Instituto Tecnológico de Buenos Aires

Final

72.40 - Ingenieria de Software II

Autores:

Ballerini, Santiago[1] - 61746
Martone, Gonzalo Alfredo[2] - 62141
Bafico, Juan Cruz [3] - 62070
Bosetti, Franco [4] - 61654
Zakalik, Agustín [5] - 62068

Docentes:

Juan Martin Sotuyo Dodero
Guido Matías Mogni

Fecha de entrega: 13 de Diciembre de 2023

[1] sballerini@itba.edu.ar
[2] gmartone@itba.edu.ar
[3] jbafico@itba.edu.ar
[4] fbosetti@itba.edu.ar
[5] azakalik@itba.edu.ar

Índice

1. Consigna

Se desea crear un juego online multijugador 3D en tiempo real similar a World of Warcraft. Se espera el mismo sea un éxito y cuente con millones de jugadores en todo el mundo participando activamente. El juego deberá contar con un sistema de chat, inventario, comercio con NPCs y otros usuarios, quests con NPCs y clanes. Es importante que el contenido del juego pueda ser actualizado en forma remota muy seguido, para mantener a los usuarios entretenidos con constantes novedades. La latencia percibida entre usuarios es fundamental para que la experiencia de juego sea fluida.

Adicionalmente, es fundamental poder moderar el sistema. Deberán por tanto existir cuentas de administradores, que desde dentro del juego puedan aplicar múltiples penas a los usuarios que no se comporten.

2. Funcionalidad Requerida

1. **Plataforma 3D:** El sistema debe contar con una plataforma visual en tres dimensiones que modele el juego en cuestión. La plataforma debe ser capaz de renderizar gráficos en 3D de alta calidad para ofrecer una experiencia inmersiva a los jugadores.
2. **Login:** Autenticación segura, recuperación de cuentas y cambio de contraseña.
3. **Quests:** Variedad de misiones y tareas proporcionadas por NPCs y clanes. Sistema de recompensas y logros para motivar la participación en misiones.
4. **Inventario:** Inventario único y persistente para cada personaje.
5. **Comercio:** Capacidad para vender y comprar artículos del inventario, ya sea con NPCs o con usuarios.
6. **Chat:** Soporte para chats públicos y privados entre jugadores en un mismo servidor.
7. **Moderación del sistema:** Herramientas de moderación efectivas, incluyendo la capacidad de monitorear y aplicar acciones correctivas en tiempo real. Registros detallados de actividades para facilitar la revisión y toma de decisiones.
8. **Clanes:** Creación y gestión de clanes con roles. Funcionalidades para realizar eventos y competiciones entre clanes.
9. **Creacion de Personaje:** Creador de personajes con opciones detalladas que permiten la personalización de la apariencia, clase, raza, y otras características. Cada usuario tiene la capacidad de crear y gestionar múltiples personajes dentro del juego.

3. Requisitos No Funcionales

- El sistema debe poder soportar millones de jugadores en todo el mundo participando activamente.
- El contenido del juego debe mantenerse actualizado de forma remota con alta frecuencia.
- La latencia percibida por los usuarios debe ser mínima para que la experiencia de juego sea fluida.
- La interfaz de usuario debe ser intuitiva.
- Se deben tolerar picos de trafico.
- Control de acceso.

4. Atributos de Calidad

1. **Disponibilidad:**

   Al ser un juego online, especialmente un MMO donde el juego no se puede jugar en modo offline, se requiere una disponibilidad máxima del sistema. La arquitectura debe garantizar la mínima interrupción posible para evitar la pérdida de jugadores debido a caídas o indisponibilidad del servicio.

2. **Performance:**

   La latencia mínima es esencial para proporcionar una experiencia de juego fluida. La optimización del rendimiento debe ser una prioridad, asegurando que los jugadores con buena conexión tengan una experiencia sin interrupciones. De lo contrario, podríamos perder una considerable base de jugadores, ya que jugar en un estado con alta latencia resulta frustrante.

3. **Escalabilidad:**

   La arquitectura debe ser altamente escalable para manejar el crecimiento esperado de millones de jugadores. Debe ser capaz de adaptarse a picos de demanda, como eventos especiales, expansiones o actualizaciones, sin afectar significativamente la calidad del servicio.

4. **Mantenibilidad:**

   Se busca una arquitectura que permita actualizaciones frecuentes y remotas sin afectar la disponibilidad del sistema. Un proceso de desarrollo eficiente es esencial para mantener a los usuarios interesados con novedades constantes.

5. **Seguridad:**

   La seguridad del sistema debe ser robusta, protegiendo las cuentas de los usuarios y toda la información asociada, como inventarios. La detección y prevención de cheats debe ser una prioridad, y se debe implementar un almacenamiento seguro de contraseñas y procesamiento de pagos.

6. **Usabilidad:**

   Dada la naturaleza visualmente intensa de los juegos MMO, la interfaz de usuario debe ser intuitiva y amigable. La usabilidad es crucial para mantener a los jugadores comprometidos y evitar la frustración asociada con interfaces complicadas.

7. **Auditabilidad:**

   Los administradores deben tener herramientas para auditar y monitorear el comportamiento de los jugadores. Los registros detallados del contenido del chat, contextualizados con la actividad del juego en ese momento, proporcionan a los administradores la información necesaria para tomar decisiones informadas sobre la moderación, como la aplicación de baneos.

5. Supuestos

- Se proyecta un máximo de 6 millones de usuarios durante el primer año, considerando que cada usuario ocupa un promedio de 1 KB en nuestra base de datos de usuarios.
- Se asume que todos los usuarios disponen de una dirección de correo electrónico para realizar el inicio de sesión y utilizar la autenticación de dos factores (2FA) cuando sea necesario.
- Se considera que los lugares seleccionados para nuestros data centers ofrecen la opción de contratar múltiples proveedores de servicios de internet.
- Nuestro MMO será lanzado para PC, por lo que suponemos que todos nuestros jugadores tienen una computadora con los siguientes requisitos mínimos:
  - SO : Windows 10 (64-bit version)
  - Procesador: Intel Core i5-2400/AMD FX-8320 o mejor
  - Memoria: 8 GB de RAM
  - Gráficos: NVIDIA GTX 670 2GB/AMD Radeon HD 7870 2GB o mejor
  - DirectX: Versión 10
  - Almacenamiento: 60 GB de espacio disponible
- Suponemos que todos nuestros jugadores tienen acceso a una conexión a internet lo suficientemente estable para poder jugar sin inconvenientes.
- La estimación es que no habrá más de 1000 administradores encargados de supervisar el comportamiento adecuado de los jugadores en el sistema.

