UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación web para el seguimiento de videojuegos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Patiño Serrano, Pablo

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web para el seguimiento de videojuegos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Pablo Patiño Serrano

Tutor: Pedro Jose Valderas Aranda

Curso 2019-2022

Resumen

El sector de los videojuegos se ha convertido en uno de los pilares en la industría del entretenimiento, y cada año que pasa su éxito es mayor, por ello, la oferta de videojuegos que hay en el mercado es tan amplia que puede llegar a ser complicado mantenerse al día con las últimas novedades o llevar un seguimiento de juegos que capten el interés del usuario. Actualmente existen aplicaciones en las que se puede explorar y gestionar listas de videojuegos, pero muchas de estas intentan abarcar demasiado y acaban implementando funcionalidades que son irrelevantes en cuanto al seguimiento de videojuegos se trata. Por ello se propone una aplicación cuya función principal sea solamente la de explorar videojuegos por diferentes criterios y guardarlos en listas para llevar un seguimiento de los mismos.

Para desarrollar esta aplicación nos hemos apoyado principalmente en estas tres herramientas: *MySQL* para gestionar la base de datos, *Node.js* para comunicarnos con la base de datos y ofrecer los recursos necesarios a la interfaz web, que ha sido creada con *Angular*, desde donde haremos las peticiones pertinentes a nuestra base de datos y a fuentes de datos externas para poder proporcionar al usuario una amplia variedad de información relacionada con el mundo de los videojuegos.

Palabras clave: Videojuegos, explorar, seguimiento, desarrollo web

Resum

El sector dels videojocs s'ha convertit en un dels pilars en la industría de l'entreteniment, i cada any que passa el seu èxit és major, per això, l'oferta de videojocs que hi ha en el mercat és tan àmplia que pot arribar a costar mantindre's al dia amb les últimes novetats o portar un seguiment de jocs que capten l'interés de l'usuari. Actualment existeixen aplicacions en les quals es pot explorar i gestionar llistes de videojocs, però moltes d'aquestes intenten abastar massa i acaben implementant funcionalitats que són irrellevants quant al seguiment de videojocs es tracta. Per això es proposa una aplicació la funció principal de la qual siga solament la d'explorar videojocs per diferents criteris i guardar-los en llistes per a portar un seguiment d'aquests.

Per a desenvolupar aquesta aplicació ens hem recolzat principalment en aquestes tres ferramentes: *MySQL* per a gestionar la base de dades, *Node.js* per a comunicar-nos amb la base de dades i oferir els recursos necessaris a la interfície web, que ha sigut creada amb *Angular*, des d'on farem les peticions pertinents a la nostra base de dades i a fonts de dades externes per a poder proporcionar a l'usuari una àmplia varietat d'informació relacionada amb el món dels videojocs.

Paraules clau: Videojocs, explorar, seguiment, desenvolupament web

Abstract

The video game sector has become one of the pillars in the entertainment industry, and with each passing year its success is greater, therefore, the offer of video games on the market is so vast that it can be difficult to keep up with the latest news or keep track of games that capture the interest of the user. Currently there are applications in which you can explore and manage video game lists, but many of them try to cover too much and end up implementing functionalities that are irrelevant when it comes to monitoring video games. For this reason, an application is proposed whose main function is only to explore video games by different criteria and save them in lists to keep track of them.

To develop this application we have relied mainly on these three tools: *MySQL* to manage the database, *Node.js* to communicate with the database and offer the necessary resources to the web interface, which has been created with *Angular*, from where we will make the pertinent requests to our database and to external data sources in order to provide the user with a wide variety of Information related to the world of video games.

Key words: Videogames, explore, tracking, web development

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Los videojuegos han adquirido una gran importancia en la sociedad actual y se han convertido en una forma de entretenimiento, comunicación y expresión cultural muy relevante. Hoy en día se estima que alrededor de 3 billones de personas juegan a videojuegos de forma frecuente [1], y los números van aumentando cada año que pasa.

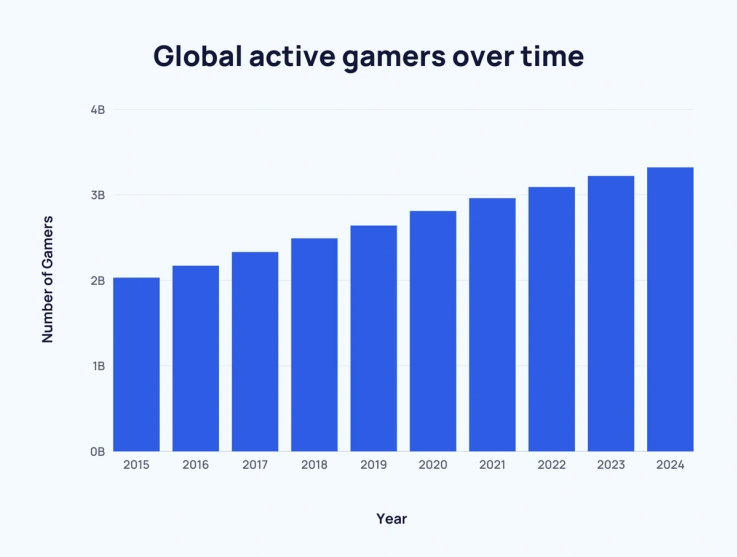

Figura 1.1: Número de jugadores activos

Estos números tienen mucho que decir de los videojuegos como sector. Según la *Digital Entertainment and Retail Association* (ERA), los videojuegos representaron el 42,1 % de los ingresos totales de entretenimiento en 2022 [2]. Para poner un poco más en contexto, *Grand Theft Auto V* estrenado en 2013 por la compañía *Rockstar*, es el producto de entretenimiento que más rápido ha alcanzado los 1000 millones de dólares en ingresos de toda la historia, y hasta la fecha ha recaudado mas de 6000 millones de dólares desde su estreno [3]. La película más taquillera de la história es *Avatar* con una recaudación de 2.92 billones de dólares [4], menos de la mitad que *GTA V*. Otro dato que respalda la importancia del sector de los videojuegos es el de *Pokemon*. *The Pokemon Company* ha sabido trasladar el éxito de la marca fuera del ámbito de los videojuegos, convirtiéndose así en la franquicia de medios más valiosa del mundo [5].

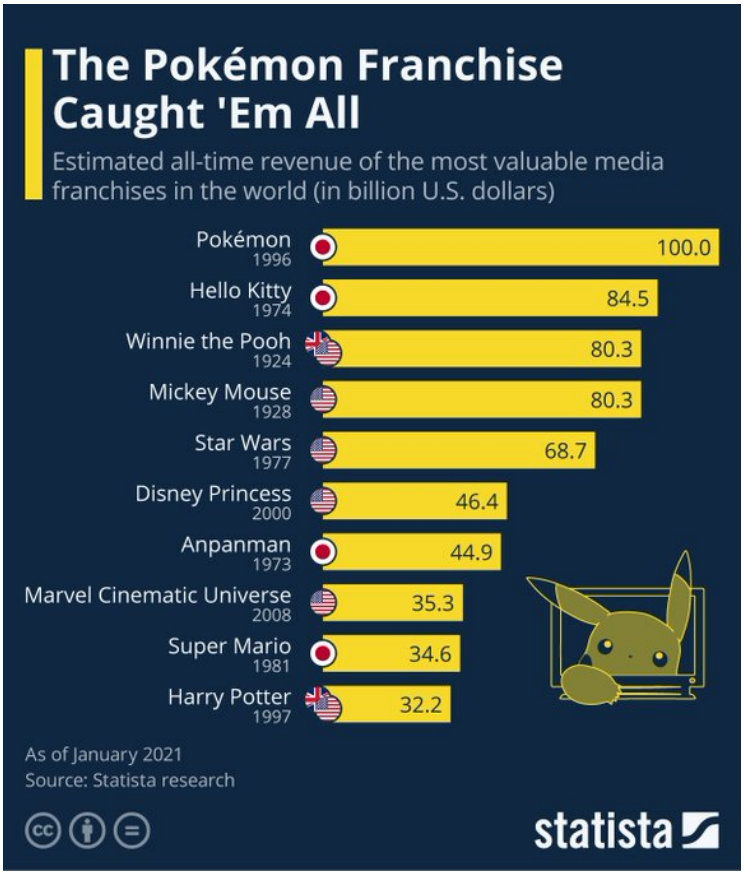

Figura 1.2: Franquicias de medios más valiosas

1.1 Motivación

Debido a los números mencionados anteriormente, los videojuegos que existen en la actualidad y los que se estrenan cada año son muchos. Por esto, llevar un seguimiento de los mismos y enterarse de las últimas novedades puede ser una tarea muy complicada. Como veremos en la sección 2, en la actualidad ya existen aplicaciones que permiten al usuario explorar y seguir videojuegos, aparte de otras funcionalidades relacionadas, a pesar de esto, se ha creido conveniente desarrollar un tipo de aplicación cuyo punto fuerte sea la facilidad y simplicidad a la hora de explorar y buscar juegos intentando eliminar cualquier funcionalidad que se aleje de este foco principal.

1.2 Objetivos

El objetivo principal de este TFG es el desarrollo de una aplicación que permita al usuario explorar y buscar entre una gran variedad de videojuegos, además de poder llevar un seguimiento de los juegos que el usuario crea conveniente. Más específicamente, los objetivos principales que busca poder cumplir la aplicación son los siguientes:

1. Buscar videojuegos por diferentes criterios y ordenar los resultados.
2. Ver juegos estrenados recientemente.
3. Poder ver información detallada de los videojuegos.

4. Mostrar noticias recientes relacionadas con el mundo de los videojuegos.
5. Guardar videojuegos en tres diferentes listas: *Pendientes*, *Jugando* o *Finalizados*
6. Gestionar las listas, puediendo eliminar juegos de estas o pasarlos de una lista a otra.

1.3 Estructura de la memória

Una vez hecha una breve introducción sobre la aplicación, en las siguientes secciones se va a entrar en detalle sobre los diferentes aspectos relevantes a la hora de diseñar y desarrollar la aplicación. Los puntos que veremos, en orden, son los siguientes:

2. **Estado del arte**: en esta sección, se llevará a cabo una investigación para identificar y analizar aplicaciones existentes que ofrecen funcionalidades similares a la aplicación propuesta. Esto ayudará a comprender el mercado, la competencia y las oportunidades para diferenciarse.
3. **Metodología**: aquí se listarán y se describirán las metodologías más usadas en el desarrollo de software, viendo sus ventajas y desventajas, y veremos cual de estas se ajusta mejor a las necesidades del proyecto.
4. **Análisis de requisitos**: en esta fase, se detallan los requisitos funcionales y no funcionales de la aplicación. Se detallarán las funcionalidades en forma de casos de uso para ver como el usuario puede realizar estas tareas.
5. **Análisis conceptual y diseño de interfaces**: se verán las relaciones de los objetos de la aplicación, como estos se gestionan en la base de datos, además de ver los prototipos utilizados en el diseño de la interfaz gráfica.
6. **Desarrollo de la solución**: en este punto, se especificarán las herramientas, lenguajes de programación y tecnologías que se utilizarán para desarrollar la aplicación. Además se respaldarán las explicaciones con ejemplos de código para entender las partes claves de la aplicación.
7. **Producto desarrollado**: mediante capturas de pantalla se verá el resultado final de la aplicación, viendo como el usuario realiza las acciones en la versión final.
8. **Validaciones**: con la aplicación ya desarrollada, se detallará que pruebas y validaciones se han realizado, sobretodo las referidas a la interfaz gráfica y de como el usuario interactúa con ella.
9. **Conclusiones**: por último se verán si los objetivos establecidos al principio han sido cumplidos y posibles mejoras para la aplicación en un futuro.

CAPÍTULO 2
Estado del arte

Debido a la popularidad creciente de los videojuegos, actualmente existen diversas herramientas y aplicaciones que nos permiten realizar acciones como las descritas en el punto anterior, pero muchas de estas aplicaciones intentan abarcar demasiado con funcionalidades que al final acaban siendo irrelevantes o que poca gente usa.

Lo que busca mucha gente simplemente es tener un sitio donde buscar juegos y poder llevar un seguimiento de los mismos, pero las aplicaciones que nombraremos a continuación se alejan de esta funcionalidad principal y algunas de ellas acaban derivando en una especie de red social en la que poca gente acaba participando.

2.1 Librerias de videojuegos

La primera aplicación que se va a nombrar es la de Rawg Games. Esta aplicación es la que más se ajusta a las necesidades descritas, también mencionar que tiene un servicio de *API*[1], que es el que se ha usado en el desarrollo de la aplicación para poder obtener información sobre los videojuegos. Es una buena fuente de datos abierta con una gran cantidad de información y metadatos[2] disponibles.

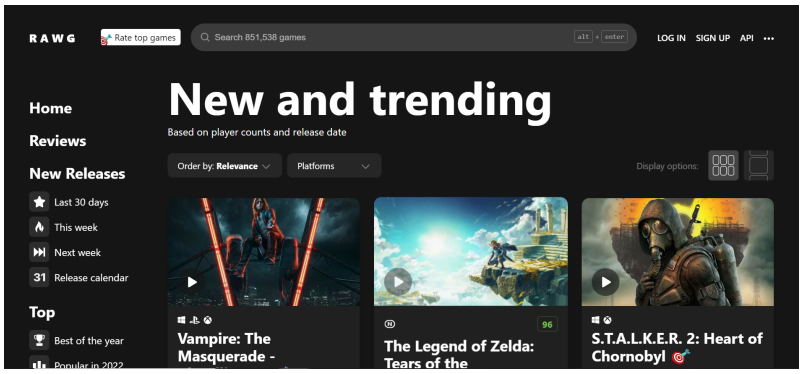

Figura 2.1: Interfaz principal de RAWG Games

Dicho esto, también tiene algunas funcionalidades que se alejan del foco principal, que es explorar videojuegos, como pueden ser la de ver tus seguidores o la gente que te sigue (figura 2.2). Debido a estas funcionalidades que hacen el uso de la aplicación más

[1] *API (Application Programming Interface)*: forma estandarizada en la que las aplicaciones pueden comunicarse y cooperar.

[2] Metadatos: conjunto de datos que aportan información extra sobre un recurso.

laborioso, mucha gente acaba optando por buscarse sus propios métodos para llevar un seguimiento de los videojuegos, como por ejemplo crearse una hoja de *excel*[3].

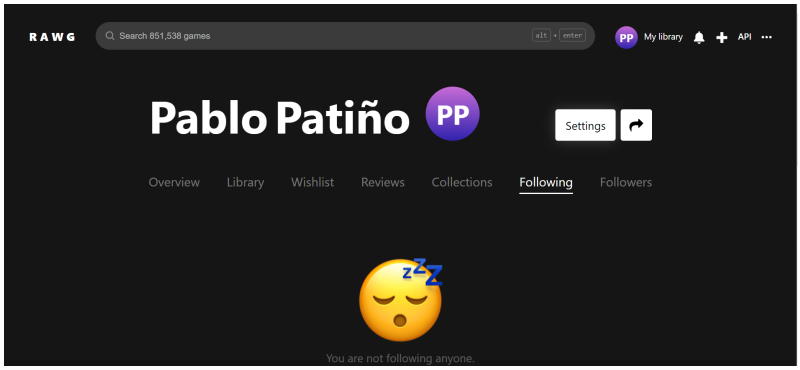

Figura 2.2: Función social de RAWG Games

Además, tambien se centra en la funcionalidad como *API* (figura 2.3), llegando a ofrecer hasta servicios de pago que permiten obtener un mayor número de metadatos. A la hora de registrarse en la aplicación se asume que también se va a hacer uso de la *API*, y hace algunas preguntas referentes al uso que se le va a dar a la misma, por lo que algunos usuarios al ver esto y no entender pueden decidir no continuar y abandonar la aplicación.

