



UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA

## Trabajo Profesional

*Decentralized Cloud Gaming Rental Service*

### Integrantes

#### Alumnos

Caserio, Juan Cruz - 104927 - jcaserio@fi.uba.ar

Gazzola, Franco - 105103 - fgazzola@fi.uba.ar

Rial, Tadeo Ezequiel - 104677 - trial@fi.uba.ar

Kelman, Axel - 104379 - akelman@fi.uba.ar

#### Tutor

Mendez Mariano - marianomendez@gmail.com

# Índice

<b>1. Resumen</b>	<b>3</b>
<b>2. Palabras Clave</b>	<b>3</b>
<b>3. Abstract</b>	<b>4</b>
<b>4. Keywords</b>	<b>4</b>
<b>5. Glosario</b>	<b>5</b>
<b>6. Introducción</b>	<b>6</b>
<b>7. Estado del arte</b>	<b>7</b>
<b>8. Problema detectado y faltante</b>	<b>9</b>
8.1. Problema detectado . . . . .	9
8.2. Faltante . . . . .	9
<b>9. Solución Implementada</b>	<b>10</b>
9.1. Alcance . . . . .	10
9.2. Arquitectura General . . . . .	10
9.3. Nodo de procesamiento . . . . .	13
9.3.1. Transmisión de datos: WebRTC . . . . .	13
9.3.2. Gstreamer . . . . .	18
9.3.3. Captura de audio y vídeo . . . . .	20
9.3.4. Reproducción de audio y vídeo . . . . .	21
9.3.5. Captura y reproducción de entrada de usuario . . . . .	22
9.3.6. Concurrencia y Sincronización: Tokio . . . . .	23
9.4. Servidor central . . . . .	23
9.4.1. ABM de Usuarios . . . . .	24
9.4.2. Comunicación con WebSockets . . . . .	24
9.4.3. Base de datos . . . . .	24
9.4.4. Autenticación con Firebase . . . . .	25
9.4.5. Integración de MercadoPago . . . . .	25
9.5. Interfaz de usuario . . . . .	26
9.6. Carga de Horas . . . . .	27
9.7. Automatización . . . . .	28
<b>10. Metodología aplicada</b>	<b>29</b>
10.1. Backlog . . . . .	29

10.2. Roles . . . . .	29
10.3. Registro de actividades . . . . .	29
<b>11. Experimentación y validación</b>	<b>31</b>
11.1. Medición de Latencia de Red entre pares . . . . .	31
11.2. Medición de Latencia en Gstreamer . . . . .	32
11.3. Resultados Obtenidos . . . . .	32
11.3.1. Primer escenario . . . . .	32
11.3.2. Segundo escenario . . . . .	35
<b>12. Cronograma</b>	<b>39</b>
<b>13. Riesgos y lecciones aprendidas</b>	<b>40</b>
13.1. Elección del lenguaje de programación . . . . .	40
13.2. Implementación de captura, transmisión y reproducción multimedia . . . . .	41
<b>14. Trabajos Futuros</b>	<b>42</b>
14.1. Captura de ventana de juego . . . . .	42
14.2. Controles y compatibilidad . . . . .	42
14.3. Extracción de dinero . . . . .	42
14.4. Multiplataforma . . . . .	42
14.5. Mejoras en el servidor . . . . .	42
14.6. Otras mejoras . . . . .	43
<b>15. Conclusiones</b>	<b>44</b>
<b>16. Referencias</b>	<b>45</b>
<b>17. Anexos</b>	<b>48</b>
17.1. Enlaces a los Repositorios e Instalador . . . . .	48
17.2. Protocolos . . . . .	48
17.3. Diseño y Accesibilidad . . . . .	50
17.3.1. Autenticación . . . . .	50
17.3.2. Menu . . . . .	52
17.3.3. Sesión de juego . . . . .	52
17.3.4. Agregar Juego . . . . .	54
17.3.5. Ofrecer Juego . . . . .	56
17.3.6. Cargar creditos . . . . .	57

## 1 Resumen

El siguiente documento presenta el informe final de nuestro Trabajo Profesional grupal para la carrera de Ingeniería Informática de la Universidad de Buenos Aires. El objetivo de este trabajo es desarrollar un servicio de Cloud Gaming descentralizado que permita a los usuarios ofrecer sus computadoras en alquiler o alquilar una para jugar videojuegos.

Expondremos nuestra solución desarrollada, enfocándonos en mejorar los aspectos de latencia, escalabilidad y usabilidad respecto a las alternativas actuales. Además, presentaremos un innovador modelo de negocio que permite a los usuarios monetizar sus recursos de hardware inutilizados. Todo esto se implementa a través de una interfaz accesible y fácil de usar.

## 2 Palabras Clave

Juegos en la Nube; WebRTC; Gstreamer; Descentralización; Alquiler de Hardware; P2P; Videojuegos.

### 3 Abstract

This document aims to present our Professional Work final report for the Computer Engineering career at the University of Buenos Aires. The objective of the project is to develop a decentralized Cloud Gaming service that allows users to either rent out their computers or rent one to play videogames.

We will expose our developed solution, focusing on improving latency, scalability, and usability in comparison to existing alternatives. Additionally, we will introduce an innovative business model that enables users to monetize their unused hardware resources. The implementation relies on a accessible and user-friendly interface.

### 4 Keywords

Cloud Gaming; WebRTC; Gstreamer; Decentralization; Hardware Rental; P2P; Gaming; Videogames.

## 5 Glosario

- Bitrate: Medida en bits por segundo (bps), que describe la cantidad de datos procesados o transmitidos en una unidad de tiempo. En el contexto de vídeo y audio, un bitrate más alto generalmente significa mejor calidad.
- Conexión Peer-to-Peer (P2P): Conexión directa entre dos sistemas terminales.
- FPS (Frames per second): La cantidad de fotogramas que se muestran por segundo en una pantalla.
- Latencia: El tiempo que demora un paquete de datos en viajar desde un origen hasta un destino.
- Offerer: Quien presta su computadora para el servicio generando ingresos.
- Peer: Un participante en una conexión de tipo peer to peer.
- QoE (Quality of Experience): Medida subjetiva de la satisfacción del usuario con un servicio, que tiene en cuenta factores como la calidad del vídeo, la latencia y la estabilidad de la conexión.
- Receiver: Quien utiliza el servicio para jugar.

## 6 Introducción

En los días que corren los videojuegos forman una gran parte de la vida de millones de personas. Muchos de estos requieren cierto nivel de tecnología de hardware que a veces puede no ser accesible para todo el mundo. Frente a este problema aparece Cloud Gaming como una posible solución. Se trata de un servicio que permite a los usuarios disfrutar de videojuegos sin la necesidad de contar con un equipo de alto rendimiento.

En este proyecto, pretendemos desarrollar un servicio de Cloud Gaming mediante el cual los usuarios puedan tener acceso a equipos con el fin de disfrutar o probar sus títulos favoritos. Para esto, buscamos proveer un sistema de alquiler en el que aparecen dos actores. Por un lado, los usuarios arrendadores, quienes ponen sus equipos a disposición de otros a cambio de una tarifa preestablecida. Por otro lado, los usuarios arrendatarios, quienes alquilan estos equipos por períodos de tiempo determinados.

Para que nuestro servicio sea elegible, este deberá brindar al usuario una alta QoE (Calidad de Experiencia). Esto se debe a que, si bien puede ser variable según el tipo, se estima que para que la experiencia de juego sea correcta, la latencia se debe encontrar entre 0-100 ms [1]. y los fps no deben caer por debajo de los 30 [2]. Con este objetivo, a lo largo del presente documento se abordarán los pasos seguidos para el desarrollo de dicho servicio de Cloud Gaming con baja latencia.

Se comentará el actual estado del arte en donde se hablará de Cloud Gaming como servicio innovador, revisando distintas métricas y mencionando referentes del servicio. Luego se expondrán los problemas detectados como la latencia, escalabilidad y usabilidad de los servicios que se ofrecen hoy en día. Ligado a esto también se hará mención de los faltantes encontrados como la necesidad de brindar un servicio que ayude a los usuarios a sacar un rédito de sus equipos con poco uso.

Además, desarrollaremos en detalle nuestra solución, incluyendo el uso del protocolo WebRTC para la transmisión de datos, mecanismos de captura y reproducción de audio, vídeo y eventos de ratón y teclado, un servidor central y la interfaz gráfica. Así como la comunicación entre los diferentes componentes involucrados. Finalmente, en las últimas secciones abordaremos la metodología de desarrollo utilizada, los resultados obtenidos a partir de los escenarios propuestos y también conclusiones y posibles trabajos futuros.

## 7 Estado del arte

Cloud Gaming es un servicio innovador que permite disfrutar de videojuegos sin la necesidad de contar con un equipo de alto rendimiento. En este contexto, un servidor se encarga de realizar todas las tareas de procesamiento intensivo, y luego, mediante la tecnología de transmisión continua de datos (streaming), los usuarios pueden visualizar y jugar a los videojuegos de manera fluida. El sector del Cloud Gaming está experimentando un auge reciente y constante. Según la plataforma NewZoo, que ofrece distintas métricas y analíticas relacionados al mundo del gaming, los usuarios de pago en servicios de Cloud Gaming en el año 2021 fueron alrededor de 21,7 millones y se obtuvieron 1.476 millones de dólares en ingresos, mientras que en el 2022 la cantidad total de usuarios fue de 31,7 millones y las ganancias se estiman por 2,379 millones de dólares [3].

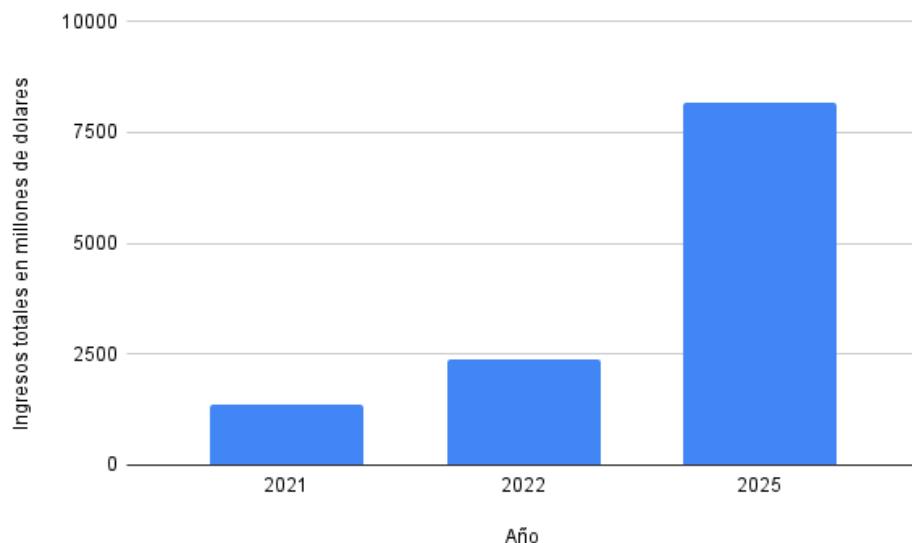


Figura 1: Ingresos anuales proyectados en la industria del Cloud Gaming

Varios servicios de grandes o medianas empresas han emergido como líderes en el mercado, donde cada uno tiene sus propias características, ventajas y desventajas. Una de estas empresas es Microsoft con su servicio Xbox Cloud Gaming. Satya Nadella, CEO de la compañía, confirmó durante la llamada de ganancias del primer trimestre del año fiscal 2023 que el servicio ha sido usado por más de 20 millones de personas [4]. Otras empresas como Nvidia también cuentan con un servicio de Cloud Gaming como lo es Geforce Now o también Amazon tiene un servicio llamado Luna. Por otro lado, Google contaba con Stadia, pero el servicio dejó de existir a principios de 2023 tras no alcanzar el impacto económico esperado.



Figura 2: Principales servicios de Cloud Gaming [5].

## 8 Problema detectado y faltante

En esta sección se describen los principales problemas y carencias detectadas en los servicios de Cloud Gaming actuales. Se aborda tanto la centralización de la infraestructura, que genera diversos inconvenientes, como la falta de mecanismos que permitan a los usuarios rentabilizar sus recursos ociosos.

### 8.1 Problema detectado

El principal problema detectado en el contexto de los servicios de Cloud Gaming es la centralización de la infraestructura. Los problemas asociados con esta centralización incluyen:

- Latencia: La transmisión de datos depende de servidores distribuidos globalmente, lo que puede introducir retrasos significativos de acuerdo a la distancia física entre el usuario y el servidor más cercano a él.
- Escalabilidad: A medida que la demanda de usuarios aumenta, una arquitectura centralizada se enfrenta al desafío de asignar más recursos para manejar eficientemente estas solicitudes. Sin embargo, hay limitaciones en la cantidad de recursos que se pueden asignar, y aumentarlos significativamente puede ser costoso. Ante la posibilidad de un aumento de la demanda que exceda la capacidad del servidor, esta arquitectura puede volverse ineficiente, dando lugar a tiempos de respuesta lentos.
- Usabilidad: Gran parte de los servicios que existen actualmente no ofrecen una forma intuitiva y práctica de aprovechar los recursos de los usuarios. La falta de una interfaz amigable y práctica limita la participación activa de los usuarios en el alquiler de sus máquinas.

### 8.2 Faltante

El principal faltante lo encontramos en la posibilidad de permitir a los usuarios obtener un rédito por prestar sus recursos a un tercero. Muchos usuarios no hacen uso de su hardware durante gran parte del tiempo y esto hace que se desperdicie el potencial del mismo. Casi ninguno o muy pocos de los servicios que existen actualmente permiten que los usuarios obtengan ganancias a partir del alquiler de sus equipos lo que también está relacionado al hecho de que son infraestructuras centralizadas y su modelo de negocio es otro. El enfoque actual del modelo de negocio de los servicios de Cloud Gaming está en la suscripción mensual o el pago por uso de recursos informáticos proporcionados por la plataforma. Sin embargo, este enfoque deja sin explotar una oportunidad significativa para optimizar el rendimiento y la eficiencia de los recursos desaprovechados.

## 9 Solución Implementada

En esta sección, se describe la solución implementada para el proyecto ‘Cloud Gaming Rental Service’. La arquitectura incluye una aplicación de escritorio en Flutter, un nodo de procesamiento en Rust y un servidor monolítico en Node.js. Además, se explican en detalle las tecnologías principales utilizadas.

### 9.1 Alcance

- Captura, transmisión y recepción de vídeo y audio sincronizado del videojuego con baja latencia.
- Control de ratón y teclado remoto.
- Automatización en el arranque del videojuego.
- Registro y autenticación de usuarios.
- Carga de videojuegos disponibles por los oferentes.
- Listado de máquinas disponibles por videojuego en tiempo real.
- Manejo automático de las sesiones de juego.
- Sistema de gestión créditos.

