**Índice**

[**1. Descripción y Justificación del Proyecto 3**](#_xlvclct2hbvt)

[1.1. Alcance 4](#_991s3tg2gvf7)

[1.2. Alternativas de mercado 5](#_ekios96ngqgs)

[1.3. Stack Tecnológico 6](#_po7l3oihyo4)

[Desarrollo del Videojuego 6](#_ve8vbyqf9q0k)

[Backend y Gestión de Datos 6](#_lhndrc12cf6p)

[Creación de Recursos Gráficos (Assets) 7](#_kvf8dfhn3pdy)

[1.4. Descripción del videojuego 7](#_nntl4rv9alk5)

[1.4.1. Forma y mecánica 7](#_j49tnnyo5umk)

[1.4.2. Arte y estilo visual 8](#_oe8bg6ppt2r3)

[1.4.3. Target de usuarios 11](#_he9c1ux9vg8b)

[**2. Objetivo y requisitos del sistema 11**](#_lb27i3oyjb62)

[2.1. Requisitos funcionales 11](#_4nhgipqeqpl0)

[2.2. Requisitos no funcionales 12](#_deyjm94op34o)

[2.3. Requisitos de interfaz 13](#_i3jo4t28m257)

[2.4. Casos de uso 14](#_m0oxeq1v6h63)

[**3. Base de datos 18**](#_lzjo4hifbnab)

[3.1. Elección de Firebase 18](#_fry46u58q2gw)

[3.2. Estructura de la base de datos. 19](#_nxajdhtilmb8)

# 

# **Descripción y Justificación del Proyecto**

**Warrior’s Attack** es un videojuego de acción 2D desarrollado en Unity. Se centra en combates intensos, ofreciendo mecánicas variadas: cuerpo a cuerpo y a distancia con la lanza. Aunque prioriza la acción, el juego forma parte de una aplicación híbrida que integra un sistema en la nube (Firebase) para la gestión. Esto permite a los jugadores sumergirse en la experiencia de Sanctum, manteniendo un control personalizado del historial y las estadísticas de cada partida. Este proyecto busca demostrar los conocimientos adquiridos en el ciclo realizando una aplicacion que combine un diseño jugable atractivo con la programación avanzada y la gestión de datos.

La aplicación se compone de varios módulos principales que trabajan de forma integrada:

* **El Videojuego (Unity):** Un entorno de acción 2D donde "Lincoln", el personaje principal, se enfrenta a hordas de enemigos zombi y a un jefe final que pondrá a prueba las habilidades del jugador.
* **Control de Acceso y Roles:** Implementado con Firebase Authentication, garantiza un acceso seguro y permite diferenciar los roles de cada usuario dentro de la plataforma.
* **Almacenamiento en la Nube:** Utilizamos Cloud Firestore para la persistencia de datos, registrando los perfiles de usuario, cada partida jugada y sus correspondientes estadísticas.
* **Experiencia de Usuario (UI):** La interfaz ha sido diseñada en Unity, estructurada en pantallas claras y diferenciadas: el acceso (*login*), el panel de gestión para administradores, la vista de jugador y la propia escena de juego.
* **Recursos Visuales:** El contenido gráfico se generó con herramientas de Inteligencia Artificial y fue editado manualmente a posterior utilizando Photoshop CS6 y Pixellab.

## **Alcance**

"Warrior's Attack" consiste en el desarrollo de un prototipo funcional avanzado, que presenta una experiencia de videojuego 2D completa e integrada con un sistema de perfiles de usuario y gestión de datos.

Para poder desarrollar el proyecto con la profundidad técnica necesaria, pero teniendo en cuenta las dimensiones y plazos del mismo, el alcance se acota a las siguientes características clave. Esta base funcional está diseñada para ser escalable, permitiendo la fácil implementación de futuras actualizaciones y nuevo contenido.

El alcance se divide en dos áreas principales: la experiencia de juego y la Gestión de Usuarios y Datos en la Nube.

**La Experiencia de Juego**

El corazón del proyecto es un videojuego de acción y plataformas *side-scroller*. El jugador tomará el control de "Lincoln", el protagonista, y deberá avanzar a través de un único nivel de juego diseñado para este proyecto.

