UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación de apoyo para Valorant, el popular shooter free-to-play de RIOT

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fayos Momparler, Víctor

Tutor/a: Sapena Vercher, Oscar

CURSO ACADÉMICO: 2023/2024

Resumen

El objetivo de este trabajo es desarrollar una aplicación que ofrezca al jugador vídeos cortos de *lineups* y *wallbangs*. Con este conocimiento, el usuario de la aplicación podrá disfrutar de una ventaja estratégica respecto al resto de jugadores de Valorant, ya que podrá dañar a un enemigo, conseguir información o defender una bomba sin el peligro de entrar en contacto directo con los adversarios. Además, no tendrá que memorizarlos previamente, algo bastante habitual entre los jugadores de Valorant más dedicados.

Un *lineup* es un punto de referencia dentro del juego desde el cual, con una inclinación concreta, es posible emplear las utilidades de tu agente contra un objetivo concreto, que se supone de interés, sin la necesidad y el peligro de estar cerca. Un ejemplo típico suele ser defender una bomba plantada desde una distancia segura. Por otra parte, un *wallbang* es un punto concreto del mapa desde donde se puede disparar y atravesar paredes y objetos, dañando a los posibles jugadores que se encuentren detrás.

Dado el poco margen de tiempo del que dispone el jugador durante la partida, la información que proporcionará la aplicación deberá adaptarse lo máximo posible a la situación del juego, filtrando de manera automática según el mapa que se esté jugando, el agente utilizado y el bando con el que se participa (ataque o defensa), o al menos, facilitando que el filtrado sea ágil. Esta información se conseguirá gracias a la API de eventos de videojuegos del SDK de Overwolf, que incluye a Valorant entre otros.

Se utilizarán principalmente TypeScript y HTML+CSS junto con un SDK proporcionado por Overwolf (overwolf.github.io), una plataforma de desarrollo que ofrece la posibilidad de publicar la aplicación en la tienda de aplicaciones de Overwolf, además de añadir anuncios y/o versiones premium para monetizarla.

Palabras clave: Juegos; TypeScript; HTML+CSS; Overwolf SDK; Valorant; Lineups; Wallbangs

Abstract

The goal of this project is to develop an application that provides short videos of lineups and wallbangs to the player. With that knowledge, the user will obtain a strategic advantage against the rest of Valorant players, as they will be able to harm an enemy, obtain information or defend a bomb without the risks involved in engaging the enemy. In addition, the user won't need to memorize them beforehand, something very usual among Valorant's most dedicated players.

A lineup is a reference point that, with a certain inclination, makes you able to use your agent's abilities against a specific target that is assumed to be of interest, without the need and danger of being close to it. A typical example is defending a planted bomb from a safe distance. On the other hand, a wallbang is a certain spot of the map where you can shoot through walls and objects, harming any player that could be hidden behind them.

Given the small spans of time that a player has during a match, the information shown by the app must be adapted, as much as possible, to the match that is being played, sorting automatically the content considering the current map, the agent selected and the side (attack or defense) that is being played or, at least, be agile enough to sort it manually. This information will be obtained thanks to the game events API of Overwolf SDK, which includes Valorant.

A TypeScript and HTML+CSS combo will be mainly used, alongside a SDK provided by Overwolf (overwolf.github.io), an app creating platform that offers the possibility of publishing your application in the Overwolf Appstore and also monetize it with ads and premium versions.

Keywords: Games; TypeScript; HTML+CSS; Overwolf SDK; Valorant; Lineups; Wallbangs

.

Tabla de contenidos

Índice de figuras

1. Introducción

En algunos videojuegos, ganar la partida y subir de rango se convierte en la aspiración principal de sus jugadores, Valorant[1] es un ejemplo de ello.

Entre los más cinco millones de jugadores diarios del título[2], son populares los vídeos con consejos para mejorar en el juego, las webs donde desarrollar estrategias y aprenderse centenares de *lineups y* las herramientas para mejorar tu precisión como Aimlabs[3].

Buscando un hueco en este mercado y también en respuesta a una necesidad personal nacida de la experiencia como jugador, el presente proyecto tiene como meta el desarrollo de una aplicación de PC que ayude a los jugadores del videojuego Valorant a sacar su máximo potencial.

La aplicación proporcionará a los usuarios vídeos cortos de *lineups* y *wallbangs,* dándoles así una información que les hará partir con ventaja respecto al resto de jugadores. Estos vídeos serán filtrados previamente para que coincidan con el mapa, agente y lado que se esté jugando en el momento.

La metodología elegida para el desarrollo es la Kanban, que forma parte de las metodologías ágiles y se centra en la gestión visual de las tareas usando un tablero.

1.1. Motivación

Aún recuerdo con ilusión cuando llegó a mis manos aquella Game Boy Color usada que mi primo, ya cansado de ella, decidió darnos a mi hermana mayor y a mi cuando yo solo tenía seis años.

Podría haberse quedado ahí, una anécdota más de mi infancia, pero fue sólo el principio de una afición que me ha acompañado toda mi vida y que, dada su estrecha relación con la tecnología, me hizo enamorarme también de la informática y cursar este grado.

Unos cuantos años después del regalo de aquella Game Boy, y tras un PC de disquetes de 3.5 con el que competía contra mi hermana en juegos de navegador, una Nintendo DS y una WII, mis padres compraron un portátil. Con él me inicié en mi adolescencia en los *shooters*[4] *online*; en este género descubrí que los videojuegos, a parte de un solitario pero entretenido pasatiempo podían ser también una competición donde intentar sacar lo mejor de ti y demostrarte cuánto has mejorado. Además de reforzar la amistad con tus amigos al competir juntos como equipo.

[1] https://playvalorant.com/es-es/
[2] https://tracker.gg/valorant/population
[3] https://aimlabs.com
[4] Un *shooter* es un videojuego cuyo objetivo principal es acabar con los enemigos mediante el uso de armas.

No ha sido sólo mi caso, ya que paralelamente, en la última década, los videojuegos en línea se han convertido en una actividad con un claro enfoque competitivo que ha relegado al jugador casual típico de antaño a un segundo plano.

Esta metamorfosis no ha sido fruto de la casualidad, sino consecuencia de varios factores clave, destacando los siguientes. En primer lugar, el aumento de jugadores: si comparamos los datos publicados en los anuarios de 2018[1] y 2023[2] por la Asociación Española de Videojuegos (AEVI)[5], en España ha habido un crecimiento de 3,2 millones de jugadores en los últimos 5 años. Por otra parte, el reconocimiento y la promoción por parte de medios de comunicación, marcas y empresas de tecnología que ha elevado la visibilidad de los deportes tecnológicos, atrayendo así a nuevos jugadores a la escena competitiva. Por último, la evolución de los propios videojuegos, dada la demanda del mercado, a un modelo más centrado en la competitividad, introduciendo modos de juego online con clasificaciones por nivel y complejos sistemas de emparejamiento de juego que buscan hacer las partidas lo más reñidas posible.

Con este contexto de videojuegos y jugadores altamente competitivos fue en el que empecé a jugar, hará cosa de 3 años, a Valorant, un juego de disparos en primera persona publicado en 2020 por la desarrolladora, RIOT Games[6], ampliamente conocida por el título League of Legends[7].

En este juego me topé con un problema, una vez llegado a rangos de competitivo más altos: muchos jugadores, que no necesariamente tenían más puntería o experiencia en *shooters* que yo, me superaban utilizando *lineups* y *wallbangs* de los que yo no podía aprovecharme por falta de conocimiento, causándome frustración y la pérdida de puntos de rango.

Con este problema en mente, y con el limitante de no querer utilizar mi tiempo libre, valioso y escaso, en memorizar toda esta información, busqué alguna web o aplicación que me ayudara con el problema. Al no encontrar ninguna solución que me convenciera del todo, decidí hacerla yo mismo aplicando los conocimientos adquiridos en el grado.

1.2. Objetivos

Este proyecto tiene como objetivo principal desarrollar una aplicación que proporcione, en el contexto de una partida de Valorant, vídeos de *lineups* y *wallbangs* que puedan ayudar al jugador y además eliminarle la necesidad de tener que memorizarlos de manera previa a la partida.

Como parte de la consecución del objetivo principal se han definido varios objetivos específicos:

- Los vídeos estarán filtrados automáticamente con los datos obtenidos de la partida en juego.
- Adicionalmente, también podrá filtrarse manualmente de una manera rápida.

[5] https://www.aevi.org.es/web/
[6] https://www.riotgames.com/es
[7] https://www.leagueoflegends.com/es-es/

- La interfaz será simple y fácil de usar.
- El usuario podrá controlar la reproducción de los vídeos a su conveniencia.
- La navegación entre vídeos debe ser fluida.
- El usuario podrá utilizar sus propios vídeos.

Además, se definen dos objetivos específicos adicionales relacionados con la monetización futura de la aplicación:

- La aplicación se convertirá en una posible fuente de ingresos pasivos.
- Se reducirá al mínimo posible el coste de mantenimiento de la aplicación.

1.3. Impacto esperado

Se espera que la aplicación resultante de este desarrollo ofrezca a los usuarios la posibilidad de aprender lineups y wallbangs rápidamente justo antes y durante la partida, evitando tener que memorizar toda esta información.

Este conocimiento hará mejorar su rendimiento en partida, permitiéndoles llegar a rangos más altos en Valorant.

Por otro lado, el filtrado se hará automáticamente a partir de la información recibida de la partida en juego. Actualmente es común que el jugador de Valorant busque en webs específicas donde tiene que filtrar manualmente los vídeos, por tanto, el uso de la aplicación ahorrará una gran cantidad de tiempo a largo plazo a los usuarios.

Además, la posibilidad de añadir vídeos propios hará que el usuario pueda disfrutar de aquellos vídeos en los que está interesado, pero no están en la aplicación.

Finalmente, viéndolo desde una perspectiva más general, se espera impulsar la creciente comunidad de Valorant al añadir una herramienta adicional de utilidad para los jugadores.

1.4. Metodología

La metodología para utilizar durante el desarrollo de este proyecto es la Kanban[3].

Kanban, o tarjeta visual en español, es una metodología para la gestión de proyectos y tareas que surgió en Toyota de la mano de Taiichi Ohno en la década de los 40, como parte de su sistema de gestión de producción JIT (just-in-time).

Mediante la metodología Kanban se podrá gestionar visualmente la carga de trabajo usando un tablero[4]. Este tablero se divide en distintas columnas que representan las etapas por las que debe pasar cada tarea y una tarjeta por cada tarea que haya por desarrollar, avanzando esta de columna a medida que es implementada.

Utilizaremos tres columnas para la gestión del desarrollo de este proyecto: To do (pendiente), In progress (en progreso) y Done (terminado).

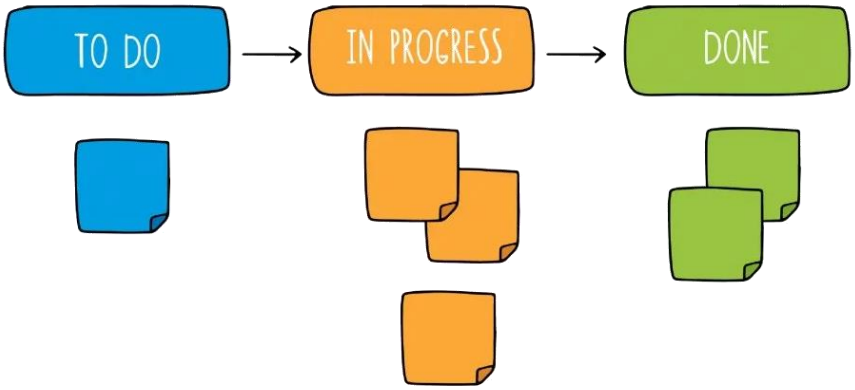


Ilustración 1: Representación simplificada de un tablero Kanban

La metodología Kanban se basa en varios principios fundamentales:

- **Calidad garantizada**: Esta metodología prioriza la calidad sobre la velocidad, con el objetivo de evitar retrabajos. El principio se basa en que, a menudo, es más eficiente hacer las cosas bien desde el principio que corregir errores posteriormente.
- **Reducción del desperdicio**: El objetivo es realizar solo lo esencial y necesario, eliminando lo superfluo o secundario. Siguiendo el principio "You Ain't Gonna Need It" (YAGNI), se evita implementar funcionalidades que no son estrictamente necesarias en el momento, enfocándose en lo que realmente aporta valor.
- **Mejora continua**: Basado en los objetivos deseados, Kanban no solo ayuda a organizar y visualizar tareas, sino que también se orienta hacia la optimización constante. Se adapta fácilmente a los cambios y busca mejorar la eficiencia y la calidad del proceso de trabajo.
- **Flexibilidad**: Esta es una característica clave de la metodología Kanban, permitiendo reorganizar el orden de las tareas en la cola de espera según las necesidades del momento.

Se ha escogido la metodología Kanban principalmente por la simplicidad de esta y la visión completa del estado de desarrollo de la aplicación que ofrece el tablero de una manera inmediata.

Adicionalmente hay otros factores que hacen decantarse por esta metodología.

Flexibilidad. Kanban permite ajustarse a cambios en el desarrollo de manera rápida y eficiente.

Planificación continua. Kanban, a diferencia de algunas metodologías *agile*, no utiliza *sprints* de un tiempo fijo, sino que se gestiona como un flujo continuo de trabajo. La gestión por *sprints* en este caso, sería contraproducente dado que el desarrollo se va a producir en ratos libres que no siguen una estructura fija en el tiempo.

Cola de tareas. La cola de tareas permite poder tener una vista clara de las tareas pendientes. Muchas de las tareas se tratan de pequeñas funcionalidades que surgen utilizando *brainstorming*

y son fácilmente olvidables en el futuro, tener el tablero para no perderlas es idóneo. Además, la vista completa de la cola de tareas ayuda a decidir la siguiente a implementar, centrándose primero en aquello que más valor aporte a la aplicación.

No obstante, también hay que estar alerta de algunos problemas que nos puede causar utilizar la metodología Kanban en este proyecto.

Falta de una estructura fija de tiempo. Aunque se ha señalado anteriormente que la planificación continua de Kanban puede ser beneficiosa, oculta el riesgo de no tener una sensación de urgencia en el desarrollo y puede hacer que este se demore más de lo necesario.

Dependencia de la disciplina. Todas las ventajas que ofrece Kanban pueden desaparecer si no se es disciplinado con la gestión del tablero.

Orden de implementación sin especificar. No hay una priorización de tareas clara como si pueden tener otras metodologías, esto puede crear momentos de inactividad entre la implementación de una tarea y la siguiente, ya que hay que determinar que tarea debe realizarse.

En resumen, se ha optado por la metodología Kanban ya que es la que mejor se adapta a las necesidades de este proyecto, sin embargo, hay que tener en consideración algunos aspectos negativos que podrían mermar el desarrollo.

1.5. Estructura

La memoria del TFG consta de ocho capítulos. A continuación, se resumirán brevemente cada uno de ellos para dar al lector una ligera idea de su contenido.

Capítulo 1 - Introducción. En este capítulo se explica la idea a implementar, la motivación para su desarrollo, los objetivos y el impacto que se espera conseguir, la metodología y, por último, la estructura del documento.

Capítulo 2 - Estado del arte. En este capítulo se explican las reglas de Valorant y se defiende la importancia de los *lineups* y *wallbangs* dentro del videojuego. Posteriormente se hace un análisis de distintas herramientas existentes en el mercado y, finalmente, se propone una idea de aplicación.

Capítulo 3 - Análisis del problema. En este capítulo se hace, partiendo de la propuesta del capítulo anterior, una especificación completa de los requisitos que debe cumplimentar nuestra aplicación y, posteriormente, se debaten las distintas soluciones tecnológicas posibles al reto propuesto, eligiendo finalmente la opción más óptima.

Capítulo 4 – Diseño de la solución. En este capítulo se elige la arquitectura de la aplicación y se detalla el contenido de cada uno de sus niveles. Posteriormente, se definen aspectos de vital importancia como las clases de la aplicación, la estructura de la base de datos y el diseño de la interfaz.

Capítulo 5 – Tecnología utilizada. En este capítulo se comentarán brevemente cada una de las tecnologías utilizadas en el proyecto, explicando sus principales características y su papel dentro del desarrollo.

Capítulo 6 – Desarrollo de la solución propuesta. En este capítulo se detalla el proceso mediante el cual se ha pasado de la propuesta a la solución final, describiendo los problemas encontrados, decisiones tomadas y algunas implementaciones relevantes.

Capítulo 7 – Pruebas. En este capítulo se detallan las pruebas realizadas para asegurar el correcto funcionamiento de la aplicación.

Capítulo 8 – Conclusiones. En este último capítulo se hace un análisis retrospectivo de todo el proceso de desarrollo, se exponen los conocimientos adquiridos durante la carrera que se han utilizado durante este y finalmente se listan distintos caminos que puede tomar la aplicación tras terminar el trabajo.

2. Estado del arte

En este capítulo se detallará el funcionamiento de Valorant, se expondrá la importancia de los *lineups* y *wallbangs* dentro del videojuego, se analizarán diversas soluciones ya existentes en el mercado y se concluirá con una propuesta de aplicación.

2.1. Valorant

Dado que el objetivo de este proyecto es el desarrollo de una aplicación de ayuda para Valorant, sería interesante familiarizarse previamente con el funcionamiento del juego[8] y la importancia que tienen los *lineups* y *wallbangs* en este.

2.1.1. Qué es Valorant y cómo se juega

Valorant es un videojuego gratuito de disparos desarrollado y publicado por RIOT Games. Desde su lanzamiento en 2020, ha destacado como uno de los títulos más populares en la escena de los deportes electrónicos[5], gracias a su enfoque táctico y competitivo.

Ilustración 2: Logo de Valorant


Ilustración 3: Experiencia de juego de Valorant

En el modo clásico de Valorant, que es el más jugado y el que se emplea en el ámbito competitivo, dos equipos de cinco jugadores cada uno se ven enfrentados en un mapa[9] elegido aleatoriamente. En el mapa hay 2 lados, atacante y defensor. Cada equipo ocupa un lado de inicio y se intercambia con el otro una vez se llegue a la mitad de la partida.

[8] https://playvalorant.com/es-es/news/announcements/beginners-guide
[9] https://playvalorant.com/es-es/maps/

Ilustración 4: Representación de Icebox, un mapa de Valorant

Previamente al comienzo de la partida, cada jugador elige un agente con el que jugará. Cada agente tiene unas habilidades únicas y pertenece a uno de los cuatro roles definidos dentro del juego[10].

Las partidas se disputan al mejor de 25 rondas o, en otras palabras, el equipo que llegue primero a 13 rondas ganadas será el ganador.

Ilustración 5: Resumen por rondas de una partida de Valorant.

Cada ronda se decide bajo el siguiente criterio:

[10] https://playvalorant.com/es-es/agents/

- **Gana el lado atacante.** Los atacantes consiguen la ronda si eliminan por completo al lado defensor o consiguen detonar una bomba llamada Spike. El explosivo es portado por un miembro del lado atacante y puede ser activado en una de las zonas de plantado, generalmente llamadas A o B. La Spike necesita siete segundos para activarse y tarda 45 segundos en explotar. Durante este tiempo los atacantes deberán evitar que el lado defensor la desactive.
- **Gana el lado defensor.** Los defensores consiguen la ronda si eliminan en su totalidad al lado atacante antes de que activen la Spike o, una vez activada, si consiguen desactivarla antes de que terminen los 45 segundos. De igual forma que para activarla, se precisan de siete segundos para desactivarla, aunque en este caso existe un *checkpoint* justo a la mitad que permite la desactivación en dos tiempos.

Ilustración 6: Intento de desactivación de la Spike

2.1.2. *Wallbangs* y *lineups*. Por qué son importantes

Primero de todo, vamos a definir los conceptos de *lineup* y *wallbang*.

Un *lineup* es un punto de referencia dentro del juego desde el cual, con una inclinación concreta, es posible emplear las utilidades de tu agente contra un objetivo concreto, que se supone de interés, sin la necesidad y el peligro de estar cerca.

Por otro lado, un *wallbang* es un punto concreto del mapa desde donde se puede disparar y atravesar paredes y objetos, dañando a los posibles jugadores que se encuentren detrás.

Clarificados los conceptos de *wallbang* y *lineup* y explicadas las normas del juego en el apartado anterior, queda claro que eliminar a un enemigo o evitar que planten/desplanten la

Spike puede hacer ganar la ronda. Es por esta razón que los *wallbangs* y los *lineups* son tan interesantes en Valorant, ya que permiten realizar estas acciones sin tomar riesgos.

Con el objetivo de dar más peso a la argumentación y acercar al lector al frenesí de una partida de Valorant se expondrá en el siguiente apartado un caso, acompañado de documentos gráficos, donde tanto *wallbangs* como *lineups* han sido utilizados en la escena competitiva.

2.1.3. *Wallbangs* y *lineups*. Un caso práctico

Desde el año 2021, Riot Games organiza el Valorant Champions Tour (VCT)[6], un circuito competitivo global que atrae a los mejores jugadores y equipos de Valorant del mundo.

En los VCT de 2022, concretamente en el Stage 2 de Norteamérica, Evil Geniuses (EG), actualmente vigente campeón de VCT tras ganar la edición de 2023[7], se enfrentó en fase de grupos contra Luminosity (LG) al mejor de tres partidas[8][11]. Vamos a utilizar la primera partida del enfrentamiento como ejemplo.

En la quinta ronda de la partida, Evil Geniuses, jugando como lado atacante, consigue activar la Spike en la zona de B.