6. Arquitectura

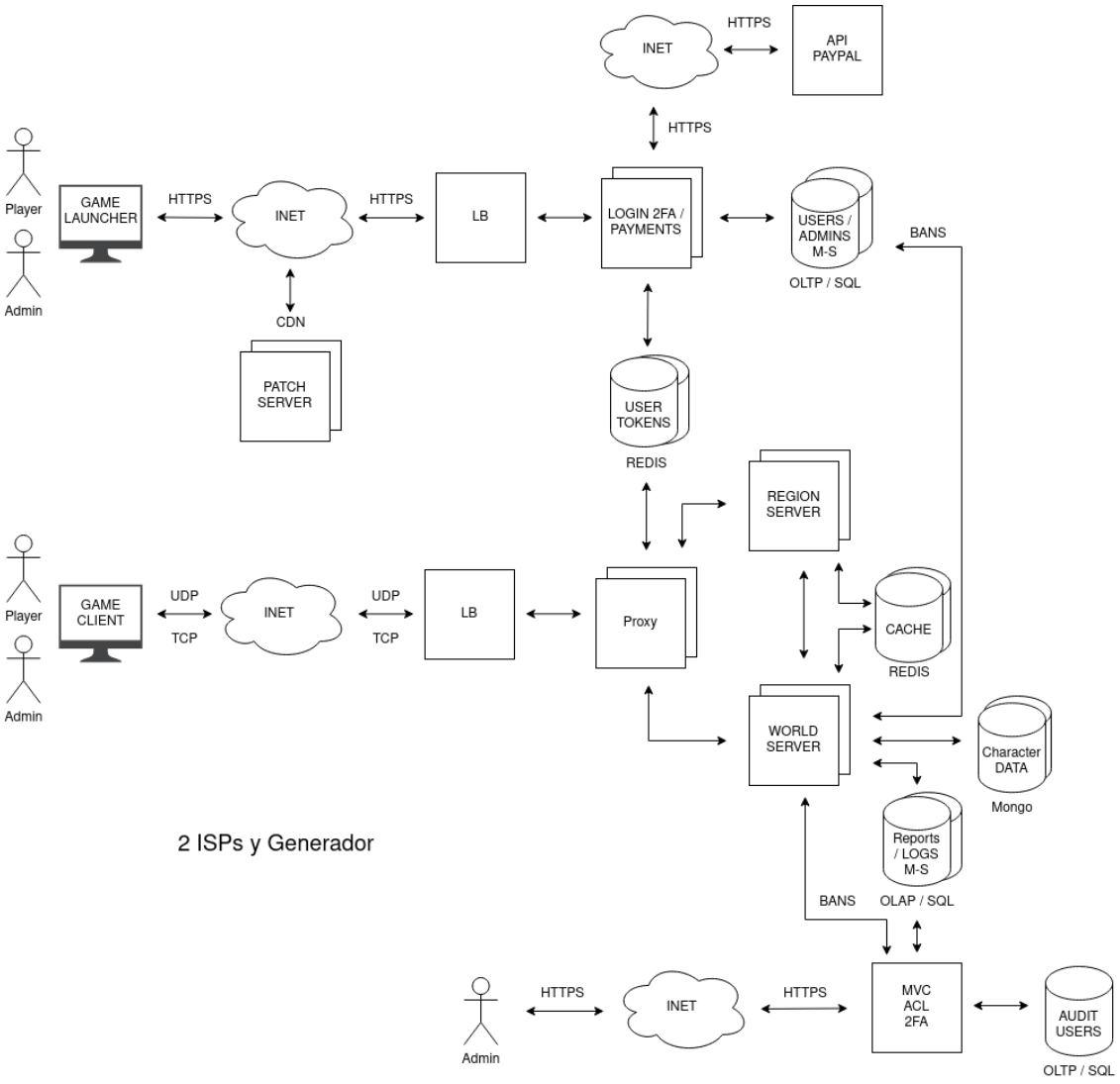

Figura 1: Arquitectura

Optamos por dividir la arquitectura en tres componentes principales:

La primera está dedicada a la actualización y autenticación, integrada por el patch server, el login server y el game launcher.

El juego en sí se implementa con un modelo cliente-servidor autoritativo, donde el servidor es la única fuente de verdad y supervisa toda la lógica del juego. En esta sección, se incluyen el game client, los proxy servers, los World servers y los Region servers.

Por último, se encuentra la sección de moderación, donde los administradores acceden a los informes de los usuarios para llevar a cabo acciones de moderación, como la imposición de sanciones.

La infraestructura se replicará en diversas ubicaciones geográficas para minimizar la latencia.

6.1. Componentes

A continuación, se detalla el papel específico de cada componente.

6.1.1. Clientes

- **Game Launcher:** Este componente establece la conexión con los servidores de inicio de sesión (login server) y los servidores de actualización (patch server) para facilitar el proceso de inicio de sesión y mantener el juego actualizado. Su función principal es gestionar el acceso del jugador al sistema y garantizar que la versión del juego esté siempre actualizada.

- **Game Client:** Este programa representa la interfaz principal con la que el jugador interactúa y experimenta el mundo del juego. Utiliza comunicación bidireccional con los servidores proxy para informar sobre las acciones del jugador y recibir actualizaciones sobre eventos en el entorno del juego, como acciones de NPCs, mobs y otros jugadores. Se utiliza el protocolo UDP para acciones críticas que requieren baja latencia, como movimientos y combates, mientras que las interacciones menos críticas, como comercio y chat, se realizan mediante TCP para garantizar la integridad de los datos. En la seccion de protocolos se profundizara sobre esto.

6.1.2. Servidores

- **Patch Server:** Este servidor se encarga de distribuir actualizaciones y parches a los jugadores. Colocado en una CDN para acelerar la descarga de archivos, garantiza que los jugadores siempre tengan la última versión del juego para acceder. Su función principal es optimizar el proceso de actualización y mantener a los jugadores en sincronía con las últimas características y correcciones.

- **Login Server:** Este es responsable de autenticar a los usuarios. También, verifica la validez de las suscripciones. Asegura que solo los usuarios autorizados tengan acceso al sistema, garantizando la integridad y seguridad de las cuentas de los jugadores. Una vez que un usuario ha sido autenticado con éxito, el Login Server le comunica al sistema del juego mediante un Session Token que usuario inicio sesión, estableciendo de esta manera una asociación única entre una sesión y un usuario específico.