### 9.2 Arquitectura General

A continuación, presentamos la arquitectura del sistema. La misma se basa en una solución integrada que combina una aplicación de escritorio desarrollada en Flutter, un programa en Rust encargado del procesamiento intensivo y un backend monolítico construido en Node.js. Para simplificar, a partir de este momento, nos referiremos al programa de Rust como el nodo de procesamiento, al programa de Flutter como la interfaz gráfica y al backend monolítico de Node.js como el servidor central.

Estos procesos están diseñados para trabajar de manera coordinada, sincronizando su estado.

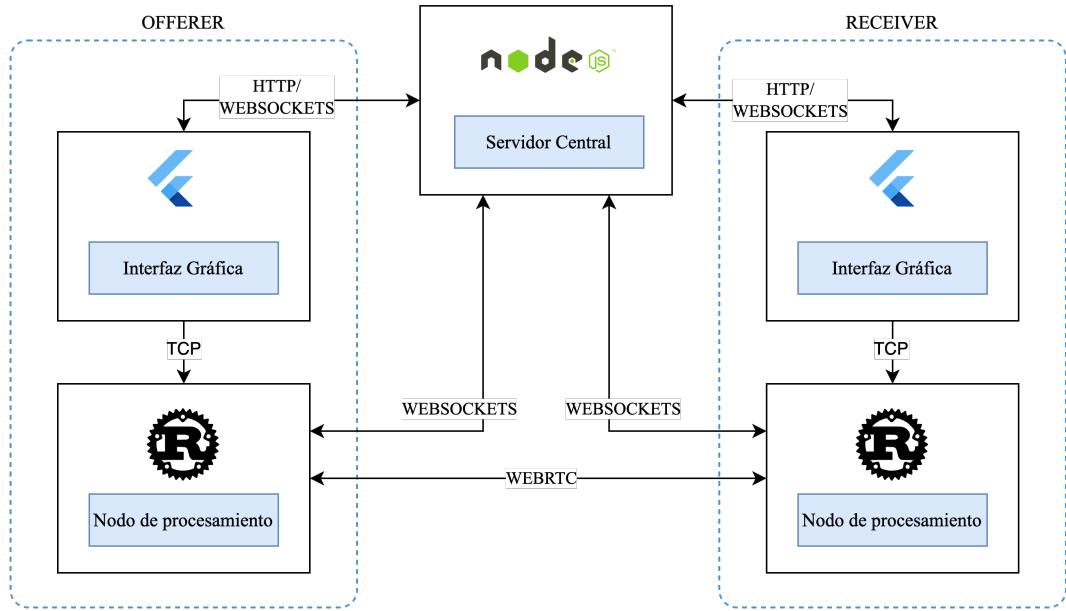


Figura 3: Comunicación entre componentes

Desarrollamos ahora los distintos protocolos utilizados para la comunicación:

■ **TCP (Transmission Control Protocol):**

Se utiliza para la comunicación interna entre el nodo de procesamiento y la interfaz gráfica debido a su practicidad para la comunicación entre procesos y su conexión persistente. Sobre este protocolo de capa de transporte, se definió un protocolo interno de capa de aplicación para coordinar diferentes escenarios del sistema. Los tipos de mensajes esperados incluyen comandos y actualizaciones de estado.

■ **HTTP:**

Este protocolo de capa de aplicación se utiliza para la comunicación sincrónica entre la interfaz de usuario y el servidor central. En particular, se implementó una API REST para las operaciones ABM (Altas, Bajas, Modificaciones) que impactarán en la base de datos. Los tipos de mensajes esperados incluyen solicitudes y respuestas de autenticación, carga de juegos y créditos.

■ **WebSockets:**

Este protocolo de capa de aplicación se utiliza para comunicación asincrónica y bidireccional entre la interfaz de usuario y el nodo de procesamiento con el servidor central. Esto permite que tanto el cliente como el servidor envíen mensajes a la otra parte en cualquier momento sin la necesidad de una solicitud previa, a diferencia del protocolo HTTP. Su principal uso fue en el manejo de las sesiones de juego, así como también para diferentes notificaciones. Los tipos de mensajes esperados incluyen solicitudes y eventos de sesión, notificaciones de estado de usuario y actualizaciones de créditos.

- **WebRTC:**

Este stack de protocolos, que se definirá en detalle en la siguiente subsección, se utiliza para la comunicación P2P entre nodos de procesamiento (siendo siempre uno de ellos Offerer y otro Receiver). Mediante esta tecnología se logra la transferencia de audio, video y otros tipos de datos.

Para proporcionar una visión completa de la arquitectura, es importante mencionar otros servicios esenciales para el funcionamiento del sistema. Firebase se utiliza para la autenticación y el almacenamiento de imágenes, mientras que una base de datos PostgreSQL alojada en Neon gestiona los datos. Otro servicio que utilizamos es un servidor STUN de Google. La API de MercadoPago administra los pagos, y un servidor TURN, alojado en una instancia EC2 (Elastic Compute Cloud) de AWS, que de ser necesario actúa como intermediario entre los pares. Cada una de estas partes será profundizada a lo largo de las siguientes secciones.

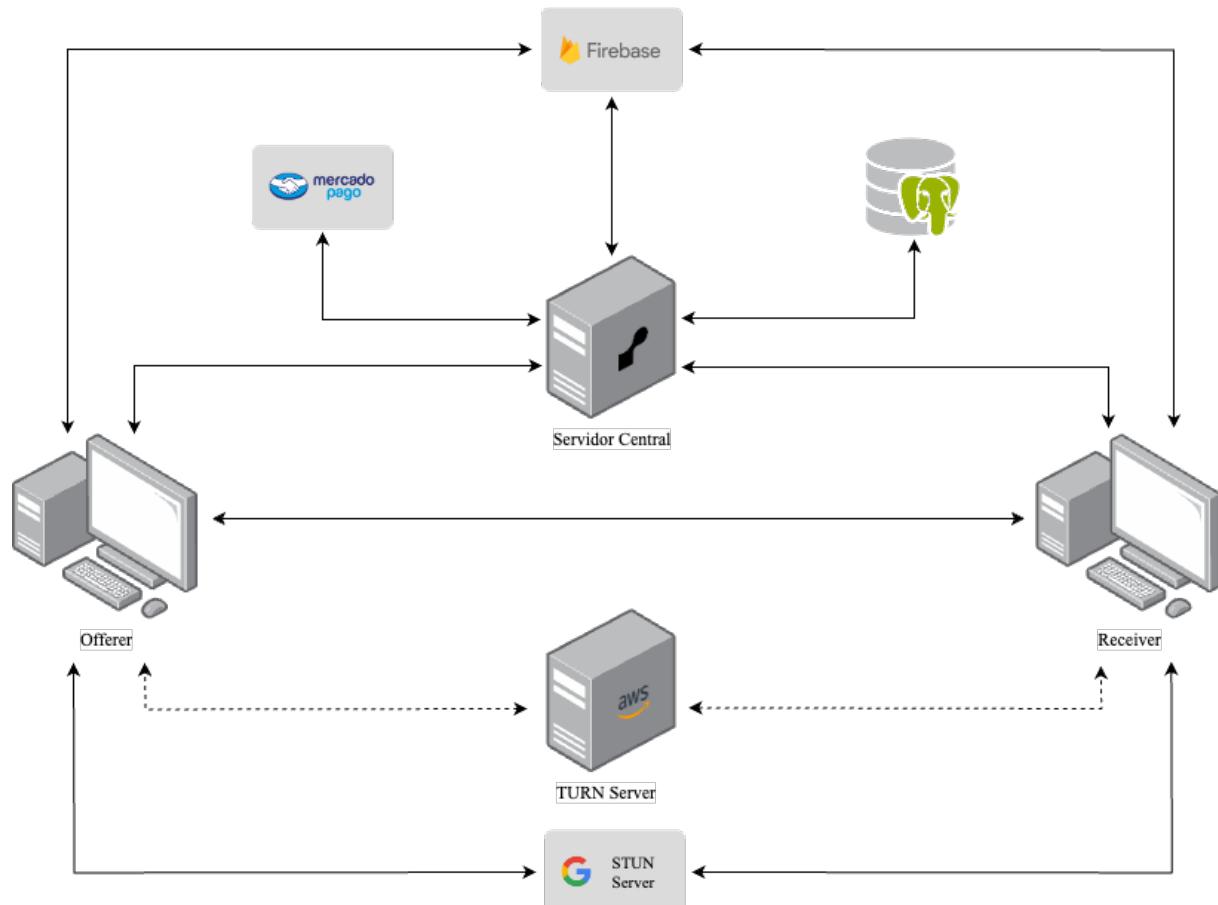


Figura 4: Diagrama de arquitectura

### 9.3 Nodo de procesamiento

Pasamos ahora a hablar del nodo de procesamiento. Este nodo se ocupa, en el caso del Offerer, de la gestión de captura de vídeo y audio del videojuego y reproducción de controles, mientras que para el Receiver se encarga de la reproducción de vídeo y audio del videojuego y captura de controles. En ambos casos la transmisión y recepción de los datos se realiza a través del protocolo WebRTC. Además para coordinar ambas funciones, como se explico previamente, este nodo se conectará con la interfaz gráfica vía TCP y a través de Websockets con el servidor.

Para la implementación de este nodo, optamos por el lenguaje de programación Rust [6]. Rust es un lenguaje de programación compilado, de tipado estático, multiparadigma y de bajo nivel. Busca ser un lenguaje seguro, potente, veloz y concurrente. También, adhiere al principio de cero costo (zero cost abstractions) que implica que las abstracciones de alto nivel no generan problemas de rendimiento. Otra importante ventaja es 'fearless concurrency' o concurrencia sin miedo, ya que Rust esta diseñado para ser muy robusto y seguro para programas concurrentes, al prevenir a través de su compilador el uso de memoria inválida y race conditions.

Dada nuestra problemática a resolver, requeríamos un lenguaje con alta performance, para optimizar al máximo el tiempo de procesamiento con el objetivo de no influir negativamente en la latencia de la experiencia de juego. Además, a lo largo del desarrollo se utilizaron diversos mecanismos de concurrencia para sincronizar puntos de acceso y ejecución.

Otro punto a favor, fue que todos los integrantes del grupo contábamos con experiencia previa por haberlo utilizado previamente en las materias de Taller de Programación I y Técnicas de Programación Concurrente y tanto WebRTC como Gstreamer tenían crates disponibles para usar en el lenguaje.

A continuación, expondremos en detalle los principales puntos de la implementación de este nodo.

#### 9.3.1 Transmisión de datos: WebRTC

Una de las tecnologías principales que utilizamos en el desarrollo de nuestra solución con el fin de lograr una comunicación P2P en tiempo real entre Offerer y Receiver es el protocolo (o stack de protocolos) WebRTC [7]. WebRTC (Web Real-Time Communication) es un proyecto libre y de código abierto que surge en el año 2011 y se compone de un conjunto de protocolos respaldados por una API intuitiva, y que cuenta con el respaldo de importantes empresas como Google, Apple, Microsoft y Mozilla. WebRTC proporciona a los navegadores y aplicaciones la capacidad de llevar a cabo Comunicación en Tiempo Real (RTC) mediante interfaces de programación de aplicaciones (API) de uso sencillo. Estas API han sido definidas por el Consorcio World Wide Web (W3C), y la especificación del protocolo se está desarrollando en colaboración con el grupo RTCWEB del Internet Engineering Task Force (IETF) a través de una serie de RFCs. Gracias a la definición de estos estándares, la tecnología está disponible en todos los navegadores modernos y aplicaciones de escritorio [8].

Para entender cómo funciona el protocolo WebRTC, podemos partirla en tres partes o pasos:

1. Conexión

## 2. Comunicación

### 3. Seguridad

Estos pasos son secuenciales, lo que significa que el paso anterior debe ser exitoso para que el siguiente paso pueda comenzar. Un hecho peculiar sobre WebRTC es que cada paso está compuesto por distintos protocolos, dentro de los cuales podemos mencionar los protocolos SCTP, DTLS, ICE, SRTP, SDP, etc.

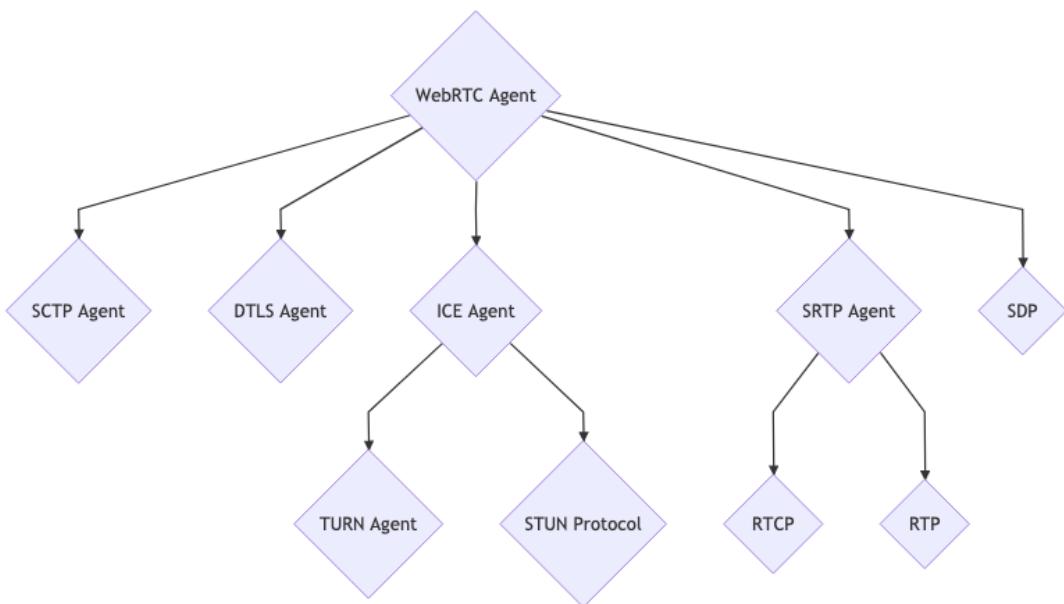


Figura 5: Protocolos que componen WebRTC [8]

## Conexión

El proceso de conexión es el que se encarga de que dos agentes WebRTC (parejas que desean conectarse) se puedan comunicar entre sí de manera directa.

Sin embargo, establecer conexión peer-to-peer no es tan sencillo. Pues dos agentes que se encuentran en redes diferentes suelen tener IPs privadas, lo que imposibilita establecer una conexión directa entre ellos.

En este contexto aparece NAT [9] (Network Address Translation) mapping. Se trata de un traductor que mapea una dirección IP privada a una pública a través de la modificación de la información de direccionamiento en tiempo real al pasar a través de un router, permitiendo así acceder a direcciones IP privadas. Sin embargo, no existe una forma única de mapeo, generando inconsistencias. Encontramos los siguientes tipos de NAT:

- **Full-Cone NAT o one-to-one NAT:** NAT mapeará la dirección IP y puerto interno a una determinada dirección y puerto público. Luego, cualquier host externo puede comunicarse con el host de

la red privada enviando los paquetes a la dirección IP y puerto externo que haya sido mapeado.