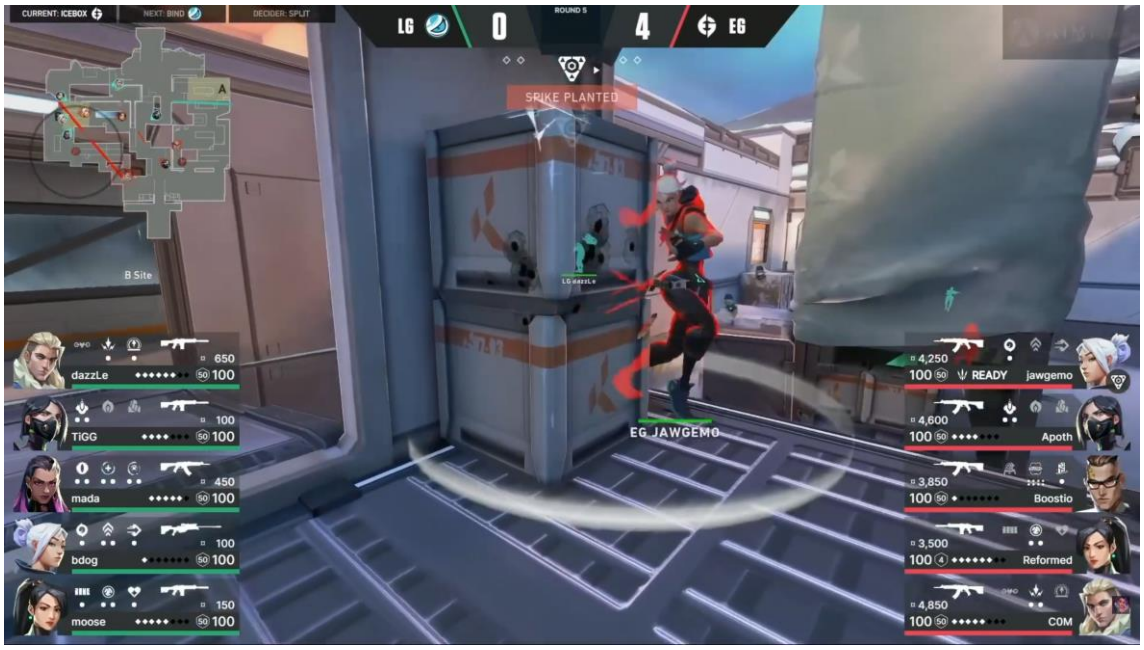


Ilustración 7: Evil Geniuses activa la Spike en B

Con la bomba ya activada, la Viper de Evil Geniuses, Apoth, utiliza un *lineup* desde una zona alejada de B para proteger la Spike de una posible desactivación con su habilidad Mordedura.

[11] https://www.youtube.com/watch?v=JX0gfuxKFQI

Ilustración 8: Aspecto de Viper

Ilustración 9: La Viper de EG, Apoth, utilizando un lineup

Ilustración 10:La habilidad Mordedura de Apoth cayendo sobre la Spike activa

Pocos segundos después, el Sova de Evil Geniuses, COM, utiliza otro *lineup* para lanzar dos veces su Proyectil Eléctrico a donde se encuentra la Spike activada. Gracias a los 2 lineups utilizados, el equipo rival, Luminosity apenas dispone de tiempo ahora para desactivar la Spike.

Ilustración 11: Aspecto de Sova

Ilustración 12: el Sova de EG, COM, utilizando un lineup

Ilustración 13: La habilidad Proyectil Eléctrico de COM cayendo sobre la Spike activa

Justo tras caer el último proyectil en la Spike, Luminosity consigue eliminar a la Apoth y se pone, con uno contra dos a su favor, a intentar desactivar la Spike.

De una manera muy inteligente ya que evita todo riesgo, COM elimina utilizando un *wallbang* desde tubo al jugador de Luminosity que estaba desactivando la Spike.

Ilustración 14: COM elimina a un enemigo utilizando un wallbang desde tubo

Aunque es eliminado inmediatamente por el último jugador restante de Luminosity, ya no queda tiempo suficiente para desactivar la Spike y Evil Geniuses consigue llevarse la ronda.

Ilustración 15. EvilGeniuses gana la ronda tras explotar la Spike

Con este desenlace de ronda, se concluyen todas las explicaciones pertinentes sobre el juego y se da por justificando la importancia de técnicas como los *lineups* y los *wallbangs* dentro de una partida. La descripción de estos conocimientos básicos sobre el juego será esencial para poder comprender este documento en su totalidad.

2.2. Aplicaciones similares

Previamente al diseño e implementación es imprescindible buscar y analizar distintas herramientas ya existentes que mantengan similitudes con la aplicación a desarrollar. Este

análisis servirá para encontrar las deficiencias y fortalezas de las soluciones actualmente presentes en el mercado. Las conclusiones extraídas del análisis ayudarán a definir las funcionalidades que nuestra aplicación deberá implementar para aportar un valor adicional a los potenciales usuarios.

Las herramientas existentes elegidas deben coincidir en la búsqueda del mismo fin que la aplicación a desarrollar: ayudar al usuario a mejorar su rendimiento en Valorant, ya sea de una manera u otra. Se ha seleccionado para su análisis las siguientes aplicaciones:

2.2.1. Valoplant

Valoplant[12] es la principal aplicación de creación de estrategias en Valorant. Valoplant nació en 2020 como una web donde crear estrategias de equipo para partidas de Valorant. Hoy en día, dispone de un apartado para *lineups* y aplicación dedicada para Mac, Windows y Android. La aplicación para Windows está disponible también para su descarga desde la tienda de Overwolf[13].

Ilustración 16: Logo de Valoplant

Ilustración 17: Apartado de lineups de Valoplant

Valoplant destaca por su interfaz simple y su funcionalidad de crear estrategias muy completas, que se ha ido puliendo con el paso de los años. La aplicación permite crear estrategias colaborativamente en un lobby, lo que lo hace idóneo para organizar tácticamente a un equipo.

Las estrategias se crean en el mapa seleccionado, arrastrando las habilidades de los agentes deseados al mapa y completándolas usando las distintas herramientas que aporta la aplicación: secuenciar por pasos las habilidades, añadir comentarios de voz, dibujar, escribir o añadir imágenes. Las habilidades seleccionadas se representan en el mapa con el mismo tamaño relativo al que ocupan en el juego. Se trata de una funcionalidad muy atractiva, ya que permite visualizar mejor el desempeño de la estrategia en creación.

Una vez situadas todas las habilidades en el mapa, se habrá creado una secuencia de acciones a acometer por el equipo, es decir, una estrategia. Esta se podrá guardar para un futuro visionado.

[12] https://valoplant.gg/es
[13] https://www.overwolf.com/app/shoast-valoplant

En el apartado de comunidad podremos tanto compartir una estrategia como visualizar e importar aquellas compartidas por la comunidad.

Adicionalmente, Valoplant ofrece un apartado de *lineups*. Estos son listados por mapa y agente. Como se muestra en la Ilustración 17: Apartado de lineups de Valoplant, cada *lineup* se representa sobre el mapa usando la imagen del agente en la posición desde la que se va a usar la habilidad, unido con una línea a un círculo que representa la zona donde se aplicará la habilidad. Los *lineups* se pueden exportar desde este apartado al apartado de estrategia, quedando representados de una manera muy clara y elegante.

Valoplant es la aplicación para Valorant más popular, sin embargo, su apartado de *lineups* está claramente enfocado para ser compatible con la creación de estrategias. La selección de un *lineup* se hace desde una representación del mapa elegido. Pese a que esto es atractivo y muy representativo, hace que visualizar los vídeos sea muy poco ágil, haciendo inviable la búsqueda y visionado de *lineups* durante una partida. Además, no permite añadir vídeos propios ni tiene un apartado de *wallbangs*, lo que resta versatilidad a la aplicación en su faceta de creación de estrategias.

Para finalizar, vale la pena destacar, dada su expansión y sus cuatro años de mantenimiento, que se trata de un proyecto que parece haber conseguido la viabilidad económica con funciones premium que no limitan en exceso la experiencia del usuario gratuito y anuncios no intrusivos.

2.2.2. Valorant Lineups

Valorant Lineups[14] es una aplicación para Windows publicada en la tienda de Overwolf en 2021. Se trata de un concepto de aplicación similar al propuesto y por tanto un competidor directo.

Ilustración 18: Logo de Valorant Lineups

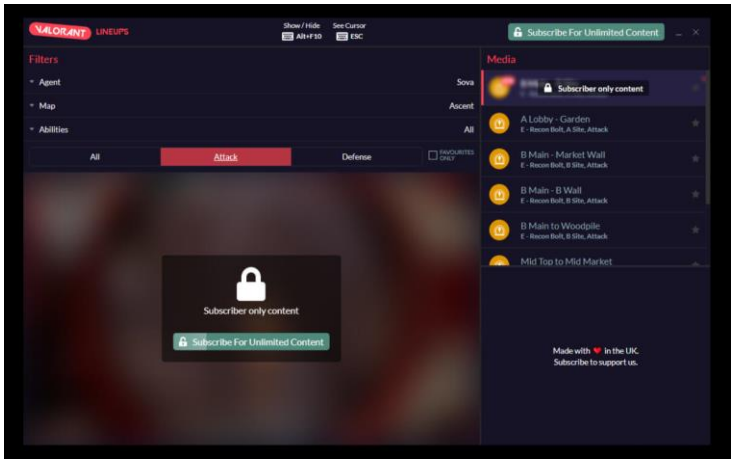


Ilustración 19: Interfaz de Valorant Lineups

Como se comentaba en el párrafo anterior, estamos ante una idea muy similar, ya que la aplicación detecta los datos de la partida y te ofrece *lineups* relacionados con el mapa, lado y agente. Pese a que goza de una interfaz simple que hace que la aplicación sea intuitiva y fácil de

[14] https://www.overwolf.com/app/gabriel_micko-valorant_lineups

utilizar, existen pocos factores positivos más a destacar. Valorant Lineups ha sido probada previamente al desarrollo de la aplicación propuesta y se trata de una de las soluciones existentes mencionadas al final del apartado de metodología y que no convencen.

Destacan varios aspectos que hacen no recomendar su uso. Principalmente, existen varios problemas con el contenido que influyen muy negativamente en la experiencia de usuario.

- **Falta de variedad.** La aplicación sólo dispone de vídeos de *lineups*.
- **El contenido no ha sido actualizado.** La aplicación apenas ha tenido mantenimiento desde que fue lanzada. Esta falta de mantenimiento ha provocado que falten muchos mapas y agentes en la aplicación y que diversos *lineups* expuestos en la aplicación hayan quedado inutilizables ya que algunos de los mapas han sufrido modificaciones durante este periodo de tiempo.
- **Exceso de contenido de pago.** Gran parte de los *lineups* que la aplicación ofrece al usuario son de pago, esto hace que pierda su utilidad en gran medida para los usuarios que quieren hacer un uso gratuito de la aplicación.

De manera menos crítica, existen un par de aspectos negativos en el desempeño de la aplicación que cabe mencionar también.

- **Problemas con la carga de vídeos desde fuera de Europa.** Como expone el usuario de Reddit NeoStarSky97, los vídeos de la aplicación tardan mucho en cargar. El desarrollador achaca este problema a que el usuario reside en India y sólo disponen de servidores en Frankfurt, Alemania[15].
- **La interfaz es mejorable.** Aunque previamente se destacaba que la interfaz es simple y fácil de utilizar hay que mencionar que la localización en pantalla, tamaño y forma del apartado de filtros carece de sentido.

No obstante, como aspecto positivo es interesante destacar que los desarrolladores han reaprovechado el trabajo realizado en crear de una web[16] similar a su aplicación, eso sí, sin la posibilidad de filtrar por los datos de partida obtenidos. Se trata de una posibilidad interesante que se tendrá en cuenta.

2.2.3. Strats.gg

Strats.gg[17] nació alrededor de 2021 como una web donde ver *lineups* de Valorant. Dos años más tarde, en 2023, publicaron su propia aplicación de PC en la tienda de Overwolf y ha sido descargada más de cien mil veces[18].

[15] https://www.reddit.com/r/ViperMains/comments/rm13jc/comment/hq0hz4e/
[16] https://lineups.fun/
[17] https://strats.gg/valorant
[18] https://www.overwolf.com/app/strats.gg-strats_app

Centraremos el análisis en la aplicación, dado que el formato coincide con nuestra idea propuesta e incluye funcionalidades adicionales respecto a la versión web.

Ilustración 20: Logo de Strats.gg

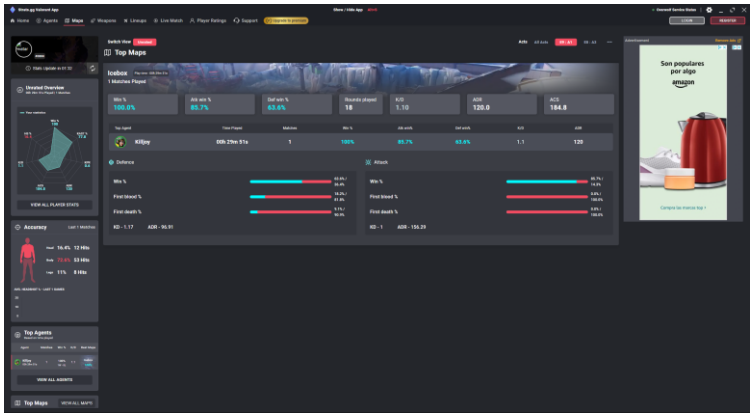


Ilustración 21: Informe de mapas de Strats.gg

Pese a ser una evolución de una web donde ver *lineups*, la aplicación de Strats.gg parece mucho más centrada en las estadísticas. Almacena las estadísticas de las partidas del jugador en un historial y las utiliza para ofrecer informes de rendimiento por mapa, agente y arma.

Ilustración 22: Apartado de lineups de Strats.gg

El apartado de *lineups* es casi idéntico al que tienen en la web y la forma de visualizarlos es muy similar a Valoplant, aplicación previamente analizada. Por tanto, comparte en este apartado las mismas ventajas e inconvenientes.

Strats.gg ofrece dos filtros adicionales, por habilidad del agente seleccionado y por dificultad del *lineup*. Estos cambios, aunque son interesantes, no solucionan los problemas que se han señalado para Valoplant. Pese a que la forma de representar los *lineups* en el mapa es muy elegante, todo el proceso previo a visualizar los vídeos es muy poco ágil. El tiempo que se tarda en filtrar y acceder a un *lineup* es excesivo como para hacerlo durante una partida.

La interfaz de usuario de Strats.gg destaca respecto a las anteriores herramientas proporcionando al usuario una aplicación muy atractiva a la vista. Tanto el agradable diseño de

la aplicación como el detalle con el que está hecho cada apartado denotan una aplicación muy trabajada.

Otro aspecto positivo que destacar es que la aplicación está muy bien integrada con Valorant, dando la posibilidad de utilizar la aplicación a modo de *overlay* y añadiendo las estadísticas recientes debajo de la ficha de cada jugador que te encuentres en partida.

En el apartado de *lineups*, que es el que nos incumbe, se han detectado otros factores negativos a destacar como la imposibilidad por parte del usuario de subir su propios *lineups*. Por otro lado, la utilidad de la herramienta de *lineups* está mermada por una limitación de 40 visualizaciones de *lineups* diarias impuesta a los usuarios que no son de pago así como la imposibilidad de usar la aplicación si no se tiene Valorant abierto.

Finalmente, cabe destacar que, pese el claro esfuerzo que han tenido en el desarrollo por integrar la aplicación en Valorant, la aplicación no filtra automáticamente los *lineups* con los datos recibidos de la partida en juego. Esta funcionalidad solventaría el problema expuesto sobre la poca agilidad del proceso de filtrado previo a la visualización.

2.3. Propuesta

Tras analizar las herramientas ya existentes en el mercado, se pretende, teniendo en cuenta sus fortalezas y flaquezas, realizar una propuesta de aplicación que tenga una aportación significativa para la comunidad de Valorant.

Hemos visto que las aplicaciones existentes proponen un repositorio de *lineups* que se filtra por mapa, agente y lado manualmente. Es imprescindible que nuestra aplicación mantenga esta funcionalidad. Además, dado que el objetivo es que pueda ser usada mientras se juega, es crucial que la aplicación filtre automáticamente con los datos de la partida en juego como hace Valorant Lineups pero, a diferencia de este, hay que evitar que el contenido de la aplicación se quede desfasado por falta de mantenimiento durante demasiado tiempo.

Otro factor diferencial importante sería que existiese más contenido de ayuda a parte de los *lineups*, como los *wallbangs*, concepto ya mencionado anteriormente y que se ha tratado desde el principio como indispensable en el resultado final. Otro tipo de contenido que podría formar parte de la aplicación podría ser los *setup*, que son maneras de posicionar estratégicamente tus habilidades al inicio de la ronda para evitar el avance enemigo.

Otro aspecto mencionado de manera reiterada en los análisis previos es la inexistente posibilidad de que los usuarios puedan usar la herramienta con sus propios vídeos. Esta idea sería interesante incluirla en el desarrollo ya que nos diferenciaría del resto de aplicaciones existentes y además ayudaría a que la aplicación pudiese seguir siendo útil en caso de que el contenido se quedara desfasado durante algún periodo de tiempo.

Por último, para hacer que sea posible utilizar la aplicación rápidamente mientras se está jugando, habría que descartar la manera de representar los *lineups* con un mapa interactivo que utiliza tanto Valoplant como Strats.gg y decantarse por una solución más simple pero efectiva como podría ser un carrusel o una lista como la que se utiliza en Valorant Lineups.

2.4. Tabla comparativa

Nombre	Aplicación propuesta	Valoplant	Valorant Lineups	Strats.gg
Recopilación de *lineups*	×	×	×	×
Recopilación de *wallbangs*	×			
Filtrado por mapa, agente y lado	×	×	×	×
Filtrado manual	×	×	×	×
Filtrado automático con datos de partida	×		×	
Contenido actualizado	×	×		×
Permite subir vídeos propios	×			
Vídeos filtrados presentados en una lista o carrusel	×		×	
Vídeos filtrados presentados del mapa seleccionado		×		×
Rápido acceso al contenido	×		×	
Funcionalidades esenciales limitadas para los usuarios no premium			×	×
Recopilación y análisis de las partidas del jugador				×
Planificación de partidas		×		

Ilustración 23: Tabla comparativa de aplcaciones analizadas

Tras analizar la tabla comparativa que enfrenta nuestra idea propuesta con las herramientas previamente estudiadas, quedan claras las características en las que destaca nuestra propuesta de aplicación y ayuda a plasmar gráficamente lo comentado en el apartado anterior.

La recopilación de *wallbangs* y la posibilidad de subir vídeos propios destacan como características que ninguna otra herramienta analizada ofrece. Adicionalmente, el filtrado automático solo lo podemos encontrar en Valorant Lineups, aplicación mermada por la falta de contenido actualizado.

Ofrecer contenido más allá de *lineups* es un factor diferencial importante, siendo prioritaria la inclusión de la categoría *wallbangs* y dejando abierta la posibilidad de incluir otras categorías de contenido menos prioritarias como los *setup*. De esta manera, el contenido de nuestra aplicación será más completo y podrá resultar interesante a un abanico de jugadores más amplio.

Por otro lado, permitir el uso de vídeos propios en la aplicación proporcionará un nivel de personalización superior al de las otras herramientas. Además, si no se consigue mantener actualizado el contenido en un futuro, esta funcionalidad permitirá que la aplicación sea en cierto grado utilizable, ya que los usuarios podrán mantener el contenido al día utilizando sus propios vídeos.

Finalmente, también resulta interesante comentar brevemente aquellas características de la tabla comparativa que no cumple nuestra propuesta:

- **Representar los vídeos dentro del mapa seleccionado.** Se ha decidido representar los vídeos en forma de lista o carrusel para hacer el acceso al contenido más rápido. Se trata por tanto de una funcionalidad que no se considera necesaria.
- **Funcionalidades esenciales limitadas para los usuarios no premium.** Se trata de una característica negativa, algunos de los competidores limitan funcionalidades esenciales para usuarios no premium. Algunos ejemplos son limitar a 40 vídeos por día o bloquear parte del contenido. Se busca diferenciarse de estos competidores descartando esta política de restricción en nuestra aplicación.
- **Recopilación y análisis de las partidas del jugador**. Se trata de una característica muy alejada de la funcionalidad principal de nuestra propuesta y no resulta una amenaza para la viabilidad de nuestra aplicación.
- **Planificación de partidas.** Del mismo modo que el punto anterior, se trata de una característica muy alejada de la funcionalidad principal de nuestra propuesta y no resulta una amenaza para la viabilidad de nuestra aplicación.

3. Análisis del problema

En este capítulo se identificarán las necesidades del desarrollo de la aplicación y sus posibles soluciones.

Un correcto análisis de las necesidades es decisivo para el desarrollo de un proyecto exitoso, ya que proporcionará una base sólida sobre la que desarrollar y mantener la aplicación. Se conseguirá con una completa especificación de requisitos. Esta se alcanzará apoyándose en el uso de la técnica de historias de usuario, que nos ayudará a comprender las funcionalidades que necesita el usuario final.

Una vez identificados los requerimientos de la aplicación, se explorarán diversas soluciones que cumplan los desafíos planteados. Finalmente, se elegirá la solución más óptima para abordar de manera efectiva el problema expuesto.

