.ACCEPTED);
}
```

En el servidor se busca de nuevo el objeto por id y se cambian los campos que estén permitidos.

```
@Override
public void updateGame(Long id,Game game) {
    Game gameUpdate=gameRepository.findById(id).get();
    gameUpdate.setTitle(game.getTitle());
    gameUpdate.setAuthor(game.getAuthor());
    gameUpdate.setDescription(game.getDescription());
    gameUpdate.setImage(game.getImage());
    gameUpdate.setPrice(game.getPrice());
    gameUpdate.setCategories(game.getCategories());
    gameUpdate.setStatus(game.isStatus());
    gameUpdate.setUpdatedAt(new Date());
    gameRepository.save(gameUpdate);
}
```

Por ultimo,se hace una subida para actualizar el elemento en la base de datos.

WebSocket (Chat a tiempo real)

Para tener una experiencia óptima con un chat entre jugadores, se debía desarrollar a tiempo real pero el protocolo http no lo permite ya que se basa en un sistema de petición y al que recibiría el mensaje, no se le actualizaría a tiempo.

Para ello, hemos utilizado Web Socket, el cual consiste en establecer un canal al que se conecten los usuarios y mientras permanezcan en la ruta, estarán siempre conectados.

Primero, deberemos establecer la configuración del socket para que este funcione.

```

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat-socket").setAllowedOrigins("http://localhost:4200").withSockJS();
    }
}

```

Como vemos tenemos que establecer a donde se conectará los clientes y desde la dirección que se conectará el cliente.

Desarrollamos un controlador específico para el uso de WebSocket.

```

@Controller
public class ChatController {

    @Autowired ChatService chatService;

    @MessageMapping("/chat/{roomId}")
    @SendTo("/topic/{roomId}")
    public List<MessageDTO> chat(@DestinationVariable String roomId, @Payload MessageDTO message) {

        chatService.saveMessage(roomId,message);

        return chatService.getMessages(Long.valueOf(roomId));
    }
}

```

Como se aprecia, establecemos el canal al que se conectarán los usuarios. Al querer que el chat sea individual, debemos crear un canal específico para cada relación de amistad. Para ello se ha utilizado el identificador de la relación entre dos usuarios como parámetro para crear una ruta específica para cada relación de amistad.

Esto hace que al recibir un mensaje, todo el que este conectado a la ruta automáticamente se le actualice el chat.

Por supuesto, guardamos el mensaje en la base de datos para su persistencia y retornamos la lista de mensajes completa.

En angular, deberemos instalar las librerías para manejar sockets para ello instalaremos las siguientes librerías:

```
npm install @stomp/ng2-stompjs @stomp/stompjs
```

```
npm install sockjs-client stompjs
```

Una vez instalado, procedemos a crear el servicio de conexión.

```
export class ChatService {
  private stompClient:any;
  private messageSubject:BehaviorSubject<ChatMessage[]> = new BehaviorSubject<ChatMessage[]>([]);

  constructor(private http:HttpClient) {
    this.connect()
  }

  connect(){
    const socket = new SockJS('//localhost:8080/chat-socket');
    this.stompClient = Stomp.over(socket);
    this.stompClient.connect({}, () => {
      console.log('Conectado');
    }, (error: any) => {
      console.error('Error de conexión', error);
    });
  }
}
```

Como se aprecia, creamos un método para establecer la conexión con el socket usando las rutas anteriores y lo invocamos en el constructor del servicio para que se conecte nada más se llame al servicio.

```
joinRoom(roomId:string){
  if (this.stompClient && this.stompClient.connected) {
    this.stompClient.subscribe(`/topic/${roomId}`, (mensaje: any) => {
      const contenidoMensaje = JSON.parse(mensaje.body);
      this.messageSubject.next(contenidoMensaje);
    });
  } else {
    console.error('Cliente STOMP no conectado');
  }
  this.joinRoom(roomId)
}
```

Ahora debemos hacer otro método para conectarse a una sala específica según a quien queramos enviar el mensaje.

Se usa otro behaviorSubject para mantener actualizado el componente html y así más tarde refrescar la lista de mensajes según se manden mensajes.

```
sendMessage(roomId:string,message:ChatMessage){
  this.stompClient.send(`/app/chat/${roomId}`, {}, JSON.stringify(message))
}

getMessageSubject():Observable<ChatMessage[]>{
  return this.messageSubject.asObservable();
}

readmessages(id:number):Observable<any>{
  return this.http.put<any>(`${baseUrl}/read-messages`, id)
}
```


Por último, establecemos los metodos para mandar el mensaje al servidor y los metodos para suscribirse al BehaviorSubject y marcar todos los mensajes de ese usuario como leídos, lo cual tendrá lugar al abrir el chat de esa persona.

Con esto tendríamos un chat a tiempo real con sistema de leído para mostrar una notificación en la cabecera de la aplicación cuando tengamos mensajes sin leer .

Carro de compra

Para implementar el carro de compra, se ha decidido guardar los datos en la base de datos para que su uso pueda estar disponible en multidispositivos. Y gracias al uso de observadores podemos controlar que al estar ya dentro del carro de la compra, no se pueda añadir de nuevo un mismo producto.