- **Proxy Server:** Estos servidores gestionan las conexiones con los clientes, realizando tareas críticas como decodificación, desencriptación y descompresión de paquetes. Deben ser altamente escalables para manejar múltiples conexiones concurrentes de manera eficiente, actuando como intermediarios eficaces entre los clientes y los servidores especializados.

- **World Server:** Estos manejan de manera integral la información de un mundo específico, que incluye las posiciones de los jugadores, las conexiones a los Region servers y la gestión de aspectos como el chat, clanes, transacciones e inventarios. Estos servidores están diseñados para ser altamente escalables y eficientes en términos de rendimiento. En caso de una eventual caída de un World Server, se implementa una persistencia frecuente de la información para minimizar la pérdida de progreso del usuario.

  Además, los World servers tienen la responsabilidad de orquestar la cantidad de Region servers que estarán disponibles en su mundo de acuerdo con la demanda actual. También responden activamente a la posible caída de un Region server, iniciando la creación de uno nuevo para mantener la estabilidad y continuidad del servicio. Este enfoque garantiza una gestión dinámica y eficaz de los recursos para adaptarse a las variaciones en la demanda y garantizar una experiencia de juego ininterrumpida para los jugadores.

- **Region Server:** Múltiples Region Servers operan bajo cada World Server y manejan la lógica de interacción de los jugadores con el entorno y otros jugadores. Están asignados a áreas específicas del mapa. Además, acceden a un caché con información relevante del entorno, facilitando una recuperación más rápida en caso de caída. Este caché también sirve para que los Region servers intercambien información de la frontera entre áreas, asegurando una transición suave y minimizando la interrupción del juego. En caso de pérdida de un Region server, cada uno dispone de una instancia de respaldo que asume el control. Posteriormente, se crea una nueva instancia de respaldo, la cual puede acceder al caché para recuperar toda la información esencial, asegurando así la continuidad del servicio sin pérdida de datos.

- **Admin Server:** El Admin server implementa un diseño MVC sencillo con Control de Acceso (ACL) y Autenticación de Dos Factores (2FA). Los administradores pueden acceder de manera segura a los reportes de usuarios, que incluyen registros de chat, intercambios de ítems, asesinatos, etc. Esto les permite tomar decisiones informadas para mantener la armonía entre jugadores.

6.2. Persistencia

- **Base de datos de usuarios:** Esta base de datos se encarga de persistir información acerca de los usuarios, como su id, nombre real, correo electrónico, contraseña (hasheada), etc. Dado que no requiere de constantes actualizaciones, se decidió que la misma será un Postgres SQL OLTP con replicación activo-pasivo puesto que una caída de esta base de datos implica que ningún jugador podrá iniciar el juego en ese periodo. Esto nos garantiza alta disponibilidad y escalabilidad, lo cual es importante para esta base de datos en especial, porque se accederá cada vez que un usuario abra el launcher para hacer login. Tomamos la decisión de no hacer sharding a esta base de datos ya que esto no sería necesario considerando nuestro supuesto de que ocupará alrededor de 6TB.

- **Base de datos de personajes:** Esta base de datos es responsable de persistir información acerca del personaje de cada usuario (puede haber más de un personaje por usuario)

  Algunos de los datos almacenados incluyen:

  - Identificador
  - Nombre del personaje
  - ID de usuario asociado de la BD de usuarios
  - Clase
  - Apariencia
  - Inventario
  - Nivel
  - Skills
  - Misiones completadas
  - Misiones en progreso
  - Pertenencia a clanes (lista de identificadores)
  - Última posición en el mundo
  - Amigos
  - Logros
  - Estados (fear, staggered, blind, entre otros)

  Además, se persiste la siguiente información de los clanes:

  - Identificador de clan
  - Nombre
  - ID de líder
  - Logo
  - Colores asociados

Dicha base de datos será MongoDB, ya que debe soportar constantes lecturas y escrituras con consistencia, además de un volumen de información muy grande. Elegimos utilizar MongoDB sobre una base de datos relacional con sharding debido a que el sharding de MongoDB se implementa de forma transparente y no requerimos todas las ventajas que ofrece el paradigma relacional para persistir la información de los personajes y clanes. Utilizaremos sharding por el ID de usuario, ya que podemos aprovechar fuertemente el principio de localidad.

- **Base de datos de moderadores:** Esta base de datos se encarga de persistir los usuarios pertenecientes a moderadores para el servidor de admins. Solo se guardaran las cosas básicas necesarias para hacer login con 2FA, entre ellas el id de moderador, email y contraseña. Dicha base de datos será un Postgres OLTP. Esta decisión no nos afecta mucho en performance, dado que la cantidad de moderadores de un juego es un porcentaje muy bajo de la base de usuarios del mismo, y la información guardada por moderador también es muy baja. Las contraseñas de esta base estarán hasheadas con el algoritmo SHA-256.

- **Base de datos de logs:** Esta base de datos registrará los incidentes reportados por los jugadores, facilitando su análisis por parte de los moderadores. Algunos de los datos almacenados incluyen los chats del usuario reportado previos al incidente, el usuario que realiza el reporte, el timestamp del reporte, y otra información contextual como intercambio de items y asesinatos entre jugadores. Todos estos datos estarán vinculados a través de un identificador de caso. Además, se registrarán las resoluciones de los casos, incluyendo un timestamp y la penalidad aplicada al jugador reportado.

  Para esta base de datos, hemos optado por utilizar un sistema Postgres SQL OLAP, ya que el volumen de datos no requiere el uso de tecnologías no relacionales. La base de datos estará replicada bajo un esquema activo-pasivo, garantizando disponibilidad y escalabilidad.

- **Almacenamiento temporal:** En nuestro sistema, utilizamos Redis Cluster para desempeñar el papel de memoria para los Region Servers. Esta configuración asegura que todos los Region Servers tengan acceso constante a una memoria compartida con velocidades de acceso muy altas. Esto les permite funcionar de manera stateless, brindando una excelente escalabilidad. Además, esta arquitectura nos permite agregar o eliminar instancias de Region Servers según sea necesario, sin inconvenientes. También garantiza una rápida recuperación en caso de la caída de una instancia de Region Server.

  Este clúster de Redis almacena información del juego, como las posiciones de los jugadores, mobs y otros objetos, y gestiona eventos para intercambiar información entre los Region Servers y actualizar a los clientes. Se proporcionará una mayor profundización sobre este aspecto en la sección de puntos críticos más adelante.