3.1. Especificación de los requisitos

En los apartados previos *1.2 - Objetivos* y *2.3 - Propuesta* se han ido pincelando las necesidades de la aplicación a desarrollar. No obstante, se requiere de una especificación de los requisitos escrita y correctamente definida para conseguir un desarrollo de software de calidad.

Con el fin de establecer de forma precisa los requerimientos de la aplicación, vamos a usar la técnica de historias de usuario. El resultado nos servirá de apoyo para posteriormente definir los requisitos funcionales y no funcionales.

Relacionándolo con la metodología Kanban que vamos a utilizar, los requisitos funcionales resultantes se convertirán, tras una fase de análisis y valoración de cada uno, en unidades de trabajo que añadiremos a la cola de tareas de nuestro tablero. Los requisitos funcionales no están restringidos a seguir una relación directamente proporcional con las unidades de trabajo, es posible que para cumplir un requisito funcional se especifiquen varias unidades de trabajo y viceversa.

Dada la flexibilidad de las metodologías ágiles y de Kanban en concreto, durante el desarrollo se podrán añadir nuevas tareas a la cola de tareas y modificar o eliminar las ya existentes, debido a problemas o cambios en los requisitos que puedan surgir.

3.1.1. Historias de usuario

Una historia de usuario es una explicación general e informal de una funcionalidad de software escrita desde la perspectiva del usuario final o cliente.

Las historias de usuario son unas pocas frases en lenguaje sencillo que describen el resultado deseado, sin entrar en detalles. Son en realidad una aproximación a la parte funcional de un

requisito. Pese a la falta de precisión, las historias de usuario son una técnica de gran utilidad para delimitar posteriormente los requisitos funcionales. Su enfoque centrado en el usuario final de la aplicación asegurará que las necesidades de este serán satisfechas en el producto final.

Una historia de usuario normalmente utiliza un formato estándar donde incluye tres elementos clave:

- **Rol**: ¿Quién se beneficiará de esta característica?
- **Funcionalidad**: ¿Qué necesita hacer el usuario?
- **Motivación**: ¿Por qué el usuario necesita esta funcionalidad?

La estructura típicamente utilizada es:

> "**Como** [Rol],
> **Quiero** [Funcionalidad],
> **Para** [Motivación]."

Siguiendo esta plantilla, se expondrán a continuación las historias de usuario generadas.

Como usuario, **Quiero** visualizar vídeos de *lineups*, *wallbangs* y *setups*, **Para** mejorar mi rendimiento en Valorant.	**Como** usuario, **Quiero** que los vídeos se filtren automáticamente con los datos de mi partida, **Para** ahorrar tiempo filtrando durante la partida.
Como usuario, **Quiero** poder filtrar vídeos manualmente por distintas categorías del juego **Para** poder encontrar el contenido específico que busco.	**Como** usuario, **Quiero** poder controlar el vídeo en reproducción, **Para** rebobinar, avanzar o parar el vídeo cuando no tengo algo claro.
Como usuario, **Quiero** poder marcar vídeos como favoritos y filtrarlos, **Para** aislar el contenido que más se ajusta a mi forma de jugar.	**Como** usuario, **Quiero** poder desmarcar los vídeos de favoritos, **Para** quitar de mi lista de favoritos aquellos vídeos que ya no se ajustan a mi estilo de juego.
Como usuario, **Quiero** poder añadir mis propios vídeos, **Para** poder disponer de vídeos que se ajusten a mi gusto.	**Como** usuario, **Quiero** poder filtrar mis propios vídeos, **Para** aislar mi contenido del resto.
Como usuario, **Quiero** poder editar los datos de mis vídeos añadidos, **Para** subsanar errores que haya cometido al añadirlos.	**Como** usuario, **Quiero** poder eliminar mis vídeos añadidos **Para** quitar aquellos que ya no me interesa tener en la aplicación.
Como usuario, **Quiero** que la aplicación utilice pocos recursos,	**Como** usuario, **Quiero** que encontrar y visualizar un vídeo sea un proceso rápido,

Para que no afecte al rendimiento de Valorant.	**Para** poder consultarlo durante una partida sin que me perjudique.
Como usuario, **Quiero** poder convertirme en usuario premium mediante pago **Para** apoyar el proyecto y eliminar el contenido patrocinado.	**Como** usuario, **Quiero** poder convertirme en usuario premium mediante pago **Para** apoyar el proyecto y obtener alguna ventaja.
Como potencial usuario premium, **Quiero** realizar el pago mediante alguna plataforma de pago segura **Para** asegurarme de que mis datos bancarios se tratarán de manera segura	**Como** usuario, **Quiero** poder cambiar de un vídeo a otro fácilmente, **Para** encontrar un vídeo que me guste de forma cómoda y ágil.
Como usuario, **Quiero** ver todos los vídeos filtrados en una lista **Para** encontrar un vídeo que me guste fácilmente.	**Como** usuario, **Quiero** que toda la información necesaria se encuentre en una única pantalla **Para** ver todo lo que necesito si lo he dejado abierto en mi segunda pantalla mientras juego.
Como usuario**,** **Quiero** que la aplicación se adapte a mi tamaño de pantalla **Para** poder ver con claridad los vídeos sea cual sea mi configuración de pantalla	

Ilustración 24: Historias de Usuario

3.1.2. Requisitos funcionales

Una historia de usuario generalmente representa lo que en metodología ágil se denomina una épica[19] o un concepto muy vago de lo que quiere hacer el usuario. Las funcionalidades resultantes son o demasiado grandes como para ser implementadas en una iteración o demasiado poco específicas así que necesitan ser divididas en funcionalidades más pequeñas[9].

Por tanto, a fin de especificar los requisitos funcionales, analizaremos, filtraremos, desglosaremos y sintetizaremos las funcionalidades surgidas en las historias de usuario.

En este apartado utilizaremos un enfoque menos centrado en la perspectiva del usuario final, adoptando una perspectiva más técnica. Esto nos permitirá identificar y definir requisitos funcionales adicionales que pueden no haber surgido de manera literal en las historias de usuario ya que no aportan valor al usuario de una forma directa.

Se han definido los siguientes requisitos funcionales agrupándolos por temáticas para así facilitar la organización y priorización de tareas en nuestro tablero Kanban:

[19] https://www.scrummanager.com/bok/index.php/Epic

Contenido de la aplicación:

- Los usuarios podrán visualizar vídeos de *lineups*, *wallbangs* y *setups* usando la aplicación. Este requisito se dividirá en diferentes tareas en el tablero Kanban y se priorizará cada una debidamente durante el desarrollo.

Vídeos:

- Los usuarios podrán pausar, avanzar y retroceder los vídeos.
- El vídeo seleccionado se reproducirá automáticamente y en bucle.
- Los usuarios podrán cambiar de un vídeo a otro.
- Los usuarios podrán ver todos los vídeos que cumplan los filtros en una lista.

Monetización:

- La aplicación mostrará publicidad que irá cambiando.
- Los usuarios podrán quitar la publicidad con una aportación económica, convirtiéndose en premium.

Pago:

- Los pagos se realizarán mediante una pasarela de pago externa.

Favoritos:

- Los usuarios podrán marcar y desmarcar vídeos como favoritos.

Contenido propio:

- Los usuarios podrán añadir sus propios vídeos e integrarlos en la aplicación.
- Los usuarios podrán modificar los datos asociados a sus vídeos añadidos.
- Los usuarios podrán eliminar sus vídeos.

Filtrado:

- Los usuarios podrán filtrar los vídeos por mapa, agente, lado, habilidad y tipo de contenido. Este requisito se dividirá en diferentes tareas en el tablero Kanban y se priorizará cada una debidamente durante el desarrollo.
- Los usuarios disfrutarán de un filtrado automático una vez entren en partida dependiendo del mapa seleccionado, agente y lado.
- Los usuarios podrán separar los vídeos favoritos del resto con un filtro.
- Los usuarios podrán separar los vídeos propios del resto de vídeos de la aplicación con un filtro.

3.1.3. Requisitos no funcionales

Finalmente, definir los requisitos no funcionales completará nuestra especificación de requisitos, abordando aspectos cruciales que, aunque son menos tangibles, son esenciales para el desarrollo óptimo de la aplicación. Estos requisitos tratan características como la eficiencia, seguridad, mantenibilidad o compatibilidad.

Definir los requisitos no funcionales asegurará que la aplicación no solo cumpla con las funciones básicas definidas, sino que también lo haga de una manera efectiva y confiable.

Siguiendo la misma estructura que la utilizada para los requisitos funcionales se han definido los siguientes requisitos no funcionales:

Compatibilidad:

- La aplicación debe ser compatible al menos con Windows, único sistema operativo con el que se puede jugar a Valorant actualmente.
- La aplicación se adaptará a cualquier tamaño de monitor.

Seguridad:

- La información no sensible de los usuarios sólo podrá ser vista y modificada por ellos mismos y los administradores de la aplicación.
- La información sensible de los usuarios no podrá ser vista por nadie y solo podrá ser restablecida por el propio usuario.
- La pasarela de pago utilizada debe ser segura y reconocible.
- Validar los datos de entrada y salida de la pasarela de pago para evitar un uso malintencionado.

Rendimiento:

- La aplicación deberá funcionar de una manera fluida, rodando constantemente en treinta fotogramas por segundo o más.
- La carga de datos de la aplicación no deberá utilizar recursos de red ni del PC en exceso, ya que está pensada para usarse mientras se está ejecutando un videojuego.
- Los vídeos, en condiciones normales, no deberán de tardar más de cinco segundos en empezar a reproducirse una vez seleccionado.

Usabilidad:

- El uso de la aplicación debe ser simple e intuitivo. La aplicación no debe estar saturada de funcionalidades extra que entorpezcan su funcionalidad principal.
- No se debería tardar más de cinco segundos en encontrar un vídeo que sea útil.
- Los vídeos no deben durar más de treinta segundos.

- La herramienta debe ser intuitiva, permitiendo al usuario un uso normal desde el principio, con una curva de aprendizaje pronunciada.

Coste:

- Se debe maximizar el procesamiento y el almacenamiento de los datos de la aplicación en local para reducir el coste de mantenimiento en las fases más tempranas de la aplicación.
- El gasto económico necesario para el funcionamiento de la aplicación debe ser nulo.
- El proyecto debe constituirse como una vía para generar ingresos pasivos a largo plazo.

Monetización:

- La aplicación no tendrá publicidad intrusiva.
- El contenido y las funcionalidades imprescindibles serán idénticas tanto para los usuarios normales como los premium.

Tras decidir que el desarrollo se iba a realizar con Overwolf SDK, se añadieron dos requisitos adicionales:

Compatibilidad:

- La aplicación debe ser compatible con la infraestructura de Overwolf.
- La aplicación debe aprovechar las funcionalidades que ofrece Overwolf.

3.2. Identificación y análisis de soluciones posibles

Una vez definidas las funcionalidades con las que debería contar la aplicación, es fundamental centrar nuestros esfuerzos en identificar y analizar posibles soluciones a los desafíos planteados.

Este apartado estará enfocado en explorar distintas alternativas que puedan satisfacer los requisitos ya definidos. El objetivo es analizarlas una por una, exponer las ventajas e inconvenientes de cada alternativa y definir unos criterios de selección que nos ayuden a identificar la mejor solución.

Los principales desafíos que presentan los requisitos definidos son la obtención de los datos de la partida en juego para el filtrado automático, el almacenamiento del contenido de la aplicación, el almacenamiento del contenido añadido por el usuario y la búsqueda rápida de contenido útil.

3.2.1. Obtención de los datos de partida

Empezamos con uno de los aspectos más específicos de la aplicación, la obtención de los datos de la partida de Valorant que está jugándose.

Tras investigar las soluciones existentes se han encontrado tres posibles opciones: utilizar la plataforma de desarrollo de Overwolf[10], utilizar la API de Valorant[11] o conseguir la información a través del reconocimiento de patrones en pantalla.

Utilizar la plataforma de desarrollo de Overwolf nos proporcionará acceso a su API[12] y un SDK[13] para la creación de una aplicación basada en Chromium[20], ya que implementa una tecnología similar a Electron[14]. La API de Overwolf da acceso a los eventos que ocurren durante la ejecución de distintos videojuegos[15]. Dentro de la amplia lista de eventos de videojuegos que proporciona Overwolf, se encuentran los distintos datos de partida de Valorant que necesitamos[16], pudiéndose obtener nada más empieza el enfrentamiento.

Esta opción nos proporciona una manera rápida y fácil de obtener los datos necesarios, un SDK con el que agilizar el desarrollo de la aplicación, la posibilidad de monetizarla con publicidad proporcionada por Overwolf[17] y mayor visibilidad al poder publicarla en su tienda de aplicaciones[21].

No obstante, hay que tener en cuenta que decantarnos por utilizar la plataforma de desarrollo de Overwolf hará que nuestra aplicación dependa muy directamente de un tercero. Dado que la API solo funciona si la aplicación se ejecuta desde su tienda de aplicaciones, será imposible publicar la aplicación en otro sitio sin adaptarla previamente para que no use ninguna funcionalidad de Overwolf.

Por otro lado, RIOT ofrece una API oficial para Valorant[18]. Esta opción no nos haría tan dependientes de un tercero como la anterior solución. Sin embargo, como se trata de una API oficial, su uso está muy restringido y es difícil conseguir una llave[22] para su uso. Además, en la documentación[19] no se especifica que se pueda acceder a los datos de una partida en marcha, dando a entender por el contexto que sólo puede utilizarse para conseguir los datos de partidas ya terminadas.

Como última posibilidad, se podría conseguir la información que buscamos utilizando reconocimiento de patrones en el videojuego. Aunque eliminaríamos cualquier dependencia hacia terceros, su viabilidad está claramente en duda. Se trata de un reto técnico de mayor envergadura que el desarrollo del resto de la aplicación. Además, es muy probable que los recursos de procesador y memoria necesarios para su funcionamiento sean incompatibles con una ejecución fluida de Valorant, incumpliendo así uno de los requisitos no funcionales: “La carga de datos de la aplicación no deberá utilizar recursos de red ni del PC en exceso, ya que está pensada para usarse mientras se está ejecutando un videojuego”.

La elección de la mejor solución al problema se realizará basándonos en su practicidad y viabilidad.

[20] https://en.wikipedia.org/wiki/Chromium_(web_browser)#Use_in_app_frameworks
[21] https://www.overwolf.com/appstore
[22] https://rapidapi.com/blog/api-glossary/api-key/

3.2.2. Almacenamiento de datos

Dado que uno de los requisitos no funcionales definidos implica que la aplicación no puede generar pérdidas económicas (“El gasto económico necesario para el funcionamiento de la aplicación debe ser nulo”), el modo en el que vamos a almacenar los datos de la aplicación se convierte en un gran desafío.

La aplicación que vamos a desarrollar ofrecerá un amplio abanico de vídeos a los que los usuarios accederán considerables veces, por tanto, es necesario encontrar una buena solución que nos asegure que los continuados accesos al contenido no repercutan en pérdidas económicas mes tras mes.

La primera posibilidad podría ser utilizar un servicio de almacenamiento en la nube. Esto nos daría la posibilidad de tener un sistema de almacenamiento propio y la potencia de un motor de base de datos. Sin embargo, hay que tener en cuenta que el acceso y almacenamiento de vídeos no es precisamente barato considerando además que no se van a obtener grandes ingresos de la aplicación. Para hacer esta opción económicamente viable habría que optimizar al máximo el número de llamadas en la aplicación para minimizar su coste.

Por otro lado, podríamos optar por una opción conceptualmente opuesta, guardar el contenido en el propio proyecto. Aunque está claro que almacenar todo en el cliente nos quitaría toda carga económica, no hay que dejar de lado las desventajas que esto conlleva. Haciendo que la aplicación contenga una gran cantidad de archivos de vídeo se pone en riesgo el rendimiento de esta.

La aplicación tendrá que estar manejando todo el rato grandes cantidades de memoria y además, todo el filtrado deberá realizarse en el lado del cliente[23], así que es más que probable que esta solución afecte al rendimiento. Tampoco hay que olvidar que el tamaño total de la aplicación se verá drásticamente aumentado. Estos dos factores hay que tenerlos en gran consideración ya que podrían hacer que la aplicación no fuese aceptada en las tiendas de aplicaciones donde la queramos publicar.

Por último, podríamos llegar a una solución más creativa y utilizar vídeos de YouTube incrustados, eso sí, guardando en la propia aplicación la referencia al vídeo y sus características necesarias para el filtrado. Esta opción nos permitiría tener almacenados los vídeos en la nube a coste cero, aunque nos haría depender de que Google no cambie sus condiciones de uso en un futuro. Además, como ya ocurría en la opción anterior, tener los datos en la propia aplicación implicaría que todo el filtrado se realice en el lado del cliente, aunque en este caso el rendimiento no debería verse tan castigado ya que no se estarían manejando archivos de vídeo propiamente, sino enlaces a YouTube.

Como ya se daba a entender al principio del apartado, el coste económico será el criterio principal en la elección. Sin embargo, no podemos dejar de lado tampoco otros factores como el rendimiento, ya que si la aplicación se ve muy perjudicada en este aspecto perderá toda utilidad.

[23] https://es.wikipedia.org/wiki/Lado_del_cliente

3.2.3. Almacenamiento del contenido del usuario

Una de las funcionalidades que queremos implementar implica que el usuario pueda añadir sus propios vídeos y utilizarlos junto al resto de contenido de la aplicación. Aunque se trate de un desafío parecido al del apartado anterior existen ciertas diferencias que podrían hacer decantarnos por otra solución en este caso.

Las soluciones planteadas son idénticas, así que vamos a centrarnos en las diferencias respecto al apartado anterior.

Utilizar un servicio de almacenamiento en la nube para este caso agravaría el problema del precio, algo inaceptable considerando que sólo el usuario que lo ha subido va a disfrutar de ese contenido. Además, necesitaríamos moderar el contenido para controlar que el uso del servicio esté siendo honesto. Podríamos aprovechar que el contenido se almacene en la nube y que se necesite moderar para añadir también aquel contenido que nos resulte interesante para el resto de usuarios, pero ya estaríamos hablando de una funcionalidad ligeramente diferente.

Aprovechar el almacenamiento local en este caso se vuelve una opción más atractiva ya que el contenido almacenado en local va a ser seguramente menos cuantioso y afectaría por tanto en menor medida al rendimiento. Además, el tamaño de la aplicación solo se vería incrementado una vez descargada, por tanto, no nos veríamos perjudicados ante cualquier restricción de tamaño que nos pudiesen imponer para publicar la aplicación.

Como última opción, utilizar vídeos de YouTube sigue siendo una opción atractiva. Sin embargo, es posible que el vídeo que el usuario quiera añadir no se encuentre en YouTube y se vea obligado a subir el vídeo allí para poder añadirlo, lo que podría suponer un problema.

Finalmente, hay que dejar claro que implementar diferentes soluciones para el almacenamiento general y el de contenido del usuario podría aumentar el tiempo necesario para desarrollar algunas funcionalidades que dependan de la tecnología usada para almacenar el vídeo, ya que habría que adaptarlo para las dos opciones. Es un factor adicional que hay que considerar a la hora de elegir.

De igual forma que en el apartado anterior, el criterio principal de elección será económico, pero sin dejar de lado otros aspectos como en qué medida puede afectar al rendimiento de la aplicación o su practicidad.

3.3. Solución propuesta

En este apartado, se presenta la solución elegida para abordar los desafíos y requisitos previamente identificados. Describiremos en detalle en qué consiste la solución y expondremos las razones tras su elección.

La solución elegida consiste en una aplicación para Windows que ofrecerá a los usuarios de Valorant un catálogo de vídeos que le ayudarán a mejorar su rendimiento durante la partida. Para que esto sea posible es crucial que el usuario llegue al contenido que necesita con un periodo de tiempo lo suficientemente corto para no entorpecer su partida. Esto se conseguirá mediante una interfaz rápida y sencilla de utilizar y un filtrado automático del catálogo por el

mapa, agente y lado que se está jugando. De todas formas, el contenido podrá también filtrarse manualmente, dando así total libertad al usuario de consultar el resto del catálogo de vídeos dentro y fuera de partida.

Para obtener los datos de la partida necesarios para la implementación del filtrado automático, hemos optado por utilizar Overwolf SDK, este kit de desarrollo incluye el acceso a los eventos que ocurren en Valorant, de los que extraeremos la información necesaria. Se ha decidido utilizar esta herramienta ya que agilizaba enormemente el desarrollo. Además, colaborar con Overwolf ayudará a mejorar la visibilidad de la aplicación ya que nos permite publicarla en su tienda de aplicaciones. Otro punto positivo a tener en cuenta es que, una vez publicada la aplicación, podremos añadir sin apenas esfuerzo publicidad y funcionalidades premium, todo lo gestiona la propia plataforma de manera que esas dos funcionalidades que queríamos implementar pueden pasar a un segundo plano para centrarnos en el grueso de la aplicación.

El amplio catálogo de vídeos del que dispondrán los usuarios se ha decidido de momento que estará constituido por enlaces a vídeos de YouTube, ya que, por un lado, no se disponen de recursos económicos para tener un servicio de almacenamiento propio en la nube y, por otro lado, almacenar una cantidad tan grande de archivos multimedia en el cliente iba a limitar seriamente su rendimiento y aumentar enormemente su tamaño. No obstante, se ha decidido dejar preparada la lógica de la aplicación de tal manera que se pueda fácilmente migrar los vídeos y su información a una base de datos cuando los ingresos de la aplicación lo permitan.