- **Address-restricted-cone NAT:** NAT mapeará la dirección IP y puerto interno a una determinada dirección y puerto público. Cuando el host de la red privada quiera comunicarse con una dirección IP específica fuera de su red. Se bloqueará todo tráfico que no venga de esa dirección IP específica.
- **Port-restricted cone NAT:** NAT mapeará la dirección IP y puerto interno a una determinada dirección y puerto público. Cuando el host de la red privada quiera comunicarse con una dirección IP y puerto específico se bloqueará todo el tráfico que no provenga de esa IP y puerto específico.
- **Symmetric NAT:** La traducción de dirección IP privada a dirección IP pública depende de la dirección IP de destino donde se quiere enviar el tráfico. Solo aquellos hosts externos que hayan recibido un paquete del host de la red privada podrán enviar paquetes.

En este contexto, resulta útil el protocolo STUN [10] (Session Traversal Utilities for NAT). Se trata de un protocolo que permite a clientes NAT encontrar su dirección IP pública, el tipo de NAT en el que se encuentra y el puerto asociado con el puerto local a través de NAT, información que resulta necesaria para configurar una comunicación entre dos hosts que están detrás de una NAT. Para nuestra implementación decidimos utilizar uno de los servidores STUN públicos que ofrece Google.

Sin embargo, no siempre es posible establecer la conexión directa entre pares, como por ejemplo en la situación en la que ambos se encuentran detrás de una Symmetric NAT. Para ese caso, resulta útil TURN.

El protocolo TURN [11] (Traversal Using Relays around NAT) resulta la solución a este problema. TURN utiliza un servidor dedicado el cual actúa como proxy entre el Offerer y el Receiver. Cuando el cliente se conecta al TURN server, este le otorga una Relay Transport Address que se puede pensar como una dirección temporal de forzamiento, a la cual otros pueden enviar tráfico a través del servidor TURN. Lamentablemente, la utilización de este servidor proxy suma latencia a la comunicación entre pares por lo que se evita utilizarlo siempre que sea posible.

Para la solución se hizo uso del proyecto de código abierto llamado COTURN [12] que implementa, entre otras cosas, un servidor TURN. El servidor es alojado en una instancia de tipo EC2 del servicio AWS.

Ahora que sabemos todo esto, ICE [13] (Interactive Connectivity Establishment) es el protocolo mediante el cual se establece la conexión entre dos pares. Se encarga de determinar todas las posibles rutas mediante las cuales dos pares podrían establecer una conexión. A estas rutas se las conoce como ICE Candidates y son una combinación de direcciones de transporte locales y remotas. Estas direcciones pueden ser la dirección IP local junto con su puerto, un mapeo NAT o una Relayed Transport Address. Estos ICE Candidates son listados a través del protocolo SDP, el cual definiremos a continuación, y de todos ellos se elige aquel que logre una mejor conexión.

El protocolo SDP [14] (Session Description Protocol) se trata de un formato de mensaje de tipo clave/valor que permite a dos agentes WebRTC enviar toda la información necesaria para la comunicación, incluyendo opciones de red, opciones multimedia y ICE Candidates, entre otras cosas. Estos mensajes

pueden ser intercambiados de diversas maneras, de hecho WebRTC no define una forma. En nuestra implementación optamos por utilizar WebSockets [15].

Una vez que dos agentes WebRTC han intercambiado SDPs, tienen suficiente información para intentar conectarse entre sí. Establecer conexiones peer-to-peer (P2P) proporciona una serie de ventajas respecto a las conexiones cliente-servidor tradicionales:

- **Costos reducidos de ancho de banda:** Debido a que la comunicación ocurre directamente entre pares, no es necesario pagar ni alojar un servidor separado para retransmitir los mensajes.
- **Latencia reducida:** La comunicación es más rápida cuando es directa. Cuando un usuario tiene que dirigir todo a través de un servidor, las transmisiones se vuelven más lentas. Este punto es particularmente importante ya que deseamos la menor latencia posible para que la experiencia de usuario sea óptima.

## Comunicación

Para la comunicación entre pares aparecen dos elementos, los Data Channels y los Tracks. Los denominados Data Channels utilizan el protocolo SCTP [16] (Stream Control Transmission Protocol) para el envío de mensajes. Se trata de un protocolo de comunicación de la capa de transporte que sirve como alternativa a los protocolos de transporte TCP y UDP. El mismo provee confiabilidad, control de flujo y secuenciación como TCP pero, a diferencia de este, los mensajes se envían en forma de datagramas, de manera similar a UDP. En el contexto de nuestra aplicación, dichos Data Channels son utilizados para el envío y recepción de eventos de teclado y mouse sobre los cuales se habla en profundidad más adelante.

Por otro lado, los Tracks se utilizan para el envío y recepción de contenido multimedia. Dichos tracks utilizan dos protocolos, RTP [17] (Real-time Transport Protocol) y RTCP [18] (RTP Control Protocol). El protocolo RTP es el que se encarga del transporte de audio y vídeo mientras que el protocolo RTCP es el que se encarga de comunicar datos de control entre el emisor y receptor de una secuencia RTP, permitiendo controlar perdidas de paquetes e implementar un sistema de control de congestión. Para evitar problemas de congestión de red, ambos protocolos trabajan juntos para ajustar el bitrate en función ancho de banda disponible actual y futuro estimado.

Nuestra aplicación utiliza estos Tracks en dos instancias, una la transmisión de vídeo de la ventana del videojuego y otra para la transmisión del audio. Cabe aclarar que a la hora de crear dichos Tracks se debe establecer el tipo de contenido que enviaremos así como también el tipo de codificación. Para el canal de audio decidimos que los frames serán encodeados en formato OPUS [19]. OPUS, un códec de audio de compresión con pérdida, funciona muy bien a tasas de bits bajas convirtiéndolo en un excelente formato para elegir al transmitir hacia o desde ubicaciones con velocidades de Internet malas y al mismo tiempo mantener un audio de alta calidad en tiempo real. Por otro lado, para el canal de video, los frames serán encodeados en formato H264 [20] ya que se trata de un codec con una alta eficiencia de compresión, ofreciendo buena calidad de vídeo con tasas de bits más bajas. Además, es compatible con la mayoría de los dispositivos y plataformas, asegurando accesibilidad global al contenido.

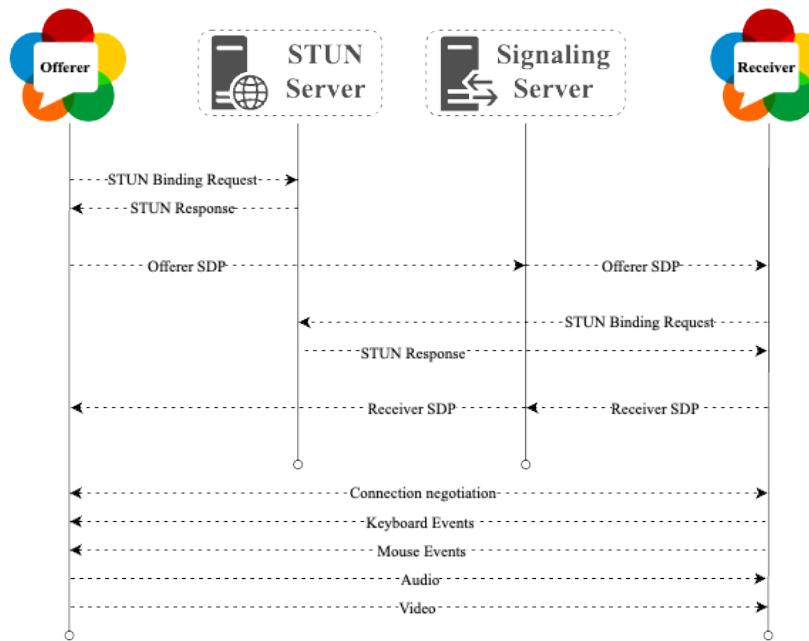


Figura 6: Conexión y Comunicación entre dos usuarios

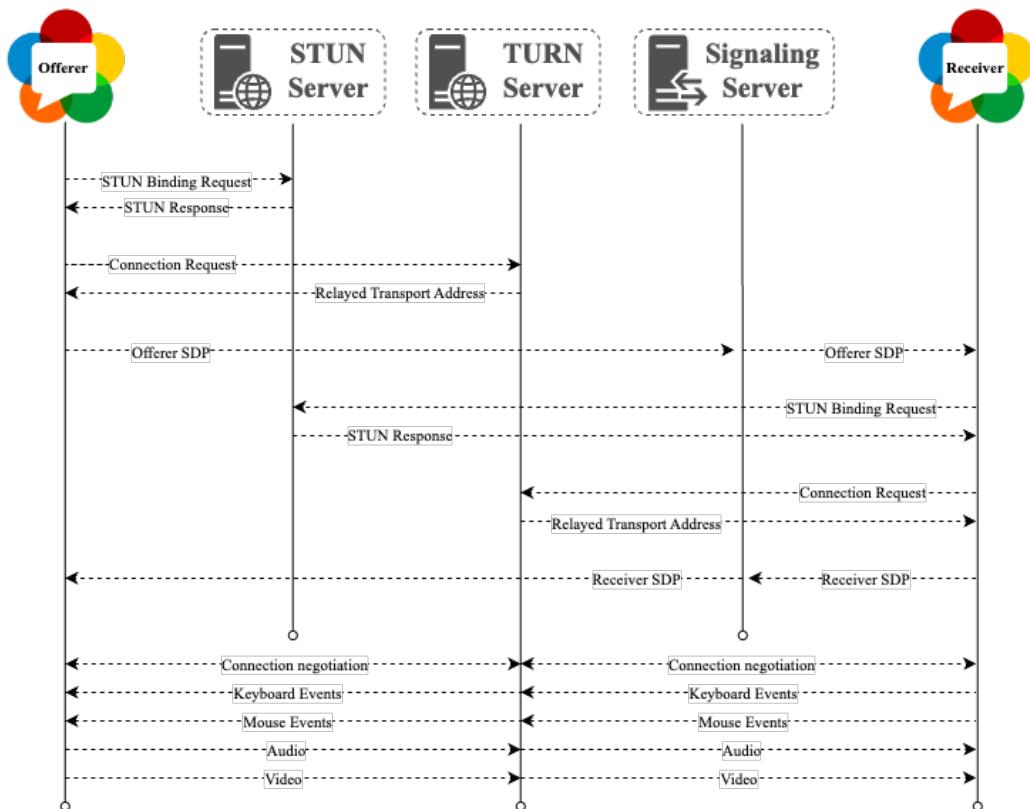


Figura 7: Conexión y Comunicación entre dos usuarios utilizando un TURN server

En las figuras 6 y 7 se busca resumir conceptualmente lo comentado hasta el momento en la sección.

## Seguridad

Finalmente mencionar brevemente los protocolos que brindan seguridad a la comunicación vía WebRTC:

- **DTLS** [21] (Datagram Transport Layer Security): Se trata de un protocolo que utiliza TLS, un protocolo de capa de transporte que brinda seguridad proporcionando un cifrado simétrico de los datos, por encima de SCTP.
- **SRTP** [22] (Secure Real-time Transport Protocol): Es un protocolo que permite encriptar paquetes de datos RTP.

### 9.3.2 Gstreamer

Introduciremos ahora el framework Gstreamer [23] utilizado para la captura, reproducción y sincronización de audio y vídeo. Se trata de una librería específicamente diseñada para el manejo de flujos multimedia y que particularmente servía a nuestro problema de lograr procesamiento de datos con la menor latencia posible.

Gstreamer cuenta con las características de ser fácilmente integrable en aplicaciones y de proveer una API muy sencilla para quien desee usarla, abstrayendo al usuario de detalles de bajo nivel pero a la vez proporcionando un alto grado de personalización. La forma en que se plantea el uso de su API es a través de la construcción de contenedores denominados pipelines. Como su nombre lo indica, en estos contenedores se colocan elementos conectados secuencialmente para formar una línea de montaje por donde fluyen los datos que son procesados según corresponda en cada elemento. A los elementos del pipeline se los denomina plugins [24], mientras que a la entrada de cada plugin, donde se reciben datos, se le llama sink y a la salida de cada plugin, donde el elemento deposita datos una vez procesados, se le denomina source [25].

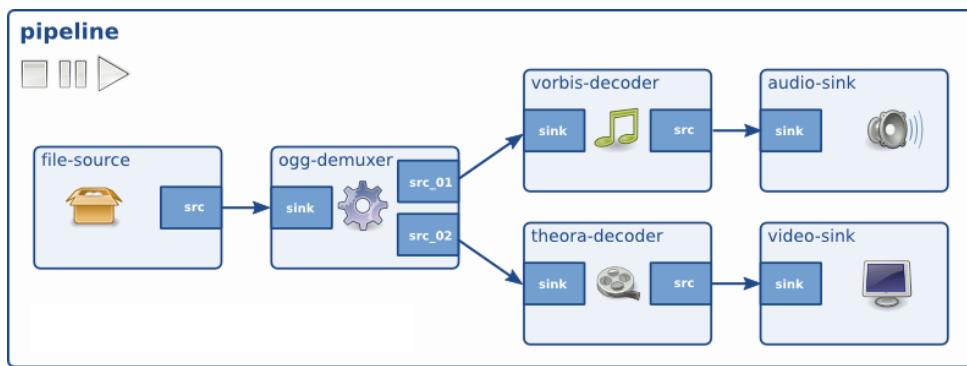


Figura 8: Ejemplo de pipeline [25]

Los plugins, que no son otra cosa que módulos de software, los cuales pueden ser implementados o bien provistos por Gstreamer pueden tener diversas funciones y se los puede clasificar según si son: sinks/sources para audio/vídeo, manejadores de protocolo, multiplexores/demultiplexores, codificadores/decodificadores, filtros/conversores, entre otros.

Todos los plugins utilizados en este trabajo son provistos por el core de Gstreamer.