  Asimismo, hemos implementado otro clúster de Redis para almacenar los Session Tokens, los cuales permiten a los Proxy Servers asociar los paquetes a los usuarios. También, se detallara este aspecto en la sección de puntos críticos posteriormente.

6.3. Decisiones de Implementación

6.3.1. Game Engine

Para el motor gráfico, consideramos diversas opciones:

- **Unreal Engine 4** un motor de Epic Games, que a lo largo de los años se volvió uno de los motores default para juegos AAA. Tiene varios sistemas para ayudar con el World Building (sistemas de paisaje, terreno, agua, cielo y iluminación), uno de los más interesantes es su sistema de world partition el cual nos ayudará con la separación de nuestros Region Servers. También incluye sistemas de animación y rendering. Al ser tan conocido e importante para la industria de videojuegos, tenemos el beneficio de que hay muchísimos desarrolladores que tienen mucha experiencia en él.

- **Unreal Engine 5** es un motor de la misma compañía que UE4 que tiene la tecnología crítica para las consolas de la nueva generación. Se volvió crucial en la ayuda del diseño 3D de mundos, con ayuda de su sistema geométrico virtualizado llamado Nanite y Lumen su sistema de iluminación. También da un enorme kit de animación, mayor que su pasada versión UE4, y una nueva feature es un sistema para manejar audio llamado MetaSounds. UE5 permite tamaños de archivo mucho mayores para poder crear tranquilamente juegos AAA para las nuevas consolas y llega a poder ser utilizado para películas. En comparación a su versión anterior (UE4), este más poderoso y avanzado a costo de ser más demandante. También vale la pena aclarar que este motor es relativamente nuevo por lo que va a haber menor cantidad de desarrolladores capacitados.

- **Unity** de la compañía Unity Technologies es también uno de los grandes pilares de motores gráficos de la industria de videojuegos. Tiene features para creación de juegos 2D y 3D, con sistemas de animación y world building. Cuenta con la posibilidad de crear juegos para Mobile y VR si se lo quisiera, como también la posibilidad de crear experiencias AR. Este motor está enfocado a ser lo más versátil posible para poder crear juegos de todo tipo y tiene una gran cantidad de desarrolladores como también una comunidad considerable.

- **Godot** es un motor gráfico argentino creado por Juan Linietsky y Ariel Manzur. Entre sus características se encuentra que es Open Source, contiene sistemas de creación de juegos 2D y 3D, como también para Mobile y XR (AR y VR). Cuenta con sistemas de renderización, shading y animación. Una desventaja que tiene este motor gráfico es el bajo número de desarrolladores que tiene.

Consideramos la posibilidad de desarrollar nuestro propio motor gráfico, pero esta opción implica una inversión significativa tanto en términos financieros como de tiempo de desarrollo. Para optimizar recursos, decidimos utilizar Unreal Engine 4 como el motor gráfico para nuestro MMO, basándonos en diversas razones. Este motor destaca en la creación de juegos AAA, alineándose con la escala y ambiciones de nuestro proyecto, a diferencia de Unity y Godot, que abarcan una mayor variedad de géneros de videojuegos.

Unreal Engine 4 ofrece sistemas y características que consideramos altamente beneficiosos para el desarrollo de nuestro videojuego. Aunque UE5 presenta nuevas características,

las percibimos como excesivas para nuestros objetivos actuales, además de contar con una menor cantidad de desarrolladores expertos disponibles en comparación con UE4.

6.3.2. Anti Cheat

Entre las alternativas evaluadas para implementar un sistema antitrampas en nuestro MMO, destacan Easy Anti-cheat, SARD Anti-Cheat y BattlEye, cada una con sus propias fortalezas y debilidades.

- **Easy Anti-cheat**, desarrollado por Epic Games, goza de confianza en la industria del gaming y ofrece capacidades de monitoreo en tiempo real para identificar y eliminar trampas, hacks y modificaciones no autorizadas. A pesar de contar con un panel administrativo fácil de usar y análisis accionables, carece de módulos de inteligencia artificial, limitando su efectividad contra bots impulsados por IA. Además, actualmente ha suspendido los servicios de soporte en tiempo real.
- **SARD Anti-Cheat** se distingue por combinar métodos de protección basados en el kernel con innovadores módulos de inteligencia artificial y aprendizaje automático. Destaca por su conjunto de herramientas altamente personalizable, permitiendo a los desarrolladores adaptar la protección a los requisitos únicos de cada juego. Sus pros incluyen módulos de IA y ML para una protección exhaustiva y una tasa de falsos positivos inferior al 0.001 %, preservando la integridad del juego con un impacto mínimo en la jugabilidad. Sin embargo, su principal desventaja es su costo, siendo una solución más cara.
- **BattlEye**, también emplea un enfoque basado en el kernel que utiliza tanto técnicas de detección específicas como heurísticas para maximizar su efectividad. Aunque ofrece soluciones personalizables y un sistema defensivo proactivo, carece de módulos de inteligencia artificial y tiene un soporte limitado, ofreciendo solo documentación y una sección de preguntas frecuentes.

Después de un análisis, se ha elegido a SARD Anti-Cheat. A pesar de su mayor costo, esta solución proporciona características importantes para nuestro MMO. Destaca por su menor cantidad de falsos positivos, esencial para la retención de nuestra base de jugadores. Además, su continua evolución se centra en la detección de diversos tipos de bots en juegos MMO, utilizando métodos que incluyen la detección de anomalías visuales y la distinción entre el comportamiento humano y el de un bot mediante la detección de control de mouse y teclado.

6.3.3. Servicio de Pago

Al ser un juego pago, es necesario un servicio de pagos que se encargue de procesar las transacciones de manera confiable y segura. Se analizaron las opciones de crear un sistema propio o utilizar uno ya existente.

Inicialmente, se descartó la idea de crear un sistema de pagos propio, dado que esto incurriría en altos costos, retrasos en el desarrollo del juego, y posibles problemas de seguridad, dado que una empresa de desarrollo de videojuegos no se especializa en asuntos contables o de seguridad informática.

Posteriormente, evaluamos dos de las opciones más destacadas: PayPal y Stripe. Optamos por PayPal, ya que está disponible en un mayor número de países en comparación con Stripe (200 países para PayPal y 145 para Stripe). Además, según datos de la página web de PayPal, el 26 % de los jugadores a nivel mundial ya ha utilizado este servicio, generando confianza en este método de pago.