Es probable que el contenido ofrecido no siempre esté actualizado con las últimas novedades o que no se adapte al gusto de todos los jugadores. Por esta razón, la aplicación ofrecerá la posibilidad de añadir tus propios vídeos. Se han debatido varias opciones de almacenamiento para los vídeos añadidos mediante esta funcionalidad y finalmente se ha optado por ofrecer la posibilidad al usuario de decidir entre importar un vídeo de YouTube a partir de su URL o un archivo de vídeo desde el explorador de archivos, manejando en cualquier caso toda la información en el lado cliente.

La decisión de no utilizar almacenamiento en la nube para este contenido era clara ya que hacer un desembolso económico por un espacio de almacenamiento que solo un usuario va a utilizar carece de sentido, sin hablar de que nos obligaría a verificar un uso honesto de la funcionalidad por parte de los usuarios. Por otro lado, implementar las otras dos opciones en vez de decantarse por una sola dotará de muchísima flexibilidad al usuario, que podrá decidir la opción que más le convenga.

En resumen, la solución elegida para abordar los desafíos identificados consiste en desarrollar una aplicación para Windows dirigida a usuarios de Valorant, que ofrece un catálogo de vídeos para mejorar su rendimiento en el juego. La interfaz será rápida y fácil de usar, con un filtrado automático del contenido según el mapa, agente y lado en juego, aunque también se podrá realizar un filtrado manual. La aplicación utilizará Overwolf SDK para acceder a los eventos del juego necesarios para el filtrado automático y así facilitar el desarrollo. Inicialmente, los vídeos se enlazarán desde YouTube para evitar costos de almacenamiento, pero se dejará preparada la aplicación para poder migrar a una base de datos propia en el futuro. Además, la aplicación permitirá a los usuarios agregar sus propios vídeos mediante enlaces de YouTube o archivos locales, priorizando la versatilidad de la aplicación y la reducción de costes económicos al mínimo

4. Diseño de la solución

Una vez identificados los requisitos funcionales y no funcionales decidiremos en esta sección cómo estructurar la aplicación para poder alcanzarlos. La fase de diseño se trata de una de las etapas más importantes en la creación de una aplicación ya que es donde se definen los pilares que tendrá el producto final.

Tomar las decisiones incorrectas durante esta fase puede tener consecuencias significativas, como aumentar la complejidad del desarrollo, generar dificultades en el mantenimiento a largo plazo o incluso comprometer el rendimiento y la escalabilidad del producto. Por tanto, es fundamental tener una planificación cuidadosa para conseguir un desarrollo exitoso.

4.1. Arquitectura del sistema

En este apartado, se define la estructura general de la aplicación, identificando los componentes principales y cómo interactúan entre sí. El objetivo es definir el funcionamiento de la aplicación sin entrar en exceso de detalle.

Como se muestra en la Ilustración 23, la arquitectura del Sistema está dividida en cuatro capas, en el diagrama se especifican los componentes y las tecnologías más relevantes que forman parte de cada capa.

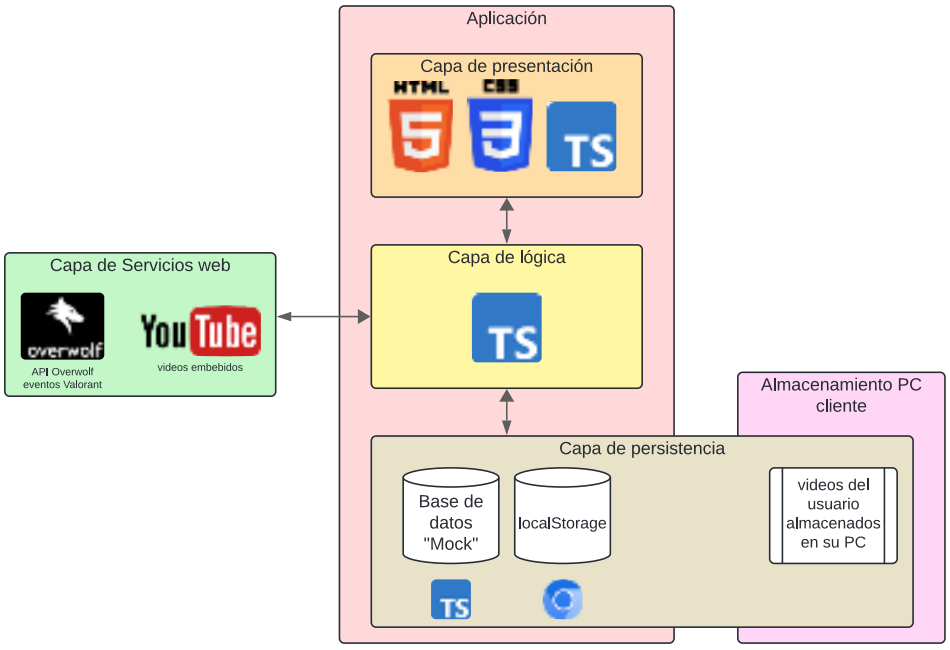


Ilustración 25: Diseño de las capas de la aplicación

4.1.1. Capa de presentación

La primera capa es la de presentación, esta se encarga de mostrar la información e interactuar con el usuario. Se desarrollará con HTML, CSS y TypeScript[20]. Dado que no existe un gran número de pantallas por las que haya que navegar, no se utilizarán *frameworks* en el *frontend* como los afamados React[21] o Angular[22]. Sin embargo, se hará uso de librerías como Boostrap[23] para agilizar el desarrollo de la interfaz.

4.1.2. Capa de lógica

La segunda capa es la de lógica, estará implementada en TypeScript y definirá toda la lógica de negocio. Aquí encontraremos todas las funciones y validaciones que facilitarán un correcto tratamiento de los datos recibidos desde las capas de persistencia y de servicios web, dotando de funcionalidad a la capa de presentación. La capa de lógica también se encargará de que las instrucciones de insertado o modificación de datos lleguen con el formato adecuado a la capa de persistencia.

En esta capa aplicaremos el patrón Fachada[24], este patrón arquitectónico nos proporcionará un acceso simple a acciones que incluyan el uso de la API de Overwolf, el acceso al almacenamiento o que interactúen con muchos subsistemas a la vez.

4.1.3. Capa de persistencia

La tercera capa es la de persistencia, responsable de almacenar los datos que utilizará la aplicación. Tendremos una clase implementada con TypeScript que almacenará todo el catálogo de vídeos que vamos a ofrecer (componente Base de datos “Mock” de la *Ilustración 25: Diseño de las capas de la aplicación*) y los devolverá en formato JSON imitando así una llamada a una API de gestión de datos. De esta manera, la aplicación quedará preparada para migrar los datos a un servidor en el futuro.

Dado que nos encontramos ante una aplicación cuyo desarrollo está basado en Chromium aprovecharemos el localStorage[25] del navegador para almacenar las referencias a los vídeos marcados como favoritos y el contenido añadido por el usuario. Tanto en la clase Typescript antes mencionada como en el localStorage cada entrada tendrá especificadas sus características principales y su enlace correspondiente a YouTube o la ruta al archivo del usuario almacenado en local.

4.1.4. Capa de servicios web

La cuarta y última capa es la de servicios web. Por un lado, a través de la API de Overwolf, esta capa nos ofrecerá el acceso a los eventos que ocurren en Valorant. Estos eventos son necesarios

para el filtrado de datos que ocurre en la capa de lógica. Por otro lado, permitirá a la capa de lógica incrustar en la capa de presentación el vídeo de YouTube referenciado.

4.2. Diseño detallado

El objetivo de este segundo apartado es desglosar la solución propuesta en un nivel de detalle mayor, definiendo cómo se implementarán los diferentes componentes de la aplicación presentados en el apartado anterior. Para ello, se dividirá el contenido en tres subapartados: Diseño de clases, Estructura de datos y uso de localStorage y Diseño de la Interfaz de Usuario.

4.2.1. Diseño de clases

A continuación, se analizarán dos diagramas que cubren, en dos niveles distintos de abstracción, el diseño de la aplicación.

4.2.1.1. Modelo de dominio

Este primer diagrama se enfoca en las clases del modelo de dominio, es decir en aquellas que representan conceptos clave para el negocio. En él se especifica la clase Video, alrededor de la cual girará toda la aplicación, y las clases necesarias para construirlas.

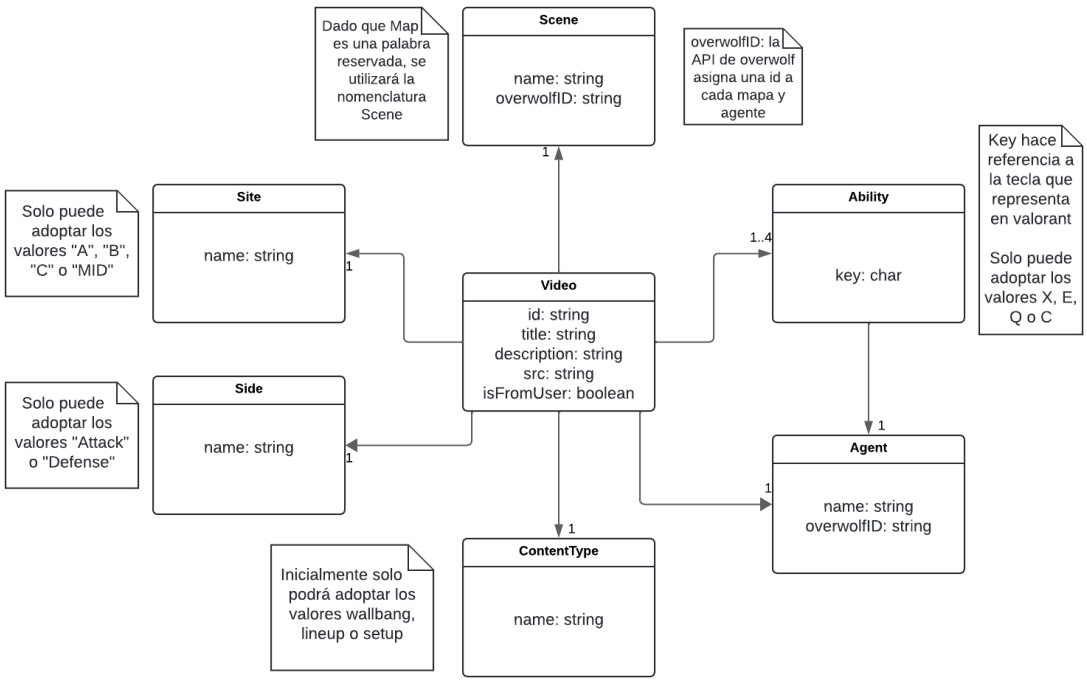


Ilustración 26: Modelo de dominio de la aplicación

Como se puede observar, la aplicación no necesita de un gran número de clases. Todo el diseño de clases se centra en especificar los atributos necesarios para filtrar los vídeos y para una correcta representación de sus características en pantalla.

4.2.1.2. Patrón de diseño Specification: Diagrama de Clases

En el segundo diagrama se detalla la implementación del patrón de diseño Specification[26], que se utilizará para el filtrado de los vídeos. En este diagrama, nos situaremos en un nivel de abstracción menor que en el anterior.

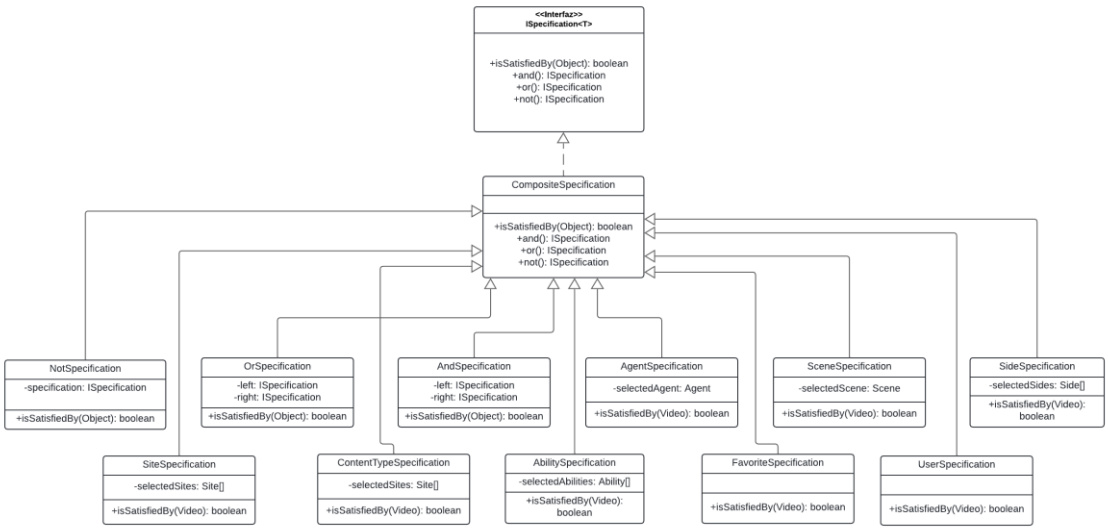


Ilustración 27: Diagrama de clases del patron Specification

Este patrón de diseño permite la creación de reglas de filtrado de manera modular y ágil, facilitando también la combinación o disyunción de diferentes criterios. Para cada atributo de un objeto Video por el que se desea filtrar, se ha definido una especificación que delimitará si el vídeo cumple con el criterio de filtrado que le llega desde la interfaz.

Adicionalmente, siguiendo la estructura típica del patrón, se han definido las clases OR, AND, y NOT. Estas clases permiten combinar las especificaciones de una manera sencilla y escalable.

Implementar el filtrado siguiendo este patrón de diseño facilitará la creación y modificación de filtros.

4.2.2. Estructura de datos y uso de localStorage

Como hemos comentado anteriormente, la aplicación no dispondrá de una base de datos real, sino que imitaremos el resultado de una llamada a una API de gestión de datos en un script que devolverá la información de un listado de vídeos en formato JSON.

Como la estructura de un JSON sigue el mismo funcionamiento clave/valor de una base de datos no relacional, es necesario definir la estructura que seguirá este JSON para asegurar la consistencia en la estructura de los datos y asegurar la fácil y completa creación de un objeto de clase Video a partir de este. Para que cada vídeo tenga una ID única, se generará un UUID[24] al momento de añadirlo.

```
id: "311b4e8b-4ed2-4173-af50-5193458bbe34"
agent: "Sova"
map: "Icebox"
side: "Attack"
title: "MID to B Recon"
description: "Recon dart from MID ramp to spot B"
ability: "E"
type: "Lineup"
src: https://youtu.be/k8cWSiQ6gOE?t=324
site: "B"
```

Ilustración 28: Estructura de almacenamiento clave/valor de la capa de persistencia.

Dado que el almacenamiento en el localStorage sigue también una estructura clave/valor, utilizaremos esta misma estructura para almacenar el contenido añadido por el usuario. Esto nos permite no tener que tratar los datos de dos maneras diferentes pese a que el origen de los datos sí lo sea.

Por último, para guardar de manera persistente aquellos vídeos que el usuario ha marcado como favoritos, utilizaremos el localStorage para guardar una lista con los id de cada vídeo que el usuario marque con la siguiente estructura.

```
{
  "favorites": [
    "f01698c5-146c-4809-84ea-5fd82c556304",
    "efc23a19-51ca-4121-8073-e9884488b4d0",
    "0bd2d445-cb32-4c73-aef8-ab0a924b186a",
    "8a2c3b69-3359-4d1d-a2a1-343095be9f17",
    "5b7fbcdd-1351-45e1-8744-0cc840e03cfb",
    "72074c60-c1b0-405f-bf33-bb3b19bbe4db",
    "50de55b7-e946-4b66-8ef1-b31ad5c38e8e"
  ]
}
```

Ilustración 29: Estructura de guardado de favoritos en el localStorage.

4.2.3. Diseño de la Interfaz de Usuario

El proceso de diseño de la interfaz ha empezado con bocetos a mano alzada que permiten reflejar la estructura básica y jugar con la disposición de los elementos en pantalla de una

[24] https://interactivechaos.com/es/wiki/identificador-unico-universal-uuid

manera rápida ya que lo que se busca una visión general de la aplicación y no dar atención al detalle. A continuación, se muestra uno de los bocetos realizados.

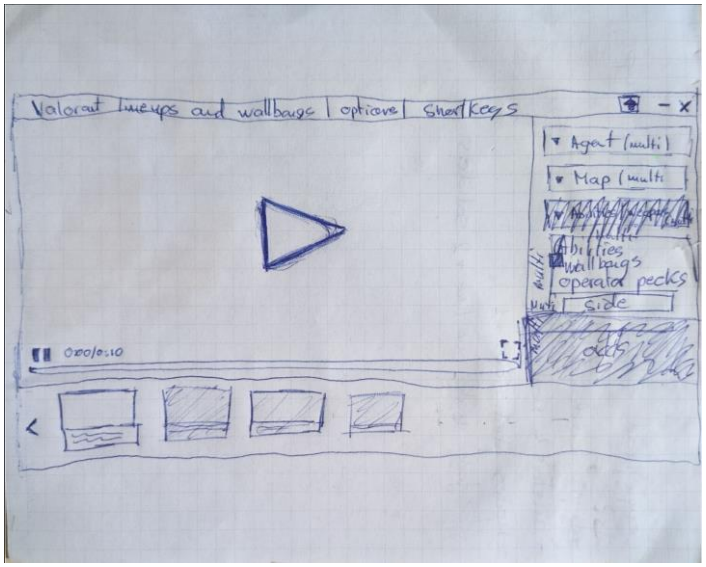


Ilustración 30: Uno de los bocetos realizados durante el proceso de diseño de la UI.

Como podemos observar en la ilustración superior, la cantidad de rectificaciones y el escaso detalle que tiene el boceto hace evidente que estamos en una fase inicial del diseño donde aún no se ha definido una estructura clara y pulida de la interfaz.

Tras varios bocetos, se ha elegido la estructura más convincente y se han definido algunas directrices que queremos que cumpla el diseño final:

- La interfaz utilizará colores agresivos, en coherencia con la temática de videojuegos.
- Evitar la predominancia de colores claros, ya que podrían fatigar la vista del jugador en largas sesiones de juego.
- El reproductor debe ser lo suficientemente grande para visualizar los vídeos sin problemas, ocupará como mínimo 2/3 de la pantalla.
- La acción de filtrar debe ser ágil, por tanto:
    - Evitar el uso de desplegables, al menos en los filtros más importantes, ya que durante el uso de la aplicación competidora Valorant Lineups, se descubrió que es un sistema de selección lento de utilizar y que empeora la experiencia de uso.
    - Utilizar pictogramas o imágenes fácilmente reconocibles.
    - Los filtros estarán ordenados de mayor a menor relevancia.
    - Las miniaturas mostrarán el mínimo de información necesaria para un correcto cribado: una *preview* del vídeo, título y descripción. De esta manera, la atención irá directamente a la información más importante, agilizando el proceso.
- Se buscará juntar la mayoría de las funcionalidades y herramientas posibles en la pantalla principal, intentando no saturarla de información.

Con estas pautas en mente, se ha utilizado Figma para crear los mockups. En el resultado conseguido ha quedado detallado el aspecto de los elementos de la interfaz, incluyendo los colores y sus cambios ante la interacción con el usuario.

A continuación, mostramos el diseño definitivo de la interfaz:

Ilustración 31: Diseño de la pantalla principal.

Ilustración 32: Diseño del apartado de filtros.

Ilustración 33: Diseño de las tarjetas del filtro de mapa.

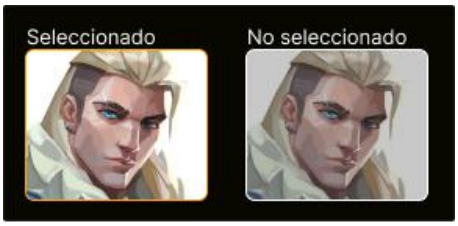


Ilustración 34: Diseño de las tarjetas del filtro de agente.

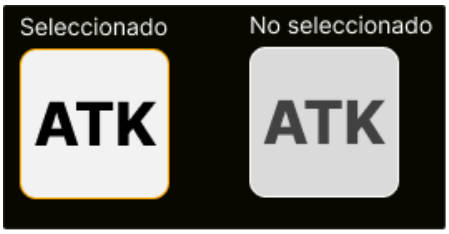


Ilustración 35: Diseño de las tarjetas del resto de filtros.

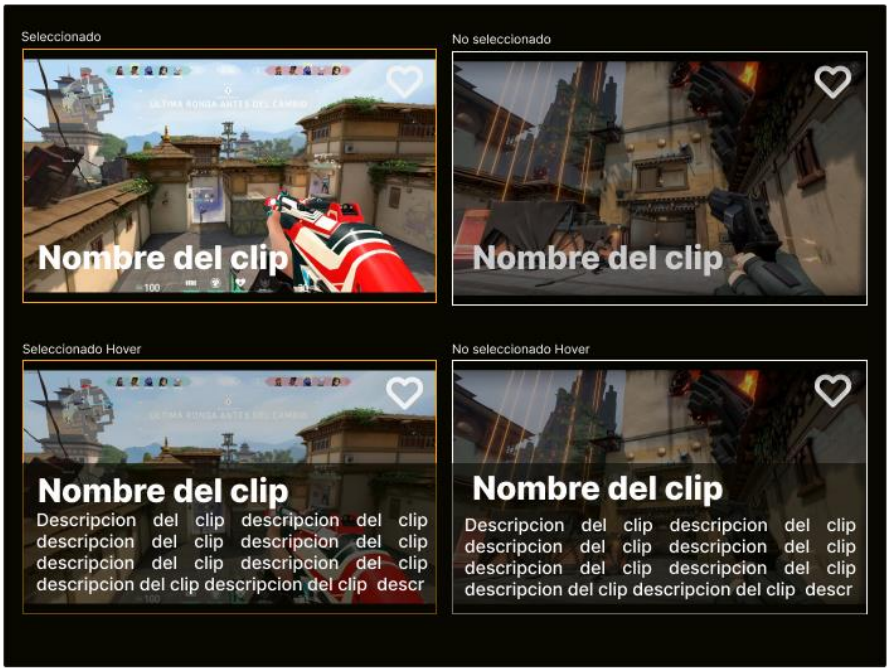


Ilustración 36: Diseño de la miniatura.