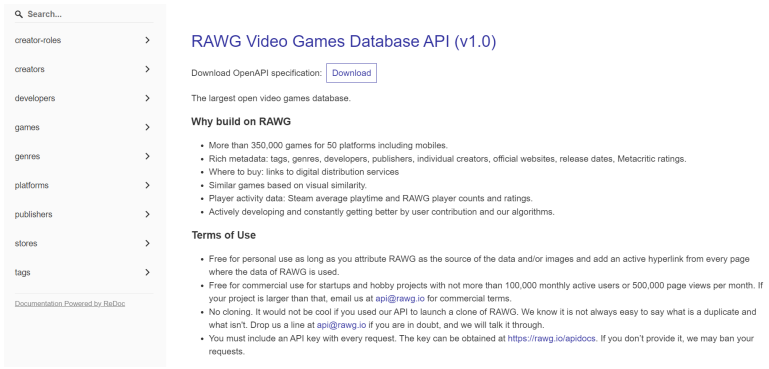

Figura 2.3: Documentación de la *API* de Rawg Games

Otra alternativa interesante para la gestión de videojuegos es la de *Playnite*. En este caso se trata de una aplicación de escritorio que permite importar los juegos de otras plataformas y tener una librería común para todos esos juegos. Si no se sincroniza con ninguna otra plataforma *Playnite* acaba siendo una opción con muchas carencias, ya que no tiene función de explorar o buscar juegos y añadirlos a tus librerías, aunque se pueden añadir juegos manualmente (figura 2.4) no es la opción más recomendable, ya que para añadir un único juego y tener una mínima información se deben completar muchos campos.

2.2 Plataformas de distribución de videojuegos

Otras aplicaciones a tener en cuenta para el estudio previo son las aplicaciones de distribución de videojuegos. Estas plataformas se centran principalmente en la venta digital de videojuegos, pero también tienen la opción de poder crear librerías y hacer un

[3] *Excel*: programa que permite gestionar hojas de cálculo.

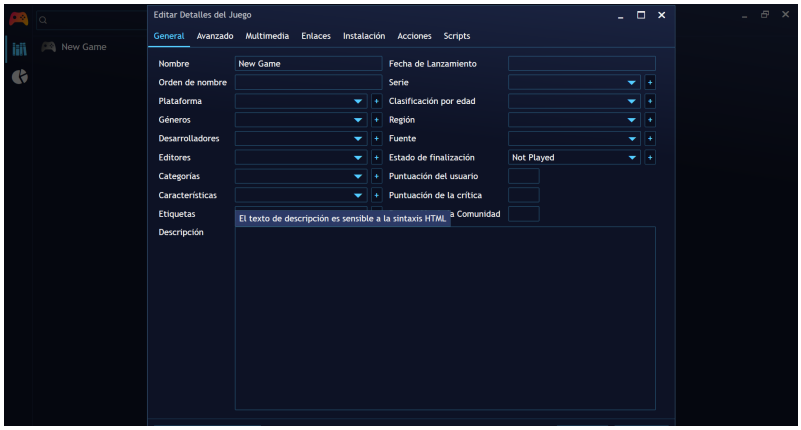

Figura 2.4: Añadir juego en *Playnite*

seguimiento de ciertos videojuegos. En este aspecto están limitadas ya que no todas las plataformas poseen todos los videojuegos por temas de derechos y exclusividades. Esta es una cuestión que enfada a los usuarios que juegan de forma habitual, ya que si se quiere estar al día con todos los lanzamientos van a tener que tener una cuenta en al menos tres o cuatro plataformas diferentes, cada una con interfaces y características distintas. Es por ello, que en la aplicación que se va a desarrollar se propone como mejora para implementar en un futuro el poder sincronizar las cuentas de estas plataformas y poder exportar los juegos en la librería de tu cuenta, indicando de alguna manera de que plataforma proviene el juego y así tener una librería común.

La aplicación más conocida y popular de este tipo es *Steam*, llegando a tener más de 33 millones de usuarios activos de forma simultánea [6]. Aparte de ser la librería online de videojuegos por excelencia, *Steam* también ofrece facilidades y herramientas a los pequeños creadores para poder publicar y vender sus videojuegos. Es cierto que *Steam* también tiene las funcionalidades de red social, pero en este caso tiene más sentido, no como en las aplicaciones que son exclusivamente librerías. *Steam* actúa como *launcher*[4] por lo que si quieres jugar con amigos tienes que ser capaz de agregar contactos y poder comunicarte con ellos.

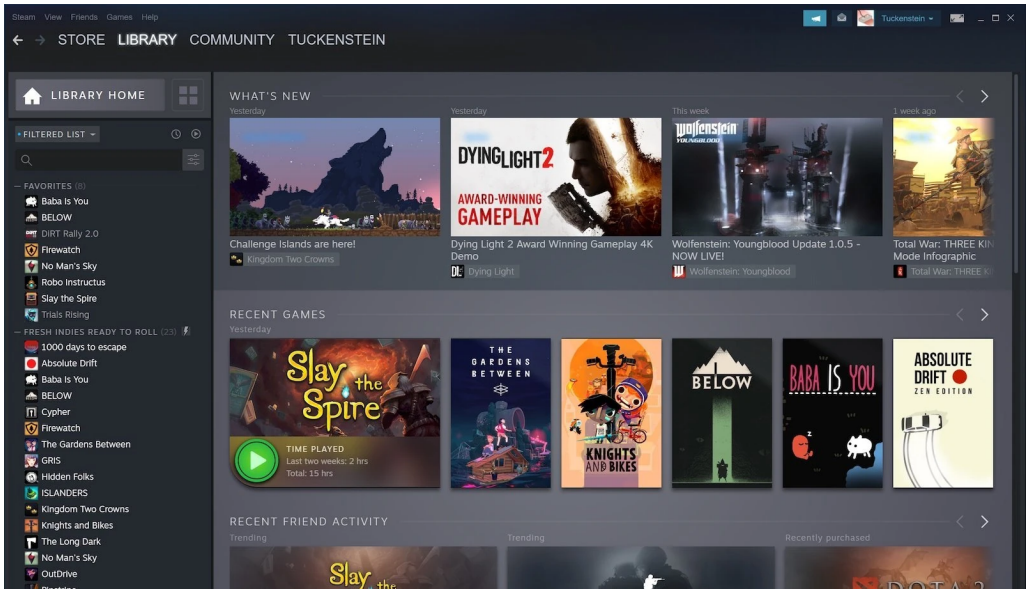

Figura 2.5: Libreria de *Steam*

[4] *Launcher* (videojuegos): aplicación desde donde se ejecutan los videojuegos para poder jugar a ellos.

Además, *Steam* también ofrece una base de datos pública en la que se pueden ver estadísticas interesantes como pueden ser los juegos más jugados y el número de jugadores activos, ventas o últimos juegos añadidos.

Y *Valve*, la empresa desarrolladora de *Steam* no se conforma con ser la plataforma online de videojuegos más utilizada, ya que hace poco lanzó al mercado un producto llamado *Steam Deck*, una especie de ordenador portátil diseñado para asemejarse lo más posible a una consola portátil, para que los usuarios puedan acceder a sus librerías desde todos los lugares.

Por poner otro ejemplo de este tipo de plataformas, tenemos a *Epic Games* (figura 2.6). Su función principal es la misma que *Steam*, ser un sitio donde comprar videojuegos y acceder a ellos de forma fácil. En cuanto a opciones de librería, Steam ofrece mucha más variedad de videojuegos, aún así el número de usuarios de *Epic Games* no se queda atrás, ya que desde esta plataforma es el único lugar desde donde acceder a *Fortnite*, uno de los videojuegos más jugados actualmente.

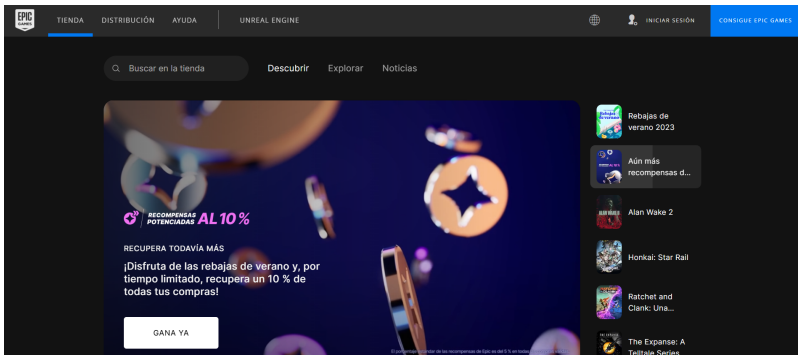

Figura 2.6: Página principal de *Epic Games*

2.3 Críticas al estado del arte

Como se ha observado en ejemplos anteriores, es común que muchas aplicaciones similares intenten incorporar una variedad de funcionalidades adicionales, lo que a menudo resulta en una experiencia complicada y puede desviar la atención del propósito central: la exploración de videojuegos. La aplicación propuesta se destaca al enfocarse de manera exclusiva en la búsqueda y seguimiento de videojuegos, adoptando un enfoque minimalista y eliminando cualquier característica que podría considerarse superflua o que podría distraer a los usuarios de su objetivo principal. Dicho esto, podemos dedicarnos plenamente a proporcionar una experiencia agradable y sin complicaciones para los usuarios a la hora de buscar videojuegos. Nuestro objetivo principal es facilitar la exploración de videojuegos de manera eficiente y efectiva, sin distracciones innecesarias, para que los usuarios puedan encontrar rápidamente lo que están buscando.

CAPÍTULO 3
Metodología

Uno de los puntos claves a la hora de desarrollar *software* es la organización. Una mala organización puede llevar fácilmente al fracaso aunque se tenga al mejor equipo de programadores. Todo proceso *software* ha de estar fundamentado en una metodología concreta, aún por muy pequeño que sea este desarrollo, ya que el *software* es un producto muy delicado y confusiones o errores de coordinación puede resultar en fracaso. El equipo de desarrollo de la aplicación propuesta está conformado por una sola persona, pero no por ello se puede empezar a programar sin tener un plan o método establecido. Es por ello que en este apartado se van a discutir distintos tipos de metodologías, sus ventajas y sus desventajas y cual de éstas se ha usado en el desarrollo de la aplicación.

3.1 Metodologías ágiles

Las metodologías ágiles son un conjunto de enfoques y prácticas para la gestión y desarrollo de proyectos que se ha vuelto muy popular en el ámbito de desarrollo de *software*, la mayoría de empresas de este sector utilizan estas metodologías. En este tipo de metodologías se valora la flexibilidad, la colaboración y la capacidad de adaptación para abordar cambios que puedan surgir durante el ciclo de vida de un proyecto [7].

Algunas de las metodologías ágiles más conocidas son *Scrum*, *Kanban* o *Extreme Programming*. Cada una de estas tiene sus particularidades pero todas comparten unas características fundamentales, que son las siguientes:

- Colaboración con el cliente.
- Iteración y entrega incremental.
- Autogestión y equipos multifuncionales.
- Adaptación al cambio.
- Enfoque en la calidad.
- Retroalimentación continua.

El proceso típico en una metodología ágil suele estar compuesto por *sprints*[1] o ciclos de trabajo cortos en los que se planifica, desarrolla, prueba y entrega un conjunto de funcionalidades concretas. Después de cada ciclo, se realiza una revisión y se ajusta el plan en función de la retroalimentación recibida.

[1] *Sprint*: período breve de tiempo en el que se trabaja para completar un trabajo determinado.

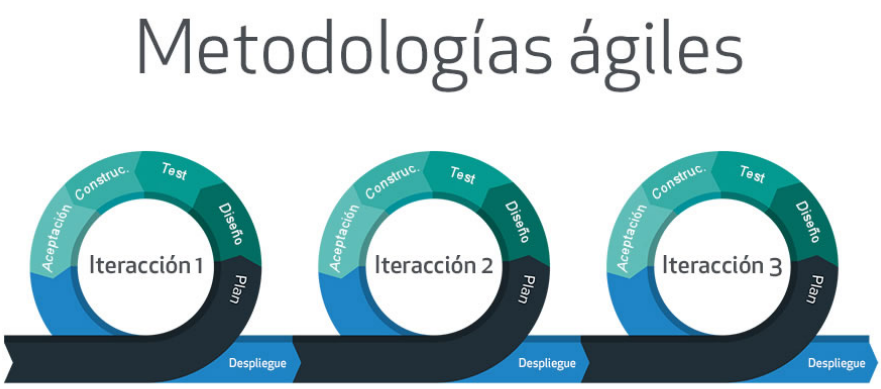

Figura 3.1: Metodologías ágiles

La aplicación propuesta va a ser desarrollada solamente por una persona, y no hay clientes que demanden o que proporcionen realimentación, por lo que los cambios que se produzcan en el desarrollo serán mínimos, principalmente serán nuevas funcionalidades o cambios de enfoque que puedan surgir sobre la marcha. Además, tampoco se trata de un proyecto tan grande como para tener en cuenta este tipo de metodología a la hora del desarrollo.

3.2 Modelo en cascada

Este modelo, al contrario que el ágil, es uno de los enfoques más tradicionales y lineales para el desarrollo de software. Se basa en una secuencia de fases bien definidas, se requiere una planificacion detallada desde el principio y sigue una estructura rígida. Ampliamente usada en el pasado, su popularidad ha decaido en favor de las metodologías ágiles. El modelo en cascada se compone de las siguientes fases, ejecutadas secuencialmente:

1. Requisitos: fase inicial en la que se definen y recopilan los requisitos del *software* en colaboración con el cliente o el equipo de de usuarios finales.
2. Diseño: los requisitos se utilizan para diseñar la arquitectura del *software*, creando un diseño técnico de como se construirá el sistema.
3. Implementación: teniendo en cuenta el diseño, el equipo de desarrollo procede a la implementación del *software*.
4. Pruebas: con el código completo, se realizan pruebas para verificar que el producto funcione correctamente.
5. Despliegue: el *software* se implementa y se pone en producción para que los usuarios finales lo utilicen.
6. Mantenimiento: conforme se vaya usando el producto, se pueden encontrar errores o necesidades de mejora.

Este modelo proporciona una estructura clara y bien definida para el desarrollo de *software*, pero tiene ciertas limitaciones. Su enfoque secuencial puede dificultar la adaptación a cambios en los requisitos o en el entorno del proyecto. Si se descubren problemas

en etapas posteriores, regresar y corregir errores en etapas anteriores puede ser costoso y consumir mucho tiempo.

3.3 Modelo incremental

Este modelo es una metodología que se basa en la entrega progresiva y secuencial de funcionalidades completas del sistema. A diferencia del modelo en cascada, que sigue un enfoque lineal y secuencial, el modelo incremental permite entregar versiones parciales del *software* a medida que se va desarrollando. En este enfoque, el *software* se construye y se entrega en múltiples incrementos, cada uno de los cuales agrega funcionalidades o características al producto final. Cada incremento representa una versión completa y funcional del *software*, aunque no necesariamente contiene todas las características planificadas. A medida que se complete cada incremento, el equipo de desarrollo recibe retroalimentación del cliente o usuarios finales, lo que permite realizar ajustes y mejoras mejoras para el siguiente incremento [8].

Este modelo es el que mejor se ajusta a las necesidades del proyecto y a la forma de trabajar que se tenía planeada desde un principio. Se tenía una visión general de como iba a ser la aplicación, pero se iba desarrollando por funcionalidades o pantallas. Gracias a *git*[2] se tenía siempre una versión estable y funcional del proyecto con las funcionalidades desarrolladas anteriormente. A la hora de empezar el desarrollo de una nueva funcionalidad, se creaba una nueva rama[3], y se programaba. Cuando ésta estaba finalizada, se hacían las pruebas pertinentes y si todo funcionaba correctamente, se integraba a la rama principal.

Para concluir este punto, podemos afirmar que la metodología usada el desarrollo de la aplicación propuesta ha sido la incremental, aunque al principio se recurrió a las fases tempranas del modelo en cascada, para tener un diseño y una visión general de la aplicación, para tener una base sólida desde donde empezar a trabajar.

[2] *Git*: *software* de control de versiones de código.

[3] rama (*git*): copia aislada del código que permite desarrollar nuevas características o corregir errores con seguridad.

CAPÍTULO 4
Análisis de requisitos

En este capítulo se entrará más en detalle sobre las funcionalidades de la aplicación web. Se listarán todas las funcionalidades y todos los requsitos que debe cumplir el sistema. Una vez listados, se detallarán cada uno de ellos en los casos de uso, donde se verá más en profundidad como el usuario puede llevar a cabo estas funcionalidades.