Este nivel pondrá a prueba las habilidades del jugador, que contará con un sistema de control que le permite correr, saltar y combatir. El personaje dispondrá de un set completo de seis animaciones principales que proporcionarán *feedback* visual sobre sus acciones y su estado de salud.

El combate es una mecánica central, permitiendo al jugador usar su lanza empleando un ataque arrojadizo a distancia. Para crear un desafío, el nivel estará poblado por tres tipos de enemigos con comportamientos de persecución y ataque: los básicos "The Bloater" y "The Stalker", y un jefe final llamado "The Overlord".

El alcance del juego cubre el ciclo de partida completo: desde que el jugador inicia el nivel hasta que alcanza la victoria (derrotando al jefe) o la derrota (perdiendo toda su salud). Al terminar, el juego registrará el *score* de la partida.

**Gestión de Usuarios y Datos en la Nube**

El videojuego está envuelto en un sistema de gestión de perfiles y datos conectado a la nube.

* **Sistema de Cuentas de Usuario:** La aplicación requerirá que los usuarios se registren y creen una cuenta personal. Esto permite que el progreso y las estadísticas de cada jugador se guarden de forma segura y persistente.
* **Perfiles y Roles:** El sistema será capaz de diferenciar entre dos tipos de usuarios. La mayoría serán "Jugadores", que acceden al juego, pero también existirá un rol de "Administrador" con permisos especiales como la gestión de los usuarios.
* **Paneles de Usuario:** La aplicación contará con varias pantallas fuera de la partida:
* **Dashboard del Jugador:** Al iniciar sesión, el jugador verá un panel personal con un resumen de su rendimiento histórico: total de enemigos eliminados, porcentaje de victorias, mejor puntuación y un historial de sus partidas recientes.
* **Ranking:** Se incluirá una tabla de clasificación pública para que los jugadores puedan competir por la puntuación más alta.
* **Panel de Administración:** El usuario "Administrador" tendrá acceso a una vista de gestión. Desde aquí, podrá supervisar la base de datos de usuarios (añadir, modificar o eliminar cuentas) y consultar las estadísticas de cualquier jugador.

## **Alternativas de mercado**

Para situar el proyecto, hemos analizado dos juegos muy conocidos que, aunque comparten el género 2D *side-scroller*, son muy diferentes entre sí:

1. **Saga *Super Mario* (Nintendo):** Es el referente clásico de las plataformas. La diversión en *Mario* se centra en el movimiento: saltar con precisión, esquivar obstáculos y avanzar por niveles con un diseño muy cuidado. El combate es sencillo (normalmente, un salto) y es secundario a la habilidad de moverse.
2. ***Hollow Knight* (Team Cherry):** Es un referente moderno de acción y exploración. Aunque también es un 2D *side-scroller*, *Hollow Knight* se centra en un combate más técnico y en batallas desafiantes contra jefes finales. También destaca por su gran mundo que el jugador debe explorar.

"Warrior's Attack" toma elementos de ambos mundos:

* Se juega como un juego de plataformas clásico, avanzando de izquierda a derecha en un nivel (como en Mario).
* Pero pone el foco en el combate (como en Hollow Knight), con enemigos que persiguen y una batalla final contra un jefe.

Aquí es donde nuestro proyecto se desmarca por completo. *Mario* y *Hollow Knight* son juegos que se guardan en la propia consola o PC del jugador (guardado local).

"Warrior's Attack" está construido de forma diferente, como una aplicación conectada a la nube. La verdadera innovación del proyecto es cómo el juego se integra con un sistema de gestión de usuarios, algo que los juegos tradicionales no ofrecen de esta manera.

Las características que realmente lo diferencian son:

* **Sistema de Cuentas:** No es un juego de "empezar y jugar". Primero, el usuario debe registrarse e iniciar sesión.
* **Guardado en la Nube:** Al terminar una partida, la puntuación no se guarda localmente, sino que se envía a una base de datos online (Cloud Firestore).
* **Ranking Público:** Gracias a esos datos en la nube, podemos tener una tabla de clasificación donde todos los jugadores compiten.
* **Panel de Administración:** El proyecto incluye un "modo" que no es para jugar: un panel para un Administrador. Desde ahí, una persona con ese rol puede ver la lista de todos los jugadores, gestionar sus cuentas o consultar sus estadísticas.