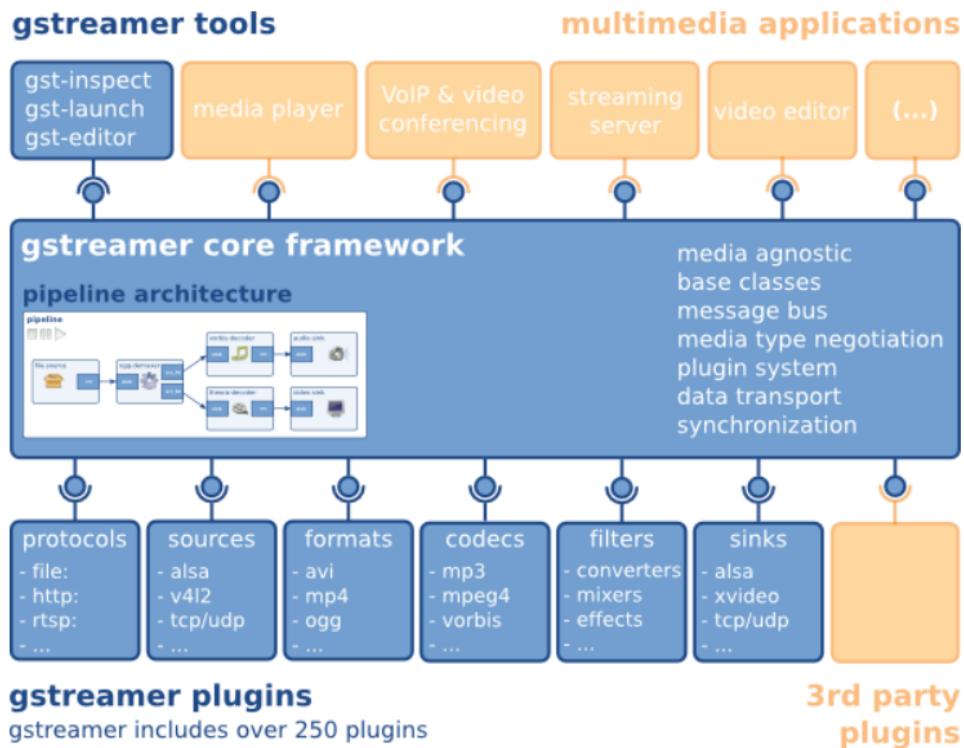


Figura 9: Gstreamer tools [23]

La transferencia de datos entre diferentes plugins del pipeline se realiza mediante buffers y para la coordinación de todo el pipeline los plugins se envían entre si mensajes denominados events. También, los elementos deben coordinar que formatos de datos van a enviar al elemento siguiente o recibir del elemento anterior en el pipeline, lo que se logra mediante la negociación de caps que son los distintos formatos que un plugin puede enviar/recibir. Así como se puede configurar diferentes propiedades de los plugins, es posible forzar a que utilicen unas ciertas caps deseadas por la aplicación. Por último, mencionar que el pipeline y la aplicación principal se comunican a través de un bus en ambos sentidos, haciendo uso de messages, queries y events. Además, es posible configurar callbacks en algunos plugins.

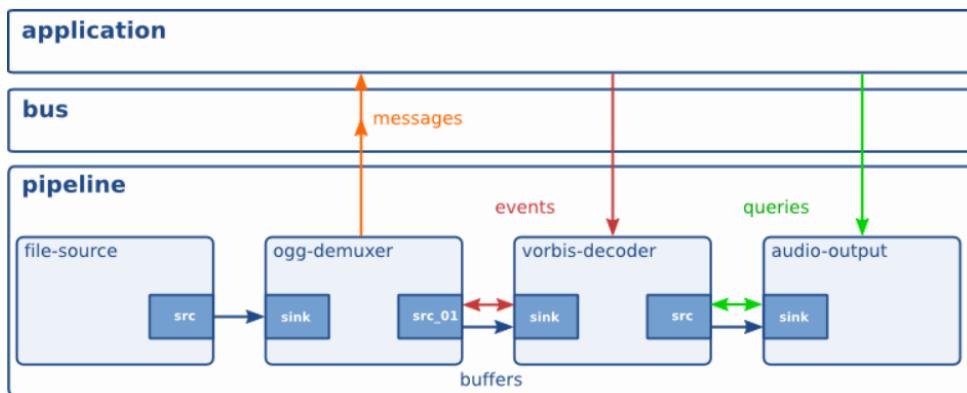


Figura 10: Ejemplo de comunicación entre aplicación principal y pipeline [25]

### 9.3.3 Captura de audio y vídeo

Habiendo introducido Gstreamer, pasamos ahora a ver los pipelines que construimos para la captura y compresión de vídeo y audio del videojuego y empaquetado en formato RTP que enviaremos a través de tracks en WebRTC hacia otro peer. No sin antes realizar algunas aclaraciones pertinentes.

En primer lugar, a la hora de configurar un pipeline es fundamental tanto el hardware donde va a ejecutarse así como también el sistema operativo, ya que los plugins pueden ser específicos a alguna de estas características. Por ejemplo, como la aplicación que desarrollamos esta pensada y solo corre en el sistema operativo Windows, para capturar el vídeo de la ventana del juego vamos a utilizar el plugin d3d11screencapturesrc en lugar de utilizar alguno específico de otro sistema operativo.

En segundo lugar, los pipelines a desarrollar son genéricos en el sentido de que puede diferir como se construyan según la computadora donde esta corriendo el programa. Según la disponibilidad de un plugin u otro se priorizará siempre la que entendemos como mejor opción. Cabe destacar que también la configuración de un plugin puede no ser la misma según el hardware subyacente. Un ejemplo de lo anterior, es la búsqueda del codificador, que dependerá de si se cuenta con una GPU integrada o dedicada, y la marca y modelo de la misma. Para nuestro caso elegimos amfh264enc y en caso de que no este disponible el programa se optará por usar mfh264enc.

En tercer lugar, tener en cuenta que a pesar de que se mencionarán a los los pipelines de captura de audio y vídeo como dos distintos, técnicamente son el mismo, ya que Gstreamer permite múltiples sink y src en un mismo pipeline, logrando de esta manera un único pipeline con una entrada de audio y otra de vídeo.

Por último, la configuración de los plugins fue trabajada mediante exhaustivas pruebas locales y remotas, hasta conseguir la más óptima relación posible entre latencia y calidad.

Pasando al pipeline de captura de vídeo, este consta del ya mencionado d3d11screencapturesrc, al cual lo configuramos enviándole la ventana que debe capturar. Luego, esto pasa a un plugin queue y llegar al plugin videoconvert, que se encarga de convertir los frames capturados en un formato BGRA [26] a NV12 [27]. Estos llegan al encoder mfh264enc que comprime a H264 y lo pasa al plugin rtph264pay que construye los paquetes RTP. Finalmente, appsink es un plugin que se puede utilizar configurando un callback, para obtener datos del pipeline y en este caso lo utilizamos para obtener los paquetes RTP que luego enviaremos por el track de WebRTC.

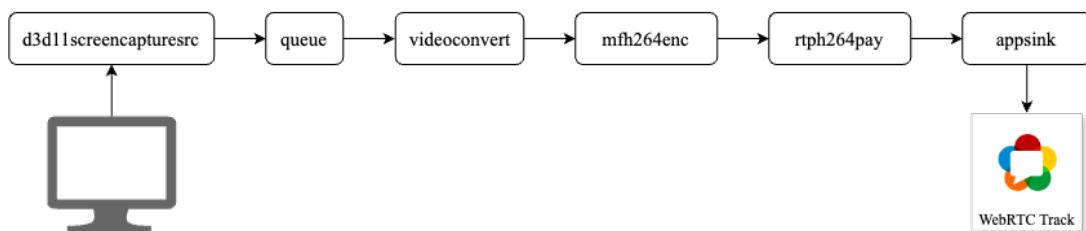


Figura 11: Pipeline de captura de vídeo

Continuamos ahora con el pipeline de captura de audio. Consta en primer lugar de wasapi2src que captura audio del dispositivo predeterminado y que luego pasa el flujo a una queue. Después se pasa por el audioconvert y el audioresample que convierten el formato de audio a S16LE para el compresor opusenc. Este plugin comprime el audio a formato opus y finalmente el plugin rtpopusspay genera los paquetes RTP. Al igual que en el vídeo se utiliza appsink para obtener los paquetes del pipeline.

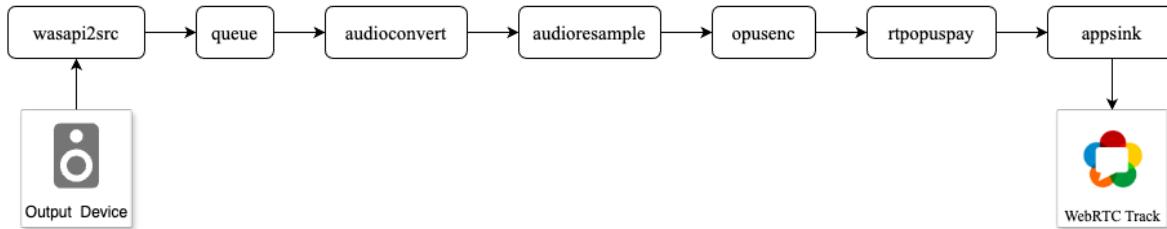


Figura 12: Pipeline de captura de audio

### 9.3.4 Reproducción de audio y vídeo

Por el lado de los pipelines de reproducción de audio y vídeo se repiten las mismas consideraciones realizadas previamente para los pipelines de captura. La diferencia sustancial es que ahora se recibirán paquetes RTP de tracks de WebRTC en lugar de enviarlos, con el fin de lograr una reproducción del contenido.

El pipeline de reproducción de vídeo consta en primer lugar de appsrc que, contrariamente a lo que hace el plugin appsink, permite injectar contenido en un pipeline. En este caso vamos a tomar los paquetes RTP recibidos en el track de WebRTC y estos pasaran al plugin rtph264pay, el cual nos dará el flujo de vídeo en formato H264. Posteriormente h264parse parseara el flujo y d3d11h264dec descomprimirá el vídeo a formato NV12. Llegará finalmente hacia una queue y luego al plugin d3d11videosink. Este plugin crea una ventana en Windows y reproduce el contenido del vídeo

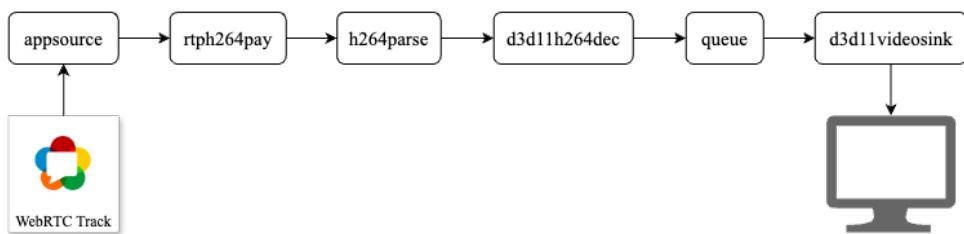


Figura 13: Pipeline de reproducción de vídeo

Por último, desarrollamos el pipeline de reproducción de audio, en donde aparece otra vez appsrc, para tomar paquetes RTP del track y pasarlo hacia una queue. El flujo continua a través de rtpopusspay, que extrae de los paquetes RTP el payload en formato OPUS y después por el opusparse que parsea el audio. El plugin opusdec descomprime a un formato adecuado. Luego, se pasa por los plugins audioconvert y audioresample y finalmente se reproduce el audio en el dispositivo predeterminado gracias a autoaudiosink. Es importante destacar que no es tan fácil predecir el formato en que se encuentra el flujo en los últimos cuatro elementos del pipeline, ya que el plugin autoaudiosink elegirá el mejor plugin

disponible y esto generará una negociación de caps en la segunda mitad del pipeline.

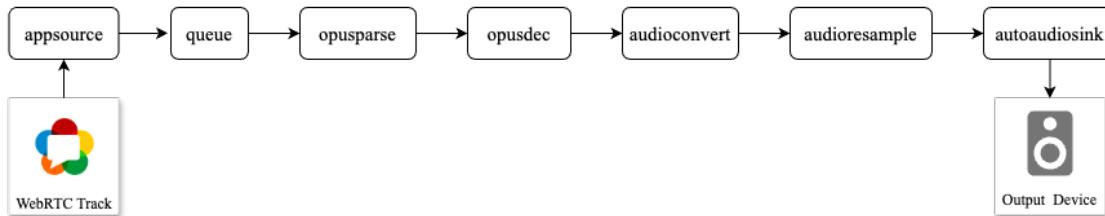


Figura 14: Pipeline de reproducción de audio

### 9.3.5 Captura y reproducción de entrada de usuario

Pasando a la capturar y reproducción de eventos de entrada de usuario, aprovechamos el crate Winput [28]. Esta librería provee una interfaz de alto nivel para el sistema de entrada de Windows, permitiendo capturar y reproducir eventos de teclado y mouse, ofreciendo también una manera sencilla de manejar estos dispositivos a través de Rust.

La librería proporciona funcionalidades para manejar eventos de teclado, incluyendo la captura de pulsaciones y liberaciones de teclas, así como su reproducción en Windows. Además, permite la captura de movimientos relativos del ratón y eventos de clic de sus botones. Facilita también la obtención y configuración de la posición del cursor en diversos sistemas de coordenadas, y soporta la rotación tanto vertical como horizontal de la rueda del ratón.

Para lograr la trasmisión de estos eventos, utilizaremos un canal de datos de WebRTC para los eventos de teclado y otro para los de ratón. Los datos a enviar son de texto y se definen a partir de un protocolo propio que sirve para interpretar si, por ejemplo, se está presionando o soltando una tecla.

La implementación se organiza en varios módulos, cada uno con responsabilidades específicas:

- **Captura de Entrada:** Este módulo es ejecutado por el Receiver y se encarga de iniciar un bucle de mensajes que captura eventos de teclado y ratón. Utilizando Winput, se registran callbacks para eventos de entrada que luego se envían a través de los canales mencionados previamente.
- **Controlador de Teclado y Ratón:** Este módulo es ejecutado por el Offerer y maneja los eventos de entrada recibidos a través de los canales de datos WebRTC. Se registran callbacks en los canales de datos para procesar los mensajes entrantes. Dependiendo del tipo de evento recibido el controlador realiza la acción correspondiente utilizando Winput.
- **Mapeo de Teclas y Botones del Ratón:** Para gestionar adecuadamente los eventos de entrada, el sistema mapea los códigos numéricos de las teclas y los botones del ratón a sus correspondientes representaciones en Winput. Esto permite una traducción precisa y una manipulación adecuada de los eventos de entrada.

Para brindar mayor seguridad al Offerer, se ha implementado una funcionalidad que bloquea ciertas teclas durante la captura de eventos. Esto previene que acciones potencialmente disruptivas o no deseadas

sean ejecutadas remotamente. Por ejemplo, una de las teclas restringidas es la tecla Windows bloqueadas para evitar que el menú de inicio de Windows se abra inesperadamente.

### 9.3.6 Concurrency y Sincronización: Tokio

En nuestra aplicación de Rust, se utilizó Tokio, un runtime asíncrono ampliamente reconocido en la comunidad de Rust, para gestionar la concurrencia y facilitar la ejecución de tareas asíncronas. Tokio nos permite lanzar múltiples tareas asíncronas que pueden ejecutarse simultáneamente, mejorando significativamente el rendimiento y la capacidad de respuesta de la aplicación. Por ejemplo, los cambios de contexto al utilizar tareas son más rápidos que si utilizásemos hilos, entre otras cosas, ya que las tareas son unidades de ejecución más livianas [29].

El uso de tareas en nuestra solución, fue conveniente debido a la gran cantidad de operaciones de entrada/salida que ocurren en nuestro programa.

Para la gestión de múltiples eventos asíncronos, utilizamos `tokio::select`, una herramienta que nos permitió la esperar simultánea de los mismos. Esto facilitó la implementación de lógica compleja dentro de una única tarea asíncrona.