Ilustración 37: Diseño icono de favorito.

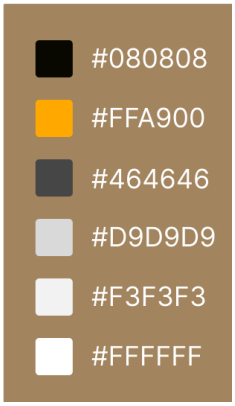


Ilustración 38: Paleta de colores.

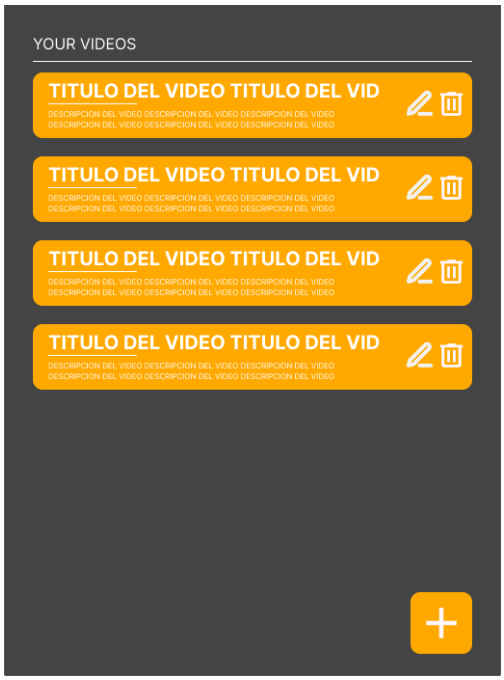


Ilustración 39: Diseño del listado de vídeos del usuario.

Ilustración 40: Diseño del formulario añadir/modificar vídeo.

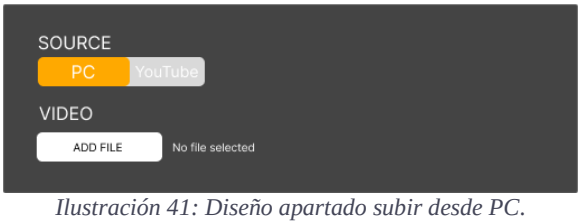


Ilustración 41: Diseño apartado subir desde PC.

Ilustración 42: Diseño apartado subir desde YouTube.

5. Tecnología utilizada

En este capítulo, se detallarán las herramientas y tecnologías empleadas a lo largo de este trabajo. Se explicarán las herramientas de diseño utilizadas para conceptualizar y modelar las ideas, las plataformas y metodologías de gestión que facilitaron la organización y seguimiento del proyecto, así como las tecnologías de desarrollo implementadas para la construcción y despliegue del producto final. La selección de estas herramientas fue clave para asegurar la eficiencia, coherencia y calidad del trabajo realizado, y su elección responde a criterios de funcionalidad, compatibilidad y necesidades específicas del proyecto.

5.1. LucidChart

LucidChart[27] es una herramienta de diagramación en línea que permite crear diagramas de flujo, organigramas, mapas mentales y otros tipos de gráficos. Se utiliza principalmente para visualizar procesos, estructuras organizacionales y relaciones entre diferentes elementos.

Dado que la herramienta está especializada en la creación de diagramas UML, LucidChart ha permitido un rápido y limpio modelado de las clases de la aplicación.

Ilustración 43: Logo de LucidChart.

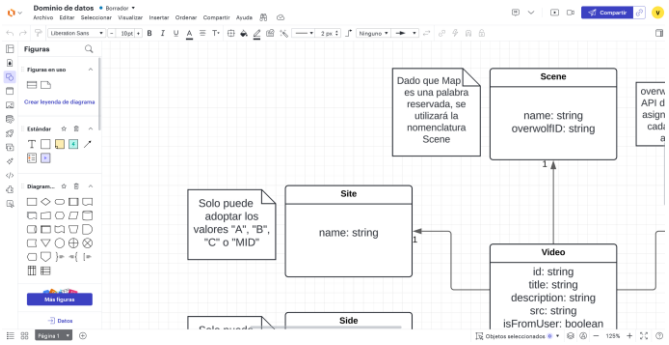


Ilustración 44: Página Web de LucidChart.

5.2. Figma

Para el diseño de las interfaces de usuario de la aplicación se necesitaba una herramienta fácil y rápida de utilizar para conseguir una propuesta de interfaz agradable sin necesidad de dedicar un tiempo excesivo. Se ha optado por utilizar Figma[28].

Figma es una herramienta de diseño colaborativo en la nube utilizada para crear interfaces de usuario, diseños gráficos y prototipos. El uso de sus herramientas más básicas es intuitivo y más que suficiente para la creación de interfaces sencillas. Esto ha permitido un diseño rápido y completo de las interfaces de la aplicación.

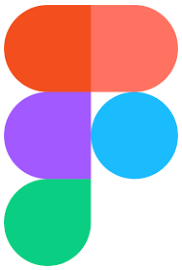


Ilustración 45: Logo de Figma.

Ilustración 46: Página Web de Figma.

5.3. Kanban Tool

Kanban Tool[29] es una aplicación visual de gestión de proyectos basada en la metodología Kanban. Tiene como objetivo facilitar la gestión y el visualizado del volumen de trabajo. Se ha utilizado su versión web para gestionar las tareas del proyecto.

Ilustración 47: Logo de Kanban Tool.

Ilustración 48: Ejemplo de tablero en Kanban Tool.

5.4. Gestión de versiones: GIT, GitHub y GitHub Desktop

El proyecto se ha alojado en un repositorio de GitHub[30], un servicio basado en la nube que permite alojar proyectos usando el sistema de control de versiones GIT[31].

GIT permite rastrear los cambios en los archivos de un proyecto a lo largo del tiempo y ofrece la posibilidad de crear diferentes ramas de trabajo y acceder al proyecto desde varios dispositivos.

Para el acceso y manipulación del repositorio se ha utilizado GitHub Desktop[32], una aplicación de escritorio que proporciona una gestión más rápida y fácil de nuestro proyecto en GitHub, pudiendo crear ramas, subir cambios o visualizar el historial de cambios entre muchas funciones.

Este conjunto de tecnologías ha permitido mantener un control de versiones, trabajar en el proyecto desde distintos ordenadores y ha protegido el código frente a posibles pérdidas de datos.

Ilustración 49: Logo de GIT.

Ilustración 50: Logo de GitHub.

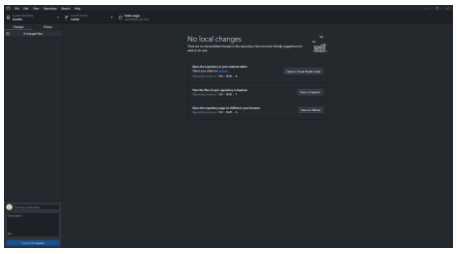


Ilustración 51: Interfaz de GitHub Desktop

5.5. Visual Studio Code

Visual Studio Code[33] es un entorno de desarrollo creado por Microsoft. La herramienta es gratuita, de código abierto y liviana, lo que la ha convertido en uno de los editores de código más populares entre los desarrolladores.

Visual Studio Code admite soporte de una gran variedad de lenguajes de programación, amplia capacidad de personalización y numerosas herramientas de manipulación de código. Estas características han permitido un cómodo desarrollo en los distintos entornos de trabajo que se han utilizado.

Ilustración 52: Logo de Visual Studio Code

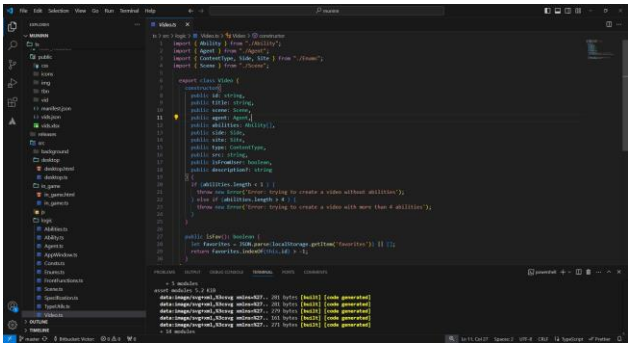


Ilustración 53: Interfaz de Visual Studio Code

5.6. Overwolf SDK

Overwolf SDK es un kit de desarrollo de software (abreviado como SDK en inglés) que permite crear aplicaciones para juegos y plataformas de streaming. El uso de este SDK incluye un *framework* basado en Chromium y proporciona acceso a la API de Overwolf que permite la obtención de los eventos que ocurren en Valorant y ofrece la posibilidad de poder publicar en la tienda de aplicaciones de Overwolf.

El uso de esta tecnología ha sido crucial ya que ha agilizado enormemente el desarrollo y ha permitido obtener los datos referentes a la partida en juego necesarios para el filtrado automático que se pretendía implementar.

Ilustración 54: Logo de Overwolf

Ilustración 55: Tienda de aplicaciones de Overwolf

5.7. HTML y CSS

Dado que se utiliza el *framework* de Overwolf que está basado en Chromium, es necesario el uso de HTML y CSS. Estas tecnologías son el pilar fundamental a la hora de crear el apartado visual de los sitios web y forman la base de todos los navegadores web desde hace ya más de tres décadas.

HTML (HyperText Markup Language) es un lenguaje de etiquetas que sirve para describir la estructura básica y el contenido de una página web, indicando al navegador los elementos que la componen como: texto, imágenes, listas, tablas, vídeos, etc.

Para poder brindar de un diseño visual a la estructura definida en HTML, se utiliza CSS (Cascading Style Sheets). Se trata de un lenguaje de diseño gráfico que se encarga de controlar distintos aspectos visuales como tamaño, color, fuente o la disposición de los elementos en pantalla.

Ilustración 56: Logo de HTML

Ilustración 57: Logo de CSS

```html
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Ilustración 58: Ejemplo de uso de HTML

```css
/* Hoja de estilos CSS */

body {
    font-family: inherit;
    background-color: lightblue;
    color: #2f2f2f;
    font-size: 1em;
    font-weight: normal;
}

.wrapper {
    border: 1px solid #999;
    border-radius: 5px;
    color: #222;
    padding: 10px;
    background: #ddd;
}
```

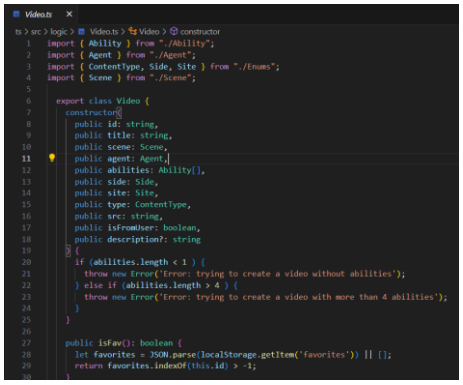
Ilustración 59: Ejemplo de uso de CSS

5.8. TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript que agrega tipos estáticos y objetos basados en clases al lenguaje, lo que ayuda a prevenir errores y mejora la legibilidad del código.

Pese a la falta de experiencia previa en este lenguaje, su estrecha relación con JavaScript, lenguaje utilizado durante el grado, me ha permitido una rápida adaptación a TypeScript, aprendiendo, a medida que avanzaba el desarrollo, cómo aprovechar al máximo las ventajas que ofrece respecto a JavaScript.

Ilustración 60: Logo de TypeScript



```
Video.ts
ts > src > logic > Video.ts > Video > constructor
import { Ability } from "./Ability";
import { Agent } from "./Agent";
import { ContentType, Side, Site } from "./Enums";
import { Scene } from "./Scene";

export class Video {
  constructor(
    public id: string,
    public title: string,
    public scene: Scene,
    public agent: Agent,
    public abilities: Ability[],
    public side: Side,
    public site: Site,
    public type: ContentType,
    public src: string,
    public isFromUser: boolean,
    public description?: string
  ) {
    if (abilities.length < 1 ) {
      throw new Error('Error: trying to create a video without abilities');
    } else if (abilities.length > 4 ) {
      throw new Error('Error: trying to create a video with more than 4 abilities');
    }
  }

  public isFav(): boolean {
    let favorites = JSON.parse(localStorage.getItem('favorites')) || [];
    return favorites.indexOf(this.id) > -1;
  }
```

Ilustración 61: Ejemplo de código en TypeScript

5.9. jQuery

jQuery[34] es una biblioteca de JavaScript que simplifica la manipulación del DOM[35] de HTML, la gestión de eventos y las animaciones, facilitando de esta manera el desarrollo de aplicaciones web interactivas.

El uso de esta biblioteca ha permitido un desarrollo más rápido del código que manipula la interfaz, consiguiendo además un resultado más compacto y legible.

Ilustración 62: Logo de jQuery

```
JS
$(".tab-list").on("click", ".tab", function(e) {
  e.preventDefault();

  $(".tab").removeClass("active");
  $(".tab-content").removeClass("show");
  $(this).addClass("active");
  $($(this).attr("href")).addClass("show");
});
```

Ilustración 63: Ejemplo de uso de jQuery

5.10. Boostrap

Boostrap es un *framework* gratuito y de código abierto para la creación de interfaces de aplicaciones web. Contiene plantillas de diseño de formularios, botones, cuadros y menús, entre otros, y clases CSS que facilitan la creación de interfaces *responsive*.

Boostrap ha permitido agilizar el desarrollo de la interfaz de la aplicación y su uso ha ayudado a conseguir un diseño consistente, facilitando la navegación y mejorando la usabilidad de la aplicación.

Ilustración 64: Logo de Boostrap

Ilustración 65: Ejemplo de uso de Boostrap

5.11. Splide.js

Splide.js[36] es una biblioteca de JavaScript ligera y flexible para crear carruseles y galerías de imágenes altamente personalizables. Su uso ha facilitado la creación del carrusel de vídeos de la pantalla principal.

Ilustración 66: Logo de Splide.js

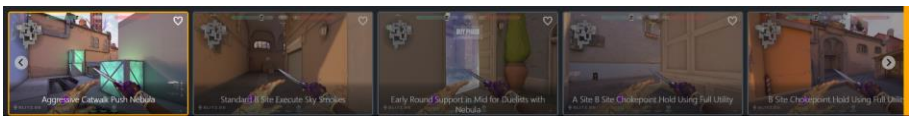


Ilustración 67: Carrusel de vídeos de la aplicación

5.12. IndexedDB

IndexedDB[37] es un sistema de almacenamiento no relacional que permite almacenar datos estructurados en el navegador o aplicación web del usuario. A diferencia del localStorage, IndexedDB permite almacenar grandes cantidades de información y estructurarlos de manera más flexible.

Tras encontrarnos con el problema de que el usuario no podía utilizar los vídeos de su explorador de archivos debido al incumplimiento del CORS, decidimos aprovechar la gran capacidad de almacenamiento que ofrece IndexedDB para guardarlos en la propia aplicación, eliminando así el problema y pudiendo mantener la funcionalidad.

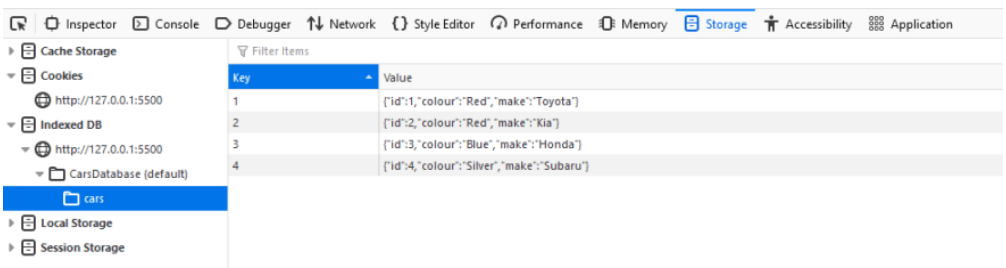


Ilustración 68: Vista de un almacén de objetos de IndexedDB

5.13. Pupeteer

Puppeteer es una herramienta desarrollada por Google que permite el control y la automatización de navegadores web. Se trata de una herramienta muy útil para tareas como pruebas automatizadas de interfaces de usuario, generación de capturas de pantalla o *scraping* de datos de páginas web.

Durante el desarrollo se ha utilizado esta herramienta para hacer *scraping* de webs de lineups, utilizando la información obtenida para poblar de datos la aplicación de manera inicial, permitiéndonos centrarnos en el desarrollo.

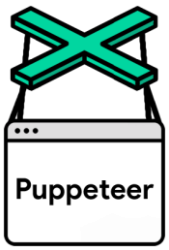


Ilustración 69: Logo de Puppeteer

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

Ilustración 70: Ejemplo de uso de Puppeteer

5.14. Jest

Jest[38] es un *framework* de pruebas unitarias que permite la creación de tests en JavaScript y TypeScript. Su uso es gratuito y puede instalarse en cualquier proyecto con los lenguajes mencionados previamente.

El uso de este *framework* ha permitido comprobar el correcto funcionamiento de cada uno de los filtros implementados en la aplicación. Poder ejecutar estas comprobaciones ha sido de especial utilidad cuando se han realizado cambios que afectaban al funcionamiento de los filtros y se necesitaba verificar que el código seguía funcionando como se esperaba.

Ilustración 71: Logo de Jest