4.1 Requisitos iniciales

Como se ha comentado en puntos anteriores, la aplicación a desarrollar permite que el usuario pueda obtener información de una amplia variedad de videojuegos, ya sea buscando por el título o por diferentes criterios como la fecha de estreno, el género o la desarrolladora. Al poder buscar por fecha también permite estar al tanto de las últimas novedades. Al seleccionar algún juego en concreto se puede ver información extra, como descripción, capturas de pantalla o juegos relacionados. Estos juegos podrán ser guardados en la listas del usuario, en *Pendiente*, *Jugando* o *Finalizados*. Además en la página principal de la web se verán noticias recientes relacionadas con el mundo de los videojuegos. Tras esta pequeña descripción de las funcionalidades básicas de la aplicación se van a listar todos los requisitos que tiene que cumplir el sistema.

- Nuevos usuarios pueden registrarse.
- Usuarios ya registrados podrán iniciar sesión.
- Usuarios con sesión iniciada podrán cerrar sesión para salir de la aplicación.
- El usuario tiene que ser capaz de ver los juegos estrenados recientemente, ya sea de la última semana, el último mes o los últimos tres meses.
- El usuario tendrá la opción de ver una lista con las desarrolladoras más populares y ver los juegos de éstas.
- También se podrán ver los juegos más populares de todos los géneros.
- Tendrá que haber una barra de búsqueda desde donde el usuario pueda buscar por el título. Se podrán seleccionar juegos que salgan en sugerencias conforme el usuario va escribiendo o ver todos los resultados del título introducido.
- En todas las páginas de resultados se tiene que poder ordenar los juegos por título, fecha de publicación o tiempo de juego.

- Seleccionando el videojuego podrás ver los detalles de éste, viendo su descripción, fecha de publicación, géneros, desarolladoras y plataformas, además de capturas de pantalla y juegos relacionados. Seleccionando alguno de estos géneros o desarrolladora te llevará a la pantalla de resultados de esa selección.
- El usuario tiene que ser capaz de añadir juegos a sus listas.
- El usuario podrá ver los juegos de sus listas.
- Las listas pueden ser gestionadas, es decir, se podrán eliminar los juegos y pasarlos de una lista a otra.
- La aplicación tendrá una sección de información referente al uso de las *APIs* y a la creación de la propia aplicación.

Listadas todas las funciones y requisitos que tiene que ser capaz de cumplir el sistema, ahora en la proxima sección se va a entrar en detalle en cada uno de estos puntos, viendo una pequeña descripción y como el usuario podrá llevar a cabo estas tareas.

4.2 Casos de uso

Primeramente, antes de listar los casos de uso es interesante representarlos en un diagrama para poder ver de una manera más general que acciones puede realizar el usuario. En este diagrama (ver figura 4.1) se han omitido las acciones de inicio de sesión y de registro, pues se ha asumido que el usuario ya se encuentra dentro de la aplicación. Para ver las acciones que es capaz de realizar el usuario lo representamos con una linea simple desde el usuario a la acción. Con la línea discontinua y la etiqueta *include* indicamos que hay algunas acciones que necesitan otra acción previa para llevarse a cabo, por ejemplo, para poder ver los detalles de algún videojuego primero es necesario realizar algún tipo de búsqueda.

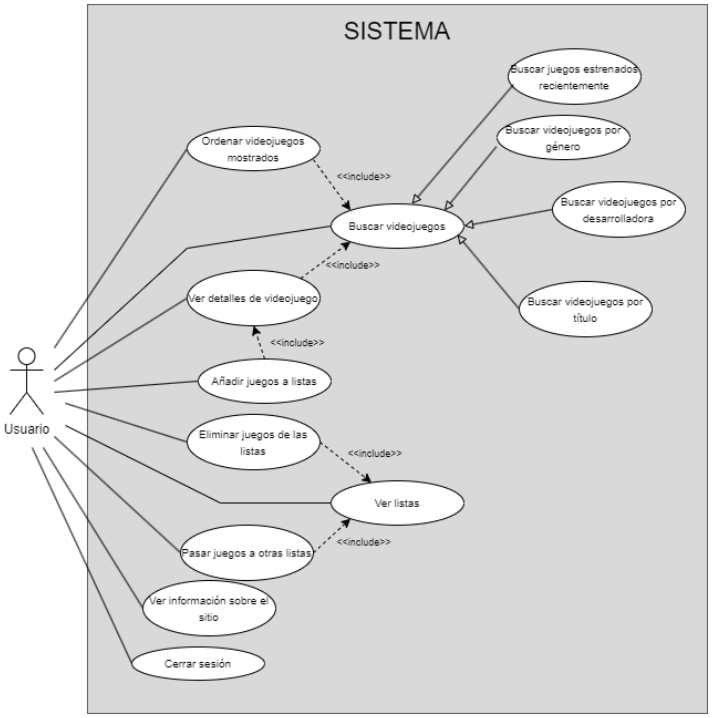

Figura 4.1: Diagrama de casos de uso

Caso de uso: **registro**

- Descripción: el usuario puede registrarse en la aplicación rellenando los campos de nombre de usuario, correo, contraseña y confirmación de contraseña. Si el nombre de usuario o correo ya están en el sistema se avisará al usuario con un mensaje. La contraseña tendrá que tener un mínimo de seguridad, si no es así el sistema también avisará
- Actores: nuevo usuario o usuario ya registrado que quiera abrirse otra cuenta.
- Flujo de eventos:
  1. En la página de inicio de sesión el usuario pulsa la opción de registrarse.
  2. En el formulario de registro el usuario completa todos los campos y presiona la opción de registro.
- Precondiciones: el usuario no tiene que tener iniciada la sesión.
- Postcondiciones: si el usuario rellena todos los campos correctamente, el sistema mostrará un *pop-up*[1] informando que el usuario se ha registrado correctamente. También redirigirá a la pantalla de inicio de sesión.

Caso de uso: **iniciar sesión**

- Descripción: el usuario introduce sus credenciales para entrar en la aplicación. Si estas credenciales son incorrectas el sistema avisará con un mensaje.
- Actores: usuario registrado.
- Flujo de eventos:
  1. En el inicio de sesión el usuario introduce sus credenciales.
- Precondiciones: el usuario no tiene que tener iniciada la sesión.
- Postcondiciones: si el usuario introduce los campos correctamente, el sistema le llevará a la página principal.

Caso de uso: **cerrar sesión**

- Descripción: el usuario con sesión iniciada puede cerrar sesión para salir de la aplicación, haciendo que ésta le devuelva a la página de inicio de sesión.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En el menú lateral de la página principal el usuario pulsa la opción de cerrar sesión.

[1] *Pop-up*: ventana emergente para mostrar información.

2. El sistema con un *pop-up* preguntará de nuevo si el usuario quiere cerrar sesión. Si el usuario pulsa que no, le devolverá a la página que estabas, si confirma el cierre de sesión lo llevará a la página de inicio de sesión.

- Precondiciones: -
- Postcondiciones: después de cerrar sesión el sistema le llevará a la página de inicio de sesión.

Caso de uso: **ver juegos estrenados recientemente**

- Descripción: en la página principal, el usuario podrá ver unos pocos de los juegos estrenados en la última semana. Si quiere ver más puede ir a la página dedicada para los juegos estrenados recientemente y ver más juegos.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En la página principal verá algunos juegos estrenados en la última semana, si quiere ver más puede pulsar la opción de "Ver más..." que hay justo debajo de los juegos o pulsar la opción del menú lateral de buscar juegos estrenados recientemente.
  2. En esta nueva página el usuario ya verá con más detalle más de los juegos estrenados en la última semana. Si quiere puede ver los juegos estrenados en el último mes o de los últimos tres meses pulsando el *radio button*[2] que lo indica.
- Precondiciones: -
- Postcondiciones: seguidos los pasos anteriores el usuario será capaz de ver una amplia variedad de resultados en forma de lista, en la que podrá ir cambiando de página o ordenar los resultados por distintos criterios.

Caso de uso: **buscar por desarrolladora**

- Descripción: el usuario podrá ver una lista con las desarrolladoras más populares, viendo el nombre y la cantidad de juegos que ha producido la misma. Pulsando en alguna desarrolladora podrá ver todos sus juegos.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En el menú lateral el usuario presiona la opción de buscar por desarrolladora.
  2. En esta nueva página el usuario podrá ver una amplia lista de desarrolladoras. Pulsa en alguna de estas.
  3. El sistema le redirigirá a una página de resultados, exactamente igual que en el caso de uso anterior.
- Precondiciones: -

[2] *Radio button*: botón de opción.

- Postcondiciones: -

Caso de uso: **buscar por género**

- Descripción: el usuario podrá ver una lista con todos los géneros de los videojuegos. Pulsando en uno de éstos el sistema le mostrará los juegos más populares del género seleccionado.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En el menú lateral el usuario pulsa la opción de buscar por género.
  2. El sistema le mostrará una página con todos los géneros disponibles. El usuario pulsa alguno de estos géneros.
  3. El usuario podrá ver la página de resultados mencionada en los casos de uso anteriores con los juegos de ese género.
- Precondiciones: -
- Postcondiciones: -

Caso de uso: **buscar por título**

- Descripción: en la cabecera de la página web hay una barra de búsqueda desde donde el usuario podrá introducir una cadena de texto y buscar juegos cuyo título coincida con esa cadena.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En la barra de búsqueda de la cabecera el usuario introduce una cadena de texto.
  2. Si hay juegos que coinciden, el sistema mostrará una pequeña lista debajo de la barra de búsqueda con algunas sugerencias de videojuegos, en donde se verá el nombre completo del juego.
  3. El usuario puede pulsar en alguna de estas sugerencias y el sistema le llevara a la página de detalles del juego.
  4. Si el usuario no está interesado en las sugerencias que le muestra el sistema puede pulsar la tecla *enter* para pasar a una nueva página de resultados donde se mostrará todos los juegos que coincidan con la cadena introducida.
- Precondiciones: tiene que existir algún juego con el título que introduzca el usuario. Si escribe por ejemplo una cadena aleatoria y no hay ningún juego con ese título aunque pulse la tecla *enter* no se mostrará ningún resultado.
- Postcondiciones: -

Caso de uso: **ordenar videojuegos mostrados**

- Descripción: cuando el usuario se encuentre en una página de resultados, este será capaz de pulsar un simple desplegable para ordenar los videojuegos por los siguientes criterios: título, fecha de publicación y tiempo de juego.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. El usuario primero tiene que realizar una búsqueda con alguno de los métodos que se han comentado en casos de uso anteriores.
  2. En la página de resultados, en la parte superior habrá una opción de ordenar, el usuario pulsa verá un desplegable con las distintas opciones de ordenación. Cuando el usuario presione alguna de estas opciones los juegos se ordenarán.
- Precondiciones: para ordenar los resultados tiene que haber más de un juego.
- Postcondiciones: -

Caso de uso: **ver detalles de videojuego**

- Descripción: el usuario pulsando en algún juego podrá ver información extra del mismo: en que plataformas está disponible, desarrolladoras, géneros, fecha de lanzamiento, una descripción del juego, capturas de pantalla y una lista de juegos relacionados.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En la página de resultados, el usuario pulsa la opción de "Detalles" del videojuego que quiera. También puede introducir el título en la barra de búsqueda y pulsar alguna de las sugerencias.
  2. El sistema le llevará a la nueva página de detalles.
- Precondiciones: -
- Postcondiciones: -

Caso de uso: **añadir juegos a listas**

- Descripción: el usuario podrá añadir juegos a sus listas, ya sea a la de *Pendientes*, *Jugando* o *Finalizados*.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. El usuario primero tiene que ir a los detalles de algún videojuego.
  2. En la página de detalles, en la parte superior derecha hay una opción de añadir. El usuario pulsa este botón y se desplegará una lista con las diferentes listas a las que añadir.

3. El usuario pulsa alguna de estas opciones.

- Precondiciones: el usuario no tiene que tener guardado ya el juego en alguna lista, sino el sistema le mostrará un *pop-up* indicando que ya tiene el juego guardado.
- Postcondiciones: si el juego se guarda correctamente el sistema lo indicará con un *pop-up*.

Caso de uso: **ver listas**

- Descripción: el usuario podrá ver todos los juegos que se ha guardado en sus listas.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En el menú lateral el usuario presiona la lista que quiera ver.
  2. El sistema le llevara a una página donde se vean los juegos que tiene guardados en esa lista.
- Precondiciones: tendrán que haber juegos guardados en las listas, si no en la página de resultados de la lista aparecerá un mensaje de que actualmente no hay juegos añadidos en esa lista.
- Postcondiciones: -

Caso de uso: **eliminar juegos de las listas**

- Descripción: el usuario desde las listas podrá eliminar los juegos que quiera.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. El usuario navega a la lista que quiera.
  2. En los resultados, debajo de cada juego aparecerá un botón de eliminar.
  3. El usuario pulsa ese botón, el sistema le muestra un *pop-up* preguntando si está seguro.
  4. Si pulsa que no, volverá a la lista, pero si pulsa que si el juego se eliminará.
- Precondiciones: tiene que haber algún juego en la lista para poder eliminarlo.
- Postcondiciones: después de eliminar el juego se recargará la página y el juego ya no aparecerá.

Caso de uso: **pasar juegos a otras listas**

- Descripción: el usuario podrá pasar juegos de una lista a otra, por ejemplo, si hay un juego en la lista de *Jugando* y el usuario ya se lo ha pasado podrá añadirlo a la lista de *Finalizados* con un simple botón.

- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. El usuario navega a sus listas.
  2. En las lista de *Pendientes* los juegos tendrán una opción de "Añadir a Jugando" y en la lista de *Jugando* la opción será "Añadir a Finalizados". El usuario simplemente tendrá que pulsar estas opciones.
- Precondiciones: la lista no puede estar vacia.
- Postcondiciones: después de pasar un juego a otra lista la página se recargará y ya no aparecerá el juego en la lista actual.

Caso de uso: **ver información sobre el sitio**

- Descripción: el usuario podrá ver una pequeña descripción del sitio web.
- Actores: usuario con sesión iniciada.
- Flujo de eventos:
  1. En el menú lateral el usuario pulsa la opción de "Sobre el sitio".
  2. El sistema le redirige a la pantalla deseada.
- Precondiciones: -
- Postcondiciones: -

CAPÍTULO 5
Análisis conceptual y diseño

Ya explicado como tiene que funcionar la aplicación, en este punto ya vamos a entrar en materia tecnológica, en ver todo lo que hay detrás para poder hacer que la web funcione como se quiere. Primero estudiaremos las relaciones de los objetos del sistema, despúes veremos que sistema de base de datos se ha usado y para finalizar este capítulo veremos los bocetos que se han utilizado como referencia a la hora de contruir las interfaces de la página web.

5.1 Diagrama de clases

Antes de empezar a comentar que sistema de gestión de base de datos se ha utilizado y el porqué, es interesante tener una visión general de los objetos que se van a gestionar en la propia base de datos. Para ello, haremos uso del diagrama de clases, donde representaremos los objetos del sistema y sus atributos.

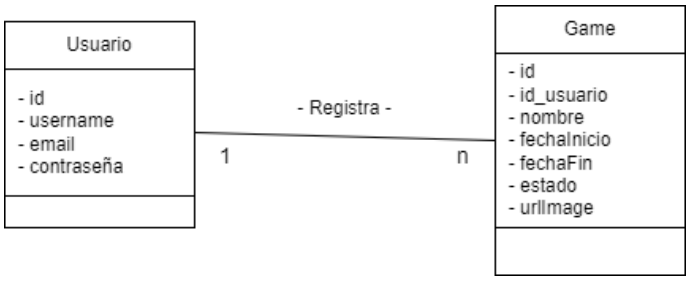

Figura 5.1: Diagrama de clases

Con un simple vistazo en el digrama observamos que en el sistema tenemos usuarios y videojuegos, y que los usuarios pueden registrar juegos, además de ver los atributos de cada uno de éstos.

5.2 Base de datos

A la hora de elegir el sistema de gestión de base de datos que se iba a utilizar, la principal característica que se ha tenido en cuenta es la simplicidad, ya que no se van a manejar objetos muy complejos. Es por eso que las principales herramientas que se tuvieron en cuenta a la hora de escoger fueron *MySQL* y *SQL Server*. Las dos utilizan el modelo relacional [9], y en cuanto a funcionalidades básicas son muy similares. *SQL Server* es un producto de *Microsoft*, por lo que todas sus ediciones son productos comerciales y requieren algún tipo de licencia. Además, esta herramienta está mas dedicada para el ámbito

empresarial y tiene algunas funcionalidades que para el proyecto que se va a desarrollar son irrelevantes, como pueden ser la integración con otras plataformas como *Microsoft Azure* o servicio de análisis de datos. Es por ello que la opción que se ha usado para la aplicación finalmente ha sido *MySQL*.