Además, estas tareas deben sincronizarse tanto en el arranque como en la finalización del sistema y comunicarse entre sí. Para lograrlo, utilizamos mecanismos como barreras, canales y semáforos:

- **Barreras:** Se emplearon para asegurar que un conjunto de tareas no avance hasta que todas estén listas para proceder, facilitando una coordinación precisa en el arranque de las operaciones del sistema.
- **Canales:** Utilizados para la comunicación y coordinación entre tareas, permitiendo el intercambio de mensajes y señales de manera eficiente. Los canales aseguran que las tareas puedan sincronizar sus acciones y responder adecuadamente a eventos y cambios en el estado del sistema.
- **Semáforos:** Empleados para gestionar la secuencia de apagado de las tareas. Cuando una tarea inicia su apagado, el semáforo coordina para que las demás tareas comiencen a apagarse, garantizando que el sistema se cierre de manera segura y controlada.

## 9.4 Servidor central

En esta sección se describe la implementación del servidor utilizando Express.js para el ABM (Alta, Baja, Modificación) de usuarios, la comunicación con la interfaz gráfica y el nodo de procesamiento a través de WebSockets. Se utilizó Firebase para la autenticación de usuarios. Para las pruebas automatizadas, se emplearon Jest junto con TestContainers, permitiendo la ejecución de pruebas en un entorno aislado y reproducible. Estas pruebas se integraron en un pipeline de GitHub Actions, el cual no solo ejecutó las pruebas automatizadas, sino que también realizó el despliegue automático del servidor en Render, el servicio de hosting elegido para la implementación. Además, se integró MercadoPago, lo que permitió la generación de órdenes de pago junto con su respectiva validación.

La arquitectura del servidor mencionado es monolítica. Tomamos esta decisión por su simplicidad, permitirnos un desarrollo más ágil y facilidad para el despliegue, testeo y mantenimiento. Además, al proveer funcionalidades sencillas, no se justificaba el uso de una arquitectura de mayor complejidad.

#### 9.4.1 ABM de Usuarios

Para gestionar los usuarios, se implementó una API REST utilizando Express.js, un framework de servidor web para Node.js. La implementación del ABM permitió almacenar y manipular información de los usuarios, incluyendo:

- **Nickname:** Nombre de usuario único para identificación.
- **Ubicación:** Información geográfica del usuario.
- **Juegos Ofrecidos:** Lista de juegos que el usuario está dispuesto a ofrecer.
- **Rutas a Juegos Ofrecidos:** Lista de paths que indican en donde se encuentra el ejecutable de cada juego.
- **Créditos Disponibles:** Saldo de créditos que el usuario tiene disponible para utilizar en el sistema.

#### 9.4.2 Comunicación con WebSockets

Como ya fue mencionado, utilizamos WebSockets para la comunicación con el nodo de procesamiento y la interfaz gráfica, ofreciendo actualizaciones en tiempo real que permiten notificaciones instantáneas y actualizaciones de los estados de los usuarios y sus sesiones de juego. Para su correcto funcionamiento diseñamos un protocolo específico para estandarizar la comunicación entre los componentes mencionados, en el anexo ?? mostramos más detalle del mismo.

#### 9.4.3 Base de datos

Para la gestión de la base de datos, se utilizó PostgreSQL, un sistema de gestión de bases de datos relacional y de código abierto. PostgreSQL fue elegido por su robustez y flexibilidad. Además, la base de datos PostgreSQL se hospedó utilizando el servicio Neon en su tier gratuita, que ofrece una plataforma escalable y gestionada para bases de datos en la nube. Por otro lado, desde nuestro código manipulamos la base de datos a través del ORM Sequelize.

Vemos a continuación las relaciones entre tablas de nuestra base de datos:

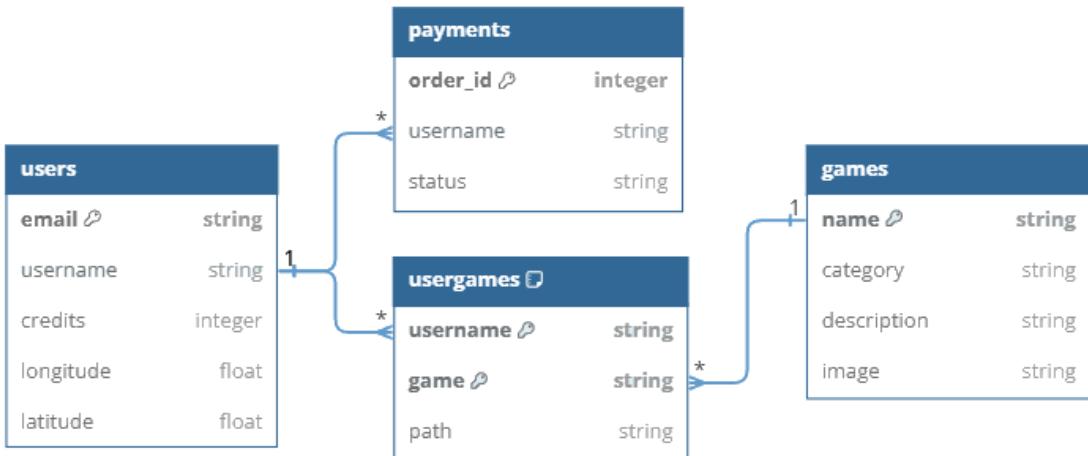


Figura 15: Relaciones entre tablas

La tabla **usergames** conecta usuarios y juegos, permitiendo que un usuario pueda guardar (para ofrecer) múltiples juegos y a su vez, un juego pueda ser jugado por múltiples usuarios. La tabla **payments** registra los pagos realizados por los usuarios.

#### 9.4.4 Autenticación con Firebase

Se utilizó Firebase para implementar un gateway que valida las peticiones a la REST API y asegura que provengan de usuarios autenticados mediante el uso de tokens. Este mecanismo garantiza que solo los usuarios con credenciales válidas puedan interactuar con el servidor, proporcionando una capa adicional de seguridad y control de acceso en la aplicación.

#### 9.4.5 Integración de MercadoPago

Para permitir a los usuarios realizar pagos y validar sus transacciones, se integró MercadoPago en el servidor. La integración se logró mediante la creación de un controlador específico que maneja tanto la generación de órdenes de pago como la recepción de notificaciones de pago.

**Creación de Órdenes de Pago:** Cuando un usuario solicita realizar un pago, el servidor crea una orden de pago utilizando la API de MercadoPago. El controlador busca al usuario en la base de datos y crea un registro de pago. Luego, se genera una preferencia de pago con los detalles del producto, incluyendo el título, la cantidad y el precio unitario. Esta preferencia se envía a MercadoPago para obtener una URL donde el usuario puede completar la transacción.

**Recepción de Notificaciones de Pago:** MercadoPago envía notificaciones al servidor cuando el estado de una transacción cambia. El controlador maneja estas notificaciones verificando el ID de la

transacción y recuperando los detalles de la misma desde MercadoPago. Si el pago es aprobado, se actualiza el estado del pago en la base de datos y se incrementan los créditos del usuario en función de la cantidad de horas compradas. Además, se notifica al usuario mediante un sistema de suscripciones en tiempo real construido sobre WebSockets.

## 9.5 Interfaz de usuario

La interfaz de usuario del sistema es una aplicación de escritorio para sistemas Windows y fue desarrollada utilizando el framework Flutter, elegido por sus ventajas significativas frente a otras tecnologías. Flutter permite desarrollar aplicaciones nativas de alta calidad con un rendimiento cercano al nativo y con la posibilidad de ser multiplataforma (ventaja que se podría aprovechar como desarrollo futuro). El desarrollo con este framework es rápido y eficiente, y su extenso conjunto de widgets permite crear interfaces atractivas y altamente personalizables. Nuestra experiencia previa con Flutter también influyó en esta decisión, permitiéndonos aprovechar al máximo sus características avanzadas para crear una interfaz intuitiva y eficiente.

Para la autenticación y el almacenamiento, se utilizaron tecnologías de Firebase debido a su integración sencilla y robusta con Flutter. Estas fueron:

- **Firebase Authentication:** proporciona un sistema seguro y fácil de implementar para gestionar la autenticación de usuarios mediante tokens, garantizando la seguridad y la simplicidad en la identificación de usuarios.
- **Firebase Storage:** ofrece una solución escalable y rápida para gestionar y acceder a recursos multimedia.

Por otra parte, como ya se mencionó, la interfaz gráfica se comunica con el nodo de procesamiento mediante TCP y con el servidor central vía HTTP (Es quien consume la API REST implementada) y WebSockets. Logrando de esta manera la sincronización de estado y eventos con los demás componentes.

Las principales características de la interfaz de usuario son:

- **Experiencia de Usuario:**
  - Interfaz intuitiva y fácil de usar.
  - Diseño atractivo y responsive.
  - Monitorización y control en tiempo real del estado de la sesión de juego, tanto para Offerers como para Receivers.
- **Autenticación y Seguridad:**
  - Autenticación mediante Firebase.
  - Uso de tokens para identificar a los usuarios de manera segura.
  - Autenticación con Google.

#### ■ Almacenamiento y Gestión de Contenidos:

- Acceso a las imágenes de los videojuegos disponibles mediante Firebase Storage.
- Gestión eficiente de recursos y acceso rápido a los contenidos.

Para ver con mayor detalle esta interfaz ir a la sección ?? del anexo.

## 9.6 Carga de Horas

Para contabilizar y gestionar el tiempo de juego disponible, implementamos un mecanismo de carga de horas. Estas horas se adquieren a través de la aplicación de escritorio, permitiendo a los usuarios mantener un saldo que pueden utilizar para jugar. A continuación, se detalla el flujo del proceso de carga de horas:

- **Solicitud de Carga de Horas:** El usuario selecciona la cantidad de horas que desea cargar en la aplicación de escritorio. Esta solicitud se envía desde la interfaz gráfica al servidor central.
- **Generación de Link de Pago:** El servidor central utiliza la API de Mercado Pago para generar un enlace de pago correspondiente a la cantidad de horas solicitadas. Este enlace se crea a partir de un ítem preestablecido que representa el costo de las horas de juego.
- **Redirección al Pago:** Al recibir el enlace de pago, la interfaz gráfica lanza el navegador de preferencia del usuario y abre el enlace. El usuario deberá iniciar sesión en su cuenta de Mercado Pago si no lo ha hecho ya y proceder con el pago del monto indicado.
- **Notificación de Pago:** Una vez completado el pago, Mercado Pago notifica al servidor central sobre la transacción exitosa.
- **Acreditación de Horas:** El servidor central procesa la notificación de Mercado Pago, realiza las acreditaciones pertinentes de las horas de juego en la cuenta del usuario y notifica a la interfaz gráfica a través de WebSocket.

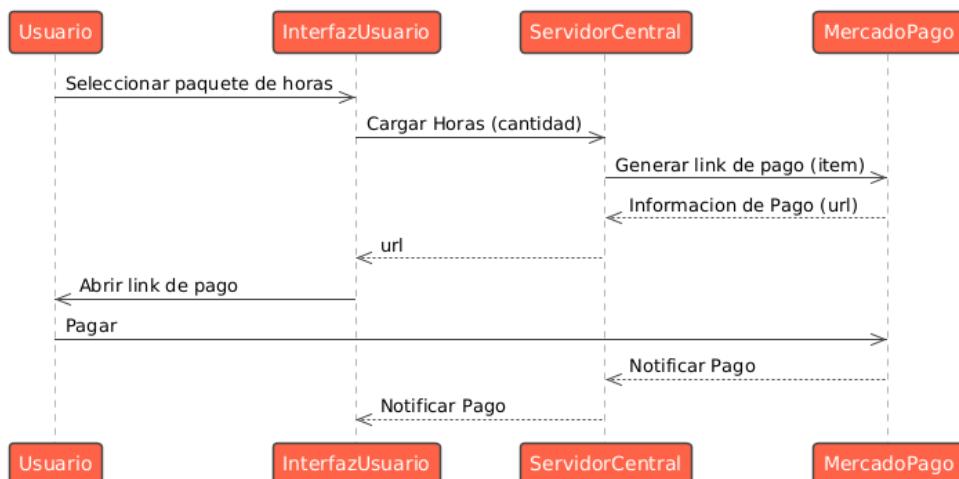


Figura 16: Diagrama de Secuencia - Carga de horas

Este proceso garantiza una experiencia de usuario fluida y segura, permitiendo la carga de horas de juego de manera sencilla. Las principales ventajas del uso de la API de Mercado Pago incluyen seguridad, eficiencia y notificaciones en tiempo real. Mercado Pago ofrece una plataforma segura para transacciones en línea, protegiendo tanto al usuario como al sistema. Además, la integración con la API permite una rápida generación y procesamiento de enlaces de pago. La comunicación mediante WebSocket asegura que el usuario sea notificado inmediatamente una vez que las horas han sido acreditadas.

## 9.7 Automatización

En este apartado describiremos como nuestro sistema maneja el arranque, control y finalización de las sesiones de juego para que estas no dependan del manejo manual de nuestros usuarios. Dividiremos estos flujos según el rol del usuario.

Por el lado del Receiver, luego de seleccionar el título deseado obtendrá la lista de los Offerers disponibles en ese momento y al elegir a uno podrá definir el tiempo de juego pretendido, se validará que este usuario cuente con los créditos necesarios y en ese caso, el servidor central facilitará que ambos pares intercambien la información necesaria para el establecimiento de la conexión, luego comenzará con el registro del tiempo y estado de la sesión. En la computadora del Receiver se abrirá una ventana en la que podrá controlar y visualizar el juego deseado. Finalmente, al pasar el tiempo de la sesión (o en otro caso de cierre) se cerrará dicha ventana.

Por otro lado, el Offerer cargará los juegos que desea compartir. Para esto, a partir de una lista de juegos soportados podrá seleccionar el mismo indicando la ruta del ejecutable. Luego podrá indicar que ya está disponible para ofrecer sesiones de juego. Cuando llegue una solicitud de sesión desde el servidor central con la información del usuario remitente y el juego solicitado, entonces se lanzará el juego a partir de la ruta previamente cargada y se enviará la información necesaria al servidor para el comienzo de la conexión. Una vez establecida la conexión se comenzará con la captura de la ventana utilizando un algoritmo que encuentra la ventana a partir de la ruta especificada y finalmente se procederá con el envío de los datos. De igual manera que con el Receiver, el Offerer será notificado ante un cierre imprevisto o no de la sesión, finalizando con la ejecución del juego volviendo a disponibilizarse para recibir nuevas peticiones.

## 10 Metodología aplicada

Para la gestión del proyecto que abordamos se utilizamos como herramienta principal Notion, un software de gestión de proyectos, que facilito la organización de la documentación de tareas y objetivos, así como también la coordinación y comunicación entre los integrantes. Los roles fueron asignados específicamente basándose en habilidades y experiencias previas con el fin de cubrir todos los aspectos del proyecto y se realizó un seguimiento constante mediante reuniones regulares y actualizaciones en Notion. Esto permitió un desarrollo organizado minimizando riesgos y desviaciones del plan inicial.