```
it('should satisfy ContentTypeSpecification', () => {
    const contentTypeSpec = new ContentTypeSpecification([ContentType.LineUp]);
    expect(contentTypeSpec.isSatisfiedBy(mockVideo)).toBe(true);
});
```

Ilustración 72: Ejemplo de test unitario en Jest

5.15. TsUML2

TsUML2[39] es una herramienta de uso gratuito publicada en GitHub que permite la creación automática de diagramas UML a partir de un proyecto TypeScript.

Su uso ha ayudado a documentar el desarrollo de la aplicación de una manera más visual y didáctica, ya que ha permitido mostrar visualmente las clases implementadas.

Ilustración 73: Logo de TsUML2

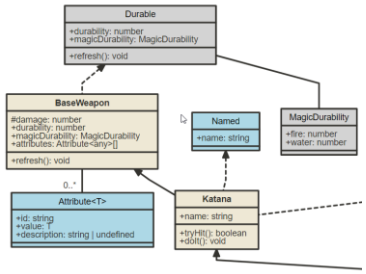


Ilustración 74: Ejemplo de diagrama generado por TsUML2

5.16. Edición de imágenes: Gimp y Paint

Gimp[40] es un editor de imágenes gratuito y de código abierto. Permite crear, editar y retocar tanto imágenes estáticas como animadas en formato GIF. Gracias a su amplio catálogo de herramientas, Gimp se sitúa como una de las mejores alternativas gratuitas a Photoshop.

Paint[41] es un editor de imágenes básico incluido en sistemas operativos Windows. Permite realizar ediciones simples como recortar, cambiar el tamaño o aplicar efectos básicos.

Ambos programas se han utilizado para pequeños retoques en las imágenes de la interfaz, como recortes, cambios de color, vectorizados, transparentar fondos o conversiones de formato.

Ilustración 75: Logo de Paint

Ilustración 76: Logo de Gimp

6. Desarrollo de la solución propuesta

En este capítulo se detalla el proceso mediante el cual se ha pasado de la propuesta a la solución final. Para ello, se explicará la implementación de cada una de las capas de la aplicación definidas en la fase de diseño, detallando sus características más relevantes y exponiendo las dificultades encontradas.

6.1. Proceso de solicitud de Overwolf

Para que funcione la API de eventos de juegos cuyo uso permite el kit de desarrollo de Overwolf, es obligatorio estar listados como desarrolladores por la plataforma. Para conseguir esto, fue necesario enviar una solicitud de desarrollo de aplicación a través de su web.

Una vez rellenada y enviada la petición, esperamos a que nos respondieran.

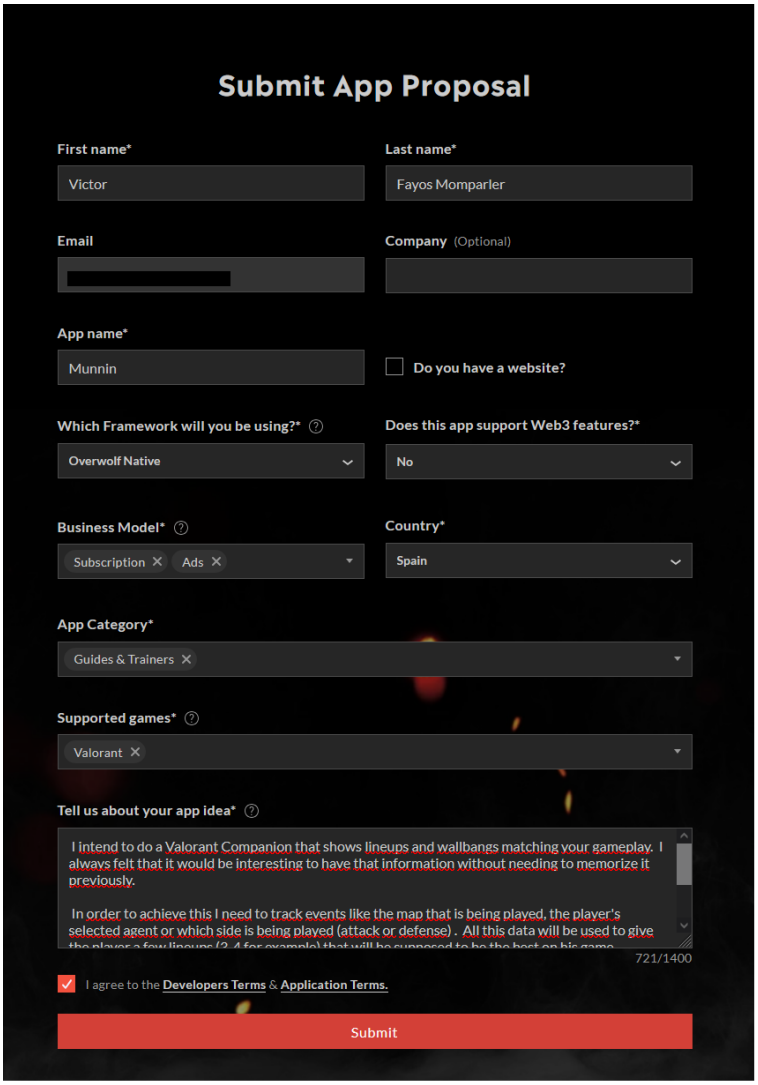
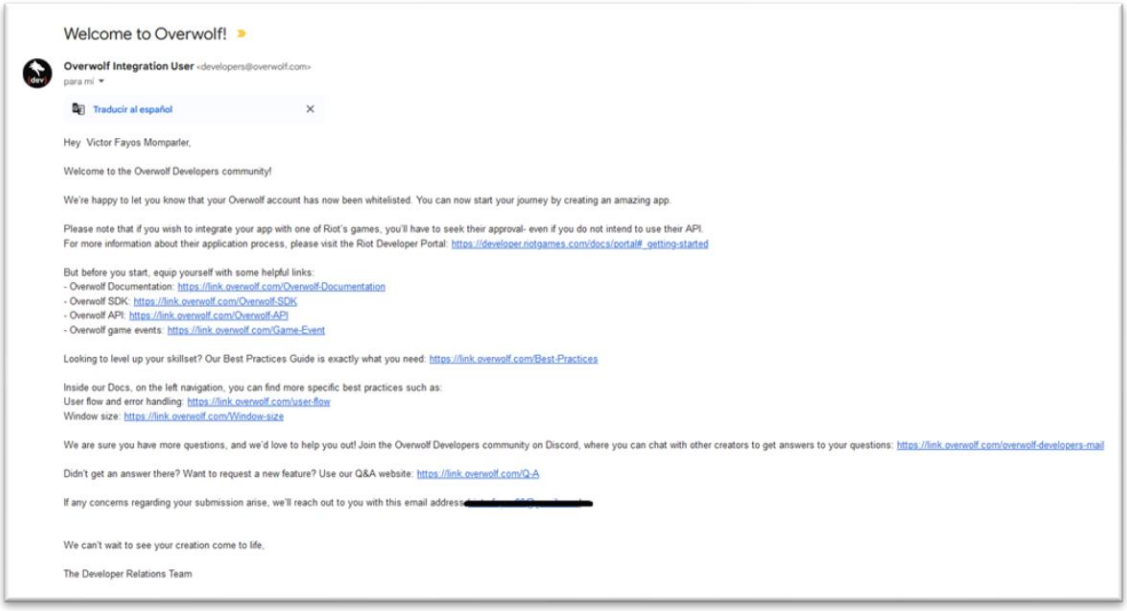


Ilustración 77: Petición de desarrollo a Overwolf

Tras unos días, la solicitud fue aprobada.



Welcome to Overwolf!

Overwolf Integration User <developers@overwolf.com>
para mí

Traducir al español

Hey Victor Fayos Momparler,

Welcome to the Overwolf Developers community!

We're happy to let you know that your Overwolf account has now been whitelisted. You can now start your journey by creating an amazing app.

Please note that if you wish to integrate your app with one of Riot's games, you'll have to seek their approval- even if you do not intend to use their API.
For more information about their application process, please visit the Riot Developer Portal: https://developer.riotgames.com/docs/portal#_getting-started

But before you start, equip yourself with some helpful links:
- Overwolf Documentation: https://link.overwolf.com/Overwolf-Documentation
- Overwolf SDK: https://link.overwolf.com/Overwolf-SDK
- Overwolf API: https://link.overwolf.com/Overwolf-API
- Overwolf game events: https://link.overwolf.com/Game-Event

Looking to level up your skillset? Our Best Practices Guide is exactly what you need: https://link.overwolf.com/Best-Practices

Inside our Docs, on the left navigation, you can find more specific best practices such as:
User flow and error handling: https://link.overwolf.com/user-flow
Window size: https://link.overwolf.com/Window-size

We are sure you have more questions, and we'd love to help you out! Join the Overwolf Developers community on Discord, where you can chat with other creators to get answers to your questions: https://link.overwolf.com/overwolf-developers-mail

Didn't get an answer there? Want to request a new feature? Use our Q&A website: https://link.overwolf.com/Q-A

If any concerns regarding your submission arise, we'll reach out to you with this email address

We can't wait to see your creation come to life,

The Developer Relations Team

Ilustración 78: Respuesta de Overwolf a la solicitud de desarrollo

Con el permiso de desarrollador ya obtenido, pudimos empezar con la implementación de la aplicación.

6.2. Estructura del directorio de archivos

A continuación, mostramos una captura completa del directorio de archivos de la aplicación. Lo hacemos con el objetivo de ofrecer una visión completa del proyecto, y de facilitar la comprensión de los apartados subsiguientes, en los que se darán detalles sobre la implementación de las distintas capas. Para ello, se han expandido y señalado aquellas carpetas que serán de especial interés en el resto del capítulo.

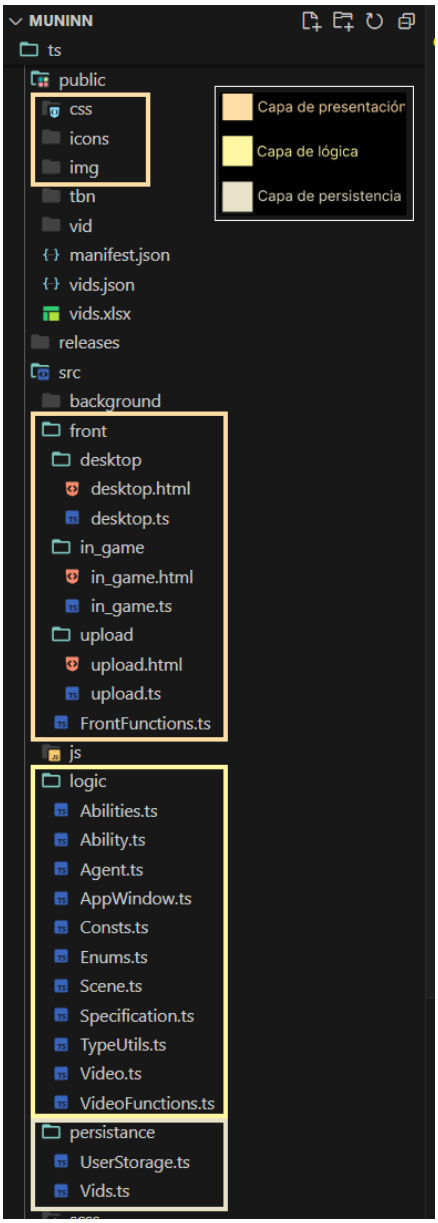


Ilustración 79: Directorio de archivos de la aplicación

6.3. Capa de presentación

En la capa de presentación se encuentra contenida la parte visual de la aplicación que ha sido desarrollada en HTML y CSS haciendo uso de librerías como Boostrap o Splide.js para agilizar el desarrollo.

6.3.1. Localización dentro del directorio y su contenido

Los archivos necesarios para construir la interfaz están resaltados en naranja en la *Ilustración 79: Directorio de archivos de la aplicación.*

Respecto a estos archivos, es necesario aclarar que Overwolf recomienda proporcionar al menos dos pantallas, una para cuando se detecte que el videojuego enlazado a nuestra aplicación está abierto (in_game) y otra cuando no lo está (desktop).

Esta estructura se puede modificar personalizando la lógica de gestión de ventanas que ofrecen, pero se ha decidido mantenerla, ya que es obligatorio mantener la estructura por defecto si se desea que la aplicación se ejecute automáticamente cuando se abre Valorant. Ahora mismo, a efectos prácticos, se trata de dos pantallas idénticas salvo por un par de nimiedades.

Por tanto, disponemos de tres archivos HTML diferentes: in_game y desktop, que como hemos comentado son idénticos y contienen la pantalla principal de la aplicación, y upload, que contiene la pantalla donde se gestiona el contenido propio del usuario.

Cada archivo HTML tiene asociado un archivo de TypeScript donde se realizan operaciones sencillas con el DOM[25] así como las llamadas necesarias a la capa de lógica. Las operaciones con el DOM de más complejidad se encuentran en el archivo FrontFunctions.ts.

Finalmente, en el directorio public encontramos los archivos CSS y los recursos gráficos que utiliza la interfaz.

6.3.2. Desarrollo de la interfaz y aspecto final

Respecto al desarrollo de la interfaz, una de las primeras tareas realizadas fue crear el esqueleto básico de la pantalla principal siguiendo el mockup. Este esqueleto delimita las zonas de la pantalla destinadas a cada funcionalidad y nos permitió ir implementando poco a poco las funcionalidades con más facilidad.

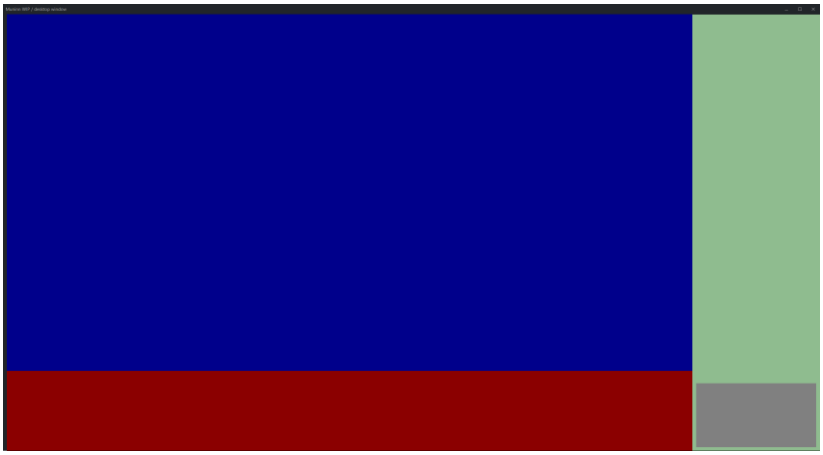


Ilustración 80: Esqueleto de la pantalla principal

Con la interfaz ya delimitada, el resto de las tareas han seguido el curso habitual de las metodologías ágiles, buscando la implementación de funcionalidades que aporten valor al producto como un conjunto de modificaciones en las capas de presentación, lógica y

[25] https://www.freecodecamp.org/espanol/news/que-es-el-dom-el-significado-del-modelo-de-objeto-de-documento-en-javascript/

persistencia. Por ejemplo, una tarea que se ha implementado ha sido añadir un filtro de tipo de contenido. Esta tarea abarca todos los cambios necesarios en las tres capas y, salvo que esté justificado por tamaño, dificultad o necesidades del equipo, esta funcionalidad no se puede dividir en una tarea por cada capa.

Las funcionalidades se han implementado siguiendo un orden de prioridad delimitado por la importancia de la funcionalidad en la aplicación final. Se empezó con el filtrado de las categorías más básicas, luego se añadió el reproductor de vídeo, posteriormente el carrusel y ya, por último, se añadieron el resto de los filtros y la funcionalidad de añadir contenido propio.

El resultado final se muestra a continuación:

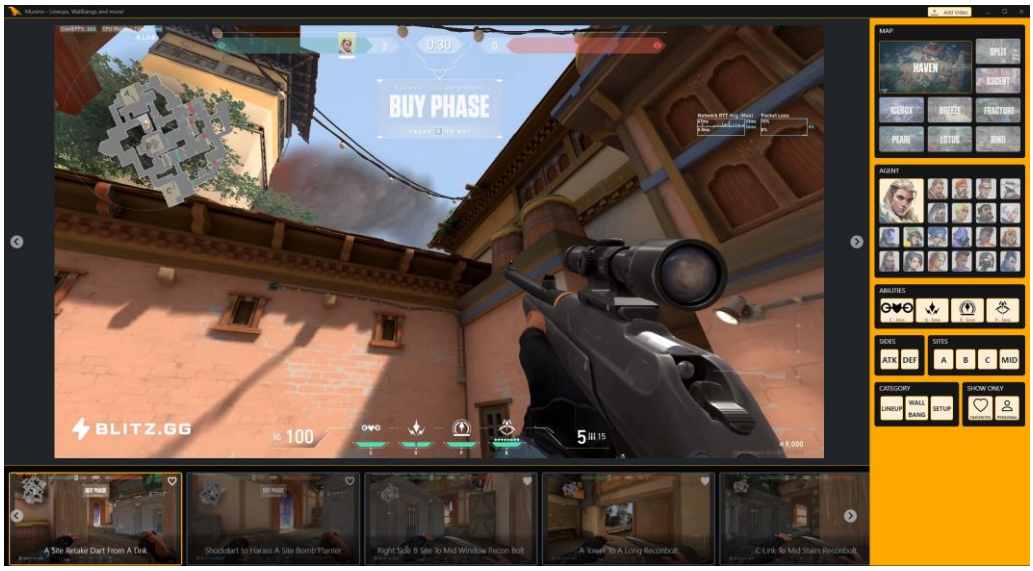


Ilustración 81: Captura de la pantalla principal de la aplicación

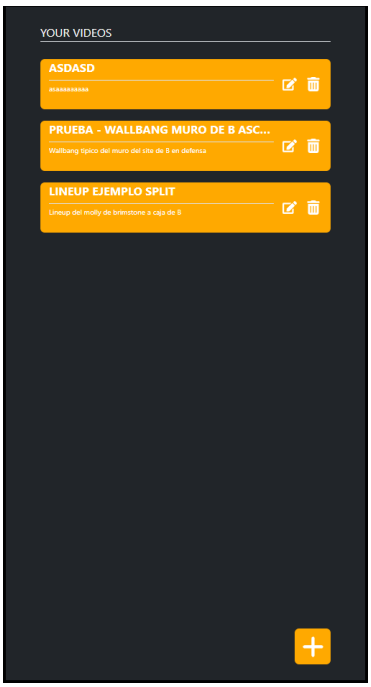


Ilustración 82: Captura de pantalla de la lista de vídeos añadidos de la aplicación

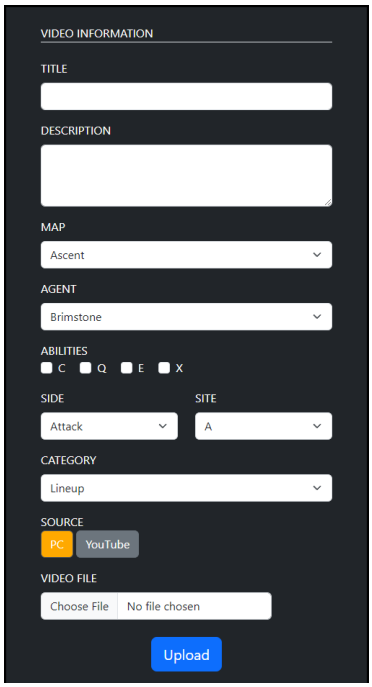


Ilustración 83: Captura de pantalla del formulario de añadir vídeo de la aplicación

Si comparamos la interfaz de usuario conseguida con los mockups mostrados al final del apartado *4.2.3 - Diseño de la Interfaz de Usuario*, queda claro que se ha sido fiel al diseño inicial. Sin embargo, durante el desarrollo se han tomado algunas libertades. La más clara se ve reflejada en la estructura de los filtros de agente y mapa.

Ilustración 84: Aspecto de los filtros de mapa y agente durante el diseño.

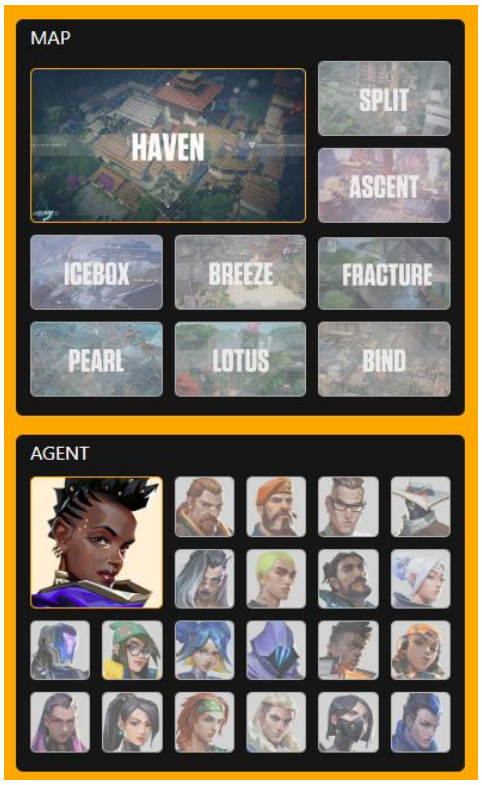


Ilustración 85: Aspecto de los filtros de mapa y agente en la aplicación

En este caso, el cambio está justificado, ya que se trata de una mejora respecto al diseño inicial que enriquecerá la experiencia de usuario. Existen otras pequeñas diferencias que han sido causadas por una mezcla entre limitaciones técnicas y practicidad en el momento del desarrollo. No obstante, no suponen una gran diferencia respecto al planteamiento inicial.

6.3.3. Patrón Fachada

Para finalizar con la capa de presentación, es interesante comentar que gracias al patrón Fachada que se ha implementado podemos hacer llamadas simples a acciones complejas desde la capa de presentación, ofreciendo un único punto de entrada a la capa de lógica y ocultando todas las operaciones que se necesitan realizar. Se puede apreciar en la siguiente llamada a la función *initialize* que se ejecuta una vez el DOM se ha cargado.

```
new AppWindow(kWindowNames.desktop);

document.addEventListener( 'DOMContentLoaded', initialize);

const openUploadButton = document.getElementById('openUploadWindowButton');
```

Ilustración 86: uso del patrón Fachada

Tras esta función se esconde la ejecución de múltiples operaciones como inicializar las variables que contienen los filtros en pantalla y sus *listeners*; la carga, filtrado y el agregado en pantalla de los vídeos; la carga de elementos necesarios para que funcione el carrusel o la carga de las miniaturas.

6.4. Capa de lógica de negocio

Paralelamente al progreso de la capa de presentación, el desarrollo de la capa de lógica se ha ido completando poco a poco a medida que se han ido implementando las diferentes funcionalidades.

Desarrollada enteramente en TypeScript, en esta capa encontramos toda la lógica necesaria para gestionar la aplicación y comunicarse con el resto de las capas en funcionamiento.

6.4.1. Estructura de archivos y su contenido

Los archivos relacionados con la implementación de la capa de lógica se representan en amarillo en la *Ilustración 79: Directorio de archivos de la aplicación*. Se resumirá brevemente el contenido de cada archivo y posteriormente se comentarán un par de implementaciones que resultan interesantes de analizar.

- **Abilities.ts:** Incluye la lógica necesaria para relacionar cada dupla de letra de habilidad y agente con su nombre en el juego y su icono.
- **Ability.ts:** Incluye la definición de la clase Ability y algunos métodos relacionados con la clase.
- **Agent.ts:** Incluye la definición de la clase Agent y algunos métodos relacionados con la clase.
- **AppWindow.ts:** Incluye la lógica necesaria para la gestión de ventanas de la aplicación.
- **Consts.ts:** Incluye las definiciones de algunas constantes como los nombres e IDs de los eventos de Valorant que necesitamos monitorear, los nombres y títulos de las distintas ventanas, etc.

- **Enums.ts:** Incluye la definición de algunos Enum[26] que se utilizan en el resto de lógica de la aplicación como Side o Site.
- **Scene.ts:** Incluye la definición de la clase Scene y algunos métodos relacionados con la clase.
- **Specification.ts:** Incluye la implementación del patrón de diseño Specification y las especificaciones creadas para el funcionamiento del filtrado de vídeos.
- **TypeUtils.ts**: Incluye la implementación de algunas operaciones relacionados con los enums y clases de la aplicación, como recuperar todos los posibles Sites o todos los Agents existentes.
- **Video.ts:** Incluye la definición de la clase Video y algunos métodos relacionados con la clase.
- **VideoFunctions.ts:** Incluye la definición de las operaciones relacionadas con los vídeos, como el filtrado o el volcado de estos desde la capa de persistencia.

Como hemos señalado previamente, vamos a detallar, dado el interés técnico de sus implementaciones, el contenido de algunos de estos archivos.

6.4.2. Definición de las clases de la aplicación

La primera implementación que vamos a analizar es la definición de las clases que nutrirán el resto de la lógica de la aplicación.

En los archivos Ability.ts, Agent.ts, Enums.ts y Scene.ts se encuentran las definiciones de las clases y enums que, en conjunto, constituirán la clase Video.

La implementación de las clases Ability, Agent y Scene goza de cierto interés ya que se ha implementado el patrón de diseño Multiton[42].

Gracias a la implementación de este patrón de diseño, la aplicación sólo maneja, como máximo, una instancia de cada objeto diferente que se cree desde estas clases. Esto se consigue mediante el método getInstance, que controla las diferentes instancias de los objetos de la clase mediante un Map, como se muestra a continuación:

```
export class Scene {
  private static instances: Map<string, Scene> = new Map();
  private constructor(public name?: SceneName, public overwolfID?: ScenesOverwolfID) {}

  public static getInstance(name?: SceneName, overwolfID?: ScenesOverwolfID): Scene {
    const instanceKey: string = `${name}-${overwolfID}`;

    if (!Scene.instances.has(instanceKey)) {
      Scene.instances.set(instanceKey, new Scene(name, overwolfID));
    }

    return Scene.instances.get(instanceKey)!;
  }
```

Ilustración 87: Implementación de la clase Scene. Método getInstance.

[26] https://www.typescriptlang.org/docs/handbook/enums.html

Aunque no se trata de algo estrictamente necesario, esta implementación liberará a la aplicación de parte de su uso de memoria y además, evita posibles erorres en las comparaciones. Esto es debido a que, en caso de que dos Scene sean idénticas, las dos partes de la comparación harían referencia al mismo objeto en memoria, así que solo necesitaría verificarse la igualdad del objeto y no la de sus atributos internos. Esta característica se puede apreciar en la siguiente ilustración:

```
constructor(private selectedScene: Scene) {super();}

isSatisfiedBy(video: Video): boolean {
  /**
   * Si no tuviesemos instancias únicas en el objeto Scene
   * necesitaríamos hacer lo siguiente:
   *
   * video.scene.name === this.selectedScene.name
   *  && video.scene.overwolfID === this.selectedScene.overwolfID
   */
  return video.scene == this.selectedScene;
}
```

Ilustración 88: Uso de una comparación directa de objetos gracias al patron Multiton.

Otra característica interesante es que los tipos de atributos Side, Site y ContentType que utiliza la clase Video se han definido como Enum. Esto hace que se limite las opciones que puede tomar un atributo de ese tipo a las definidas en su implementación.