En resumen, mientras las alternativas se centran al 100% en la experiencia de un solo jugador sin conexión, "Warrior's Attack" demuestra cómo construir un juego sobre una arquitectura de usuarios y datos en línea, que es el núcleo de este proyecto.

## 

## **Stack Tecnológico**

#### **Desarrollo del Videojuego**

* **Unity (Motor de Juego):** Es el pilar sobre el que se construye el videojuego. Se eligió Unity por ser el motor líder en el desarrollo 2D y multiplataforma. Su robusto motor de físicas 2D, su intuitivo sistema de animación y la gestión de escenas combinado con los conocimientos adquiridos en el grado sobre este motor lo convierten en la opción ideal para un *side-scroller* de acción.
* **C# (Lenguaje de Programación):** Es el lenguaje que utilizan los scripts originalmente de Unity. Se utiliza para programar absolutamente toda la lógica del juego: el control de Lincoln, la inteligencia artificial de los enemigos, la mecánica de la lanza y la comunicación con la base de datos.

#### **Backend y Gestión de Datos**

Para toda la parte *online* del juego, la decisión fue usar Firebase, una plataforma de Google.

Firebase nos proporciona un *backend* ya hecho (lo que se conoce como "Backend as a Service") que nos soluciona toda la infraestructura de servidor. Esto nos permitió centrarnos en programar la lógica del juego en Unity y la aplicación.

Los servicios concretos de Firebase que utilizamos son:

* **Firebase Authentication:** Este es el servicio que usamos para todo lo relacionado con las cuentas de usuario. Se encarga de forma segura del proceso de registro, del inicio de sesión y de mantener la sesión del jugador activa.
* **Cloud Firestore:** Esta es nuestra base de datos en la nube. Elegimos Firestore porque es una base de datos NoSQL, lo que la hace muy flexible para guardar diferentes tipos de información (como perfiles, partidas y estadísticas) sin una estructura fija y complicada. Aquí es donde guardamos *toda* la información importante:
  + Los perfiles de usuario (incluyendo su rol de "Admin" o "Jugador").
  + El historial de las partidas jugadas.
  + Las estadísticas totales de cada jugador (kills, victorias, etc.).
  + Las puntuaciones para la tabla del ranking.

#### **Creación de Recursos Gráficos (*Assets*)**

* **IA Generativa y Pixellab.ai:** Para la creación de los *sprites* (personajes, enemigos, objetos) y sus animaciones, se utilizaron herramientas de IA como Pixellab.ai o Gemini para generar la base del personaje. Esto permitió obtener un resultado funcional y resolutivo sin necesidad de conocimientos avanzados en *pixel art* o animación.
* **Adobe Photoshop CS6:** Este software se ha utilizado para el trabajo de edición de los gráficos. Con él se han editado y retocado las bases de los sprites generados por IA, se han añadido efectos y se ha procurado cuidar algunos detalles al máximo.

## **Descripción del videojuego**

**"Warrior's Attack"** te sumerge en una aventura de supervivencia y acción en el corazón de **Sanctum**, un mundo selvático y hostil que ha caído víctima de una misteriosa plaga.

En este caos, tomarás el control de **Lincoln**, un guerrero solitario y uno de los pocos supervivientes que se atreven a hacer frente a la amenaza. Las otrora vibrantes junglas de Sanctum se han convertido en un cementerio verde, un laberinto mortal patrullado por hordas de zombies.

Armado únicamente con tu fiel lanza, tu misión es abrirte camino a través de este territorio perdido. No será fácil: el peligro acecha en cada sombra. Criaturas zombificadas, como los lentos pero resistentes Bloaters y los ágiles y letales Stalkers, intentarán detenerte a cada paso.

Tu viaje es una prueba de habilidad y resistencia. Tendrás que usar tu lanza arrojadiza con precisión para mantener a raya a los enemigos mientras avanzas hacia el origen de la infección. En las profundidades de la jungla te espera el desafío definitivo: una criatura titánica conocida como "The Overlord", el presunto líder de la horda.

¿Podrás guiar a Lincoln a través de los peligros de Sanctum y enfrentarte a la pesadilla que ha consumido el planeta?

## **Forma y mecánica**

"Warrior's Attack" es un videojuego de acción y plataformas 2D de tipo *side-scroller* (desplazamiento lateral). El *gameplay* se centra en el combate estratégico a distancia y la superación de un nivel lineal.