Una vez elegido el gestor tenemos que escoger una herramienta pasa poder manejar las diferentes opciones que nos proporciona. Es cierto que se puede realizar todo desde la consola, pero no es una opción muy recomendable debido a su complejidad. Finalmente, las dos herramientas que se utilizaron fueron *MySQL Workbench* y *TablePlus*. *MySQL Workbench* se ha usado principalmente para la propia creación de la base de datos y para la modificación de algunos aspectos sobre las opciones de las tablas. La creación de las tablas se ha hecho con una librería del *backend*, la cual se explicará con detalle en puntos posteriores. Aunque esta herramienta proporcione todas las opciones necesarias para gestionar una base de datos sencilla, su principal problema es la interfaz de usuario, ya que no es nada intuitiva a la hora de ver la información relativa a las tablas, además de estar muy cargada de elementos, llegando a tener cinco ventanas distintas en pantalla (figura 5.2).

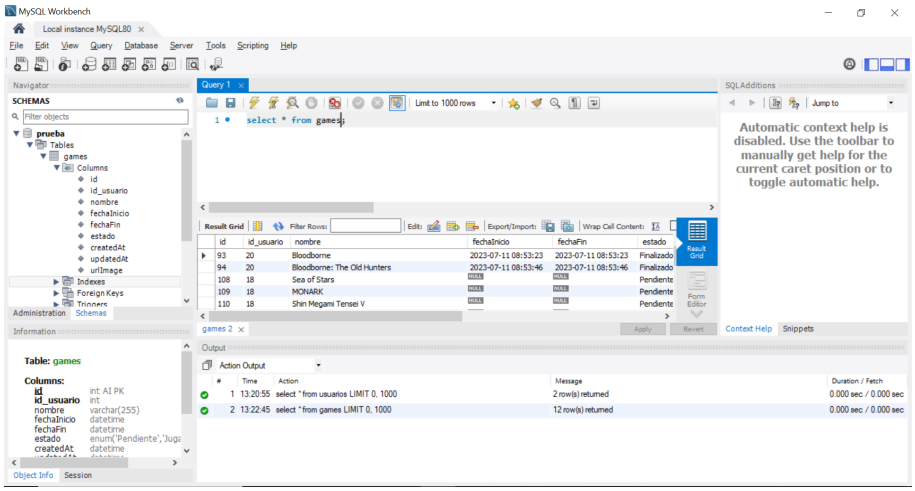

Figura 5.2: Interfaz principal de *MySQL Server*

Table Plus, al contrario que la *MySQL Server* ofrece una interfaz mucho más limpia y agradable (figura 5.3), es por ello que para ver la información de las tablas y para hacer consultas es una opción mucho más recomendable. Es cierto que no proporciona tantas opciones para la gestión de la base de datos, pero para hacer simples pruebas esta herramienta es mucho mejor que *MySQL Server*.

Como se ha comentado antes, los dos objetos principales que vamos a gestionar en nuestra base de datos van a ser los usuarios que se registren en la aplicación y los videojuegos que añadan a sus listas los usuarios. En el siguiente diagrama (figura 5.4), generado con *MySQL Workbench* podemos ver los objetos de nuestro sistema con sus diferentes columnas y como se relacionan.

Ahora veremos más en detalle lo que representan las diferentes columnas de nuestras tablas.

Tabla de usuarios

- Id: identificador de usuario generado automáticamente al crear la tabla.
- *Username*: nombre de usuario.
- *Email*: correo electrónico del usuario.

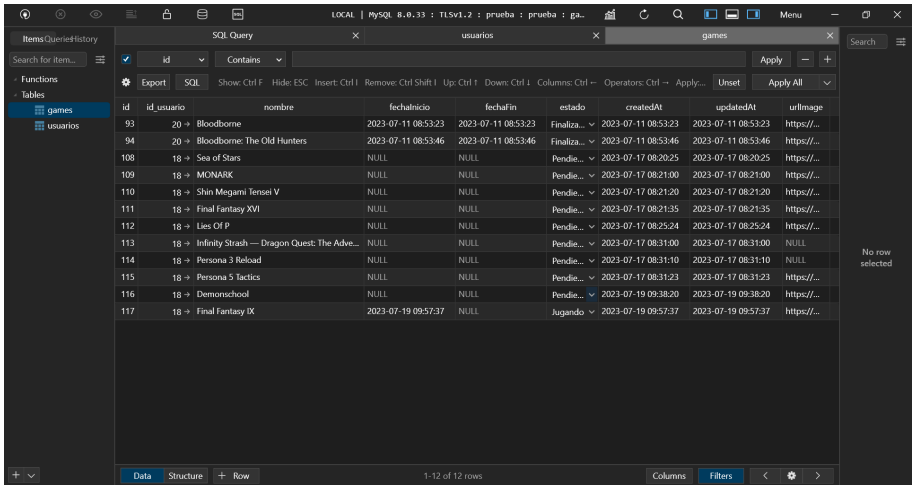

Figura 5.3: Información de la tabla *games* en *TablePlus*

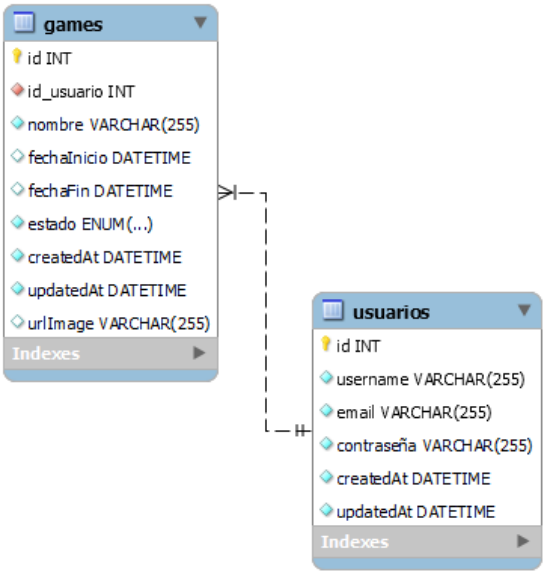

Figura 5.4: Diagrama de clases de la base de datos

- Contraseña: contraseña del usuario encriptada.
- *CreatedAt*: columna generada automáticamente por el *backend* indicando en que momento se crea un usuario.
- *UpdatedAt*: columna generada automáticamente que indica la última fecha en la que se ha modificado un usuario.

Tabla de videojuegos

- Id: identificador del videojuego generado automáticamente al crear la tabla.
- id_usuario: clave foránea que hace referencia a la tabla de usuarios.
- Nombre: título del videojuego.
- FechaInicio: fecha en la que se ha comenzado a jugar al videojuego.
- FechaFin: fecha en la que se finaliza el videojuego.

- Estado: estado en el que se encuentra el videojuego. Puede ser una de estas tres opciones: Pendiente, Jugando o Finalizado.
- *CreatedAt*: columna generada automáticamente por el *backend* indicando en que momento se registra un videojuego.
- *UpdatedAt*: última fecha en la que se ha actualizado un videojuego.
- *UrlImage*: cadena de texto que guarda la *url* de la imagen identificativa del videojuego.

Las fechas de inicio y fin de los videojuegos se actualizan según el estado. Si el videojuego se registra con un estado *Pendiente*, la fecha de inicio y fin se dejarán a nulo, si se guarda o actualiza a *Jugando*, la fecha de inicio se guardará con la fecha del día actual. Guardar un juego a *Finalizados* guardará la fecha de inicio y fin como el día actual, en cambio si se pasa un juego de *Jugando* a *Finalizados*, la fecha de inicio será la que ya estaba guardada y la de fin será la actual.

5.3 Diseño de interfaces

A la hora de diseñar las interfaces para la página web se han tenido que considerar diferentes aspectos para crear una experiencia de usuario efectiva y atractiva. Algunos elementos clave que se han tenido en cuenta son los siguientes:

- Usabilidad y facilidad de uso: la interfaz tiene que ser intuitiva y fácil de entender para los usuarios. Los elementos de navegación y las acciones deben ser claras y accesibles, haciendo que los usuarios encuentren la información sin dificultad.
- Diseño responsivo: aunque los bocetos que se van a mostrar a continuación han sido diseñados para web, las interfaces serán adaptables a diferentes dispositivos y tamaños de pantalla.
- Consistencia en el diseño: para hacer que la web tenga una coherencia visual se ha hecho uso de estilos y patrones consistentes para hacer que la interfaz sea más fácil de entender y usar.
- Retroalimentación de usuarios: en las acciones que lo requieran se mostrará al usuario ventanas de confirmación y de información extra.

Los bocetos que se van a mostrar a continuación han sido creados con la herramienta *draw.io*, que permite tanto crear interfaces, como diagramas de flujo o organigramas. Son bocetos de un nivel intermedio, su función ha sido la de dejar clara la estructura principal de la interfaz y la disposición de sus elementos. Aspectos como paleta de colores o algunas imágenes son orientativos, no son definitivos.

- **Inicio de sesión**: en esta pantalla podemos iniciar sesión en la aplicación o ir a la pantalla de registro. También vemos el logo (figura 5.6), que en el momento de hacer los bocetos éste aún no estaba creado. Hay que destacar también el uso del inglés como el idioma principal, ya que la *API* que se ha usado para obtener la información sobre videojuegos está solamente en inglés.
- **Registro**: aquí (figura 5.7) el usuario podrá crear una cuenta introduciendo el nombre de usuario, correo y contraseña

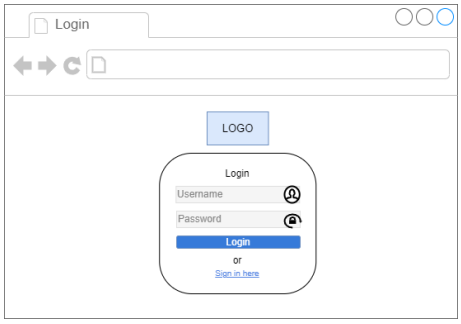

Figura 5.5: Pantalla de inicio de sesión

Figura 5.6: Logo de la aplicación

- **Menú lateral y *header***[1]: estos dos elementos (figura 5.8) siempre estarán presentes en todas las páginas. El *header* contendrá el logo y una barra de búsqueda, y en el menú lateral podremos navegar por las distintas páginas de la aplicación. En el centro de la pantalla se mostrará el contenido de las distintas páginas.

- **Juegos estrenados recientemente**: en esta página (figura 5.9) podremos ver en detalle los juegos estrenados, ya sean en la última semana, en el último mes o en los últimos tres meses, pudiendo escoger con el *radio button*. Además podrás ordenar los juegos con el botón de *"Sort by"*. En la imagen se muestran solamente dos filas, pero en la web se verán más juegos y podrás cambiar de página para ver más juegos.

- **Página principal**: esta es la página (figura 5.10) en la que se encontrará el usuario después de iniciar sesión. Podremos ver diferentes juegos estrenados recientemente y noticias relacionadas. En el menú lateral se resaltará la sección en la que se encuentre el usuario. En la imagen se ha alargado la pantalla para poder observar todo el contenido, pero de normal el usuario tendrá que hacer *scroll*[2].

[1] *Header*: parte superior de la página web
[2] *Scroll*: desplazamiento hacia arriba o hacia abajo de los contenidos de una página web

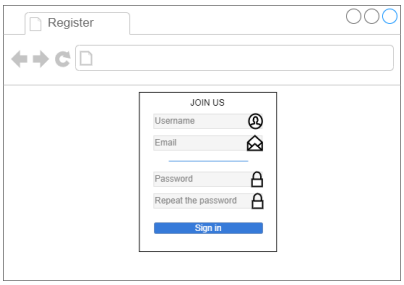

Figura 5.7: Pantalla de registro

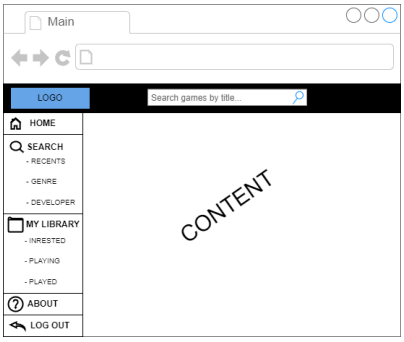

Figura 5.8: Menú lateral y *header*

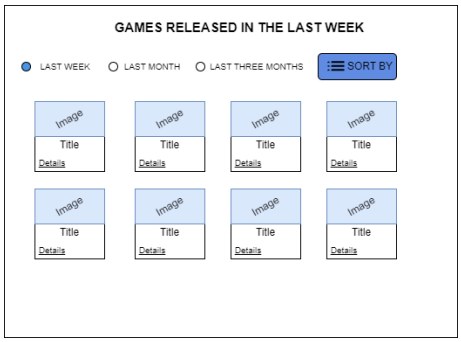

Figura 5.9: Juegos estrenados recientemente

- **Explorar géneros/desarrolladoras**: se podrán ver diferentes géneros y desarrolladoras. En esta imagen (figura 5.11) solamente se ha hecho con los géneros, pero es exactamente igual con las desarrolladoras.
- **Página de resultados**: cuando el usuario haga alguna búsqueda, ya sea mediante la barra de búsqueda, o presione algún género o desarrolladora, podrá ver la página de resultados (figura 5.12).
- **Detalles de videojuegos**: en esta página (5.13) el usuario podrá ver la descripción del videojuego, la desarrolladora, los géneros, la fecha de lanzamiento, algunas capturas y juegos relacionados. Pulsando en los géneros en las desarrolladoras llevará de vuelta a la página de resultados. Además con un botón podrá guardar el juego en las listas.

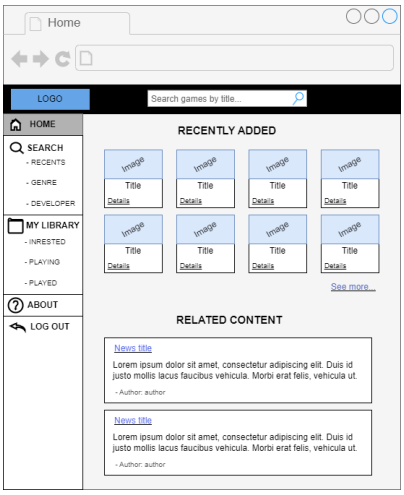

Figura 5.10: Página principal

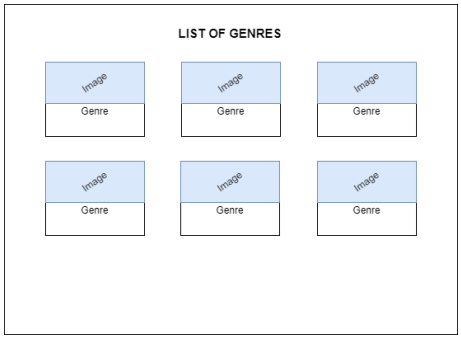

Figura 5.11: Explorar géneros/desarrolladoras

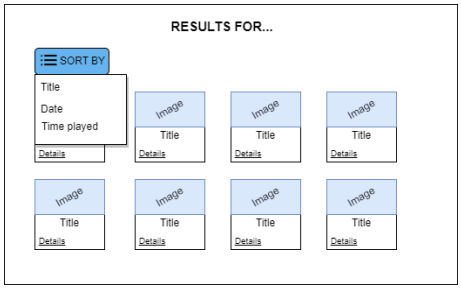

Figura 5.12: Página de resultados

- **Listas**: aquí (figura 5.14) el usuario tendrá los juegos que ha registrado. Podrá eliminarlos o pasarlo a otra lista. En este caso se ha mostrado la lista de *"Playing"*, las otras son similares.