### 10.1 Backlog

El backlog fue una herramienta fundamental para la gestión de tareas y actividades del proyecto. Utilizamos un tablero de Notion para organizar y priorizar las tareas pendientes, en progreso y completadas. En el backlog, se registraron todas las tareas que debían ser realizadas a lo largo del desarrollo del proyecto. Estas tareas incluían desde la investigación de problemas técnicos específicos, la implementación o mejora de funcionalidades, hasta la documentación o pruebas de lo desarrollado.

The screenshot shows a Notion board interface with the following structure:

- Backlog (4 items):**
  - [REFACTOR] Tamaño del buffer de los channels
  - [BUG] Shift no funciona
  - [FEAT] Elegir el dispositivo de salida/entrada del audio
  - [INFORME] Metricas elementos del pipeline
- To Do (2 items):**
  - [FEAT] Que se abra el juego que queremos
  - [ENTREGA] Mergear el codigo a main y generar el tag del entregable 2
- In Progress (4 items):**
  - [ENTREGA] Informe para entregable 2
    - Kelman Axel Franco Gazzola
  - [FRONT] Prototipado de pantallas.
    - Franco Gazzola Juan Cruz Caserío Tadeo Rial Kelman Axel
  - [REFACTOR] Cambiar los channels del receiver de la std a los de tokio
    - Franco Gazzola Tadeo Rial
  - [INFORME] Agregar mediciones de latencia
    - Kelman Axel Franco Gazzola Tadeo Rial
- Done (5 items):**
  - [FEAT] Registrar sesion de juego
    - Franco Gazzola Kelman Axel
  - Documentación del código
    - Franco Gazzola Tadeo Rial Kelman Axel
  - [RESEARCH] Investigar y probar encoder h265
    - Tadeo Rial Franco Gazzola
  - [FEAT] Seleccionar solo la ventana del juego
    - Tadeo Rial Kelman Axel
  - [FEAT] Empaqueando nodo de procesamiento e interfaz
    - Juan Cruz Caserío Tadeo Rial

Figura 17: Ejemplo del backlog utilizado

### 10.2 Roles

Para asignar las tareas buscamos aprovechar la experiencia previa y los conocimientos de cada integrante. También, en base a las capacitaciones realizadas por cada uno, los roles asignados fueron:

### 10.3 Registro de actividades

Durante el desarrollo del trabajo hicimos uso de una base de datos de Notion para llevar el registro del avance de las tareas que se hacían día a día. En el mismo registramos aspectos como quien/quienes llevaron a cabo las tareas, cantidad de horas trabajadas, fecha en la que se realizó la actividad, tipo de

Tarea	Integrantes
Interfaz de usuario	Juan Cruz Caserío, Tadeo Rial
Servidor Central	Axel Kelman, Franco Gazzola
Captura y reproducción de audio y video	Axel Kelman, Tadeo Rial
Captura y reproducción de entrada de usuario	Juan Cruz Caserío, Franco Gazzola
Sistema de comunicación	Todos

Cuadro 1: Distribución de tareas

actividad que se desarrolló (ej: debug, coding, reunion), tema sobre el que se trabajo (ej: audio, webrtc, frontend, etc.) y una descripción donde esta el detalle de lo trabajado.

Personas	Fecha	# Horas	Actividad	Tema	Descripción
J Juan Cruz Cas T Tadeo Rial K Kelman Axel	23/01/2024	3	Coding Testing	WebRTC	Refactor. Pruebas de conexión entre Axel Juan y Tadeo. A todos les conecto.
T Franco Gazzola K Kelman Axel T Tadeo Rial	24/01/2024	3	Testing	WebRTC	Pruebas de conexión entre Axel Franco y Tadeo. Conecta entre todos menos Franco y Tadeo.
K Kelman Axel	24/01/2024	2	Capacitación	WebRTC	Profundización de WebRTC para troubleshooting
T Franco Gazzola J Juan Cruz Cas K Kelman Axel T Tadeo Rial	25/01/2024	4	Coding	Video	Comienza el encodeo de video a h264. Empezamos usando: <a href="https://crates.io/crates/openh264">https://crates.io/crates/openh264</a> . Detectamos problema de compatibilidad con captura (RBGA) y encoder (RGB)
J Juan Cruz Cas	26/01/2024	1.5	Capacitación	Video	Busqueda de crates para encodeo. <a href="https://crates.io/crates/ffmpeg-next">https://crates.io/crates/ffmpeg-next</a>
T Franco Gazzola J Juan Cruz Cas K Kelman Axel T Tadeo Rial	27/01/2024	1.5	Reunión		Reunión para organizarnos, definir scope del entregable y dividir las tareas.
J Juan Cruz Cas T Franco Gazzola K Kelman Axel T Tadeo Rial	27/01/2024	1	Reunión	Diseño	Reunión de equipo, definición de entregable 1 y división de tareas
J Juan Cruz Cas	27/01/2024	1	Capacitación	WebRTC	Lectura del apartado de latencia de "Webrtc for the curious" para empezar a investigar sobre el tema.

Figura 18: Registro de actividades

Esta herramienta fue indispensable para mantener el control del avance de cada una de las tareas, poder identificar posibles cuellos de botellas en las mismas y a su vez tener un registro fidedigno de lo realizado.

## 11 Experimentación y validación

En esta sección desarrollaremos los mecanismos utilizados para la validación de QoE de nuestro sistema. Primero abordaremos como realizamos la medición de la latencia correspondiente a la red, luego a la generada por el procesamiento de cada pipeline y plugin relevante de Gstreamer y finalmente a partir de diferentes escenarios de pruebas expondremos los resultados obtenidos. Un punto previo a considerar también es que además de las métricas tomadas fue de suma importancia hacer numerables pruebas con diferentes configuraciones, juegos y hardware para asegurar que la experiencia de juego fuera óptima y satisfactoria.

### 11.1 Medición de Latencia de Red entre pares

El objetivo de este proceso es medir la latencia de la red entre dos pares utilizando un servidor Simple Network Time Protocol [32] (SNTP). En este caso por latencia nos referimos al tiempo que tarda un paquete de datos en viajar de un punto a otro en la red y es un factor crítico en aplicaciones sensibles al tiempo, como es el caso del Cloud Gaming.

El motivo por el cual decidimos utilizar un servidor SNTP es por que este proporciona una referencia de tiempo externa, precisa y sincronizada, lo cual es esencial para calcular la latencia con exactitud, asegurar que los pares utilicen la misma referencia de tiempo, evitar discrepancias debidas a la falta de sincronización y reducir los errores causados por la desincronización de los relojes internos de los dispositivos.

El procedimiento para realizar la medición de latencia comienza con el par Offerer. Utilizando un canal de datos de WebRTC para enviar mensajes de texto, el par Offerer envía cada 2 segundos un mensaje con la hora obtenida a través de una solicitud UDP al servidor SNTP. Al recibir este mensaje, el par Receiver también obtiene la hora actual del servidor SNTP mediante una solicitud similar. Luego, el par Receiver calcula la latencia determinando la diferencia entre ambas horas recibidas. En ambos casos a la hora recibida por el servidor SNTP se le resta el tiempo de retardo en la respuesta.

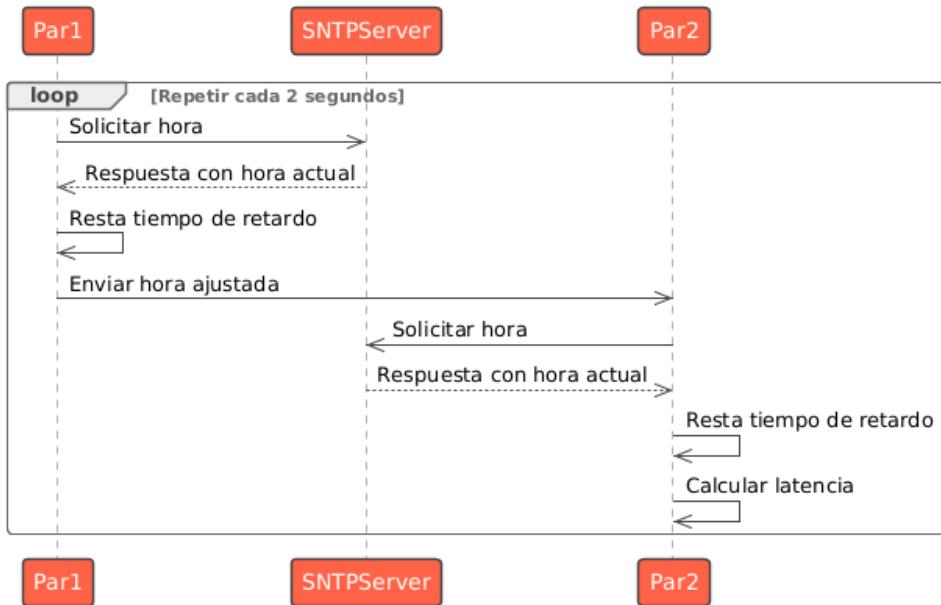


Figura 19: Diagrama de Secuencia - Evaluación de Latencia de Red

## 11.2 Medición de Latencia en Gstreamer

Gstreamer cuenta con un modulo llamado Tracing que nos permite medir la latencia en tiempo real de punta a punta del pipeline así como también la latencia en cada plugin del mismo. Cabe aclarar que las primeras lecturas pueden ser altas debido al tiempo de inicialización.

Consideramos que los principales puntos a exponer son los codificadores y decodificadores tanto de vídeo como de audio, así como también los totales obtenidos en cada pipeline. Los tipos de plugin mencionados son aquellos que tienen mayor incidencia en la latencia total del pipeline.

## 11.3 Resultados Obtenidos

A continuación, se enumerarán los escenarios de prueba realizados junto a sus resultados.

### 11.3.1 Primer escenario

En este primer escenario se realizó una sesión de 10 minutos jugando al juego Q.U.B.E. 2, registrando la latencia de red y los tiempos de procesamiento de Gstreamer. En este caso, dado que la conexión fue directa, no se requirió del uso del servidor coturn como intermediario, además la tasa de bitrate para la captura del video fue de 6000Kbps y los fps 50.

Componente	Offerer	Receiver
CPU	i5-13400	i5-1135G7
GPU	AMD Radeon 6700XT	Intel(R) Iris(R) Xe Graphics
Memoria	32GB	16GB
SO	Windows 11	Windows 11
Conexión	Ethernet	Wi-Fi