**Controles y Habilidades del Jugador**

El jugador controla a Lincoln con un esquema sencillo de movimiento:

* **Movimiento:** Control de movimiento horizontal (izquierda y derecha).
* **Salto:** Una habilidad de salto simple para esquivar enemigos y avanzar por la jungla.
* **Salud:** Lincoln cuenta con un estado de salud que se reduce al recibir daño por parte de los enemigos.

**Mecánica de Combate Principal**

El combate es el pilar de la jugabilidad. A diferencia de otros juegos del género, "Warrior's Attack" se enfoca en el combate a distancia:

* **Lanza Arrojadiza:** La única forma de ataque de Lincoln es arrojar su lanza.
* **Funcionamiento:** Al pulsar el botón de ataque, Lincoln atacará a distancia con su lanza.
* **Daño:** Cada impacto de la lanza inflige 10 HP de daño a los enemigos.

**Enemigos e Inteligencia Artificial**

Los enemigos (The Bloater y The Stalker) están diseñados para ser una amenaza constante. Poseen una IA básica de persecución: al detectar a Lincoln dentro de su rango de visión, abandonarán su posición inicial y avanzarán directamente hacia el jugador para atacarlo por contacto.

**Condiciones de Victoria y Derrota**

El ciclo de juego es claro y completo:

* **Derrota:** La partida termina si la salud de Lincoln llega a 0 HP.
* **Victoria:** Para completar el prototipo, el jugador debe avanzar por todo el nivel y derrotar al jefe final, "The Overlord".

Al finalizar la partida (ya sea por victoria o derrota), el juego registra el rendimiento y las estadísticas y lo envía al sistema de datos en la nube.

## **Arte y estilo visual**

El estilo visual de "Warrior's Attack" se ha desarrollado a través de un proceso híbrido, buscando fusionar la eficiencia de las herramientas de Inteligencia Artificial con el control y la personalización manual. El objetivo ha sido crear una estética Pixel Art que, aunque sea un estilo clásico y muy utilizado, se siente moderno gracias a sus animaciones detalladas y a una paleta de colores que refuerza la ambientación oscura y hostil de Sanctum.

La elección de este estilo responde perfectamente a la coherencia visual. Cada elemento gráfico ha sido creado y editado para que contribuya al “realismo” y a la inmersión del jugador en el juego.

**Proceso de Creación de *Assets***

1. **Generación Asistida por IA:** La base de los *sprites* y las secuencias de animación del personaje principal, Lincoln, y de los distintos enemigos, se ha generado utilizando herramientas de Inteligencia Artificial como Pixellab.ai y Gemini. Este enfoque ha permitido establecer rápidamente el concepto visual y la estructura de las animaciones (idle, run, jump, attack, hurt y dead).
2. **Edición y Retoque Manual (Adobe Photoshop CS6):** Los *sprites* generados por la IA eran una buena base, pero no el resultado final. Por eso, todos pasaron por un trabajo de edición y retoque manual en Adobe Photoshop CS6. Este paso fue fundamental para varias cosas:

* **Corrección y Coherencia:** Lo primero fue arreglar fallos, como píxeles sueltos o colores que no encajaban. Se trabajó para que todos los gráficos tuvieran el mismo estilo de *pixel art* y usaran una paleta de colores similar, asegurando que parecieran parte del mismo juego.
* **Detalles de Ambientación:** Aquí se añadieron los detalles que la IA no puede crear. Lo más notable son los efectos de daño(como la sangre), que hacen que el combate se sienta más bruto y refuerzan la atmósfera hostil de la jungla.
* **Optimización:** Finalmente, se prepararon todos los *sprites* (ajustando tamaños, recortando transparencias, etc.) para que se pudieran importar correctamente en Unity y funcionaran de manera eficiente dentro del juego.

**Presentación de Personajes**

Los principales protagonistas y antagonistas del mundo de "Warrior's Attack" son los siguientes:

#### 

#### 

**Lincoln**

El personaje principal y el último bastión de esperanza en Sanctum. Lincoln es un guerrero ágil y determinado, armado únicamente con su lanza. Sus *sprites* y animaciones reflejan su constante estado de alerta y su capacidad para moverse y atacar con precisión.