```
export enum Side {
  Defense = "Defense",
  Attack = "Attack",
}
```

Ilustración 89: Implementacióin del enum Side.

Además, estos Enum se han implementado de tal manera que se puede acceder a sus valores desde cualquier parte del código para evitar comparaciones con literales cuando se está tratando con atributos del mismo tipo.

Por ejemplo, un objeto de tipo Side puede tener como valor “Attack” o “Defense”. Si en cualquier punto del código queremos comprobar que el Video es sobre el lado atacante, podremos hacer la siguiente comparación: *video.side === Side.Attack*. De esta manera se evitan la comparación directa con literales *(video.side === “Attack”)* que, además del riesgo constante de cometer un error tipográfico o *typo* al escribir el literal, necesitan ser cambiados en todos lados en caso de modificar la clase.

```
isSatisfiedBy(video: Video): boolean {
  return video.type === ContentType.WallBang || video.agent.ov
}
```

Ilustración 90: Uso de los posibles valores de un Enum como constante.

Por último, analizaremos el contenido del archivo Video.ts. En él nos encontramos con la definición de la clase Video, en la que se incluyen todos los atributos necesarios tanto para hacer funcionar correctamente los distintos tipos de filtrado que se han definido como para su correcto visionado en el carrusel y el reproductor de la capa de presentación. A continuación, se muestra su implementación.

```
export class Video {
  constructor(
    public id: string,
    public title: string,
    public scene: Scene,
    public agent: Agent,
    public abilities: Ability[],
    public side: Side,
    public site: Site,
    public type: ContentType,
    public src: string,
    public isFromUser: boolean,
    public description?: string
  ) {
    if (abilities.length < 1 ) {
      throw new Error('Error: trying to create a video without abilities');
    } else if (abilities.length > 4 ) {
      throw new Error('Error: trying to create a video with more than 4 abilities');
    }
  }

  public isFav(): boolean {
    let favorites = JSON.parse(localStorage.getItem('favorites')) || [];
    return favorites.indexOf(this.id) > -1;
  }
}
```

Ilustración 91: implementación de la clase Video.

6.4.3. Patrón Specification

El segundo y último caso de interés es la implementación del patrón Specification, descrito en la fase de diseño. Su implementación, situada en el archivo Specification.ts, se muestra a continuación:

```
export interface ISpecification {
  isSatisfiedBy(object: unknown): boolean;
  and(other: ISpecification): ISpecification;
  or(other: ISpecification): ISpecification;
  not(): ISpecification;
}

export abstract class CompositeSpecification implements ISpecification {
  abstract isSatisfiedBy(object: unknown): boolean;

  and(other: ISpecification): ISpecification {
    return new AndSpecification(this, other);
  }

  or(other: ISpecification): ISpecification {
    return new OrSpecification(this, other);
  }

  not(): ISpecification {
    return new NotSpecification(this);
  }
}

  export class AndSpecification extends CompositeSpecification {
    constructor(private left: ISpecification, private right: ISpecification) {super();}

    isSatisfiedBy(object: unknown): boolean {
        return this.left.isSatisfiedBy(object) && this.right.isSatisfiedBy(object);
    }
  }

  export class OrSpecification extends CompositeSpecification {
    constructor(private left: ISpecification, private right: ISpecification) {super();}

    isSatisfiedBy(object: unknown): boolean {
        return this.left.isSatisfiedBy(object) || this.right.isSatisfiedBy(object);
    }
  }

  export class NotSpecification extends CompositeSpecification {
    constructor(private specification: ISpecification) {super();}

    isSatisfiedBy(object: unknown): boolean {
        return !this.specification.isSatisfiedBy(object);
    }
  }
```

Ilustración 92: Implementación de la estructura básica del patron Specification.

```
export class SideSpecification extends CompositeSpecification {
  constructor(private selectedSides: Side[]) {super();}

  isSatisfiedBy(video: Video): boolean {
    return  this.selectedSides.some(selectedSide => selectedSide == video.side);
  }
}
```

Ilustración 93. Implementación de una de las especificaciones creadas.

Como se puede apreciar, una vez ya definida la estructura principal (*Ilustración 92: Implementación de la estructura básica del patron Specification.*) la creación de criterios de filtrado adicionales (especificaciones) es muy sencilla, necesitando definir solamente la condición que debe cumplir el vídeo en cuestión (*Ilustración 93. Implementación de una de las especificaciones creadas.*).

Además, esta implementación permite que el filtrado de vídeos sea sencillo, legible y fácilmente escalable, como se puede apreciar en la función *filterVideos* situada en el archivo VideoFunctions.ts:

```
const sceneSpec: SceneSpecification = new SceneSpecification(selectedMap);
const agentSpec: AgentSpecification = new AgentSpecification(selectedAgent);
const sideSpec: SideSpecification = new SideSpecification(selectedSides);
const siteSpec: SiteSpecification = new SiteSpecification(selectedSites);
const abilitySpec: AbilitySpecification = new AbilitySpecification(selectedAbilities);
const contentTypeSpec: ContentTypeSpecification = new ContentTypeSpecification(selectedContentTypes);
const favoriteSpec: FavoriteSpecification = new FavoriteSpecification();
const userContentSpec: UserSpecification = new UserSpecification();

const specs: ISpecification[] = [sceneSpec, agentSpec];

if (isSideFiltered) {
    specs.push(sideSpec);
}
if (isSiteFiltered) {
    specs.push(siteSpec);
}
if (isAbilityFiltered) {
    specs.push(abilitySpec);
}
if (isTypeFiltered) {
    specs.push(contentTypeSpec);
}
if (isFavoriteFiltered) {
    specs.push(favoriteSpec);
}
if (isUserContentFiltered) {
    specs.push(userContentSpec);
}

const combinedSpec = specs.reduce((acc, spec) => acc.and(spec));

filteredVideos = allVideos.filter(video => combinedSpec.isSatisfiedBy(video));
```

Ilustración 94: Uso de las especificaciones para el filtrado de vídeos .

6.4.4. Diagrama de clases resultante

Utilizando la herramienta TsUML2, se ha autogenerado un diagrama de clases a partir de la carpeta logic del proyecto. Este se mostrará a continuación, con el fin de contribuir a un mejor entendimiento por parte del lector de la lógica que se ha implementado.

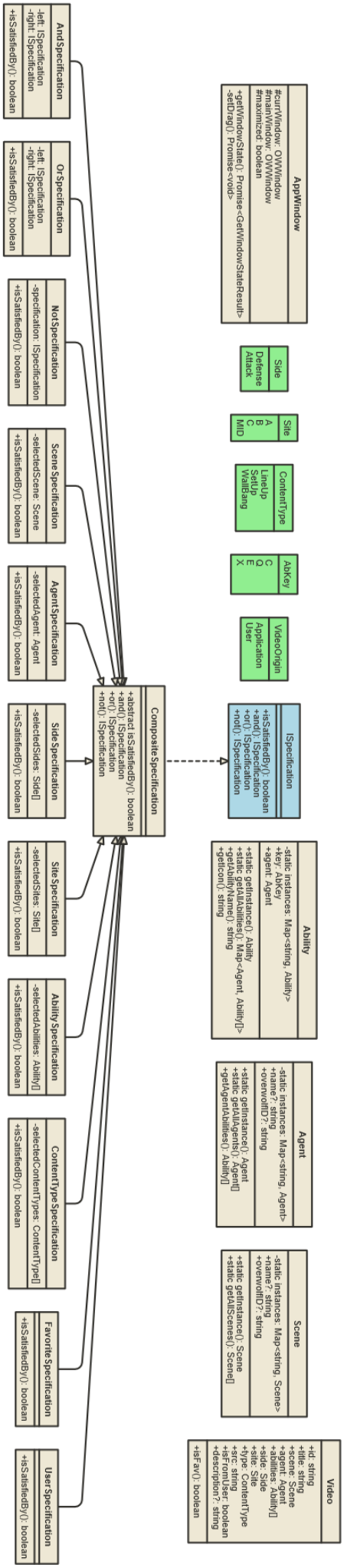


Ilustración 95: Diagrama de clases de la aplicación

6.4.5. Refactorizaciones

Por último, como ya se menciona previamente en el apartado *5.8 – Typescript,* es interesante destacar que el desarrollo de esta capa se ha realizado en Typescript en su totalidad, lenguaje del que no se tenía experiencia previa, aunque sí se estaba familiarizado con su lenguaje "hermano" JavaScript.

Durante el avance del desarrollo se ha ido aprendiendo sobre las particularidades de este lenguaje. Esto ha provocado la necesidad de refactorizar algunas de las funcionalidades implementadas, ya que no estaban aprovechando o no utilizaban correctamente las ventajas que ofrece este lenguaje frente a JavaScript, como el tipado estático, la creación de clases, interfaces, tipos y enums o el manejo de nulos (*null safety*)[27].

Aunque estas refactorizaciones han provocado cierto retraso en el desarrollo, el esfuerzo realizado por conseguir un código de mejor calidad se verá recompensado a futuro, ya que se reducirá el número de errores en ejecución y el mantenimiento de la aplicación será más sencillo.

6.5. Capa de persistencia

Finalmente, se comentará el desarrollo de la capa de persistencia, una capa cuya implementación ha sido algo peculiar ya que el almacenamiento de datos se encuentra dentro de la propia aplicación, algo no siempre habitual. Los archivos encargados de manejar la capa de persistencia se encuentran en la carpeta persistance del proyecto, como se ve reflejado en gris en la *Ilustración 77: Petición de desarrollo a Overwolf*.

6.5.1. Creación y poblado de Vids.ts

Completada la implementación de los principales filtros, el reproductor y el carrusel nos vimos necesitados de llenar la aplicación de contenido para poder probar el correcto funcionamiento de la aplicación.

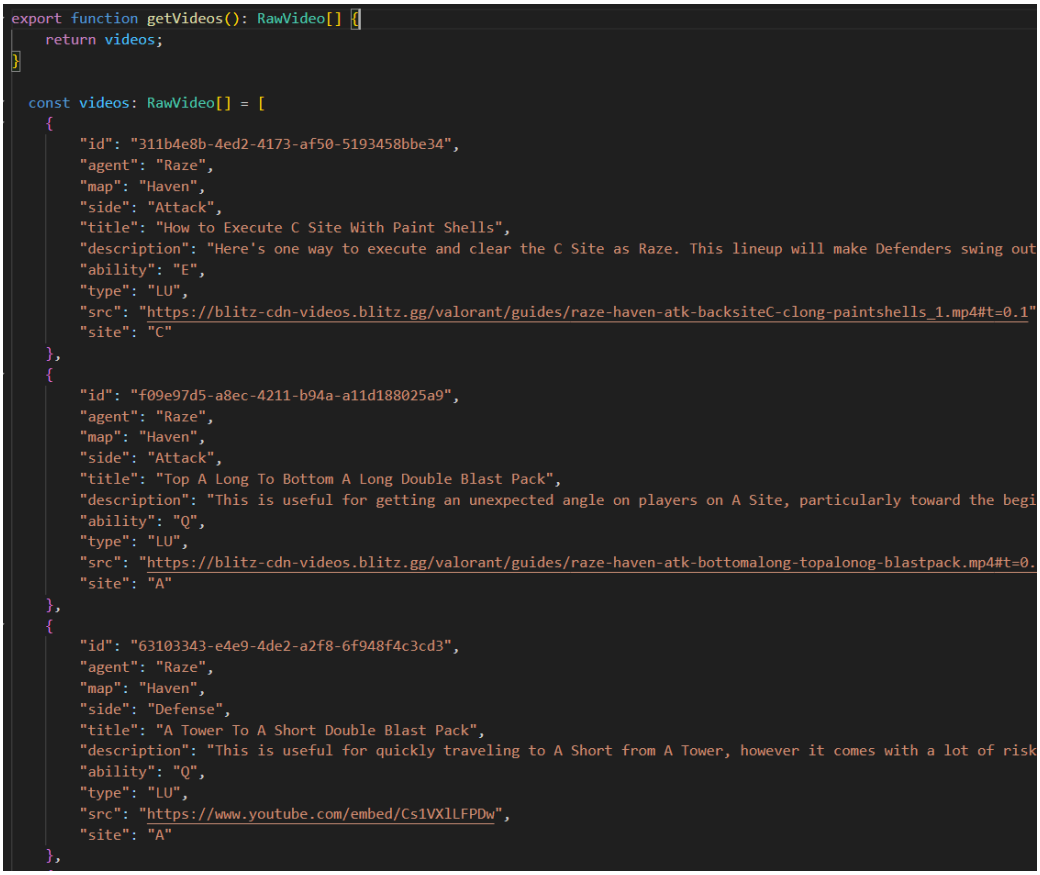
Con esta necesidad en mente, se creó el archivo Vids.ts. Este archivo se encarga de almacenar en formato JSON los datos de los vídeos. Para conseguir un correcto encaje con el resto de la aplicación, la capa de lógica recupera este contenido mediante una llamada a la función *getVideos* (parte superior de la *Ilustración 96: Captura de parte del contenido de Vids.ts*.) y se encarga de convertir el JSON en un objeto de la clase Video expuesta en el apartado *6.4.2 – Definición de las clases de la aplicación*. De esta manera, se imita el comportamiento de una API de almacenamiento de datos y dejamos preparada así nuestra aplicación para añadirle la llamada a una base de datos en un futuro.

[27] https://armanco.medium.com/null-undefined-safety-in-typescript-165fb4977194

Sin embargo, aún quedaba pendiente rellenar de datos el archivo. Buscando una solución rápida al problema, se creó un *scraper* utilizando Puppeteer, con el que, a base de prueba y error, alcanzamos nuestro cometido.

Haciendo *scraping* sobre la web de Strats.gg, conseguimos los datos de un centenar de vídeos de lineups. Gracias a la lógica definida en el *scraper*, los datos se sustrayeron cumpliedo ya la estructura JSON especificada en el apartado *4.2.2 – Estructura de datos y uso de localStorage*, así que, una vez añadidos los datos obtenidos a Vids.ts, la aplicación ya estaba lista para poder probarse.

A continuación, se muestra una captura del archivo Vids.ts una vez poblado con los datos obtenidos mediante *scraping*.