Cuadro 2: Especificaciones técnicas - Primer Escenario

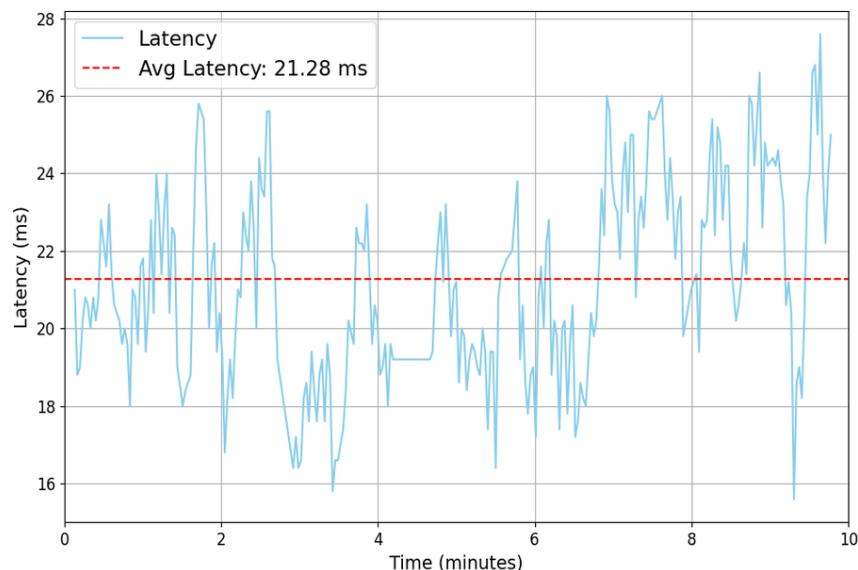


Figura 20: Latencia de Red entre pares - Primer Escenario

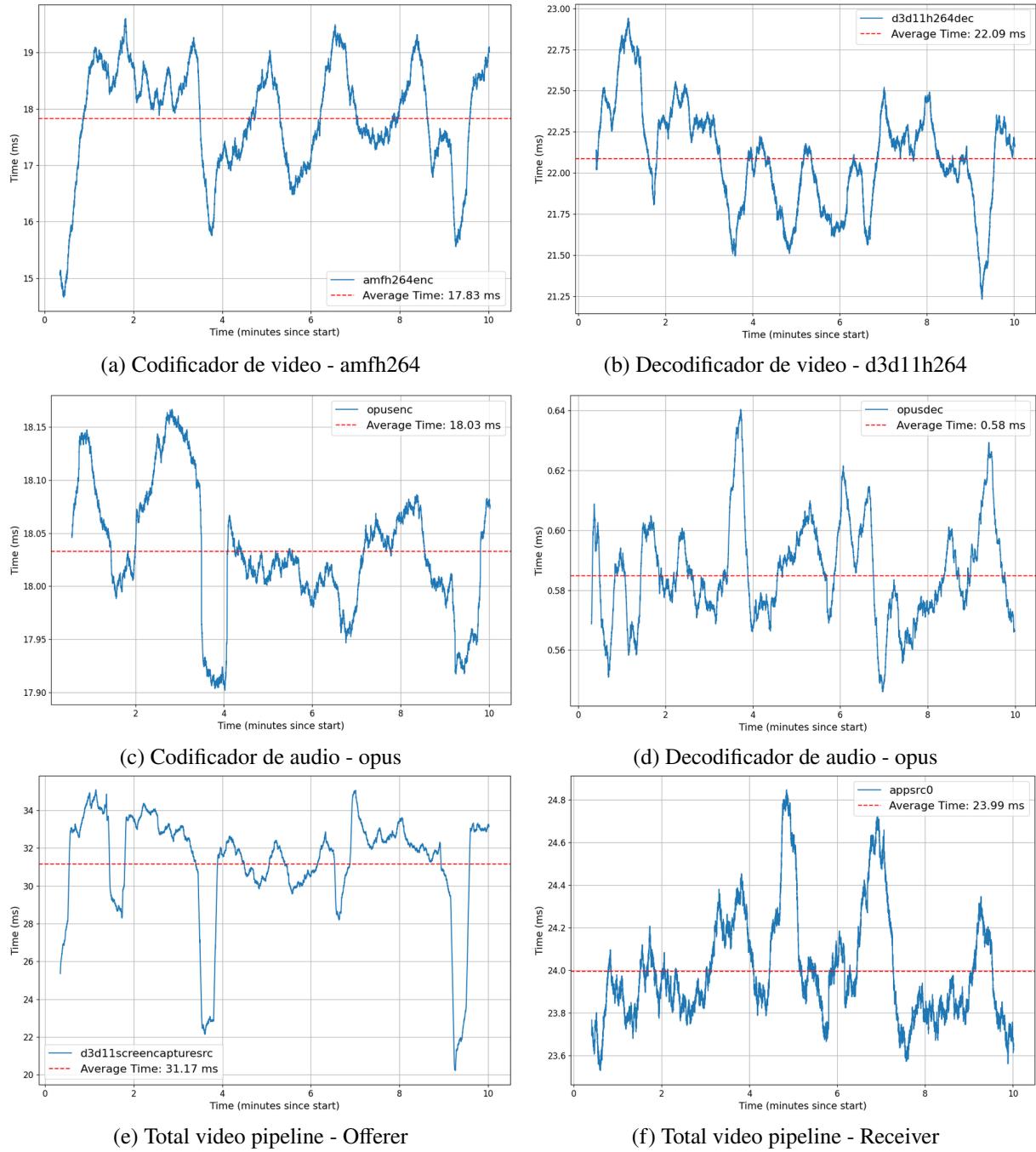


Figura 21: Tiempos de procesamiento - Primer Escenario

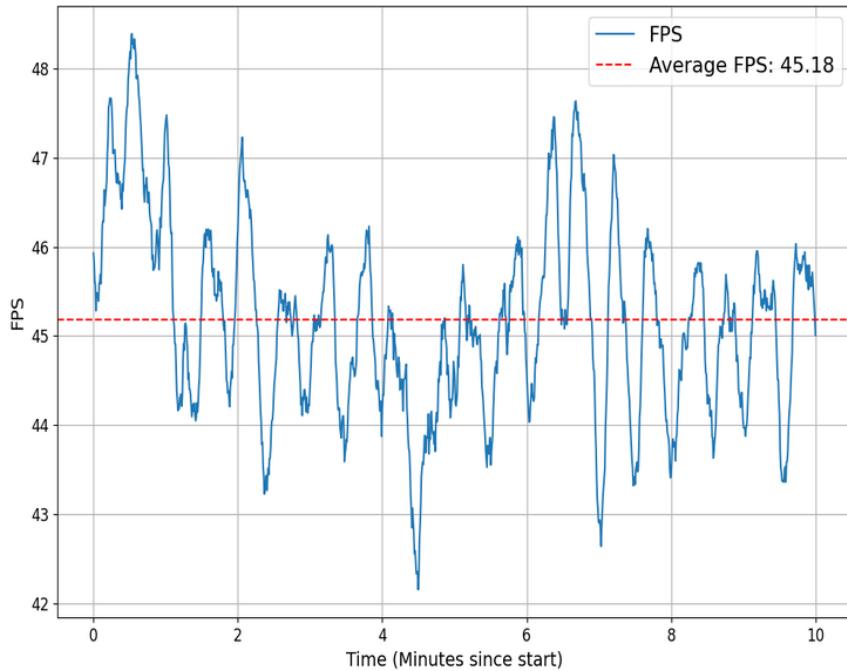


Figura 22: FPS Reproducción - Primer Escenario

Este primer escenario sería uno ideal para el uso de nuestro sistema ya que como se observa en las mediciones tuvimos una conexión medianamente estable y adecuada, teniendo además un Offerer con el hardware recomendado para soportar el procesamiento, proporcionando un rendimiento satisfactorio.

Dado que el tiempo de captura y reproducción de vídeo es mayor que el de audio tomamos estos valores para realizar un calculo aproximado de la latencia total promedio. A partir de los promedios obtenidos en el escenario planteado calculamos:

$$\left\{ \begin{array}{l} \text{Latencia total promedio} \approx \text{Latencia de captura de video} + \text{Latencia de red} + \text{Latencia reproducción de video} \\ \text{Latencia total promedio} \approx 31,17 \text{ ms} + 21,28 \text{ ms} + 23,99 \text{ ms} \\ \text{Latencia total promedio} \approx 76,44 \text{ ms} \end{array} \right.$$

Podemos observar entonces, que registramos una latencia promedio de 76,44ms y un promedio de fps en la reproducción de 45,18, superando entonces los parámetros mínimos para una correcta QoE (latencia inferior a 100ms y tasa de fotogramas promedio por segundo superior a 30fps).

### 11.3.2 Segundo escenario

Para el segundo escenario se realizó una sesión de 10 minutos jugando a Terraria, registrando las mismas métricas. La conexión también fue directa pero la tasa de bitrate para la captura del video fue de 3000Kbps y los fps fueron 50.

Componente	Offerer	Receiver
CPU	i5-8300H	i5-13400
GPU	Intel® UHD Graphics 630	AMD Radeon 6700XT
Memoria	16GB	32GB
SO	Windows 11	Windows 11
Conexión	Wi-Fi	Ethernet

Cuadro 3: Especificaciones técnicas - Segundo Escenario

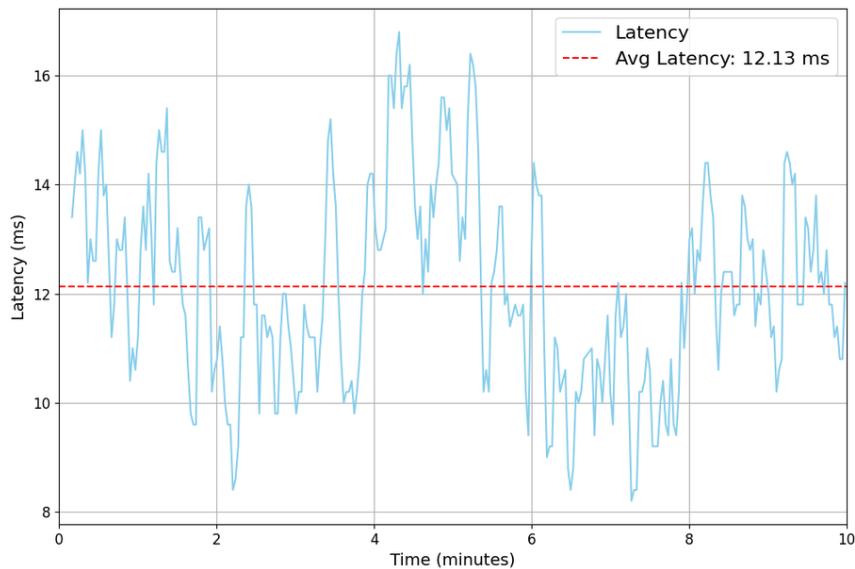


Figura 23: Latencia de Red entre pares - Segundo Escenario

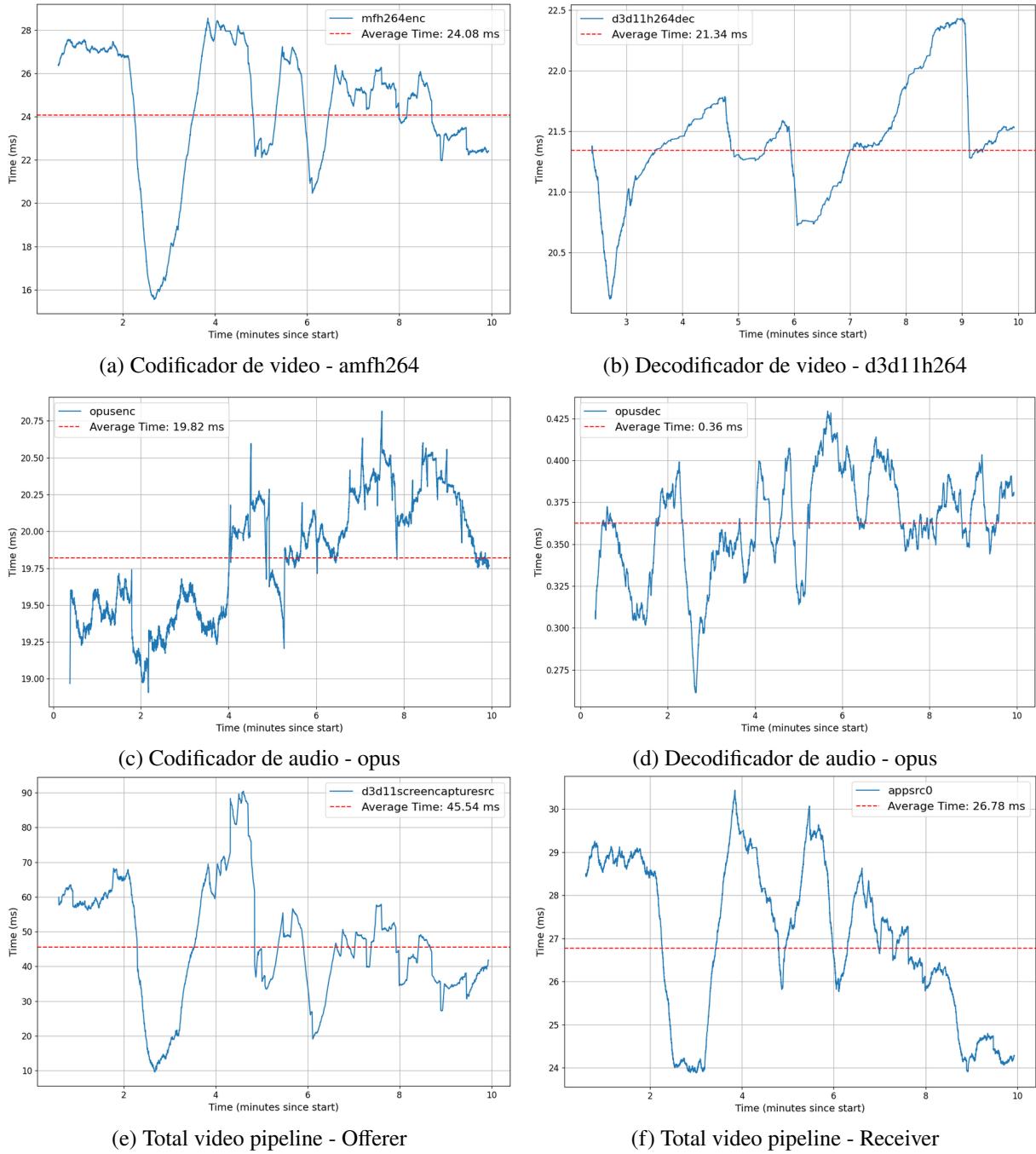


Figura 24: Tiempos de procesamiento - Segundo Escenario

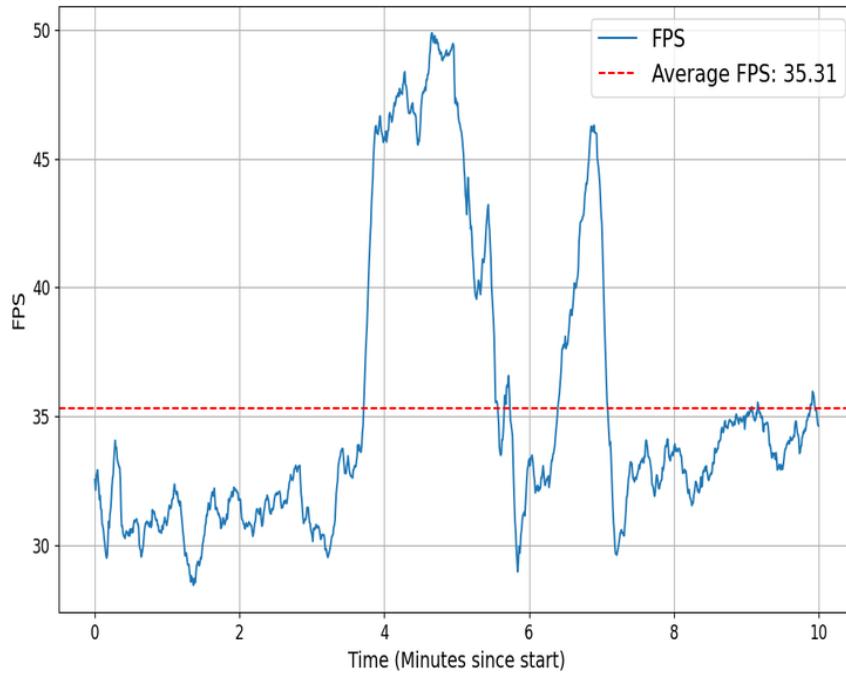


Figura 25: FPS Reproducción - Segundo Escenario

Si bien este segundo escenario no era el ideal ya que el Offerer no contaba con una computadora de gran rendimiento (lo cual se reflejo en las mediciones del codificador de vídeo) las pruebas resultaron mejores de los esperado. Por otro lado nos vimos favorecidos por una mejor conexión a internet. En este caso la latencia total promedio fue aproximadamente:

$$\left\{ \begin{array}{l} \text{Latencia total promedio} \approx \text{Latencia de captura de video} + \text{Latencia de red} + \text{Latencia de reproducción de video} \\ \text{Latencia total promedio} \approx 45,54 \text{ ms} + 12,13 \text{ ms} + 26,78 \text{ ms} \\ \text{Latencia total promedio} \approx 84,45 \text{ ms} \end{array} \right.$$

Por lo tanto, a pesar de lo mencionado, se alcanzó un QoE adecuado dado al resultado de la latencia promedio de 84,45ms y a los fps promedio 35,31ms.

## 12 Cronograma

A continuación, presentamos el cronograma de las actividades realizadas, puntuizando en hitos alcanzados a lo largo del proyecto.

Descripción de Tarea	Inicio	Fin
Capacitación WebRtc y primeras pruebas de conexión	20/11/23	20/12/23
Investigación para implementación de WebRTC	20/12/23	23/12/23
Primera implementación con C++ (transmisión básica vía WebRTC)	23/12/23	09/01/24
Investigación para captura y codificación de audio y vídeo con C++	23/12/23	09/01/24
Captura, codificación y transmisión de audio en Rust	09/01/24	16/01/24
Investigación de captura y codificación de vídeo en Rust	25/01/24	22/02/24
Captura, reproducción de vídeo mediante Gstreamer	23/02/24	29/02/24
Set up coturn en instancia EC2 de AWS	23/02/24	24/02/24
Captura, transmisión y reproducción de mouse y teclado	01/03/24	04/03/24
Transmisiones sencillas multimedia	04/03/24	05/03/24
Captura, reproducción de audio mediante Gstreamer	05/03/24	08/03/24
Frontend - Diseño y desarrollo de funcionalidades principales	18/03/24	26/06/24
Servidor - Implementación ABM usuarios	20/03/24	05/04/24
Frontend - Integración con Backend	04/04/24	14/04/24
Servidor - Implementación de Web Sockets	09/04/24	23/04/24
Mejoras en los mecanismos de captura de audio y video	19/04/2024	08/06/2024
Búsqueda y captura de ventana de juego	04/05/24	26/05/24
Implementación de Mercado Pago para la carga de horas	11/05/24	25/05/24
Manejo de escenarios de desconexión forzada o por timer	01/06/24	26/06/24

Cuadro 4: Cronograma

## 13 Riesgos y lecciones aprendidas

A continuación, detallamos los riesgos y lecciones aprendidas durante el desarrollo del trabajo. Los puntos principales se dieron en la elección del lenguaje de programación y el desarrollo de la captura, transmisión y reproducción multimedia.