#### 

#### 

**The Bloater**

Estos zombis corpulentos son los enemigos más básicos. Su baja velocidad es compensada por su resistencia y su capacidad para aparecer en grupo. Su diseño subraya su naturaleza de carne putrefacta y lenta descomposición.



#### 

**The Stalker**

Los Stalkers son la contraparte ágil de los Bloaters. Son más pequeños y rápidos, capaces de cerrar la distancia con Lincoln rápidamente, lo que exige una reacción veloz por parte del jugador.

#### 

#### 



**The Overlord, el Jefe Final**

La criatura alfa, el origen o el líder de la infección en Sanctum. The Overlord es un enemigo de gran tamaño y poder, diseñado para ser el desafío máximo del prototipo. Su diseño es imponente y aterrador.

## **Target de usuarios**

El público objetivo de "Warrior's Attack" se define por varios factores, combinando el interés en el género del videojuego con el perfil de un usuario acostumbrado a aplicaciones conectadas.

El *target* principal se compone de:

* **Aficionados a los Juegos de Acción 2D:** Jugadores que disfrutan de los *side-scrollers* clásicos y modernos, donde la habilidad del jugador (reflejos, precisión en el combate) es el factor principal. Se busca atraer a quienes aprecian un desafío directo, inspirado en la jugabilidad de referentes como *Hollow Knight* pero con un enfoque más lineal y arcade.
* **Jugadores Competitivos:** La inclusión de un sistema de *score* por partida, estadísticas detalladas y un ranking global está orientado para atraer a un público que no solo quieren completar el juego, sino dominarlo. El perfil de jugador que disfruta compitiendo por la mejor puntuación y comparando su rendimiento con el de otros es un objetivo clave.
* **Público Joven (12-30 años):** Aunque la estética *pixel art* es universal, la temática zombis y el nivel de desafío de los enemigos y el jefe final orientan el juego a un público acostumbrado a los videojuegos.

# **Objetivo y requisitos del sistema**

Ahora que ya hemos descrito la idea general del proyecto y su alcance, en esta sección vamos a definir el objetivo técnico principal. Es decir, explicaremos qué tiene que hacer exactamente el programa para que el prototipo se considere terminado y funcional.

El objetivo general es desarrollar un prototipo de "Warrior's Attack" que funcione de principio a fin, demostrando que el juego (hecho en Unity) se conecta correctamente con el sistema de usuarios en la nube (Firebase).

## **Requisitos funcionales**

Aquí detallamos las funciones principales del proyecto. Es una lista de las acciones y características específicas que el *software* debe hacer para cumplir con los objetivos. Lo hemos dividido en dos partes: las funciones de la aplicación (usuarios, menús) y las del propio videojuego.

**Funciones de la Aplicación y Gestión de Datos**

* **RF-001:** El programa debe permitir que un usuario nuevo se registre y que uno existente inicie sesión.
* **RF-002:** El programa debe saber si el usuario que entra es "Admin" o "Jugador", y mostrarle menús diferentes.
* **RF-003:** El usuario “Jugador” debe poder ver un panel (Dashboard) con sus estadísticas (kills, victorias...) y su historial de partidas.
* **RF-004:** El usuario "Administrador" debe poder ver una lista de todos los usuarios, modificar sus datos y eliminarlos.
* **RF-005:** El programa debe mostrar un Ranking público con las mejores puntuaciones de todos los jugadores.
* **RF-006:** El programa debe guardar en la base de datos la puntuación y las estadísticas cuando termina una partida.

**Funciones del Videojuego (Gameplay)**

* **RF-007:** El jugador debe poder mover a Lincoln (izquierda/derecha) y hacerlo saltar.
* **RF-008:** El jugador debe poder atacar (arrojar la lanza).
* **RF-009:** El jugador debe tener una barra de vida (HP) que baje cuando los enemigos le golpeen.
* **RF-010:** Los enemigos deben perseguir al jugador cuando lo vean.
* **RF-011:** El juego debe saber cuándo la partida termina, ya sea por "Victoria" (al matar al jefe final) o por "Derrota" (si la vida del jugador llega a 0).