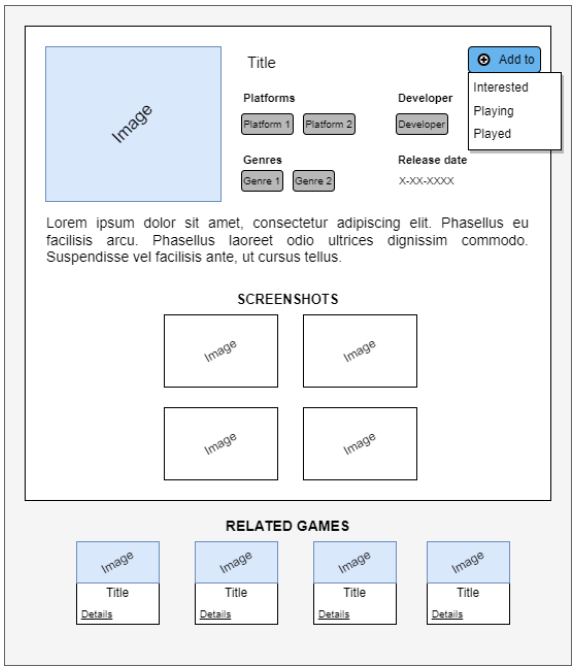

Figura 5.13: Detalles de videojuegos

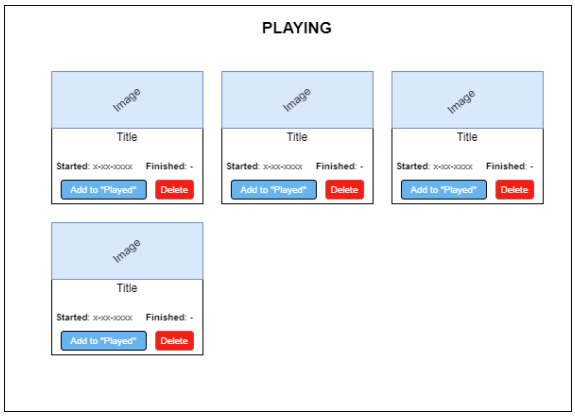

Figura 5.14: Lista del usuario

CAPÍTULO 6
Desarrollo de la solución

En este capítulo procederemos a hablar de la elección de arquitecturas y herramientas para el desarrollo de los elementos que conforman la aplicación.

6.1 Arquitectura

Al hablar de la arquitectura de una página web nos referimos a la estructura y organización general de como está construida la solución. Define como se comunican y relacionan entre sí los diferentes componentes y módulos que componen la aplicación web. En este caso se ha optado por una arquitectura de tres capas. Tenemos la base de datos, el *backend* y el *frontend*.

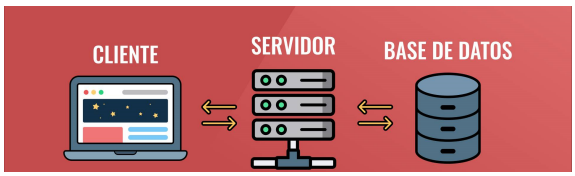

Figura 6.1: Arquitectura de tres capas

- Base de datos: como se ha comentado en puntos anteriores, en esta capa almacenamos y gestionamos los elementos de nuestra aplicación.
- *Backend* (servidor): capa que se encarga de procesar las solicitudes del cliente, interactuar con la base de datos y enviar respuestas al cliente.
- *Frontend* (cliente): es la parte de la aplicación que interactúa directamente con los usuarios. Es la interfaz que los usuarios ven y con la que interactúan en sus navegadores web o dispositivos. Desde esta capa también se realizan las peticiones a las *APIs* externas y al *backend* para obtener y presentar la información que nos interese.

Ahora pasaremos a explicar las tecnologías que se han usado en el desarrollo general de la aplicación y seguidamente pasaremos a entrar en detalle en las herramientas usadas en el *backend* y *frontend*.

6.2 Herramientas generales

- ***Visual Studio Code***: ya sea para el *backend* o para el *frontend* es importante escoger un editor de código con el que uno se sienta cómodo. Una de las principales razo-

nes por las que se ha escogido este editor es por la personalización, ya que permite descargar y usar una muy amplia variedad de extensiones que nos facilitan el desarrollo. Además de tener una interfaz muy limpia y agradable para el usuario posee una terminal integrada con la que ejecutar comandos sin necesidad de cambiar a una terminal externa.

- ***Postman***: para poder probar nuestro *backend* fácilmente se ha hecho uso de esta herramienta. Aparte de tener una interfaz agradable para el usuario, tiene otras características que hacen que esta herramienta sea ideal para hacer pruebas en nuestra *API* o en *APIs* externas y ver las respuestas de una forma clara. Se pueden agrupar las solicitudes en colecciones para poder llevar un mayor control de éstas, además tiene un historial para ver las solicitudes que se hicieron anteriormente. A la hora de realizar solicitudes tiene muchas opciones que hacen mucho más fácil esta tarea, como la definición de variables que facilita la configuración dinámica de la URL, y también admite varios métodos de autenticación, lo que facilita la prueba de *APIs* seguras y protegidas.

- ***Github***: *github* utiliza el sistema de control de versiones de *Git*, que a la hora de realizar un proyecto es imprescindible para llevar un seguimiento de los cambios del código a lo largo del tiempo. Tanto en el *backend* como para el *frontend* se han creado respositorios para poder llevar un mejor control y seguimiento del código.

  - Repositorio *backend*: `https://github.com/PabloPS21/Frontend`
  - Repositorio *frontend*: `https://github.com/PabloPS21/Backend`

Figura 6.2: Logos de *Visual Studio Code*, *Postman* y *Github*

6.3 *Backend*

El *backend*, también conocido como lado del servidor es la parte responsable de gestionar la interacción con la base de datos, autenticar y autorizar usuarios, y proporcionar la funcionalidad necesaria para que el *frontend* pueda interactuar con la aplicación de manera eficiente y segura. A continuación se van a describir todas las herramientas utilizadas en esta capa, que lenguaje de programación se ha usado y las librerías externas que se han empleado, además de proporcionar ejemplos de código para una mejor comprensión de las partes clave de la aplicación.

- ***Node.js***: entorno de código abierto basado en el motor de *JavaScript V8* de Google Chrome [10]. Además de ser ampliamente utilizado en el desarrollo de aplicaciones web y tener una comunidad activa, se ha optado por esta plataforma de desarrollo por su facilidad a la hora de instalar y gestionar módulos y bibliotecas de terceros con *NPM (Node Package Manager)*. A la hora de escoger el lenguaje para el *backend* también se tuvo en cuenta *php*, pero debido al desconocimiento del mismo y a la tendencia en alza del uso de *Node.js* en la actualidad, al final se optó por éste.

- ***Typescript***: en el punto anterior de ha dicho que *Node.js* utiliza *JavaScript* como lenguaje, pero en este proyecto de decidió usar *TypeScript*. Este lenguaje es un *superset*[1], es por ello que ofrece algunas ventajas que a la hora de programar facilitan mucho las cosas. Algunas de estas ventajas son las siguientes [11]:

  - Tipado estático: *TypeScript* proporciona tipado estático, lo que permite declarar el tipo de las variables, parámetros de funciones y valores de retorno. Es cierto que esto hace que sea más laborioso que con *JavaScript*, donde no hace falta declarar tipos, pero la declaración de tipos permite detectar errores de compilación, lo que mejora la calidad del código y reduce errores en tiempo de ejecución.
  - Mejor manteniblidad: el tipado estático ofrece una orientación a objetos más solida, haciendo que el código de *TypeScript* sea más claro y fácil de mantener.
  - Ayuda en *IDEs*[2]: al ser un lenguaje más acotado que *JavaScript*, las extensiones de *Visual Studio Code* que admiten *TypeScript* ofrecen un autocompletado más inteligente y una asistencia más detallada en comparación con *JavaScript* puro.
  - Interfaces y tipos personalizados: *TypeScript* permite definir interfaces y tipos de datos personalizados, lo que facilita la cohesión entre diferentes partes del código y mejora la legibilidad y la robustez del código. Al usar *APIs* esto permite crear tipos específicos automáticamente con la respuesta que obtenemos en *JSON*[3] utilizando una web llamada *JSON to Typescript*.

  Dicho esto, aunque se escriba el código en *TypeScript* no se va a ejecutar este, pues lo que hace *Node.js* es traducir el código de *TypeScript* a *JavaScript* y ejecutarlo. En el proyecto se tendrán las clases con extensión .ts para los archivos de *TypeScript* y al compilar se crearán los mismos archivos con extensión .js traducidos.

- ***Sequelize***: biblioteca de *Node.js* que se utiliza como *ORM*[4] para trabajar con bases de datos relacionales que proporciona una forma sencilla y poderosa de interactuar con bases de datos *SQL*. Esta librería ofrece una abstracción de la base de datos, permitiendo manipular los datos utilizando objetos y métodos en lugar del lenguaje *SQL* directamente [12]. En un punto anterior se ha comentado que los atributos de las tablas se creaban con una librería, pues *sequelize* permite definir modelos que representan las tablas de la base de datos. Cada modelo define la estructura y las relaciones de las tablas, lo que facilita el acceso y manipulación de los datos. Para poder utilizar *sequelize* lo primero que tendremos que hacer es realizar la conexión a nuestra base de datos utilizando las credenciales que establecimos en la creación de la base de datos. Una vez conectados tendremos que crear una instancia de *sequelize* y exportarla para utilizarla posteriormente en otras clases.

```
1 import {Sequelize} from 'sequelize'
2
3 //Declaramos variables con las credenciales usadas al crear la base de
      datos
4 const baseDatos: string = 'prueba';
5 const username: string = 'root';
6 const password: string = 'root';
7
8 //Creamos nueva instancia de sequelize
```

[1] *Superset* (programación): lenguaje que incluye todas las características y funcionalidades de otro lenguaje, además de añadir características adicionales.

[2] *IDE* (entorno de desarrollo integrado): aplicación que ayuda a los programadores a desarrollar *software* de manera eficiente.

[3] *JSON*: formato de intercambio de datos.

[4] *ORM*: herramienta que se utiliza para transformar datos a un formato concreto.

```
const db = new Sequelize(baseDatos, username, password, {
    host: 'localhost',
    dialect: 'mysql',
    logging: false //true permite ver en la consola los comandos SQL que
        se ejecutan cuando se realiza alguna accion en la base de datos
};

export default db;
```

Una vez realizado este paso podemos pasar a crear los modelos. Para esto tendremos que utilizar el método *define* de *sequelize* para crear nuestras tablas y definir sus columnas y propiedades.

```
import {DataTypes} from 'Sequelize'
import db from '../database/connection'

const Usuario = db.define('Usuario', {

    //Definimos los atributos del Usuario, con sus tipos y restricciones
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    username: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true,
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false
    }

});
```

En este caso observamos como se ha creado la tabla de 'Usuarios' (aunque a la hora de definir ponemos el nombre en singular, *sequelize* se encarga de crear la tabla con el nombre en plural), definiendo sus columnas con sus respectivos tipos y restricciones. Cuando se hayan creado todos los modelos tendremos que sincronizarlos para que la primera vez que ejecutemos la aplicación se creen o se modifiquen las tablas en la base de datos.

```
//Conexion a la base de datos
async databaseConnection(){
    try {
        //Conexion con los modelos de la base de datos
        await Usuario.sync();
        await Game.sync({alter: true});
        console.log("Conectado a la base de datos");

    } catch (error) {
        throw error;
    }
}
```

El método *sync* de los modelos permite pasar como parámetro *alter* que permitirá modificar la tabla según su valor:

- Sin valor: si se realiza algún cambio en el modelo pero no especificamos ningún argumento, la tabla del modelo no se modificará en la base de datos.
- *true*: en la base de datos se modificarán solamente las columnas que se hayan modificado en la clase del modelo. Los elementos que estén guardados en la base de datos no se modificarán.
- *force*: la tabla se borrará completamente y se creará de nuevo vacía con las modificaciones que hayamos realizado en el modelo.

- ***Express***: esta librería sirve como un *framework*[5] web para simplificar el desarrollo de aplicaciones web y *APIs* de manera rápida y sencilla [13]. Proporciona una capa de abstracción sobre el servidor HTTP nativo de *Node.js*, lo que facilita la creación de rutas o el manejo de solicitudes y respuestas. Podemos definir rutas y asociar funciones de controlador que se ejecutan cuando llega una solicitud HTTP espcífica. Combinando esta librería con sequalize podemos crear el *CRUD*[6] de nuestra aplicación muy fácilmente. Mediante un flujo muy sencillo podemos ver los pasos que se seguirían para llegar a interactuar con los objetos de la base de datos.

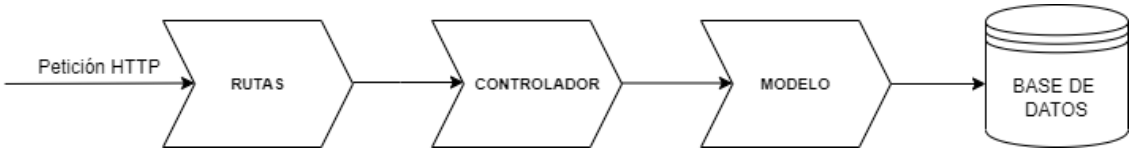

Figura 6.3: Orden de operaciones cuando llega una petición HTTP

Cuando llegue una petición HTTP a nuestra *API*, se ejecutará el controlador referente a la ruta de la petición. En el controlador, utilizaremos los modelos y los métodos ofrecidos por *sequalize* para hacer las consultas o modificaciones pertinentes en la base de datos.

En los siguientes ejemplos de código veremos como configurar las rutas para nuestra apliación. Lo primero que tendremos que hacer es definir la base de la ruta.

```
//Definimos las bases de las rutas
private apiEndpoints = {
    usuarios: '/api/usuarios', //ruta base para acceder a los usuarios
    games: '/api/games' //ruta base para los videojuegos
}
```

Teniendo la base de las rutas, tendremos que especificar a partir de ésta los diferentes *endpoints*[7] y los controladores asociados a estas rutas.

```
const gameRouter = Router();

//Definimos las rutas y las asociamos al controlador
gameRouter.get("/", getGames); //Obtencion de todos los juegos
gameRouter.get("/:id", getGame); //Obtencion de un juego por su id
gameRouter.post("/", registrarJuego); //Crear un nuevo juego
gameRouter.delete("/:id", eliminarJuego); //Eliminar un juego por su id
gameRouter.put("/:id", modificarJuego); //Modificar un juego por su id

//Exportamos el router para usarlo en otras clases.
export default gameRouter;
```

[5] *Framework*: marco de trabajo.
[6] *CRUD*: concepto que utiliza para referirse a las cuatro operaciones básicas que se pueden realizar en la mayoria de base de datos, las cuales son crear, leer, modificar y borrar.
[7] *Endpoint*: punto de acceso a un determinado recurso.

Claro está, tendremos que tener definidos los métodos de los controladores. Las bibliotecas *sequalize* y *express* ofrecen muchas facilidades a la hora de interactuar con la base de datos y con las peticiones HTTP.

```
//Obtener un juego por ID
export const getGame = async(req: Request, res: Response) => {

    //Obtenemos la id especificada en la peticion
    const { id } = req.params;

    //Obtenemos desde la base de datos el juego con esa id
    const game = await Game.findByPk(id);

    //Si el juego existe lo mandamos en la respuesta, si no respondemos
        con un mensaje de error
    if(game) {
        res.json(game);
    } else {
        res.status(404).json({ message: "Juego no encontrado" });
    }

}
```

Gracias a las librerías antes mencionadas podemos acceder a la petición HTTP y enviar respuestas fácilmente, además vemos como para obtener el juego utilizamos el método *findByPk* del modelo *Game* y no es necesario realizar la consulta con el lenguaje *SQL*. Definidas las rutas y los controladores asociados, el último paso es indicarle a la aplicación que debe usar esas rutas.

```
//Metodo encargado de utilizar las rutas establecias
routes(){
    this.app.use(this.apiEndpoints.usuarios, routerUsuarios);
    this.app.use(this.apiEndpoints.games, routerGames);
}
```

- ***Cors***: esta librería (*Cross-Origin Resource Sharing*) es una herramienta que se utiliza para gestionar y configurar las políticas de seguridad de intercambio de recursos entre diferentes dominios de aplicaciones web y *APIs* [14]. *Cors* es una medida de seguridad cuando se realizan solicitudes HTTP entre diferentes dominios. Esta librería se utiliza como *middleware*[8] en aplicaciones que usan *express*, permitiendo habilitar *Cors* de manera global para todas las rutas o configurar reglas específicas para dominios permitidos. En nuestro caso al tratarse de una aplicación local el único origen permitido es *http://localhost:4200*, además habilitaremos la opción de credenciales, ya que algunas rutas están protegidas y necesitan que se envíen las credenciales del usuario en la solicitud.

```
//Habilitar Cors en la aplicacion
this.app.use(cors({
    origin: 'http://localhost:4200',
    credentials: true
}));
```

- ***Dotenv***: esta herramienta se utiliza para cargar variables de entorno desde archivos *.env* en el entorno de desarrollo. Las variables de entorno son globales para la aplicación y pueden afectar al comportamiento y a la configuración de la misma. Esta librería no es totalmente imprescindible en nuestra aplicación, pero facilita la gestión y modificación de algunas variables que son usadas globalmente en la aplicación, como pueden ser los número de puertos o claves de *APIs*.