Es importante mencionar que esta elección también conlleva algunos aspectos negativos, ya que PayPal aplica comisiones que oscilan entre el 1.5 % y el 3 % para todas las transacciones, lo que podría traducirse en costos significativos a lo largo del tiempo.

6.3.4. Protocolos de Comunicación

TCP se utilizará para acciones que no sean en tiempo real, aquellas que sean más "transaccionales", como el uso del chat o el intercambio de objetos ya sea con NPCs o jugadores. Al realizar estas actividades, no es crucial reducir al mínimo la latencia, pero es de mayor importancia la certeza del orden e integridad de la información. Además, los mismos serán encriptados para evitar ataques del tipo "man-in-the-middle".

UDP se utilizará para la transmisión de inputs por parte del game client y la recepción del estado de game objects por parte de los Region Servers. Al tener menos overhead que TCP, nos permitirá tener menor latencia, lo que es de suma importancia para la comunicación en tiempo real. Con el fin de mitigar la pérdida de paquetes que puede llegar a tener UDP, cada paquete estará formado por el input actual y **N** inputs anteriores (los que entren en el datagrama) junto con sus timestamps (para evitar correr comandos ya ejecutados). De esta forma, es posible reconstruir el input del usuario sin importar si algún paquete se pierde en el camino. Debido a que aun así una mínima latencia es inevitable, el cliente realizará predicciones de acciones mientras espera la respuesta del servidor (spatial interpolation), permitiendo mostrar animaciones fluidas.

HTTPS se utilizará al comunicarse desde el game launcher con el login server y además para la comunicación por parte de los audit users con el servidor MVC de reportes. Como la información se encripta de punta a punta, garantizamos evitar ataques del tipo "man-in-the-middle", incrementando de esta forma la seguridad de las cuentas.

6.3.5. Lenguajes de Programación

Se tuvieron en cuenta distintos lenguajes de programación para los distintos servidores teniendo en cuenta sus requerimientos.

- **Login Server, Patch Server, Admin Server (Java)**: Java se elige para estos componentes debido a su robustez, facilidad de desarrollo, amplio soporte de librerías y una velocidad adecuada. Java es comúnmente utilizado en este tipo de entornos, lo que facilita el desarrollo y mantenimiento de estos servidores. Además, la elección de Java permite utilizar el framework de Spring MVC para el servidor de Admins. Es importante destacar que, dado que los requisitos para estos servidores son más laxos, existen varias alternativas válidas, como lo son Node, Ruby y C#. Nos hemos inclinado por Java debido a su amplio uso en la industria y la disponibilidad de numerosos desarrolladores

- **Proxy Server (Rust)**: Rust ha sido seleccionado como el lenguaje para el Proxy Server debido a su capacidad para manejar alta concurrencia y su enfoque en la seguridad. Aunque Rust no es un lenguaje funcional puro, su énfasis en la verificaciones en tiempo de compilación lo convierte en una opción sólida. Se consideraron alternativas como Erlang, un lenguaje funcional probado en la industria conocido por su manejo de la concurrencia. Sin embargo, la elección de Rust se fundamenta en su compilación a código nativo, lo que proporciona la alta performance necesaria para este servidor. Aunque Rust puede no tener la misma extensa historia en la industria que Erlang, se considera un lenguaje maduro y listo para producción.

- **Region Server, World Server (C++)**: Con respecto a los servidores críticos, como el Region Server y el World Server, se requiere un lenguaje de bajo nivel para optimizar su rendimiento al máximo. En consideración a esto, se evaluaron opciones como C, C++ y Rust. C fue descartado a causa de su falta de herramientas, lo cual dificulta la programación en dicho lenguaje. Entre Rust y C++, se optó por C++ debido a su excelente rendimiento y a las funcionalidades que facilitan el desarrollo. Además, C++ es ampliamente utilizado en la industria del desarrollo de videojuegos, lo que garantiza la disponibilidad de numerosos desarrolladores. Por último, pero no menos importante, la elección de C++ facilitará la integración con el código de Unreal Engine, el motor gráfico seleccionado.

6.4. Resolución de Atributos

6.4.1. Disponibilidad

El Patch Server se encuentra en una CDN para optimizar la distribución de actualizaciones. En cuanto al Login Server, se implementa con múltiples instancias respaldadas por un load balancer para garantizar la disponibilidad; en caso de que una instancia falle, otra asume automáticamente su función. La base de datos de usuarios utiliza un sistema SQL OLTP con replicación activo-pasivo, lo que significa que, si una instancia falla, otra toma su lugar de manera inmediata, asegurando la continuidad del servicio.

Los Proxy Servers, esenciales para gestionar las conexiones con los clientes, están replicados detrás de un load balancer. Esto proporciona redundancia activa, permitiendo que otro proxy server intervenga en caso de una falla, manteniendo así la estabilidad en las comunicaciones con los clientes.

Cada World Server, representando un mundo específico del juego, tiene un enfoque de tolerancia a fallos. Si un World Server experimenta una caída, los jugadores se desconectan temporalmente, pero tienen la opción de reconectarse a otro World Server sin pérdida significativa de progreso, ya que la información crítica se persiste regularmente. Además, la base de datos de personajes, basada en MongoDB, utiliza replicación para garantizar la disponibilidad.

Los Region Servers, responsables de regiones específicas del mapa, cuentan con un sistema de recuperación ante fallos. En caso de una caída su respaldo toma el mando y el World Server orquesta la creación de una nueva instancia de respaldo, que recupera su estado anterior gracias a la información almacenada en el caché. El uso de Clusters de Redis para el caché también asegura disponibilidad mediante replicación.

Para mitigar problemas por cortes en el servicio de internet y de luz, contamos con múltiples proveedores de internet en nuestros datacenters y disponemos de generadores de electricidad para garantizar la disponibilidad.

6.4.2. Performance

El rendimiento en la arquitectura se optimiza a través de varias estrategias clave. Los Servidores Proxy desempeñan un papel crucial al eliminar la carga de overhead asociada con la comunicación cliente-servidor, permitiendo que los World y Region Servers respondan a los clientes con una menor latencia. La elección del protocolo UDP contribuye a esta eficiencia al ser una opción más liviana y rápida.

La división del mundo en Region Servers, cada uno responsable de áreas específicas, se traduce en una reducción de la latencia al distribuir la carga en múltiples instancias. En los Proxy, Region y World server, donde el rendimiento es crítico, se emplean lenguajes de bajo nivel que aseguran una alta performance.