## **Requisitos no funcionales**

Además de las funciones principales, el proyecto debe cumplir con ciertas características de calidad y rendimiento. Estos son los requisitos no funcionales:

* **RFN-001 (Rendimiento):** El juego debe sentirse ágil y responder al instante a los controles. No debe sufrir tirones o ralentizarse, ni siquiera cuando aparezcan varios enemigos a la vez en la pantalla.
* **RFN-002 (Usabilidad):** La interfaz de usuario (menús, botones, *dashboard*) debe ser clara e intuitiva. El usuario debe entender cómo navegar por la aplicación sin necesidad de un manual.
* **RFN-003 (Compatibilidad):** El proyecto debe ser ejecutable como una aplicación independiente.
* **RFN-004 (Seguridad):** Las contraseñas de los usuarios deben gestionarse de forma segura (esto lo cumple Firebase Authentication automáticamente, al no almacenarlas en texto plano).
* **RFN-005 (Mantenibilidad):** El código fuente en C# debe estar organizado (en carpetas de *Scripts*) y comentado en sus partes más complejas, para facilitar su comprensión y futuras modificaciones.
* **RFN-006 (Fiabilidad):** La conexión con Firebase debe ser estable. El juego debe gestionar correctamente los posibles fallos de conexión.

## **Requisitos de interfaz**

Estos requisitos definen los elementos visuales y de navegación que la aplicación debe presentar al usuario para garantizar una experiencia clara y funcional.

**Pantalla de Acceso (Login/Registro)**

* **RI-001:** La pantalla de inicio debe presentar al usuario dos opciones claras: "Iniciar Sesión" y "Registrarse".
* **RI-002:** El formulario de registro debe solicitar un nombre de usuario, un correo electrónico y una contraseña.
* **RI-003:** El formulario de inicio de sesión debe solicitar un nombre de usuario y una contraseña.
* **RI-004:** El sistema debe mostrar mensajes de error visibles si los datos son incorrectos.

**Panel del Jugador (Dashboard)**

* **RI-005:** Tras el login, el jugador debe ver una pantalla principal (Dashboard) que muestre su nombre de usuario.
* **RI-006:** El dashboard debe contener botones de navegación claros para "Jugar", "Ver Ranking" y "Cerrar Sesión".
* **RI-007:** El dashboard debe mostrar de forma destacada las estadísticas clave del jugador (Kills, Victorias, Mejor Puntuación).
* **RI-008:** El dashboard debe incluir una sección o pestaña para consultar el historial de partidas.

**Panel de Administración**

* **RI-009:** El usuario "Administrador" debe ver un panel diferente al del jugador.
* **RI-010:** El panel de administrador debe mostrar una tabla o lista con todos los usuarios registrados (mostrando su nombre de usuario, email y rol).
* **RI-011:** Cada usuario en la lista debe tener controles visibles (botones o iconos) para "Modificar" y "Eliminar".

**Interfaz del Juego (HUD)**

* **RI-012:** Durante la partida, el jugador debe ver en pantalla un HUD fijo.
* **RI-013:** El HUD debe mostrar, como mínimo, la salud actual de Lincoln.

## 

## **Casos de uso**

A continuación, se describen los cuatro casos de uso principales que ilustran la interacción entre los actores (Jugador y Administrador) y el sistema "Warrior's Attack".

| **ID** | **CU-001** |
| --- | --- |
| **Nombre** | Registrar un nuevo usuario |
| **Actor(es)** | Usuario |
| **Descripción** | Describe el proceso que sigue un nuevo usuario para crear una cuenta en la aplicación. |
| **Precondición(es)** | El usuario está en la pantalla de Acceso. |
| **Flujo Básico** | 1. El usuario selecciona "Registrarse". 2. El sistema muestra el formulario (nombre de usuario, email, contraseña). 3. El usuario introduce los datos y pulsa "Confirmar". 4. El sistema valida que el nombre de usuario y el email son únicos. 5. El sistema crea la cuenta en Firebase Auth (con email) y el documento en Firestore (con nombre de usuario, email, rol "Jugador"). 6. El sistema inicia sesión al usuario. 7. El sistema redirige al usuario al "Panel del Jugador". |
| **Postcondición(es)** | El usuario está registrado en el sistema y tiene una sesión activa. |
| **Flujos Alternativos** | **(Datos ya existen):** Si el nombre de usuario o el email introducidos ya están en uso, el sistema muestra un mensaje de error (ej. "Datos no válidos") y el flujo se detiene, permitiendo al usuario corregirlos. |