```
@Entity
@Table(name = "shopping_carts",uniqueConstraints = {@UniqueConstraint(columnNames = {"user_id", "juego_id"})})
public class ShoppingCart {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonBackReference
    private User user;

    @ManyToOne
    @JoinColumn(name = "juego_id")
    private Game game;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "added_date")
    private Date addedDate;
```

Establecemos la entidad, relacionandola con el usuario y el juego. Como solo puede tener un mismo producto por usuario, formamos una clave foranea unica para no repetir los campos de las claves foraneas.

Para obtener el carrito de la compra de un usuario concreto, filtramos por id del usuario, que este a su vez se consigue a través del token de la cabecera de la petición

```
@Repository
public interface ShoppingCartRepository extends JpaRepository<ShoppingCart,Long> {

    @Query("SELECT c FROM ShoppingCart c WHERE c.user.id = :userId")
    List<ShoppingCart> getCartFromUser(@Param("userId") Long userId);
```

```

public List<ShoppingCart> getCartFromUser(String username) {
    try {
        User user=userRepository.findByUsername(username).orElse(null);
        if (user!=null) {
            return shoppingCartRepository.getCartFromUser(user.getId());
        }else {
            return new ArrayList<>();
        }
    } catch (Exception e) {
        e.printStackTrace();
        return new ArrayList<>();
    }
}

```

```

@GetMapping("/cart")
public ResponseEntity<List<ShoppingCart>> getShoppingCart(Principal principal) {
    List<ShoppingCart> cart= cartService.getCartFromUser(principal.getName());

    return new ResponseEntity<List<ShoppingCart>>(cart,HttpStatus.OK);
}

```

A parte de obtener los elementos del carrito, tendremos otros dos métodos en el controlador para borrar y para añadir al carrito. Se hace a través del id del juego y la información obtenida del token.

```

@Autowired
private ShoppingCartService cartService;

@PostMapping("/add")
public void addToCart(Principal principal,@RequestBody Long gameId) {
    cartService.addToShoppingCart(principal.getName(),gameId);
}

@DeleteMapping("/remove/{id}")
public void removeFromCart(@PathVariable Long id) {
    cartService.removeFromShoppingCart(id);
}

```

```

public void addToShoppingCart(String username,Long gameId) {

    User user=userRepository.findByUsername(username).get();
    Game game = gameRepository.findById(gameId).get();
    ShoppingCart shoppingCart=new ShoppingCart(user,game);
    shoppingCartRepository.save(shoppingCart);
}

public void removeFromShoppingCart(Long shoppingCartId) {
    shoppingCartRepository.deleteById(shoppingCartId);
}

```

Compras

A la hora de comprar, se podrá hacer una compra de todo el carrito o de un solo producto, por lo cual a la hora de guardar se hará mediante una lista.

```
@PostMapping("/order/new")
public ResponseEntity<String> newOrder(Principal principal ,@RequestBody List<Game>games){
    try {
        orderService.addOrder(principal.getName(), games);
        return new ResponseEntity<String>(HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<String>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

La estructura y el funcionamiento de las compras es similar al del carrito pero aquí solo podrán consultarlas y hacer una nueva compra.

```
@GetMapping("/order/history")
public ResponseEntity<List<Order>> getOrdersFromUser(Principal principal){
    try {
        return new ResponseEntity<List<Order>>(orderService.getOrderFromUser(principal.getName()),HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<List<Order>>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Hay que tener en cuenta que al hacer una compra sobre un elemento del carrito o sobre todo el carrito, este se vacíe y no permita realizar compras sobre productos ya comprados.

```
@Transactional
@Override
public void addOrder(String username, List<Game> games) {

    try {

        User user=userRepository.findByUsername(username).get();
        List<ShoppingCart> cart=cartRepository.getCartFromUser(user.getId());
        List<ShoppingCart> cart2=new ArrayList<>();
        cart2.addAll(cart);

        if (user.getBalance()-getTotal(games)<=0.00) {
            throw new Exception("Insufficient balance");
        }

        for (Game game : games) {
            Order order= new Order(user,game);
            orderRepository.save(order);
            for (ShoppingCart shoppingCart : cart2) {
                if (shoppingCart.getGame().getId()==game.getId()) {
                    cartRepository.deleteById(shoppingCart.getId());
                    cart.remove(shoppingCart);
                }
            }
        }
        for (ShoppingCart shoppingCart : cart) {
            shoppingCart.setUser(user);
        }
        user.setShoppingCart(cart);
        user.setBalance(user.getBalance()-getTotal(games));
        userRepository.save(user);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

7. Bibliografía.

Angular:

- <https://www.youtube.com/watch?v=rSEFTQwpmnQ>
- <https://www.youtube.com/watch?v=soInCF7nbDw>
- https://www.youtube.com/watch?v=RK_ZPKn_I3I

WebSocket:

- <https://www.youtube.com/watch?v=wyMAggMz2PE>
- <https://www.youtube.com/watch?v=7TGAe5pG8Yg>
- <https://www.youtube.com/watch?v=RtFEFEIstL4&t=323s>

Formularios reactivos angular:

- <https://www.youtube.com/watch?v=HqRVzOp1v4k>
- https://www.youtube.com/watch?v=PqJ_PkcIoWQ
- <https://www.youtube.com/watch?v=Yi1rjRoEaJ0>

Spring Boot Security:

- <https://spring.io/guides/gs/securing-web>
- <https://www.youtube.com/watch?v=ZzpDylJizjo>
- <https://www.youtube.com/watch?v=pmSJTrOWi7w>

Rutas Angular:

- <https://docs.angular.lat/tutorial/toh-pt5>
- <https://www.youtube.com/watch?v=DiZ08Tzihq8>
- <https://www.youtube.com/watch?v=cqydP5Aq51I>

BehaviorSubject:

- <https://www.learnrxjs.io/learn-rxjs/subjects/behaviorsubject>
- <https://www.youtube.com/watch?v=vUC-ospC-t0>
- https://www.youtube.com/watch?v=u6Y6_QmjuF8

Bootstrap:

- <https://getbootstrap.com/docs/5.3/getting-started/introduction/>