[8] *Middleware*: componente que actua como intermediario entre diferentes sistemas.

- ***Jsonwebtoken***: esta librería permite trabajar con *JSON Web Tokens* (JWT), facilitando la creación, validación y manipulación de estos objetos.

  El *JWT* es un estándard abierto para representar información de manera segura entre dos partes como un objeto *JSON*[15]. En la práctica se trata de una cadena de texto que tiene tres partes codificadas en Base64, cada una de ellas separada por un punto.

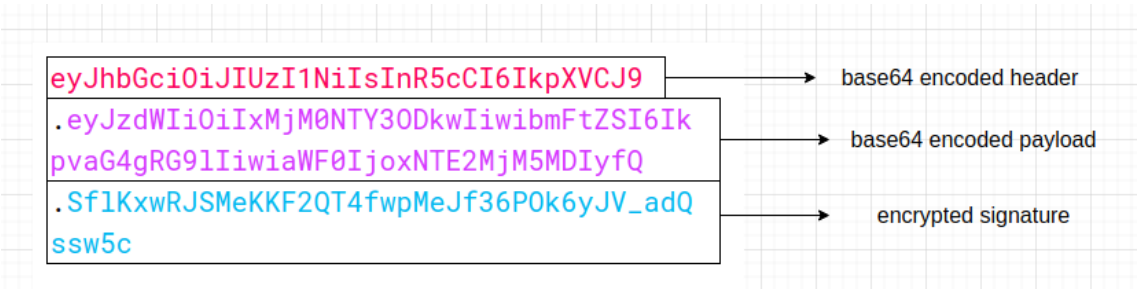

Figura 6.4: Ejemplo de *JWT*

- *Header*: encabezado donde se indica el algoritmo y el tipo de *token*.
- *Payload*: donde aparecen los datos del usuario, además de toda la información que creamos conveniente añadir en esta parte.
- *Signature*: firma que permite verificar si el *token* es válido.

En nuestro caso el *JWT* se ha usado para la identificación del usuario en la aplicación [16], además de tener una ruta protegida con este *token*. Cuando el usuario inicia sesión se genera un *JWT* firmado con una clave que se ha establecido en el *backend*. Este *token* identifica al usuario y se guarda en el almacenamiento del navegador. Cuando el usuario quiera acceder a sus juegos, en la petición se incluirá el *JWT* y en el *backend* se harán las comprobaciones pertinentes para confirmar que este usuario tiene acceso al recurso, y si lo tiene se le mandará la respuesta.

Figura 6.5: Funcionamiento del *JWT*

Ahora veremos lo explicado anteriormente en el código. Cuando el usuario inicia sesión, después de comprobar que las credenciales que ha introducido son las correctas se genera el *JWT* en el que incluimos en el *payload* el nombre de usuario y el id para usos posteriores. Además firmamos el *token* con una clave que hemos

establecido una variable de entorno, o en caso de que no encuentre esta variable firmamos con otra cadena, por ejemplo *'supersecretkey'*. Para concluir incluimos el *token* en la respuesta que le mandamos al cliente.

```
//Una vez el usuario se ha validado, generamos el jwt con la informacion del usuario
const token = jwt.sign({
    username: username,
    id: id,
}, process.env.JWT_KEY || 'supersecretkey',);

res.json({usuario, token}); //En la respuesta enviamos el usuario junto con el JWT
```

En el *frontend* guardaremos el *token* en el almacenamiento local del navegador. Así si el usuario cierra la aplicación, cuando vuelva a entrar se comprobará si tiene el *token* para que no tenga que volver a iniciar sesión. A la hora de cerrar sesión lo que haremos es destruir este token. Cuando el usuario quiera acceder a sus juegos, se incluirá en la cabecera de la petición al *backend*.

```
//Metodo para obtener los juegos del usuario
obtenerJuegosUsuario(): Observable<RegisteredGame[]> {

//Obtenemos la id del usuario desde el token
const idUsuario: this.tokenService.getIdUsuarioFromToken();

if(idUsuario) {
    //Incluimos en la respuesta en la cabecera de seguridad el token
    const headers = new HttpHeaders().set('Authorization', 'Bearer ' + this.tokenService.getToken());

    //Realizamos la peticion
    return this.htpp.get<RegisteredGame[]>('http://localhost:8000/api/usuarios/' + idUsuario + '/games', {headers});
}

    throw new Error('No se puedieron obtener los juegos')

}
```

En el *backend*, como la ruta está protegida primero ejecutaremos la validación del *token*, y si esta validación se completa dejamos pasar para que se pueda ejecutar el método asociado al controlador.

```
userRouter.get("/:id/games", tokenValidacion, getUserGames); //Para poder ejecutar el controlador primero tendremos que validar el token
```

```
//Metodo para validar el token
const tokenValidacion = (req: Request, res: Response, next: NextFunction) => {

    //Obtenemos el token de la cabecera de seguridad de la peticion
    const headerToken = req.headers['authorization']

    //Comprobamos primero si tiene token
    if(headerToken != undefined && headerToken.startsWith('Bearer ')) {

        try {
            //Nos quedamos con el token sin la cadena 'Bearer'
            const token = headerToken.slice(7);

            //Verificamos el token con la clave secreta establecida en el fichero env
```

```
            const verifiedToken: any = jwt.verify(token, process.env.
                JWT_KEY || 'supersecretkey');


            // Comprobamos si el ID en la URL coincide con el ID del
                usuario autenticado
            if (req.params.id != verifiedToken.id) {
                return res.status(403).json({ message: 'Acceso denegado'
                    });
             }

            next();

        } catch (error) {

            res.status(401).json({ message: 'No autorizado' });

        }

    //Si no tiene token no dejamos pasar
    } else {
        res.status(401).json({ message: 'No tienes autorizacion' });
    }
}
```

Es importante no incluir información que comprometa la privacidad de los usuarios, como contraseñas o números de tarjetas ya que alguien podría interceptar el mensaje y decodificar el *JWT* para ver la información.

- ***Bcrypt***: en un punto anterior se ha comentado que la contraseña de los usuarios se almacena encriptada en la base de datos, esto es gracias a la librería *bcrypt*, que se utiliza para hacer el *hashing* de contraseñas. Lleva incorporado un valor llamado sal[9], con este valor se añade un grado de complejidad que evita que el hash asociado a una contraseña sea único. *Bcrypt* permite elegir el valor de *saltRounds*, que nos da el control sobre el coste de procesado de datos. Cuanto más alto es este número mayor será la seguridad y mayor será el tiempo que requiere el pc para calcular el hash.

  Veamos ahora con un ejemplo como se generan las contraseñas. El usuario al registrarse introduce la contraseña *123456a*. La librería, con el valor de *saltRounds* que hemos establecido genera una cadena aleatoria, por ejemplo *H8p6hqv3q87BN9anj15*, además también se añadirán delante unos elementos de control para saber que algoritmo se está utilizando y el número de *saltRounds*, por ejemplo *$2a$10$*. Usando el *salt* se encriptará la contraseña y se generará otra cadena a la que añadirán los elementos de control, quedando algo como *$2a$10$H8p6hqv3q87BN9anj15/23AwPd36mJNq1Apw091ZAad*[17].

  Ahora veremos en el código la implementación y uso de la librería. Se hará uso en el controlador del usuario, a la hora de crear un nuevo usuario, es decir registrarnos, además de en el inicio de sesión para la comprobación de las credenciales. A la hora de registrarnos simplemente tendremos que aplicar el método *hash* de la biblioteca *bcrypt* a la contraseña que ha introducido el usuario indicando el número de *saltRounds*, en nuestro caso se ha aplicado 10 ya que es la opción recomendada.

```
//Metodo para registrar usuario
export const registrarUsuario =  async (req: Request, res: Response) => {

    //Obtenemos los datos del usuario de la peticion
```

[9]Sal: conjunto de valores aleatorios que se añaden para hacer el hash más seguro.

```
    const {username, email, password} = req.body;

    //Encriptar password
    const hashContrasena = await bcrypt.hash(password, 10)

     try {
        //Almacenamos el usuario en la base de datos
        await Usuario.create({
            username: username,
            email: email,
            password: hashContrasena
        })

        res.json({
            msg: 'Usuario registado con exito'
        });

    } catch(error){
        res.status(400).json({
            msg: 'Ocurrio un error',
            error
         });
    }

}
```

Aplicando el hash vemos como en la base de datos se almacena la contraseña de una forma segura.

id	username	email	contraseña	createdAt	updatedAt
18	PabloPS21	xxpablo...	$2b$10$eh3mCxopG4oSzpHVmS3/guyzgFWVPZKEaX43n08...	2023-06-0...	2023-06-0...
20	Pablo	pablose...	$2b$10$FwkZqOFC9Hg2LT/PZA177eP.1putZ07DmCohESzw...	2023-07-1...	2023-07-1...

Figura 6.6: Contraseña almacenada encriptada

Y a la hora de hacer el login y comprobar la contraseña que introduce el usuario con la que está almacenada en la base de datos se utilizará el método *compare* de la biblioteca que devolverá un *boolean* indicando si las contraseñas coinciden, si es así dejaremos pasar al usuario.

```
//Metodo para loggear usuario
export const loginUsuario =  async (req: Request, res: Response) => {

    //Obtenemos los datos que ha introducido el usuario de la peticion
    const { username, password } = req.body;

    //Validar credenciales
    const usuario: any = await Usuario.findOne({where:{username: username
        }});

    //Si no existe un usuario con ese nombre de usuario avisamos al
        usuario
    if(!usuario) {
        return res.status(400).json({ msg: "User not found" });
    }

    //Comprobar password hasheada con la que manda el usuario
    const passwordValida = await bcrypt.compare(password, usuario.
        password)

```

```
18      if(!passwordValida) {
19          return res.status(400).json({ msg: "Incorrect password" });
20      }
```

6.4 *Frontend*

El termino *frontend* se refiere a la parte visible de la aplicación y con la que interactúan los usuarios en sus dispositivos. Es la capa que se encarga de la presentación y la interacción directa con el usuario.

Las tecnologías que se han usado en esta capa son las siguientes:

- ***Angular***: para el entorno de desarrollo de esta capa se optó por angular, uno de los *frameworks* de *JavaScript* más populares a la hora de crear sitios web dinámicos [18]. *Angular* engloba las tecnologías de *HTML*, *CSS* y *TypeScript*, aunque en el navegador se ejecute en *JavaScript*. Además de usar *HTML* y *CSS* de la manera convencional *Angular* ofrece directivas para poder hacer el envío de datos de los controladores (clases de *TypeScript*) a las vistas, donde representaremos la información. Al tratarse de una aplicación con diversas páginas *Angular* es ideal para esto, pues permite agrupar los diferentes elementos de la aplicación en módulos y hacer uso de ellos con etiquetas *html* específicas, por ejemplo, yo quiero crear una carta en la que se represente la información de los videojuegos, pues creo un nuevo módulo para este componente y cuando lo quiera usar simplemente con una etiqueta *html* puedo hacer uso de el, además de pasarle la información que me interese. También permite el uso de rutas, para que cada página tenga su ruta específica y puedas acceder a ella de manera fácil, además de poder pasar información específica en la url. A la hora de trabajar con *APIs Angular* ofrece una serie de módulos propios como *HttpClient* que facilitan la interacción con servicios externos. Actualmente, Angular es uno de los *frameworks* más utilizados en el campo de desarrollo web, por lo que tiene una comunidad activa y una documentación muy extensa, es por ello que esta tecnología es un imprescindible en este campo.

  Entre las partes más destacables del código en el *frontend* podemos encontrar las peticiones que se hacen a nuestra *API* y a las *APIs* externas. A continuación vemos como realizar una llamada a la *API* local, más concretamente al *endpoint* de *login* del controlador del usuario.

```
export class LogInService {

  //URL de la api local
  private apiUrl = 'http://localhost:8000/api';

  //Utilizamos el objeto httpClient
  constructor(private http: HttpClient) { }

  //Con el username y la password que proporciona el usuario hacemos la
      llamada al backend y devolvemos la respuesta
  logUsuario(username: string, password: string): Observable<any> {
    const url = `${this.apiUrl}/usuarios/login`;
    const body = { username, password };
    return this.http.post<any>(url, body);
  }
}
```

  Seguidamente, podemos ver como se hacen las llamadas a la *API* de *RAWG Games*, en este caso veremos como obtener información de un determinado videojuego.

```
export class ObtainGameDetailsService {

  //Obtenemos la clave de la api definida en el archivo .env
  private apiKey = environment.gameApi_key;

  //url de la api de RAWG Games
  private readonly baseUrl = 'https://api.rawg.io/api/games';

  constructor(private http: HttpClient) { }

  //Obtener detalles del videojuego por el id
  getGameDetailsBySlug(id:number): Observable<GameById> {

    const url = `${this.baseUrl}/${id}?key=${this.apiKey}`;
    return this.http.get<GameById>(url);
  }

  //Obtener capturas de pantalla de un videojuego por su id
  getGameScreenshots(id:number): Observable<any> {
    const url = `${this.baseUrl}/${id}/screenshots?key=${this.apiKey}`;
    return this.http.get<any>(url);
  }

  //Obtener juegos relacionados por el id de un juego concreto
  getRelatedGames(id:number): Observable<any> {
    const url = `${this.baseUrl}/${id}/game-series?key=${this.apiKey}`;
    return this.http.get<any>(url);
  }
}
```

Estas peticiones se han realizado en clases a las que llamamos servicios, encargadas de hacer las llamadas y obtener las respuestas correspondientes. Para hacer uso de esta información que se ha obtenido simplemente tendremos que importar este servicio en la clase de *TypeScript* correspondiente y hacer uso de esta como mejor nos convenga, normalmente guardando la respuesta que obtengamos en una variable para así poder acceder a ella y mostrarla fácilmente en el *HTML*.

- ***Angular Material***: la parte visual es una de los elementos más importantes a la hora de desarrollar el *frontend*, es por ello que se hizo uso de *Angular Material* [19] para construir los elementos que iban a formar parte de la interfaz de usuario. Esta herramienta ofrece una amplia biblioteca de componentes y estilos diseñada por el equipo de *Angular* para facilitar la creación de interfaces de usuario coherentes y atractivas en aplicaciones diseñadas con este *framework*. Su facilidad de uso hacen que esta librería de componentes sea una de las más usadas actualmente, pues para usar un componente nuevo simplemente hay que importarlo y añadirlo con etiquetas *html* en cualquier parte de la aplicación. Uno de los principales inconvenientes de esta herramienta es la poca personalización que permiten los componente, llegando a tener que cambiar archivos de configuración de *Angular Material* para poder cambiar márgenes o colores de algunos componentes. *Bootstrap* también se tuvo en cuenta para construir la interfaz gráfica, pues esta ofrece una mayor personalización a la hora de crear los elementos que constituyen la interfaz. Aún así, la facilidad de uso de los componentes de *Angular Material* se impuso y se optó por usar esta herramienta.

CAPÍTULO 7
Producto desarrollado

Una vez con la aplicación desarrollada se van a mostrar capturas de pantalla de la interfaz y se darán las explicaciones necesarias para ver como el usuario puede realizar los casos de uso descritos anteriormente.

- **Registro**: el usuario se registrará en la aplicación desde esta pantalla. Si alguna credencial que introduce no es correcta, ya sea porque ya existe ese nombre de usuario o porque la contraseña no es lo suficientemente segura el sistema le avisará.