| **ID** | **CU-002** |
| --- | --- |
| **Nombre** | Jugar una partida completa |
| **Actor(es)** | Jugador |
| **Descripción** | Describe el flujo del jugador desde que inicia una partida hasta que la finaliza (por victoria o derrota) y se guardan sus datos. |
| **Precondición(es)** | El Jugador ha iniciado sesión y está en el Dashboard. |
| **Flujo Básico** | 1. El Jugador pulsa "Jugar". 2. El sistema carga la escena de juego. 3. El Jugador controla a Lincoln, avanza y derrota enemigos. 4. El Jugador derrota al jefe final ("The Overlord"). 5. El sistema detecta la "Victoria". 6. El sistema recopila y guarda las estadísticas en Cloud Firestore. 7. El sistema muestra la pantalla de "Victoria" y redirige al Dashboard. |
| **Postcondición(es)** | La partida ha finalizado y las estadísticas están guardadas. |
| **Flujos Alternativos** | **(Derrota):** Si durante la partida la salud de Lincoln llega a 0, el sistema detecta "Derrota". A continuación, recopila y guarda las estadísticas de la partida, muestra la pantalla de "Derrota" y redirige al Jugador de vuelta al Dashboard. |

| **ID** | **CU-003** |
| --- | --- |
| **Nombre** | Gestionar usuarios |
| **Actor(es)** | Administrador |
| **Descripción** | Describe cómo el Administrador modifica o elimina usuarios del sistema. |
| **Precondición(es)** | El Administrador ha iniciado sesión y está en su panel. |
| **Flujo Básico** | 1. El Administrador ve la lista de usuarios. 2. El Administrador selecciona un usuario y pulsa "Modificar". 3. El Administrador cambia el rol del usuario y pulsa "Guardar". 4. El sistema actualiza el dato en Cloud Firestore. 5. El sistema refresca la lista. |
| **Postcondición(es)** | Los datos del usuario han sido modificados. |
| **Flujos Alternativos** | **(Eliminar usuario):** El Administrador pulsa "Eliminar" en un usuario. El sistema pide una confirmación. Si el Admin confirma, el sistema elimina al usuario (de Auth y Firestore) y refresca la lista. |

| **ID** | **CU-004** |
| --- | --- |
| **Nombre** | Consultar el ranking |
| **Actor(es)** | Jugador |
| **Descripción** | Describe cómo un jugador accede y visualiza la tabla de clasificación global. |
| **Precondición(es)** | El Jugador ha iniciado sesión y está en el Dashboard. |
| **Flujo Básico** | 1. El Jugador pulsa "Ver Ranking". 2. El sistema consulta a Cloud Firestore las mejores puntuaciones. 3. El sistema muestra la tabla de clasificación. 4. El Jugador pulsa "Volver" y regresa al Dashboard. |
| **Postcondición(es)** | El Jugador ha consultado la clasificación. |
| **Flujos Alternativos** | **(Error de conexión):** Si el sistema no puede conectarse a Cloud Firestore para obtener los datos, muestra un mensaje de error (ej. "No se pudo cargar el ranking"). |

# **Base de datos**

Toda la arquitectura de la aplicación "Warrior's Attack" gira en torno a la persistencia de datos en la nube. Un sistema de gestión de usuarios, roles, estadísticas e historial de partidas requiere una base de datos robusta, flexible y de acceso rápido.

En esta sección se detalla la tecnología de base de datos seleccionada, las razones de dicha elección y el modelo de datos diseñado para dar soporte a todos los requisitos funcionales del proyecto.

## **Elección de Firebase**

La decisión tecnológica más importante del proyecto fue elegir Google Firebase para toda la parte *online*.

Al principio se consideraron otras opciones. Por ejemplo, usar una base de datos local como SQLite, pero se descartó porque no permite guardar datos en la nube (y, por tanto, no tendríamos ni perfiles *online* ni ranking global).

La otra opción era almacenar una base de datos (como MySQL) en un servidor alojado en una máquina virtual.

Al ver las ventajas que proporcionaba Firebase dándonos un sistema de login seguro y una base de datos NoSQL flexible ya listos para usar y sin mantenimiento, la alternativa de la máquina virtual quedó descartada por ser más compleja y menos eficiente para este prototipo.