```
export function getVideos(): RawVideo[] {
    return videos;
}

const videos: RawVideo[] = [
    {
        "id": "311b4e8b-4ed2-4173-af50-5193458bbe34",
        "agent": "Raze",
        "map": "Haven",
        "side": "Attack",
        "title": "How to Execute C Site With Paint Shells",
        "description": "Here's one way to execute and clear the C Site as Raze. This lineup will make Defenders swing out
        "ability": "E",
        "type": "LU",
        "src": "https://blitz-cdn-videos.blitz.gg/valorant/guides/raze-haven-atk-backsiteC-clong-paintshells_1.mp4#t=0.1"
        "site": "C"
    },
    {
        "id": "f09e97d5-a8ec-4211-b94a-a11d188025a9",
        "agent": "Raze",
        "map": "Haven",
        "side": "Attack",
        "title": "Top A Long To Bottom A Long Double Blast Pack",
        "description": "This is useful for getting an unexpected angle on players on A Site, particularly toward the begi
        "ability": "Q",
        "type": "LU",
        "src": "https://blitz-cdn-videos.blitz.gg/valorant/guides/raze-haven-atk-bottomalong-topalonog-blastpack.mp4#t=0.
        "site": "A"
    },
    {
        "id": "63103343-e4e9-4de2-a2f8-6f948f4c3cd3",
        "agent": "Raze",
        "map": "Haven",
        "side": "Defense",
        "title": "A Tower To A Short Double Blast Pack",
        "description": "This is useful for quickly traveling to A Short from A Tower, however it comes with a lot of risk
        "ability": "Q",
        "type": "LU",
        "src": "https://www.youtube.com/embed/Cs1VXlLFPDw",
        "site": "A"
    },
```

Ilustración 96: Captura de parte del contenido de Vids.ts.

De esta manera pudimos, sin perder demasiado tiempo, seguir con el desarrollo de la aplicación y posponer el rellenado auténtico de datos para el final, algo importante ya que el contenido del videojuego está en constante cambio e interesa hacer el poblado lo más cercano posible a la fecha de publicación.

6.5.2. Almacenamiento interno y problemas con el CORS

Respecto a las funcionalidades que hacen uso del almacenamiento interno para guardar la información referente al usuario, la primera en implementarse fue la de marcar vídeos como

favoritos. Para completar esta funcionalidad se pudo, sin demasiado problema, almacenar las referencias a los vídeos marcados como favoritos en el localStorage tal y como se planificó en el apartado *4.2.2 – Estructura de datos y uso del localStorage.*

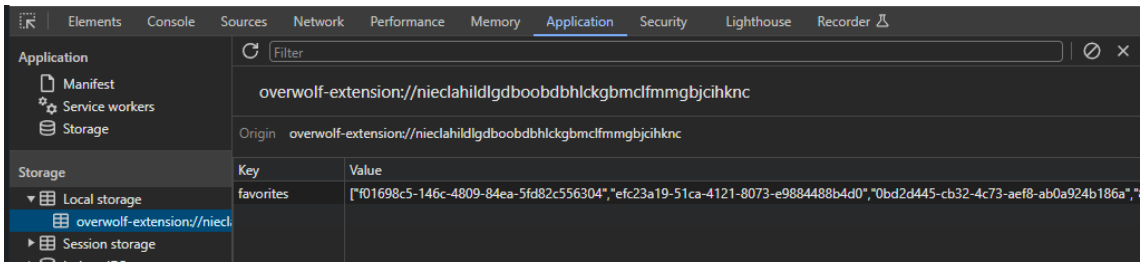


Ilustración 97: Referencias a los vídeos marcados como favoritos almacenados en el localStorage

Sin embargo, cuando intentamos implementar la funcionalidad que permite al usuario añadir sus propios vídeos nos encontramos con el siguiente problema: una vez guardado en el localStorage el vídeo añadido mediante la funcionalidad con su ruta apuntando al explorador de archivos del ordenador, si se cerraba la aplicación y se volvía a iniciar, esta no dejaba reproducir el archivo ya que, por un error CORS[43][28], no se podía acceder al contenido de un archivo cuya ruta se sitúa fuera de los dominios permitidos por la aplicación.

Para nuestro caso en particular, debería ser suficiente con añadir la propiedad *"allow_local_file_access": true* en el archivo *manifest*[29] tal y como indica la documentación de Overwolf[44], sin embargo, esta acción no solucionó el problema.

Buscando resolver el problema de una manera distinta, se optó por no guardar una referencia al archivo de vídeo seleccionado por el usuario sino, por el contrario, almacenar directamente el contenido del archivo en el almacenamiento interno de la aplicación.

Para conseguir esto, no podíamos hacer uso del localStorage, ya que sólo permite almacenar cadenas de texto hasta un máximo de 50Mb. Era necesario utilizar IndexedDB, tal y como recomienda Overwolf en su documentación para estos casos[45]. Los principales motivos son que, por un lado, IndexedDB no establece un límite de almacenamiento, y, por otro lado, esta tecnología permite el guardado de archivos multimedia en formato Blob[46].

La incorporación de IndexedDB a la aplicación dio verdaderos quebraderos de cabeza, ya que, pese a haber basado el código en los ejemplos proporcionados por distintas páginas referentes en el ámbito del desarrollo web como Medium[30], Web.dev[31] o Mozilla Developers[32], este nunca terminaba de funcionar como debía.

Fueron varios los fallos a los que tuvimos que enfrentarnos: errores al crear la tabla, al intentar acceder a la tabla en una sesión diferente a la que se ha creado, duplicado de tablas con nombre idéntico o que la inserción de datos funcionaba en algunos casos y en otros no sin razón

[28] CORS es un mecanismo de seguridad presente en todas las aplicaciones web, como lo es esta.
[29] El archivo manifest es un documento en formato JSON que contiene parámetros de inicio y valores predterminados necesarios para la ejecución de una aplicación web.
[30] https://medium.com/
[31] https://web.dev/
[32] https://developer.mozilla.org/es/

aparente. Tras un largo proceso de investigación sobre la tecnología y *debugging* se consiguió finalmente solventar todos estos errores.

Tras todos estos contratiempos, pudimos finalmente terminar la funcionalidad que permite al usuario añadir sus propios vídeos. A continuación, se muestra cómo queda la tabla creada con IndexedDB tras añadir un vídeo mediante la funcionalidad.

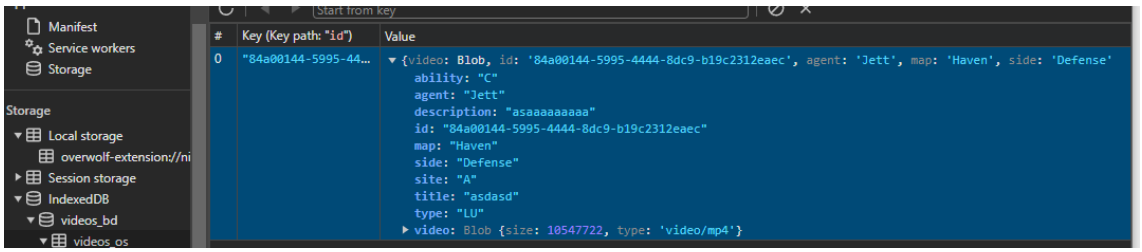


Ilustración 98: Vídeo guardado mediante el uso de IndexedDB

Antes de dar por finalizado este apartado, es necesario indicar, ya que no se ha tenido ocasión de hacerlo previamente, que toda la lógica referente al manejo de IndexedDB y al control e inserción de favoritos en el localStorage se encuentra en el archivo UserStorage.ts.

7. Pruebas

Las pruebas son fundamentales en el desarrollo de una aplicación para asegurar el cumplimiento de los requisitos fijados y su correcto funcionamiento. Si no se realizan pruebas, es altamente probable que el código contenga errores significativos.

A lo largo de este capítulo se detallarán las pruebas realizadas en nuestra aplicación, en ellas se buscará verificar el funcionamiento de esta.

7.1. Limitaciones del entorno de desarrollo

Antes que nada, es necesario exponer que algunas limitaciones que nos impone el entorno de desarrollo nos han perjudicado a la hora de testear.

La limitación de uso que Overwolf impone para su API hace imposible poder ejecutar la aplicación sin tener permiso de desarrollador (véase *6.1 – Proceso de solicitud de Overwolf*), lo que imposibilita su uso por terceras personas hasta una vez publicada la aplicación en la tienda.

De esta manera, el peso de verificar el correcto funcionamiento de la aplicación recae enteramente sobre el desarrollador, algo poco recomendable, ya que las pruebas tienden a ser sesgadas a causa del conocimiento que se tiene sobre el código y a ser insuficientes por una mezcla de limitaciones de tiempo y exceso de confianza sobre las funcionalidades implementadas.

Pese a todo, se ha hecho especial esfuerzo en verificar el correcto comportamiento de las distintas funcionalidades implementadas en la aplicación. Durante el proceso de desarrollo, cada funcionalidad terminada fue probada antes de seguir con la implementación de la siguiente, se crearon tests unitarios para verificar el correcto filtrado de los vídeos y finalmente se probó toda la aplicación en conjunto una vez terminada.

7.2. Pruebas unitarias con Jest

Al añadir las primeras especificaciones para el filtrado del mapa, agente y lado, se vio necesaria la creación de tests unitarios que ayudasen a verificar el correcto funcionamiento de las especificaciones existentes y futuras. De esta manera, se podría asegurar de manera objetiva que cada uno de los criterios de filtrado especificados es correcto, ya sea al final de su implementación o tras acometer modificaciones que afecten en su comportamiento.

Para las pruebas unitarias, utilizamos JEST, un framework de testing compatible con TypeScrpit. Se definieron distintos casos que debían cumplimentarse: para cada especificación se definió mínimo un caso en el que debe cumplirse y otro en el que no. Adicionalmente, se definieron casos donde se prueba la correcta unión y disyunción entre distintas especificaciones.

A continuación, se muestra una captura del inicio del archivo creado para las pruebas unitarias, en el que se puede ver el primer test unitario definido, donde se verifica que un vídeo satisface la especificación de su mismo mapa.

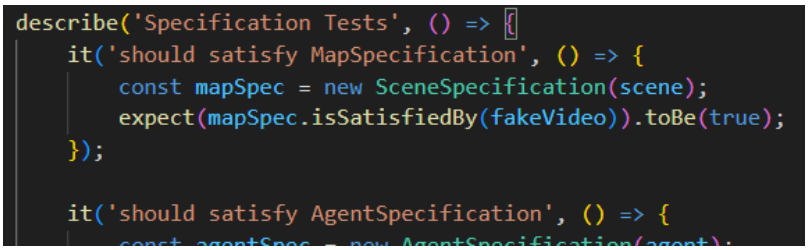


Ilustración 99: Ejemplo de prueba unitaria

7.3. Pruebas finales

Una vez terminado el desarrollo, se establecieron una serie de pruebas de uso para asegurar el correcto funcionamiento de la aplicación en conjunto. Las pruebas realizadas fueron las siguientes:

- Pantalla principal
  - Botones de minimizar, expandir y cerrar ventana.
- Reproductor de vídeo
  - Operaciones de control: pausar, avanzar, retroceder y reanudar.
  - Reproducción automática de los vídeos.
- Carrusel
  - Navegación fluida del carrusel.
  - Visionado de las miniaturas.
  - Cambio de vídeo seleccionado.
  - Actualización del contenido tras filtrar.
- Favoritos
  - Marcar/desmarcar vídeos como favorito.
  - Los cambios realizados persisten al cerrar y volver a ejecutar la aplicación.
- Filtrado
  - Filtrado de cada categoría por separado.
  - Correcta unión de distintas categorías de filtrado.
  - Filtrado automático al entrar en partida.
- Añadir contenido
  - Añadir un vídeo a la aplicación.
  - Visualización correcta del vídeo añadido.
  - Correcta persistencia de los datos al volver a iniciar la aplicación.
  - Modificar los datos de un vídeo añadido.
  - Listado correcto de todos los vídeos añadidos.

8. Conclusiones

La sección de conclusiones es fundamental en cualquier trabajo académico, ya que nos permite ver en retrospectiva el trabajo realizado y nos impulsa a reflexionar sobre este. Nos ofrece la oportunidad de evaluar el cumplimiento de los objetivos planteados, los errores cometidos durante el desarrollo y el crecimiento tanto profesional como personal logrado a lo largo del proyecto.

En términos generales, se ha conseguido satisfacer el objetivo principal y los objetivos específicos han sido cumplimentados. Sin embargo, aún es pronto para evaluar el éxito en la meta de convertir la aplicación en una fuente de ingresos pasivos, ya que este objetivo va más allá del desarrollo inicial.

El resultado es sin duda satisfactorio: se ha resuelto el problema planteado; el diseño de la interfaz de usuario de la aplicación, teniendo en cuenta que no se trata del ámbito de estudio, es llamativo y agradable a la vista; su uso es intuitivo y su rendimiento, pese a ser mejorable, es más que aceptable si se tiene en cuenta el marco educativo en el que nos encontramos.

8.1. Desafíos enfrentados y errores cometidos

El desarrollo no ha estado exento de problemas, aunque no ha sido una continua fuente de inconvenientes. Hay tres situaciones que han afectado al progreso del proyecto:

1. **Curva de aprendizaje de las tecnologías utilizadas.** Al principio del desarrollo surgieron complicaciones debido a la falta de experiencia previa con algunas tecnologías y a la documentación insuficiente de Overwolf, situación que se vio agravada por la escasez de recursos extraoficiales debido al contexto específico de esta tecnología.
2. **Balance rendimiento-coste**. Conseguir un equilibrio entre un rendimiento aceptable y un coste económico bajo fue un desafío continuo, forzándonos a buscar soluciones innovadoras y no siempre convencionales.
3. **Limitación de CORS e implementación de IndexedDB**. La limitación de CORS fue un contratiempo inesperado que ralentizó el desarrollo, obligándonos a optar por utilizar IndexedDB. A pesar de que Overwolf recomendaba su uso[47], la implementación de esta tecnología presentó algunos problemas en los que se tuvo que invertir un tiempo adicional para resolverlos.
4. **Cantidad de contenido a añadir**. La cantidad de vídeos necesaria para conseguir una aplicación completa es muy elevada. Cada vídeo debe seleccionarse y categorizarse con criterio y esto ha supuesto sin duda un gran desafío. Debido a la magnitud de la tarea, tal y como se comenta en el apartado *6.5.1 – Creación y poblado de Vids.ts*, se buscaron soluciones alternativas y se pospuso el poblado de datos para más adelante.

Por otro lado, el desarrollo tampoco ha estado exento de errores. Uno de los errores más significativos es que la interfaz se ha recargado con demasiadas animaciones e imágenes. Esto ha provocado un empeoramiento de la experiencia de usuario, ya que hace que la aplicación se ralentice en ordenadores con menos recursos[48], especialmente al aplicar filtros.

Este problema subraya la importancia de priorizar la eficiencia y simplicidad en el diseño de interfaces, especialmente cuando se trabaja con dispositivos de capacidades variables.

8.2. Reflexión personal

A nivel personal, el desarrollo de la aplicación ha sido en general una experiencia enriquecedora, aunque durante algunas etapas ha sido ciertamente estresante. Conciliar este proyecto con una rutina diaria ya de por sí muy demandante no siempre ha sido fácil. Esta experiencia me ha enseñado a ser más eficiente y disciplinado y comprender que, aunque el progreso no siempre es lineal, es crucial ser autocrítico para poder detectar cuándo no nos estamos esforzando lo suficiente y así evitar estancarse por falta de constancia.

Por otro lado, haber podido finalizar este proyecto ha sido la confirmación de que estoy preparado para emprender más proyectos personales en el futuro. El desarrollo de esta aplicación me ha ayudado a mejorar mis conocimientos sobre desarrollo de software y me ha permitido investigar y probar nuevas herramientas, lenguajes y metodologías.

Además, desarrollar una aplicación con el objetivo de lanzarla a un público real me ha forzado a tener en cuenta factores como el coste económico o su visibilidad en el mercado, lo que ha enriquecido mi enfoque y fomentado la creatividad en la resolución de problemas.

8.3. Relación del trabajo desarrollado con los estudios cursados

Es relevante destacar que el conocimiento adquirido en diversas asignaturas del grado ha sido crucial para el desarrollo de la aplicación. A continuación, se listan las más relevantes:

- **PSW (11571 – Proceso de software) y PIN (11574 - Proyecto de ingeniería de software):** Estas asignaturas han proporcionado conocimientos teóricos y prácticos sobre las metodologías ágiles y la gestión de proyectos. Adicionalmente, en estas asignaturas se utilizó Scrumban, una metodología que evoluciona de la Kanban utilizada en el desarrollo y que por tanto comparte grandes similitudes.
- **ISW (11555 – Ingeniería del software):** La creación de diagramas UML aprendida en esta asignatura ha sido crucial a la hora de diseñar la arquitectura y las clases de la aplicación.
- **DDS (11565 - Diseño de software):** El cursado de esta asignatura ha sido de gran utilidad a la hora de aplicar patrones arquitectónicos y de diseño y de refactorizar el código.

- **AER (11570 - Análisis y especificación de requisitos):** Los conocimientos adquiridos en esta asignatura han sido cruciales en el análisis de requisitos, especialmente en el uso de historias de usuario.
- **MES (11569 – Mantenimiento y evolución de software):** La asignatura ha proporcionado unos conocimientos sobre control de versiones y uso de GIT que han sido de gran utilidad para el seguimiento y gestión de los cambios realizados en la aplicación a lo largo del desarrollo.
- **IPC (11556 – Interfaces persona computador):** Los principios de diseño aprendidos en la asignatura han ayudado a mejorar el diseño de la interfaz garantizando una interacción con el usuario más intuitiva, fluida y eficaz.
- **IEI (11572 - Integración e interoperabilidad):** El uso y aprendizaje sobre herramientas de *scraping* que se da en las prácticas de la asignatura han sido de gran ayuda a la hora de hacer un poblado de datos a partir del contenido de otras páginas web de la misma temática, permitiendo así poder continuar con el desarrollo durante sus etapas más iniciales.
- **TSR (11563 – Tecnología de sistemas de información en la red):** el uso de JavaScript en esta asignatura forma parte del conocimiento previo que se tenía de este lenguaje de programación y que ha facilitado la adaptación a TypeScript.
- **IIP (11541 - Introducción a la informática y a la programación), PRG (11543 - Programación), EDA (11551 - Estructura de datos y algoritmos), etc.:** A lo largo de la carrera se han utilizado lenguajes orientados a objetos y fuertemente tipados como Java o C#. La familiarización obtenida con lenguajes de estas características ha facilitado también la adaptación a TypeScript.
- **LTP (11557 - Lenguajes, tecnologías y paradigmas de la programación):** Cursar esta asignatura ha ayudado a conocer las ventajas e inconvenientes de un lenguaje con tipado estático frente a uno con tipado dinámico, ayudando a aprovechar al máximo las ventajas que ofrece TypeScript.

8.4. Trabajos futuros

Con la realización de este TFG hemos obtenido una aplicación funcional y completa. Sin embargo, se podrían implementar más funcionalidades. Estas podrían ser: enlazar el canal del propietario del vídeo de YouTube que se está referenciando para darle crédito, añadir un filtro de búsqueda por texto, la posibilidad de añadir contenido en forma de secuencia de imágenes, controlar la interfaz con atajos de teclado o un apartado de gestión de favoritos.

Tampoco hay que olvidar que queda pendiente publicar la aplicación en la tienda Overwolf, se mandará la petición de publicación una vez se haga el poblado de datos de la aplicación.

Por último, una vez publicada la aplicación podremos añadir publicidad a la aplicación y, si las cosas van bien, los ingresos recibidos nos abrirán la posibilidad de migrar todos los vídeos y sus datos relacionados a una base de datos propia, mejorando así el rendimiento de la aplicación y eliminando la dependencia que sufre la aplicación actualmente con YouTube.

9. Referencias

[1] Asociacion Española del Videojuego, «La industria del videojuego en España - Anuario 2018».

[2] Asociación Española de Videojuegos, «La industria del videojuego en España en 2023».

[3] «Kanban: una breve introducción | Atlassian». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://www.atlassian.com/es/agile/kanban

[4] «¿Qué es un tablero kanban? | Atlassian». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://www.atlassian.com/es/agile/kanban/boards

[5] «Most Popular Esports Games 2024 | Esports Charts». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://escharts.com/top-games?order=peak

[6] «Valorant Champions Tour - Wikipedia». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Valorant_Champions_Tour

[7] Mariano Cortasa, «Evil Geniuses se consagró campeón del VALORANT Champions 2023 - ESPN». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://espndeportes.espn.com/esports/nota/_/id/12500431/evil-geniuses-campeon-valorant-champions-2023-paper-rex

[8] «Luminosity vs. Evil Geniuses | Champions Tour North America Stage 2: Challengers | Group Stage | Valorant match | VLR.gg». Accedido: 30 de julio de 2024. [En línea]. Disponible en: https://www.vlr.gg/99705/luminosity-vs-evil-geniuses-champions-tour-north-america-stage-2-challengers-w3

[9] D. Leffingwell y P. Behrens, «A User Story Primer», 2009, Accedido: 30 de julio de 2024. [En línea]. Disponible en: www.scalingsoftwareagility.wordpress.com

[10] «Easily create apps for PC games on the Overwolf framework | Overwolf». Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/

[11] «Riot Developer Portal - Valorant API». Accedido: 6 de agosto de 2024. [En línea]. Disponible en: https://developer.riotgames.com/apis#val-console-match-v1

[12] «Overwolf API Overview | Overwolf». Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/api

[13] «Overwolf SDK Introduction | Overwolf». Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/start/getting-started/sdk-introduction

[14] «Electron (software) - Wikipedia, la enciclopedia libre». Accedido: 6 de agosto de 2024. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Electron_(software)

[15] «Supported Games IDs | Overwolf». Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/api/games/ids

[16] «Valorant Game events | Overwolf». Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/api/live-game-data/supported-games/valorant

[17] «Ads SDK API | Overwolf». Accedido: 6 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/api/general/ads-sdk

[18] «Riot Developer Portal». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://developer.riotgames.com/docs/valorant

[19] «Riot Developer Portal». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://developer.riotgames.com/apis#val-match-v1/GET_getMatch

[20] «TypeScript: JavaScript With Syntax For Types.» Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.typescriptlang.org/

[21] «React». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://es.react.dev/

[22] «Home • Angular». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://angular.dev/

[23] «Bootstrap · The most popular HTML, CSS, and JS library in the world.» Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://getbootstrap.com/

[24] «Facade». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://refactoring.guru/es/design-patterns/facade

[25] «Window: localStorage property - Web APIs | MDN». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[26] E. Evans y M. Fowler, «Specification». Accedido: 5 de septiembre de 2024. [En línea]. Disponible en: https://www.martinfowler.com/apsupp/spec.pdf

[27] «Lucidchart | Diagramming Powered By Intelligence». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.lucidchart.com/pages/?

[28] «Figma». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.figma.com/

[29] «Kanban Tool – Tablero Kanban para Empresas | Software Kanban | Kanban Tool». Accedido: 6 de agosto de 2024. [En línea]. Disponible en: https://kanbantool.com/es/

[30] «GitHub: Let’s build from here · GitHub». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://github.com/

[31] «Git». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://git-scm.com/

[32] «GitHub Desktop | Simple collaboration from your desktop · GitHub». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://github.com/apps/desktop

[33] «Visual Studio Code - Code Editing. Redefined». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://code.visualstudio.com/

[34] «jQuery». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://jquery.com/

[35] «¿Qué es el DOM? El significado del Modelo de Objeto de Documento en JavaScript». Accedido: 3 de septiembre de 2024. [En línea]. Disponible en: https://www.freecodecamp.org/espanol/news/que-es-el-dom-el-significado-del-modelo-de-objeto-de-documento-en-javascript/

[36] «Splide - The lightweight, flexible and accessible slider/carousel». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://splidejs.com/

[37] «Trabaja con IndexedDB | Articles | web.dev». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://web.dev/articles/indexeddb?hl=es-419

[38] «Jest · Delightful JavaScript Testing». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://jestjs.io/es-ES/

[39] «GitHub - demike/TsUML2: Generates UML diagrams from TypeScript source code». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://github.com/demike/TsUML2

[40] «GIMP - GNU Image Manipulation Program». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.gimp.org/

[41] «Dibuja, crea y edita con Paint | Microsoft Windows». Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.microsoft.com/es-es/windows/paint

[42] «What is the Multiton Design Pattern? | by Göksu Deniz | Medium». Accedido: 3 de septiembre de 2024. [En línea]. Disponible en: https://justgokus.medium.com/what-is-the-multiton-design-pattern-eeeb5dd8bc7d

[43] «Cross-Origin Resource Sharing (CORS) - HTTP | MDN». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[44] «manifest.json | Overwolf Developers». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://dev.overwolf.com/ow-native/reference/manifest-json/

[45] «Choosing your App's Client-Side storage technology | Overwolf Developers». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://dev.overwolf.com/ow-native/guides/general-tech/choosing-your-apps-client-side-storage-technology/#indexeddb

[46] «Blob - Referencia de la API Web | MDN». Accedido: 4 de septiembre de 2024. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/Blob

[47] «Choosing your App's Client-Side Storage Technology | Overwolf». Accedido: 15 de agosto de 2024. [En línea]. Disponible en: https://overwolf.github.io/topics/best-practices/data-persistence#indexeddb

[48] «How Too Many Animations Ruin User Experience | by sikiru | Medium». Accedido: 15 de agosto de 2024. [En línea]. Disponible en: https://medium.com/@sikirus81/how-too-many-animations-ruin-user-experience-a2eb7040c1a8

Anexo

OBJETIVOS DE DESARROLLO SOSTENIBLE

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

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

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

- **ODS 3 – Salud y bienestar:** La aplicación desarrollada ayuda a mejorar la experiencia de juego de los usuarios de Valorant, lo que provoca un mayor disfrute y satisfacción tras la sesión de juego, contribuyendo así al bienestar emocional del jugador. Adicionalmente, el uso de la aplicación evita la necesidad de invertir muchas horas en memorizar información sobre el juego, lo que podría a ayudar a prevenir cansancio mental en el usuario.

- **ODS 8 – Trabajo decente y crecimiento económico**: La aplicación desarrollada ofrece un modelo de negocio que, pese a no requerir de una gran inversión económica, tiene el potencial de generar ingresos económicos tanto al desarrollador como a creadores de contenido de YouTube. Este potencial escenario abriría una puerta a la contratación de personal para el mantenimiento y expansión de la aplicación.

- **ODS 9 – Industria, Innovación e infraestructuras:** La aplicación contribuye al crecimiento de la industria de los eSports, al ofrecer una herramienta que ayuda a los jugadores a mejorar su rendimiento se está contribuyendo a crear un entorno más competitivo, haciendo de esta manera que los eSports sean un espectáculo más atractivo y potenciando también la generación de empleo en sectores relacionados como el coaching, la creación de contenido online o el análisis de videojuegos.