La utilización de servidores geográficamente distribuidos contribuye a minimizar la latencia y mejorar el rendimiento para jugadores en diferentes regiones. Esto asegura una experiencia de juego fluida y sin demoras significativas.

6.4.3. Escalabilidad

La escalabilidad se garantiza de varias formas. El Login Server cuenta con un Load Balancer que opera en conjunto con replicación activa, asegurando así la distribución equitativa de la carga. La base de datos de usuarios utiliza un enfoque de replicación activo-pasivo para escalar las lecturas.

En el caso del Patch Server, se aprovecha una CDN para distribuir las actualizaciones de manera eficiente y permitiendo una escalabilidad flexible según las necesidades de los jugadores.

La escalabilidad de los Proxy Servers se logra mediante la implementación de Load Balancers y replicación activa, garantizando una distribución uniforme de las solicitudes entre múltiples instancias para mantener un rendimiento óptimo. La utilización de Redis para almacenar los Session Tokens contribuye a la escalabilidad al hacer que estos servidores sean stateless, facilitando una expansión sencilla y eficiente de recursos según las demandas del sistema.

Para manejar la creciente demanda de jugadores, la arquitectura incorpora múltiples World Servers, distribuyendo a los jugadores en los mismos. MongoDB, utilizado para almacenar datos de personajes, es altamente escalable, adaptándose eficientemente a un aumento en el número de usuarios.

La distribución de carga en los Region Servers, junto con la elección de Redis como sistema de caché, contribuye significativamente a la escalabilidad. Redis, conocido por su capacidad para manejar grandes volúmenes de datos con baja latencia, garantiza un rendimiento constante incluso en escenarios de alta demanda.

6.4.4. Mantenibilidad

Hemos tomado diversas decisiones para garantizar la mantenibilidad y permitir un desarrollo eficiente de nuevos contenidos. La implementación de Patch Servers asegura la entrega eficiente de nuevas versiones a los usuarios, facilitando la actualización y la introducción de mejoras continuas en el juego. Además, se establece un énfasis significativo en la calidad del código mediante un alto nivel de cobertura de pruebas. La incorporación de pruebas exhaustivas minimiza la introducción de errores en el código, mejorando la robustez del sistema y facilitando la identificación y corrección de posibles problemas en etapas tempranas del desarrollo.

Para mantener la consistencia y calidad del código, se emplearán herramientas de análisis estático de código, lo que incluye la aplicación de estándares y formatos específicos. Esta práctica contribuye a la uniformidad y legibilidad del código, facilitando la colaboración.

Se implementará un pipeline de CI que incluirá las pruebas automatizadas y el análisis estático de código, asegurando que solo se agreguen cambios que hayan pasado exitosamente todas las pruebas, minimizando así la posibilidad de errores.

Finalmente, se hará especial énfasis en mantener pull requests con un número manejable de líneas de código, lo que simplifica el proceso de revisión de código y facilita la identificación y corrección de problemas antes de la integración en el código principal.

6.4.5. Seguridad

Se utiliza HTTPS para asegurar la comunicación entre el launcher y los servidores de inicio de sesión y de parches. Además, se garantiza la encriptación de los paquetes TCP entre el game client y el proxy server. El sistema de Session Tokens se configura para requerir credenciales válidas, evitando que usuarios no autorizados accedan al juego. Los jugadores opcionalmente pueden usar 2FA, pero los administradores lo **deben** utilizar.

Para prevenir trampas y la presencia de bots, se introduce un sistema anti-cheat en el juego. Este sistema trabaja para mantener la integridad del juego y garantizar una experiencia justa para todos los jugadores.

En cuanto a la seguridad del servidor de logs para administradores, se establece un Control de Acceso (ACL) con autenticación de dos factores (2FA). Además, la comunicación con este servidor se cifra mediante HTTPS, proporcionando una capa adicional de seguridad.

En el ámbito de la base de datos, se implementa un sólido enfoque de seguridad. Las contraseñas se almacenan de forma segura utilizando el algoritmo SHA-256, conocido por su robustez. Además, se asegura la ubicación física de las bases de datos en instalaciones propiedad de la empresa, equipadas con cámaras de vigilancia y personal entrenado, garantizando la seguridad física de la información almacenada.

6.4.6. Usabilidad

La usabilidad de nuestro juego es esencial para evitar frustraciones por parte de los jugadores. Para garantizar una interfaz amigable y efectiva, hemos establecido un enfoque estructurado:

Iniciaremos contratando a un equipo de diseñadores especializados que estudiarán detenidamente las interfaces de juegos destacados, como World of Warcraft y Final Fantasy XIV. Este análisis proporcionará insights valiosos sobre las características compartidas y las mejores prácticas en el diseño de interfaces de juegos similares.

Una vez adquirido un conocimiento detallado de estas interfaces, comenzaremos a generar varios bocetos que representen posibles diseños para nuestro juego. Cada pantalla tendrá múltiples variaciones, explorando diversas opciones.

Posteriormente, organizaremos un grupo de personas diverso, compuesto por jugadores con diferentes niveles de experiencia (desde principiantes hasta expertos en este tipo de juegos). Este grupo evaluará cada boceto y proporcionará opiniones valiosas sobre su coherencia y usabilidad general.

Con base en el feedback recopilado en el paso anterior, ajustaremos y mejoraremos los bocetos, y procederemos a la fase de programación de las interfaces seleccionadas.

Una vez que las interfaces estén programadas, convocaremos nuevamente al grupo de evaluadores para realizar pruebas prácticas y determinar si se pueden realizar mejoras adicionales para optimizar la usabilidad.

6.4.7. Auditabilidad

La auditabilidad del sistema se garantiza mediante un proceso de reportes, el cual le otorga la capacidad a los administradores para analizar eventos y tomar decisiones informadas.

En el sistema, los chats no se persisten automáticamente, permanecen en la memoria del World Server. Sin embargo, cuando un usuario reporta a otro, se persiste automáticamente el historial de mensajes del usuario reportado, junto con otros datos contextualizadores como registros de asesinatos e intercambios entre jugadores, proporcionando una visión completa de la situación reportada.

Este historial de chats y datos contextualizados puede ser analizado por los administradores a través del Admin Server. Esta interfaz permite a los administradores revisar y evaluar los eventos reportados para tomar decisiones sobre posibles acciones, como aplicar prohibiciones a los jugadores involucrados.