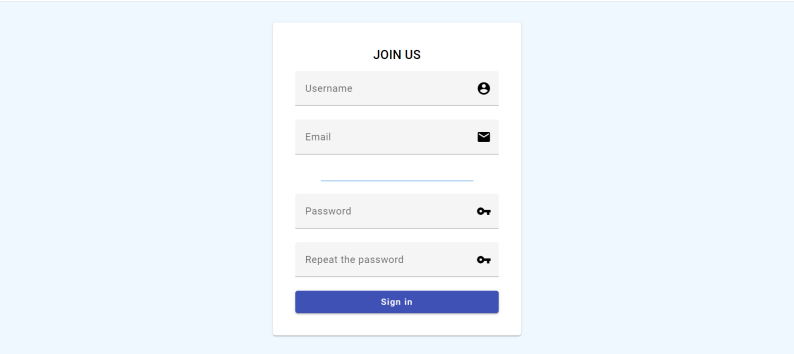

Figura 7.1: Pantalla de registro

- **Iniciar sesión**: en esta pantalla el usuario introducirá sus credenciales para poder entrar a la aplicación. Si no introduce las credenciales correctas el sistema le avisará.

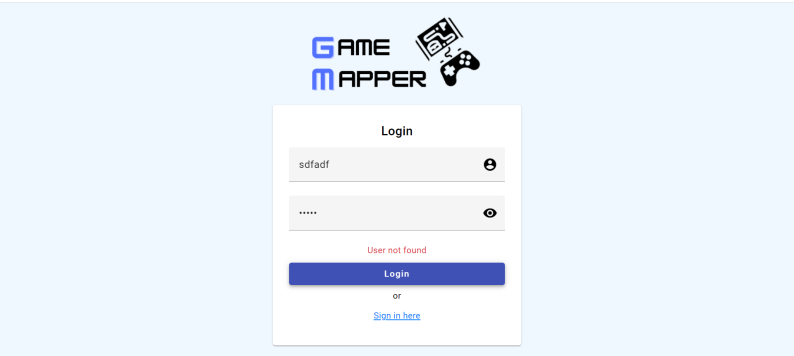

Figura 7.2: Pantalla de inicio de sesión

- **Cerrar sesión**: una vez dentro de la aplicación en el menú lateral tenemos las diferentes acciones que puede hacer el usuario. Para cerrar sesión simplemente pul-

samos la opción de *Log Out* del menú lateral, y el sistema mostrará un *pop-up* para confirmar la selección. Si cierras sesión el sistema te redirigirá a la página principal.

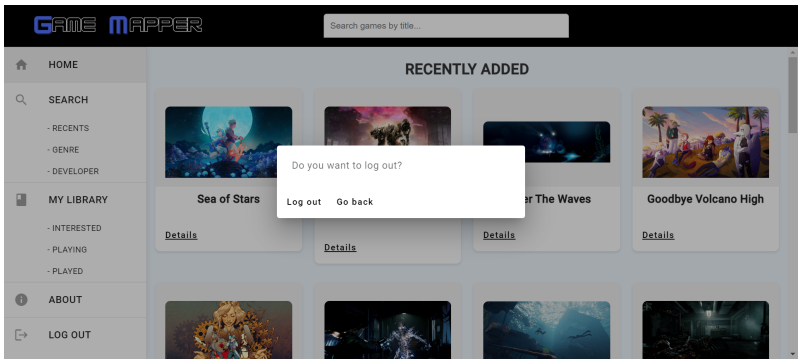

Figura 7.3: Cerrar sesión

- **Ver juegos estrenados recientemente**: pulsando la opción *Recents* del menú lateral iremos a la pantalla de juegos estrenados recientemente, donde podremos ver que videojuegos se han estrenado en la última semana, en el último més o en los últimos tres meses, puediendo cambiar estas fechas desde el *radio button*. Además podremos ordenar los resultados pulsando el botón de *Sort by*. Los juegos se dividirán en 5 filas con 4 juegos cada fila, teniendo un selector de páginas para poder navegar por los resultados. Además como se ha comentado en puntos anteriores todas las pantallas son ajustables a diferentes tamaños de dispositivo donde se redistribuirán los elementos y se ajustará su tamaño para seguir ofreciendo una interfaz agradable para el usuario.

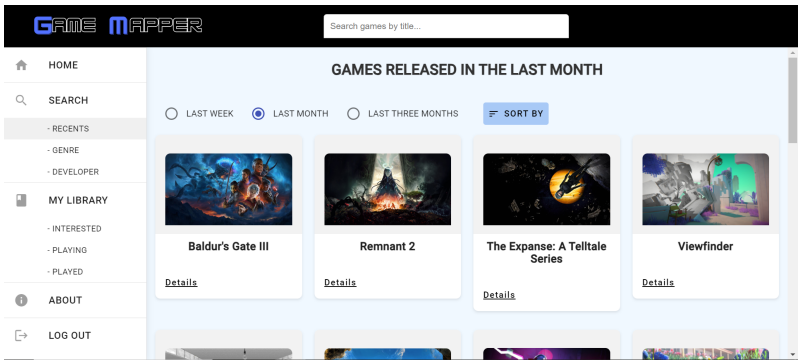

Figura 7.4: Videojuegos estrenados recientemente

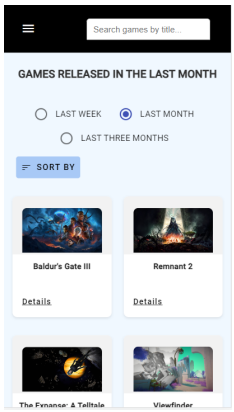

Figura 7.5: Videojuegos estrenados recientemente en resolución móvil

- **Buscar por género y desarrolladora**: en la pantalla de buscar por género encontraremos una lista con todos los géneros, y si pulsamos en uno de ellos nos llevará a una página de resultados con los juegos más populares de ese género. En cuanto a la desarrolladora, tendremos una lista con las desarrolladoras más populares indicando el nombre y el número de juegos que han producido, y pulsando en una de ellas nos redirigirá a la página de resultados con todos sus juegos. En el apartado de detalles se muestra los géneros y la desarrolladora de un cierto videojuego, pulsando en esos botones también podremos ver los resultados.

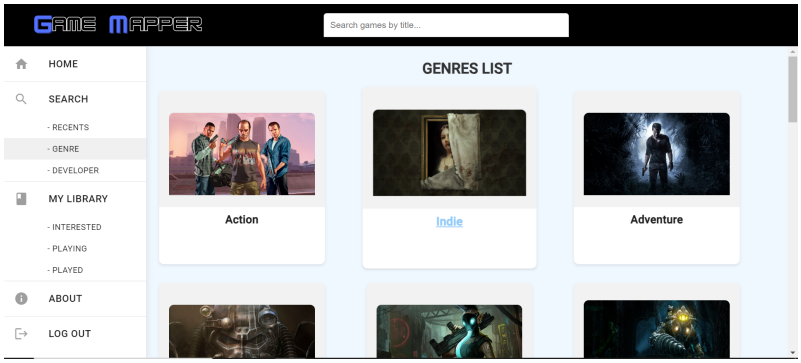

Figura 7.6: Lista de géneros

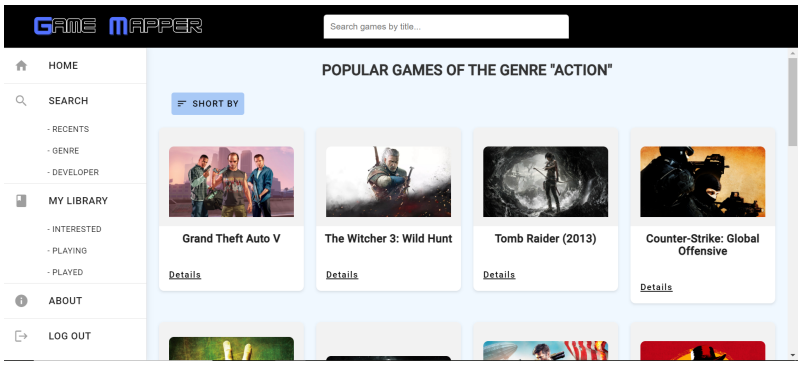

Figura 7.7: Resultados de géneros

- **Buscar por título**: en el *header* tenemos una barra de búsqueda en la que podemos buscar videojuegos por su título. Si mientras escribimos hay coincidencias el sistema nos las mostrará en forma de lista debajo de la propia barra, pudiendo pulsar en algún resultado para ir a los detalles de ese juego. Si queremos ver las coincidencias con la cadena que hemos escrito pulsamos la tecla *enter* y podremos ver todos los resultados.

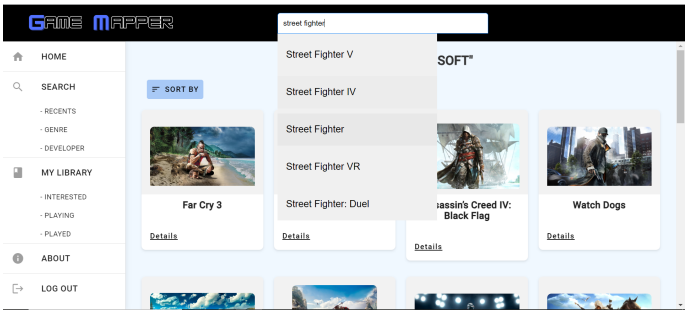

Figura 7.8: Sugerencias de la barra de búsqueda

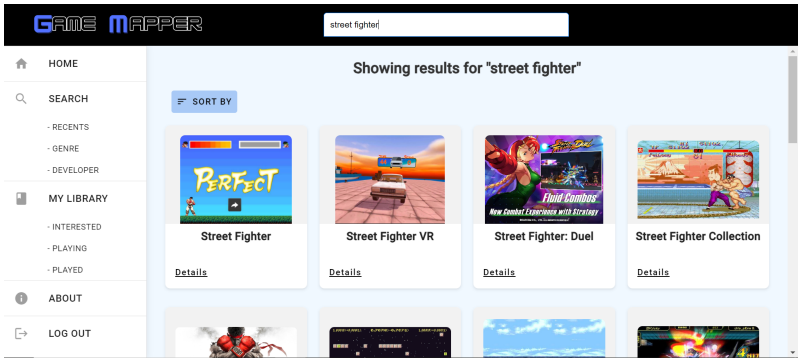

Figura 7.9: Pantalla de resultados de la cadena *"Street Fighter"*

- **Ordenar videojuegos mostrados**: una vez busquemos algún videojuego mediante alguno de los métodos descritos, en la página de resultados tendremos un botón de ordenar, en el que al pulsar veremos un desplegable con las diferentes opciones de ordenación. Pulsando en una de estas los videojuegos se ordenarán.

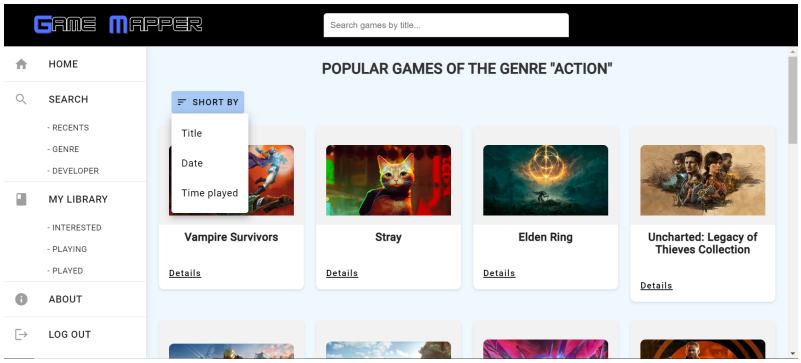

Figura 7.10: Ordenar resultados

- **Ver detalles de videojuego**: pulsando en la opción de *Details* de las cartas de videojuegos o desde las sugerencias de la barra de búsqueda podremos ver información extra sobre el videojuego, además de poder añdirlo a nuestras listas. Se mostrará una descripción, las desarrolladoras, los géneros , las plataformas en las que se puede juegar, la fecha de publicación, capturas de pantalla y juegos relacionados.

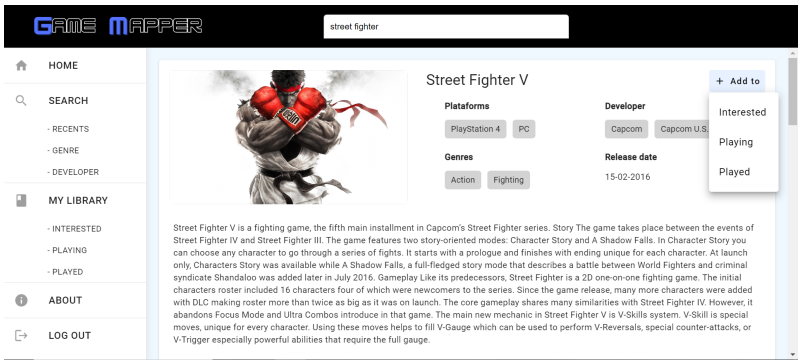

Figura 7.11: Pantalla de detalles

Figura 7.12: Pantalla de detalles en resolución móvil

- **Añadir juegos a listas**: en la pantalla de detalles de videojuego en la esquina superior derecha hay un desplegable en el que podremos ver las diferentes listas a las que añadir los juegos (figura 7.11). Si el juego se añade con éxito el sistema mostrará un *pop-up* indicándolo, si el juego ya estaba en alguna lista el sistema también avisará.

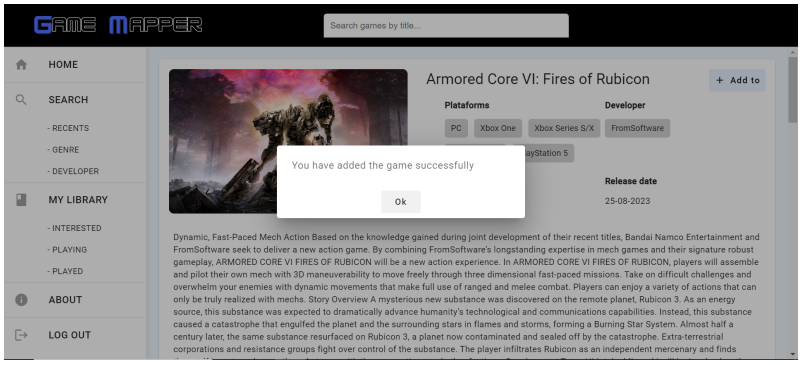

Figura 7.13: Juego añadido correctamente

- **Ver listas**: en el menú lateral, en la sección de *My Library* pulsaremos alguna de las opciones para ver las listas con sus respectivos juegos. Si en la lista no hay ninún juego el sistema lo indicará con un mensaje.

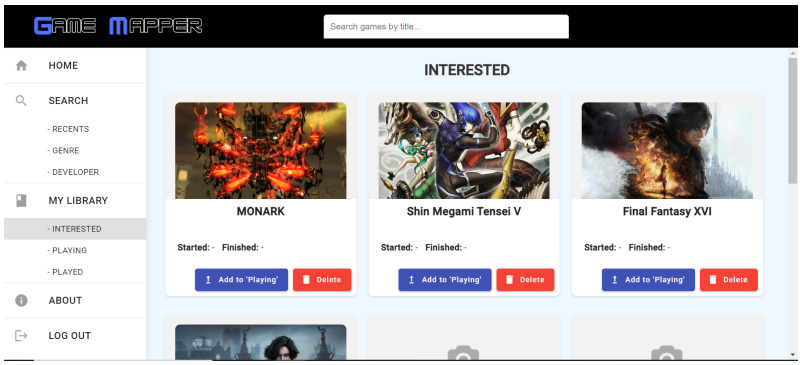

Figura 7.14: Lista *Interested*

- **Eliminar juegos de la lista**: en cada una de las listas, si estas tienen algún juego, debajo de éste tendremos un botón de eliminar para borrarlo de la lista. Pulsando el botón el sistema mostrará un *pop-up* preguntando si estamos seguros, si pulsamos

que si el juego se eliminará y se recargará la página para ver que el juego ya no aparece.