Firebase fue la solución perfecta porque es un "Backend as a Service" (BaaS), es decir, un *backend* ya hecho que nos da todo lo que necesitábamos:

1. **Es un "todo en uno":** Firebase no es solo una base de datos. Nos da dos servicios que funcionan perfectamente juntos:
   * **Firebase Authentication:** Se encarga de todo el sistema de cuentas (registro, login, contraseñas seguras) sin que tengamos que programar la seguridad nosotros mismos.
   * **Cloud Firestore:** Es la base de datos donde guardamos la información *del juego* (perfiles, *scores*, roles, etc.).
2. **Nos ahorra montar un servidor:** No tenemos que preocuparnos de alojar el *backend*, de la seguridad, ni del mantenimiento. Google se encarga de todo eso. Esto nos permitió centrarnos en lo importante: programar el juego en Unity y la lógica de la aplicación.
3. **Es una base de datos flexible:** Firestore guarda los datos en "documentos" (similares a un archivo JSON) en lugar de en las tablas fijas de SQL. Esto es ideal para un juego, porque es muy fácil guardar el perfil de un jugador, una lista de sus partidas o sus estadísticas sin tener que definir una estructura súper complicada.
4. **Se conecta fácil con Unity:** Firebase tiene un "SDK" (un kit de herramientas) oficial para Unity. Esto hizo que la conexión entre el juego (C#) y la nube fuera muy directa.
5. **Puede crecer (Escalabilidad):** Aunque "Warrior's Attack" es un prototipo, está construido sobre una tecnología que usan aplicaciones con millones de usuarios. Si el día de mañana quisiéramos añadir más contenido o más jugadores, la base de datos aguantaría sin problemas.

## **Estructura de la base de datos**

Como hemos elegido Cloud Firestore, que es una base de datos NoSQL, no usamos un Modelo Entidad-Relación (E-R) tradicional. Nuestra estructura se basa en **Colecciones** (que son como carpetas) y **Documentos** (que son como archivos).

La organización de nuestros datos es simple y está centrada en el usuario:

**Colección usuarios**

Esta es la "carpeta" principal (Colección raíz) de toda la base de datos. Contiene todos los usuarios que se han registrado en el juego.

**Documentos de usuario**

Dentro de la colección “usuarios”, cada jugador tiene su propio "documento" o ficha personal. Este documento guarda toda su información importante:

* Sus datos de perfil (nombre de usuario, email y su rol de "Jugador" o "Admin").
* Sus estadísticas totales (el resumen de kills, victorias, mejor puntuación, etc.).

Guardamos las estadísticas resumidas aquí para que, cuando el jugador entre en su Dashboard, la información cargue de forma instantánea.

**Subcolección partidas**

Dentro de la ficha de cada jugador, creamos una "subcarpeta" (Subcolección) llamada “partidas”. Aquí es donde guardamos su historial. Cada vez que el jugador termina una partida (gane o pierda), creamos un nuevo "documento" en esta sub-carpeta con los detalles de esa partida:

* El resultado ("Victoria" o "Derrota").
* La puntuación que consiguió.
* Los enemigos que eliminó en esa sesión.

En resumen, la estructura es así:

* Usuarios (La carpeta de todos los jugadores)
  + Ficha de FierroGamer538 (Documento)
    - Datos (Rol, Email, Stats...)
    - Partidas (La carpeta de historial de FierroGamer538)
      * Partida 1 (Documento: Victoria, 5000 pts)
      * Partida 2 (Documento: Derrota, 1200 pts)
  + Ficha de OtroJugador (Documento)
    - Datos (...)
    - partidas (...)

Esta forma de organizar los datos es muy eficiente: para ver el perfil de un jugador solo leemos su ficha, y solo si quiere ver su historial, leemos la subcarpeta.

# **Guía de estilos y prototipado**

* Guía de estilos: [https://www.figma.com/design](https://www.figma.com/design/a2w5BpygBWLWOveepLq1Oo/Gu%C3%ADa-de-estilos---La-caja-del-Mart%C3%ADn?node-id=0-1&p=f&t=zbV25w3vXOFc3Cyr-0)
* Prototipo: [https://www.figma.com/design](https://www.figma.com/design/lKGbOrc6fT1pH0t2A3AVIA/Untitled?node-id=0-1&p=f&t=w2fwGi23PkS6Aapc-0)