Además, los administradores tienen la capacidad de ingresar al sistema con un personaje en "god-mode" para monitorear en tiempo real las interacciones en el mundo del juego. Esto proporciona una capa adicional de control y visibilidad, permitiendo a los administradores intervenir y tomar medidas apropiadas según sea necesario.

En conjunto, este enfoque asegura que los eventos críticos, como reportes y acciones administrativas, sean registrados y estén disponibles para su revisión, garantizando así la auditabilidad del sistema y permitiendo una respuesta adecuada a situaciones problemáticas en el entorno del juego.

7. Puntos Críticos

A continuación se detalla cómo interactúan los distintos componentes para resolver puntos críticos del sistema.

7.1. Autenticación

Cuando un usuario inicia sesión, el Login Server verifica tanto el nombre de usuario como la contraseña. Posteriormente, se realiza la validación del estado de pago del usuario y se verifica que no esté sujeto a un baneo.

Una vez completadas exitosamente todas estas validaciones, el Login Server genera un token de sesión vinculado a la cuenta del usuario. Este token se almacena de forma segura en el sistema Redis de User Tokens, estableciendo una asociación entre el token y la identidad del usuario. Simultáneamente, el token se envía de manera segura al Launcher del juego, donde se almacena localmente para su posterior lectura por parte del Game Client.

El Game Client incorpora este token en cada paquete que envía al servidor, permitiendo su identificación. En el lado del servidor, el Proxy Server valida el token en cada solicitud y lo asocia al jugador si es válido. Esta metodología asegura que los Region Servers y World Server siempre reciban paquetes de jugadores autenticados e identificados.

Al cerrar sesión, el token se destruye, y todos los tokens tienen un tiempo de expiración. Cuando un token está próximo a expirar, el cliente y el Proxy Server pueden negociar un nuevo token sin requerir que el jugador se autentique nuevamente, permitiendo así una transición fluida sin interrupciones en la experiencia del usuario.

7.2. Persistencia del Progreso

La persistencia de los avances de los jugadores en el juego es esencial para evitar pérdidas de progreso. Aunque se utiliza Redis como caché, es necesario tener un mecanismo de persistencia permanentemente que sea actualizado en intervalos no muy largos. Dada la considerable cantidad de jugadores prevista, se ha establecido un enfoque selectivo para determinar qué información se persiste en la base de datos de personajes y con qué frecuencia se actualiza.

Con el objetivo de aliviar la carga del sistema, se ha definido actualizar los datos en la base de datos de personajes después de eventos específicos:

- La ubicación del jugador se actualiza cada 1 minuto.
- El inventario se persiste cada vez que experimenta algún cambio.
- Una misión se marca como completa cuando un jugador la finaliza.
- Cuando un jugador se une a un clan.
- Al aceptar una misión.
- Cuando un jugador sube de nivel o aprende una habilidad.
- Al cerrar sesión o cuando el usuario pierde conexión.

Estos eventos son desencadenados por el World Server, asegurando que la información crítica de los jugadores se actualice de manera oportuna y eficiente en la base de datos de personajes. Este enfoque selectivo garantiza que solo los eventos relevantes y significativos sean persistidos, optimizando así el rendimiento del sistema.

7.3. Baneo de un Usuario

Los administradores tienen la capacidad de aplicar baneos a los usuarios mediante dos métodos: a través del Admin Server, analizando reportes de incidentes, o directamente desde el juego si observan que un jugador está actuando de manera indebida.

En el primer caso, al prohibir al jugador desde el administrador, este servidor notificará a los World Servers que el usuario ha sido baneado. En este punto, cada World Server verifica si el usuario está en sesión en ese servidor y lo expulsa inmediatamente si está presente. Posteriormente, el World Server indica en la base de datos de usuarios que el jugador ha sido baneado, evitando que el Login Server permita su inicio de sesión durante el período del baneo.

En el segundo caso, al banear al jugador desde el juego, el World Server donde se encuentra el jugador baneado lo expulsa de la sesión y notifica a la base de datos de usuarios sobre el baneo.

7.4. Interacción entre región servers

Mecanismo de anuncio de eventos

La comunicación de eventos entre los servidores se lleva a cabo mediante la implementación de un patrón de diseño de publicación/suscripción, para esto usamos el sistema Pub/Sub de Redis. Cada servidor de región se suscribe a los eventos en las fronteras de los servidores de región adyacentes, permitiendo así que los servidores se notifiquen mutuamente cuando se produce un evento cerca de sus límites compartidos.

Ahora bien, analicemos cómo los servidores comunican estos eventos a los clientes. Para cada jugador en la región de un servidor, el servidor desempeña dos funciones fundamentales:

1. Procesa las acciones recibidas del cliente.

2. Administra la suscripción o anulación de la suscripción del cliente para recibir eventos que ocurren en su "Área de Interés". Por ejemplo, cuando un jugador está en movimiento, el servidor suscribe al jugador a los eventos que están ocurriendo en su nueva área de interés. Si esta área abarca más de un servidor, el cliente recibe eventos de todos los servidores relevantes a esa área.

Eventos cerca de limites del servidor

Para gestionar de manera adecuada estos eventos, es esencial cumplir con los siguientes requisitos de consistencia:

1. El orden de los eventos que afectan el estado de cualquier objeto debe ser el mismo para todos los clientes.

2. Existe un orden global de los eventos que es coherente con el orden de los eventos que afectan a cualquier objeto. Para lograrlo, la secuencia numérica que identifica a los eventos será monótonamente creciente.

Cuando se produce un evento cerca del límite de un servidor, este lo ejecuta y luego notifica a los servidores que comparten ese límite a través del mecanismo de anuncios de eventos mencionado anteriormente. Una vez completado esto, procede a notificar a los clientes pertinentes de la misma manera, y estos realizan las acciones necesarias, como la actualización de la vista del jugador.

Si el cliente recibe un nuevo evento antes de haber concluido el manejo del evento anterior, simplemente lo encola. Un evento particular que merece mención especial es la transferencia de un objeto de una región de un servidor a otra región de otro servidor. Esto se lleva a cabo de la misma manera que se mencionó en el párrafo anterior, pero la ejecución del evento también implica la transferencia del objeto de un servidor a otro.

El escenario más complejo se presenta cuando el objeto es un jugador. Sin embargo, en este caso, no solo transferimos el objeto entre servidores, sino que también los eventos que ocurren mientras la transferencia está en proceso deben procesarse de manera normal. Esto asegura que, desde la perspectiva del jugador, todo continúe sin problemas.