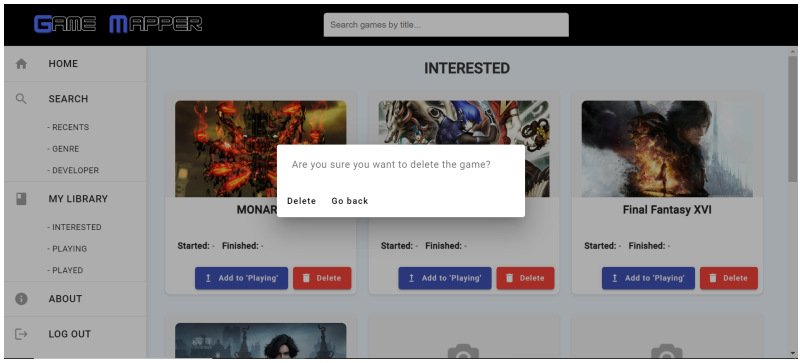

Figura 7.15: Eliminar juego

- **Pasar juegos a otra lista**: igual que con el caso anterior, en las listas, debajo de cada juego tendremos la opción de pasar a la siguiente lista, es decir, de *Pendiente* a *Jugando* y de *Jugando* a *Finalizados*. Pulsando en alguna de estas opciones la página se recargará y veremos que el juego ya no se encuentra en la lista actual.
- **Información sobre el sitio**: pulsando en el botón de *About* del menú lateral veremos la página de información. Esta pantalla ha sido creada para aportar información extra sobre la aplicación además de dar el crédito correspondiente a las *APIs* externas.

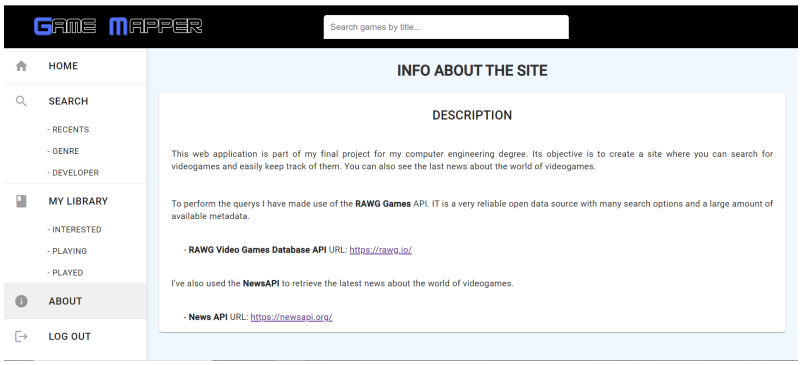

Figura 7.16: Pantalla de información

CAPÍTULO 8
Validación

La validación de cualquier tipo de producto software es una etapa crucial en el desarrollo, ya que garantiza que el producto final funcione sin problemas, sea accesible para todos los usuarios y cumpla con los estándares de calidad. En una página web es importante asegurarse que los clientes o usuarios encuentren la interfaz agradable y puedan realizar todas las funciones sin problemas. Es por ello que una forma de validación muy común de páginas web es hacer que un grupo reducido de usuarios, o los mismos clientes prueben la aplicación y proporciones el *feedback* necesario para arreglar errores que hayan podido surgir. La validación de la interfaz web es necesaria que se haga en etapas tempranas del desarrollo, normalmente en los prototipos, para que el grupo de usuarios pueda ver la disposición de los elementos y ver como se realizan las diferentes funciones. Es importante que los usuarios en los prototipos puedan encontrar y raealizar sin problemas todas las funciones de la web. En nuestra aplicación, en etapas tempranas, mientras se empezaba a desarrollar la interfaz web se fue preguntando y pidiendo opiniones a algunos amigos y conocidos que estuviesen familiarizados con los videojuegos y con este tipo de webs para sugerir mejoras o corregir fallos en la interfaz.

Actualmente existen diversar herramientas online que permiten validar tu página web según distintos estándares o criterios [20], como pueden ser el validador *WC3*, que comprueba la sintaxis del *HTML* tanto como el *CSS*. Estas páginas pueden ser útiles a la hora de validar, pero no por ello tenemos que excluir otro tipo de validaciones mas exhaustivas, entre las que encontramos la validación heurística [21]. Este tipo de validación se basa en análisis de expertos para detectar problemas de usablilidad en la web. Solucionando estos problemas lograremos que la experiencia de usuario sea agradable y satisfactoria. Una de las referencias más usadas a la hora de detectar problemas de usabilidad en la interfaz son los 10 principios de usabilidad de Jakob Nielsen [22]. Estos principios son las pautas generales y básicas que debe cumplir toda interfaz de usuario, es por ello que para evaluar nuestra interfaz web vamos a enumerar cada uno de estos principios y ver si se cumplen. Los 10 principios de usabilidad de Jakob Nielsen son los siguientes:

- **Visibilidad del estado del sistema**: el usuario tiene que saber en todo momento que está ocurriendo o donde se encuentra en la página web. En nuestro caso indicamos en todo momento en que pantalla se encuentra al usuario resltando de otro color la sección del menú donde está.
- **Relación entre el sistema y el mundo real**: es importante que el usuario aunque no haya entrado nunca a la página sea capaz de reconocer elementos familiares que le faciliten la interacción con la interfaz. Estos elementos los encontramos en formas de figuras o iconos en el menú para que con un simple vistazo el usuario reconozca las secciones.

- **Control y libertad del usuario**: el sistema siempre tiene que ofrecer alguna opción para deshacer o abandonar la tarea que está realizando. Para ello en nuestra web, en las acciones que lo requieran y que sean delicadas siempre se ofrecerá al usuario una ventana de confirmación para que el usuario pueda deshacer la acción.

- **Consistencia y estándares**: la interfaz grafica debe ofrecer elementos consistentes y mantener una cierta coherencia en todas las pantallas, además de implementar estándares que el usuario entienda, como puede ser el menú de hamburguesa utilizado en el menú de la resolución móvil. Gracias a *Angular Material* utilizamos elementos consistentes que mantienen una cierta cohesión visual.

- **Prevención de errores**: el sistema debe de ayudar al usuario a que no cometa errores. En nuestro caso, hay pocos errores que pueda cometer el usuario, aun así nos aseguramos de que el usuario no los cometa mostrándole mensajes de confirmación en las acciones que no tengan vuelta atrás.

- **Reconocimiento antes que recuerdo**: las opciones tienen que estar bien representadas para que el usuario no tenga que recordar la información de como navegar entre secciones o páginas.

- **Flexibilidad y eficiencia de uso**: deben proporcionarse facilidades o aceleradores para que todos los usuarios, ya sean novatos o expertos puedan disfrutar de una experiencia de uso satisfactoria. La aplicación desarrollada no dispone de acciones complejas que requieran muchos pasos por lo que en este aspecto respecta, este principio de usablilidad no se ha tenido en cuenta.

- **Estética y diseño minimalista**: no se debe cargar la interfaz con elementos innecesarios que puedan entorpecer la experiencia de usuario. En nuestro caso *Angular Material* ofrece elementos simples y minimalistas, además de proporcionar una paleta de colores visualmente agradable.

- **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores**: si el usuario comete algún error el sistema debe proporcionar mensajes de advertencia que indique claramente en que se ha equivocado el usuario y como puedo solucionarlo. Esto lo podemos ver bien en los formularios de inicio de sesión y registro en los que se informa con precisión que error ha cometido el usuario sin lugar a ambigüedades.

- **Ayuda y documentación**: a veces es necesario que los sitios web aporten información o algún apartado extra de ayuda. Nuestra aplicación usa recursos externos de otras *APIs* por lo que ha parecido conveniente crear una pantalla extra en la que se da crédito a las web de las *APIs* además de ofrecer información extra sobre el desarrollo del sitio web.

Una vez listados todos los principios de usabilidad, podemos concluir que se ha realizado un buen trabajo en el diseño de la interfaz de la aplicación, ya que la mayoría de los principios se cumplen en cierto modo. Aún así es cierto que algunos de estos principios no se han tenido en cuenta o se quedan un poco cortos ya que la aplicación y sus funciones no son muy complejas.

CAPÍTULO 9
Conclusiones

Una vez finalizado el proyecto listaremos los objetivos descritos al principio de la memoria y veremos si éstos se han cumplido y propondremos algunas futuras mejoras.

El hecho de que hayan tantos videojuegos y que cada año se estrenen tantos títulos hace difícil el seguimiento de éstos, es por ello que se propuso una aplicación web para poder explorar una amplia variedad de juegos y llevar un seguimiento de ellos. Antes de comenzar el desarrollo se hizo un estudio previo de distintas aplicaciones similares a la propuesta para extraer ideas y referencias. Es importante que antes de comenzar el desarrollo se concretará el método empleado en el mismo, es por ello que se explicaron diferentes métodos empleados en el desarrollo de *software* y se discutió cual era el que mejor se ajustaba a las necesidades del proyecto. Para dejar claro el alcance de nuestra aplicación se describieron en forma de casos de uso todas las funcionalidades y requerimientos que debe de cumplir el sistema, además de establecer los objetos involucrados y la base de datos utilizada. También utilizamos prototipos para observar la interfaz gráfica y la disposición de los elementos que la forman.

Para comenzar el desarrollo, se han listado todas las herramientas que se ha utilizado, tanto en el *frontend* como en el *backend*. En esta sección se hizo una descripción de estas herramientase y se explió con ejemplos de código como se han usado en nuestra aplicación. Para resumir el desarrollo se ha podido ver como crear nuestra propia *API* local y gestionar los recursos, además de gestionar peticiones a esta *API* y a *APIs* externas desde el *frontend* para obtener información y presentarla en la interfaz. Una vez terminado el desarrollo, se observó como el sistema cumple con las funcionalidades propuestas ofreciendo capturas de pantalla de la aplicación final.

Para finalizar los objetivos, se validó la solución desarrollada. Se empleo el método heurístico, utilizando los 10 principios de usabilidad de Jakob Nielsen para comprobar si nuestra interfaz cumple dichos principios y ofrece al usuario una experiencia agradable.

En cuento a futuras mejoras que se pueden proponer para un futuro podríamos exportar las librerias de otras plataformas y así tener una librería común en nuestra aplicación. Dentro de las listas del usuario, se podrían implementar algunas mejoras, como separar los juegos por series o poder crear tus propias listas con las preferencias deseadas. Y por último, de momento se trata de una aplicación local, por lo que si se llegan a implementar estas mejoras lo ideal sería desplegar la aplicación web para que sea pública.

Bibliografía

[1] Cuantos jugadores hay? Josh Howarth https://explodingtopics.com/blog/number-of-gamers

[2] La industria de los videojuegos ya genera más ingresos que la música y el cine juntos. Jorge Arias https://theobjective.com/economia/2023-06-04/videojuegos-ingresos-musica-cine/

[3] GTA 5 supera los 160 millones de unidades vendidas y pulveriza nuevos récords. Sergio González https://as.com/meristation/2022/02/08/noticias/1644296229_254671.html

[4] 'Avatar: El Sentido del Agua' es la cuarta película con más ingresos de todos los tiempos. JL Hodgson https://en.as.com/entertainment/avatar-the-way-of-water-now-fifth-highest-grossing-movie-of-all-time-n/

[5] Pokémon es "la franquicia de medios más valiosa del mundo", según un reciente estudio. Óscar Tejedor https://www.marca.com/videojuegos/nintendo/2022/05/04/62725628268e3e557f8b4600.html

[6] Steam supera los 33 millones de usuarios activos de forma simultánea. Eduardo Medina https://www.muycomputer.com/2023/01/09/steam-33-millones-usuarios-activos/

[7] ¿Qué es la metodología ágil? https://www.redhat.com/es/devops/what-is-agile-methodology

[8] Diferencia entre el modelo de cascada y el modelo incremental. Rudeus Greyrat https://barcelonageeks.com/diferencia-entre-el-modelo-de-cascada-y-el-modelo-incremental/

[9] ¿Qué es una base de datos relacional (sistema de gestión de bases de datos relacionales)? https://www.oracle.com/es/database/what-is-a-relational-database/

[10] Qué es *NodeJS* y para qué sirve. Jesús Lucas https://openwebinars.net/blog/que-es-nodejs/

[11] TypeScript vs. JavaScript: Your Go-to Guide. Daniele Monesi https://www.toptal.com/typescript/typescript-vs-javascript-guide

[12] Sequelize documentation https://sequelize.org/docs/v6/

[13] Express/Node introduction https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

[14] How to use CORS in Node.js with Express. Joseph Chege https://www.section.io/engineering-education/how-to-use-cors-in-nodejs-with-express/

[15] Qué es *Json Web Token* y cómo funciona. Luis Miguel López Magaña https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/

[16] *JWT* y *Node.js*: Cómo crear un sistema de autenticación. M.Domínguez https://todoxampp.com/jwt-y-nodejs-como-crear-un-sistema-de-autenticacion/

[17] Encriptación de password en *NodeJS* y *MongoDB*: *bcrypt* https://www.izertis.com/es/-/blog/encriptacion-de-password-en-nodejs-y-mongodb-bcrypt

[18] *Angular docs* https://angular.io/docs

[19] *Angular Material* https://material.angular.io/

[20] Validadores y herramientas para consultorías de accesibilidad y usabilidad https://www.usableyaccesible.com/recurso_misvalidadores.php

[21] Análisis Heurístico para UX - Cómo Ejecutar una Evaluación de Usabilidad. Miklos Philips. https://www.toptal.com/designers/usability-testing/analisis-heuristico-para-ux-como-ejecutar-una-evaluacion-de-usabilidad

[22] Los 10 principios de usabilidad de Jakob Nielsen: *be user friendly*. Beatriz Allas Miguelsanz. https://profile.es/blog/los-10-principios-de-usabilidad-web-de-jakob-nielsen/

APÉNDICE A

ANEXO: OBJETIVOS DE DESARROLLO DISPONIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Una aplicación cuya función principal es la exploración de videojuegos puede parecer, a primera vista, una herramienta diseñada para el entretenimiento y la diversión. Sin embargo, como parte de un mundo cada vez más conectado y digital, esta aplicación puede tener un impacto más amplio y complejo en la sociedad y la economía, lo que la relaciona indirectamente con el Objetivo de Desarrollo Sostenible (ODS) de Ïndustria, Innovación e Infraestructuras".

En primer lugar, es importante reconocer que la industria de los videojuegos es una de las más dinámicas y creativas en la actualidad. Los videojuegos no son solo formas de entretenimiento, sino también plataformas de expresión artística, narrativa y tecnológica. La constante innovación tecnológica en esta industria ha dado lugar a avances notables en términos de gráficos, procesamiento de datos, inteligencia artificial, realidad virtual y mucho más. Cada nuevo juego desafía los límites de lo que es posible en el ámbito tecnológico y creativo.

En este contexto, la aplicación de exploración de videojuegos puede desempeñar un papel relevante al destacar y promover videojuegos innovadores. Al proporcionar a los jugadores un espacio para descubrir títulos vanguardistas, la aplicación no solo les brinda entretenimiento, sino que también puede influir en la dirección futura de la industria de los videojuegos. Los desarrolladores, al ver que los juegos innovadores son apreciados y ganan reconocimiento, pueden sentirse motivados a explorar nuevas tecnologías y enfoques creativos para el desarrollo de videojuegos.

Además de promover la innovación tecnológica, esta aplicación también tiene el potencial de ampliar la audiencia de los videojuegos. Los videojuegos ya no son exclusivamente para un público joven o masculino; la diversidad de géneros, estilos y temáticas ha ampliado considerablemente su atractivo. Al destacar una amplia gama de videojuegos a través de la aplicación, se pueden presentar títulos que aborden diferentes intereses y que puedan atraer a una audiencia más diversa en términos de edad, género y experiencia de juego. Esto contribuye al crecimiento de la industria al ampliar su alcance y su base de jugadores.

Asimismo, esta aplicación puede funcionar como un escaparate para desarrolladores independientes y estudios de menor tamaño. Muchos videojuegos innovadores son creados por equipos pequeños que a menudo carecen de los recursos y el presupuesto de las grandes empresas de la industria. Al darles visibilidad en la aplicación, se les brinda la oportunidad de competir en un mercado altamente competitivo. Esto no solo fomenta la competencia saludable, sino que también puede contribuir al surgimiento de nuevas voces creativas y al estímulo de la diversidad en la industria.

En resumen, aunque una aplicación de exploración de videojuegos pueda parecer, en un principio, una herramienta de entretenimiento, su influencia se extiende más allá de proporcionar diversión. Al promover la innovación, la creatividad y la expansión de la audiencia en la industria de los videojuegos, esta aplicación puede desempeñar un papel importante en el cumplimiento del ODS de Ïndustria, Innovación e Infraestructuras". Su contribución al crecimiento y mejora de la industria no solo tiene implicaciones económicas, sino que también influye en el desarrollo tecnológico y en la promoción de la creatividad en una industria que sigue evolucionando rápidamente.