8. Riesgos

- La caída de la BD de Audit Users implicaría el no funcionamiento del MVC de reportes ya que la misma no posee un esquema Master-Slave. Por ende, esto causaría no se podrían ver los logs mediante la interfaz, ni efectuar bans desde la misma hasta que se recupere.
- La falla de uno de los World Servers provocaría la desconexión inmediata e inevitable (con posibilidad de posterior reconexión) de todos los usuarios conectados al mundo.
- La caída del servicio de PayPal frenaría todos tipo de transacciones por parte de los usuarios. Al no ser nuestro propio método de pago, dependemos de cuán rápido solucionen ellos el inconveniente.
- Ambos load balancers (el utilizado para las conexiones con el Launcher y con el Client) presentan un single point of failure. Si alguno de estos falla, se finalizarán todas las conexiones asociadas. Tampoco se podrán establecer nuevas hasta el restablecimiento del servicio.

9. No Riesgos

- El robo de credenciales de un usuario administrador no presenta un riesgo ya que para iniciar sesión los administradores **deben** utilizar 2FA.
- En caso de que el World Server presente una falla, no se perderá progreso ya que la información importante fue persistida en la base de datos de characters. En caso de que haya datos más recientes del personaje en el caché Redis se utilizaran estos.
- Se evitan accesos indebidos a nuestras instalaciones gracias a la presencia de seguridad física (cámaras, personal de vigilancia, etc.).
- En caso de un acceso indebido a la base de datos, las contraseñas de los usuarios se encuentran cifradas, por lo que la misma no se verá vulnerada.
- Tener problemas en nuestros datacenters por falta de electricidad o conexión a internet, dado que contamos con un generador y dos conexiones a internet de alta velocidad de proveedores distintos.
- Una persona utiliza bots para beneficio propio a costa de otros jugadores. Este jugador es baneado gracias a SARD anti-cheat que contiene sistemas especializados para detección de bots.
- Se experimenta un aumento repentino en la demanda debido a una actualización anticipada. La escalabilidad del sistema facilita la adición rápida y sencilla de instancias adicionales de proxy, World y Region servers para hacer frente a este pico.
- PayPal es una de las plataformas de pago más seguras en el mercado, por lo que el robo de datos bancarios de los usuarios no presenta un riesgo.
- Un jugador exhibe un comportamiento inapropiado cuando no hay administradores supervisando el mundo. El sistema de reportes posibilita la aplicación de sanciones al jugador de manera asincrónica.
- Ataques de tipo “man-in-the-middle” no son posibles ya que los paquetes TCP y HTTPS se encuentran encriptados a la hora de ser enviados.

10. Tradeoffs

Usabilidad vs. TTM

El desarrollo de las interfaces requiere de hacer bocetos de alta fidelidad, e interactuar con grupos de personas para obtener feedback en dos ocasiones. Este proceso nos asegura tener una interfaz que sea fácilmente usable, pero hace que se extienda el time to market.

Seguridad vs. Usabilidad

La utilización de 2FA nos aporta mayor seguridad. Es importante evitar que existan accesos malintencionados a las cuentas de nuestros jugadores moderadores, porque podría haber graves consecuencias, como el baneo injusto de jugadores u otros factores que hagan que los usuarios abandonen el juego. Sin embargo, esto reduce la usabilidad y experiencia de usuario, dado que el moderador debe abrir su casilla de correo para hacer login.

Performance vs. Mantenibilidad

Se utilizarán lenguajes de bajo nivel, tales como C++ para el desarrollo de los Region y World server. Esto garantiza que los mismos serán más rápidos y eficientes en recursos. Sin embargo, usar lenguajes de bajo nivel puede reducir la mantenibilidad del código, un caso serían los accesos inseguros a memoria que permite C++ pueden ocasionar bugs y memory leaks. En el caso del Proxy server, utilizamos Rust. Este lenguaje tiene una performance muy alta, similar a la de C, pero con más seguridad a la hora de acceder a memoria. Sin embargo, sigue siendo más difícil de programar y mantener que otros lenguajes de más alto nivel.

Costo vs. Disponibilidad

Dado que la cantidad de moderadores de nuestro juego será baja, decidimos no colocar un load balancer para el MVC de moderadores. Esta elección contribuye a reducir los costos operativos, evitando la necesidad de mantener instancias adicionales de un load balancer y múltiples instancias de nuestro MVC, ya que esta actividad no requiere acción inmediata. Aún así, esto podría llegar a ocasionar problemas de disponibilidad si nuestro servidor que hostea el MVC se cae.

Disponibilidad vs. Costo

Para garantizar la máxima disponibilidad del sistema, hemos tomado medidas como mantener varias instancias de nuestros servidores, replicaciones de bases de datos, contar con dos proveedores de servicios de internet y disponer de un generador eléctrico. Aunque estas acciones aumentan los costos operativos, consideramos que la inversión justifica la mejora significativa en la disponibilidad del sistema.

Seguridad vs. Costo

La elección de PayPal como método de pago aporta significativamente a la seguridad, ya que la plataforma prioriza la confidencialidad de la información del usuario. Además,

el hecho de no tener que almacenar datos de pagos en nuestras bases de datos reduce la responsabilidad y nos permite centrarnos en el desarrollo del juego en lugar de invertir recursos en la creación de un medio de pago propio. No obstante, es importante tener en cuenta que a medida que pasa el tiempo, los costos asociados con el uso de esta plataforma pueden acumularse y llegar a representar un gasto considerable. Al aumentar la cantidad de jugadores, el costo que se deberá afrontar a causa de las comisiones de PayPal aumentará de manera proporcional.

11. Referencias

- A Distributed Architecture for MMORPG - Marios Assiotis and Velin Tzanov
- MMO Architecture: Source of truth, Dataflows, I/O bottlenecks and how to solve them
- How does any MMO Games backend work? - Narendra Lakshmana Gowda
- Online games System design backend
- A Journey Into MMO Server Architecture
- Gaming Anti-Cheat Solutions To Consider For Your Game
- SARD Battling MMO game bots
- MMORPG and the ol' UDP vs TCP
- Is the TCP protocol good enough for real-time multiplayer games?
- Why aren´t top games (Battlefield, WoW, LoL) created in other languages other than C++? Why not Java or C#?
- What are the best concurrent programming languages?
- Persistence in Massively Multiplayer Online Games
- How often to save player's state in persistent online games?
- Unreal Engine
- Unity
- Godot Docs
- Redis Pub/Sub