### 13.1 Elección del lenguaje de programación

Al haber tomado la decisión de utilizar WebRTC en nuestro trabajo y dadas las diversas problemáticas que el mismo presenta, se nos presentó la necesidad de elegir un lenguaje de programación idóneo. Esto implicaba que cuente con una librería adecuada de WebRTC, que en lo posible sea conocido para los integrantes, que facilite el desarrollo y que a su vez sea altamente performante.

Dada esta situación, desde del principio nuestras opciones se limitaron a C++, Go y Rust. En primer lugar optamos por Go, ya que nos encontramos con Pion [30], una librería de WebRTC que parecía ser de las más utilizadas, proveía muchos ejemplos y contaba con una comunidad muy activa. Luego de las primeras semanas de desarrollo lo relacionado a WebRTC avanzaba correctamente, sin embargo, surgió la necesidad de utilizar ciertas funcionalidades específicas del sistema operativo Windows relacionadas a la captura de vídeo, pero ninguna librería de Go resultaba de utilidad. Luego de esta primer experiencia, tuvimos que considerar esas funcionalidades a la hora de cambiar a un nuevo lenguaje.

En este punto, decidimos volcarnos hacia C++. Esto fue por que creímos útil el hecho de que este sea un lenguaje de aún más bajo nivel, contábamos con una librería de WebRTC [31] y de la API de Windows bien documentadas, y también con algunos ejemplos básicos. Sin embargo, los problemas comenzaron cuando quisimos utilizar tracks para transmisión multimedia sin éxito y no encontrábamos respuestas en su comunidad, además a medida que crecía el proyecto se fue tornando más complejo el desarrollo, considerando también que este no era un lenguaje en el que contábamos con mucha experiencia.

Teniendo en cuenta lo anterior, finalmente optamos por Rust, ya que cumplía todos los requisitos que necesitábamos. Aquellas primeras semanas si bien fueron fructíferas, tomaron más tiempo del esperado al tener que llevar a cabo migraciones de lo implementado hacia los distintos lenguajes.

## 13.2 Implementación de captura, transmisión y reproducción multimedia

Inicialmente, este fue el punto de nuestro trabajo donde más capacitación requerimos todos los integrantes. Al comenzar el desarrollo del proyecto nuestra idea era construir nuestra propia forma de capturar y reproducir vídeo y audio. Con este propósito, exploramos diferentes herramientas alternativas, como la librería de gráficos de Windows o diferentes crates en Rust como cpal, opus-rust u openh264. Si bien llegamos a una primera versión con transmisión de audio entre los pares, esta no funcionaba de forma totalmente correcta. Por otro lado, al intentar comprimir el vídeo para enviarlo los tiempos obtenidos eran muy lejanos a algo medianamente aceptable.

Por lo tanto, nos dimos cuenta de que la forma en la cual estábamos intentando resolver el problema no era la mejor opción y decidimos buscar una alternativa superadora, este fue el caso de Gstreamer. Gstreamer resolvía muchos de los problemas que teníamos hasta ese momento y además contaba con una gran variedad de ejemplos, una extensa y detallada documentación y casos de uso reales similares a lo que necesitábamos. Al llegar a este punto ya habían pasado varios meses desde el comienzo del proyecto, prolongándose considerablemente lo estimado para esta parte del trabajo.

Finalmente, si bien Gstreamer nos facilitaba en gran parte la complejidad de la captura y reproducción multimedia, también tuvo una elevada curva de aprendizaje para lograr su funcionamiento y optimizar su uso. Dedicándole muchas horas de lectura de documentación y pruebas locales o entre pares para evaluarlo.

## 14 Trabajos Futuros

Esta sección tiene como objetivo mencionar algunas mejoras y funcionalidades que podrían implementarse en un futuro, ya que las mismas escapaban del alcance de nuestro trabajo.

### 14.1 Captura de ventana de juego

El sistema cuenta con la limitación de solo poder funcionar con juegos que se ejecuten en una única ventana durante todo su ciclo de vida. Por ejemplo, títulos como Valorant o League of Legends primero abren una ventana para el launcher y luego gestionan otras ventanas después de esta. La mejora consistiría en la implementación de un mecanismo que pueda alternar la captura de pantalla entre las ventanas pertenecientes al juego de modo de mantener el ambiente aislado para el jugador.

### 14.2 Controles y compatibilidad

No todos los juegos capturan la entrada de comandos del jugador de la misma manera. Por ejemplo, para algunos juegos el ratón funciona pero el teclado no. Esto puede tener que ver con mecanismos de prevención de trampas que adoptan los mismos. La mejora en este caso podría ser hallar una manera de que los comandos que envíe el usuario sean aceptados por cualquier juego que se ejecute en el sistema o al menos ampliar mas la compatibilidad.

### 14.3 Extracción de dinero

Actualmente se cuenta con la posibilidad de ingresar dinero en la plataforma a través de Mercadopago. Queda pendiente la implementación de la funcionalidad de extracción de dinero.

### 14.4 Multiplataforma

La implementación realizada solo funciona en Windows pero podría soportar también otros sistemas operativos. Para desarrollar esto sería necesario construir los pipelines de Gstreamer con plugins específicos, utilizar alguna alternativa a la librería Wininput y revisar algunas cuestiones de los mecanismos de automatización.

### 14.5 Mejoras en el servidor

Dado el contexto de nuestro trabajo no resultó necesario implementar un servidor escalable y tolerante a fallos. En el caso de contar con múltiples usuarios interactuando en simultáneo con el servidor o en el escenario de desplegar el software en producción sería importante solventar los puntos mencionados. Podría considerarse también, en el caso de aumentar notablemente las funcionalidades, pasar a una arquitectura de microservicios.

## 14.6 Otras mejoras

Finalmente, mencionaremos algunas ideas que podrían mejorar la experiencia del usuario, como el uso de coordenadas para búsquedas por cercanía, la incorporación de reseñas de las sesiones de juego, la posibilidad de buscar juegos por nombre o género, la exposición de detalles del hardware del Offerer y la capacidad de obtener mejores offerers según el ping.

## 15 Conclusiones

El desarrollo llevado a cabo en este Trabajo Profesional nos permitió reexplorar conceptos abordados a lo largo de toda la carrera, profundizando en algunos e incorporando otros, además de integrarlos de manera efectiva. Nuestro servicio descentralizado abarca temas como concurrencia, arquitectura cliente-servidor y P2P, bases de datos y una interfaz gráfica. También tuvimos que aprender y aplicar nuevos conocimientos, como el protocolo WebRTC, el framework GStreamer y nociones relacionadas con la captura, transmisión y reproducción de contenido multimedia.

En cuanto a los resultados obtenidos, destacamos el cumplimiento de los principales objetivos propuestos al comenzar el proyecto. En primer lugar, como observamos en la sección de validación, logramos transmitir videojuegos a una latencia adecuada para jugar de manera confortable según lo que entendíamos necesario desde un principio, siempre y cuando se cumplan las condiciones necesarias. En segundo lugar, la interfaz de usuario se diseño y construyó para ser intuitiva y fácil de usar, permitiendo a los usuarios ofrecer y alquilar computadoras para jugar de manera sencilla.

Finalmente, otro punto a destacar que no fue visto al analizar otros servicios de Cloud Gaming y diferencia al nuestro, es el modelo de negocio innovador. Nuestro servicio les da a los usuarios la oportunidad de aprovechar su hardware inutilizado para obtener un rédito al ofrecer su computadora para que otros que quizá no tenían la posibilidad, ahora puedan jugar una numerosa cantidad de juegos. Esto permitiría crear una comunidad de usuarios que se benefician mutuamente, lo que no se observa en los servicios de Cloud Gaming tradicionales.

## 16 Referencias

- [1] British Esports. *Ping, latency and lag: What you need to know* (2022). Recuperado de <https://britishesports.org/general-esports-info/ping-latency-and-lag-what-you-need-to-know>
- [2] LogicalIncrements. *Information About Frame Rate* (2023). Recuperado de <https://www.logicalincrements.com/articles/framerate>
- [3] McDonald, Emma. *Cloud Gaming Revenues to Hit \$2.4 Billion in 2022, up +74 % Year-on-Year; Revenues Will Triple by 2025*. Recuperado de <https://newzoo.com/resources/blog/cloud-gaming-revenues-to-hit-2-4-billion-in-2022-up-74-year-on-year-revenues-will-triple-by-2025>
- [4] Cueto, Héctor. *Más de 20 millones de personas han usado Xbox Cloud Gaming, según Microsoft* (2022).. Recuperado de [https://businessinsider.mx/xbox-cloud-gaming-rebasa-20-millones-usuarios-microsoft\\_tecnologia](https://businessinsider.mx/xbox-cloud-gaming-rebasa-20-millones-usuarios-microsoft_tecnologia)
- [5] McDonald, Emma. *The Cloud Gaming Ecosystem*. Recuperado de <https://newzoo.com/resources/blog/cloud-gaming-ecosystem-diagram-2022>
- [6] Steve Klabnik and Carol Nichols, with contributions from the Rust Community. *The Rust Programming Language* Recuperado de <https://doc.rust-lang.org/book/>
- [7] WebRTC Sitio oficial de WebRTC Recuperado de <https://webrtc.org/?hl=es-419>
- [8] WebRTC for the Curious. *WebRTC for the Curious* (2022). Recuperado de <https://webrtcforthecurious.com/>
- [9] J. Rosenberg, J. Weinberger, dynamicssoft, C. Huitema, Microsoft, R. Mahy, Cisco. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* (2003). Recuperado de <https://www.rfc-editor.org/rfc/rfc3489.html>
- [10] J. Rosenberg, Cisco, R. Mahy, P. Matthews, D. Wing. *Session Traversal Utilities for NAT (STUN)* (2008). Recuperado de <https://www.rfc-editor.org/rfc/rfc5389.html>
- [11] T. Reddy, Ed, McAfee,A. Johnston, Ed.,Villanova University,P. Matthews,Alcatel-Lucent,J. Rosenberg,jdrosen.net *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* (2020) Recuperado de <https://www.rfc-editor.org/rfc/rfc8656.html>

- [12] Coturn. *Github: Coturn TURN server* Recuperado de <https://github.com/coturn/coturn>
- [13] A. Keranen, C. Holmberg, Ericsson, J. Rosenberg, jdrosen.net. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal (2018)* Recuperado de <https://www.rfc-editor.org/rfc/rfc8445.html>
- [14] A. Begen, Networked Media, P. Kyzivat, C. Perkins, University of Glasgow, M. Handley, UCL *SDP: Session Description Protocol (2021)* Recuperado de <https://www.rfc-editor.org/rfc/rfc866.html>
- [15] I. Fette, Google, Inc, A. Melnikov, Isode Ltd. *The WebSocket Protocol (2011)* Recuperado de <https://www.rfc-editor.org/rfc/rfc6455.html>
- [16] A. Begen, Networked Media, P. Kyzivat, C. Perkins, University of Glasgow, M. Handley, UCL *Stream Control Transmission Protocol (2021)* Recuperado de <https://www.rfc-editor.org/rfc/rfc4960.html>
- [17] C. Perkins, M. Westerlund, J. Ott, University of Glasgow, Ericsson, Technical University Munich. *RFC 8834: Media Transport and Use of RTP in WebRTC (2021)* Recuperado de <https://www.rfc-editor.org/rfc/rfc8888.html>
- [18] Z. Sarker, C. Perkins, V. Singh, M. Ramalho, Ericsson AB, University of Glasgow, callstats.io, AcousticComms *RTP Control Protocol (RTCP) Feedback for Congestion Control (2021)* Recuperado de <https://www.rfc-editor.org/rfc/rfc8888.html>
- [19] JM. Valin, Mozilla Corporation, K. Vos, Skype Technologies S.A., T. Terriberry, *Definition of the Opus Audio Codec (2012)* Recuperado de <https://www.rfc-editor.org/rfc/rfc6716.html>
- [20] Y.-K. Wang, R. Even, Huawei Technologies, T. Kristensen, Tandberg, R. Jesup, WorldGate Communications *RTP Payload Format for H.264 Video (2011)* Recuperado de <https://www.rfc-editor.org/rfc/rfc6184.html>
- [21] E. Rescorla, H. Tschofenig, N. Modadugu, Mozilla, Arm Limited, Google, Inc.. *RFC 9147: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 (2022)* Recuperado de <https://www.rfc-editor.org/rfc/rfc6184.html>
- [22] M. Baugher, D. McGrew, Cisco Systems, Inc., M. Naslund, K. Norrman, Ericsson Research (2004 ) *The Secure Real-time Transport Protocol (SRTP)* Recuperado de <https://www.rfc-editor.org/rfc/rfc3711.html>

- [23] Gstreamer *Documentación oficial de Gstreamer* Recuperado de <https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html?gi-language=c>
- [24] Gstreamer Plugins *Documentación oficial de Gstreamer* Recuperado de [https://gstreamer.freedesktop.org/documentation/plugins\\_doc.html?gi-language=c](https://gstreamer.freedesktop.org/documentation/plugins_doc.html?gi-language=c)
- [25] Gstreamer Foundations *Documentación oficial de Gstreamer* Recuperado de <https://gstreamer.freedesktop.org/documentation/application-development/introduction/basics.html?gi-language=c>
- [26] IDS *Color pixel formats* (2024) Recuperado de <https://www.1stvision.com/cameras/IDS-IDS-manuals/en/basics-color-pixel-formats.html>
- [27] Paul Bourke *NV12 yuv pixel format* (2016) Recuperado de <https://paulbourke.net/dataformats/nv12/>
- [28] Karl Tastroff *winput* Recuperado de <https://crates.io/crates/winput>
- [29] DOCS.RS *Module tokio::task* Recuperado de <https://docs.rs/tokio/latest/tokio/task/index.html>
- [30] Golang Community *WebRTC - Pion* Recuperado de <https://pkg.go.dev/github.com/pion/webrtc/v3>
- [31] Paul-Louis *libdatachannel - C/C++ WebRTC network library* Recuperado de <https://github.com/paullouisageneau/libdatachannel>
- [32] D. Mills, University of Delaware *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI* (2006) Recuperado de <https://www.rfc-editor.org/rfc/rfc4330.html>

## 17 Anexos

En esta sección se presentan los anexos correspondientes al proyecto. Se incluyen los enlaces a los repositorios de los diferentes componentes del sistema, así como el protocolo diseñado para la correcta comunicación y funcionamiento del mismo.

### 17.1 Enlaces a los Repositorios e Instalador

- **Nodo de Procesamiento**
- **Servidor Central**
- **Interfaz Gráfica**
- **Instalador**

### 17.2 Protocolos

Esta sección está destinada a exponer el protocolo implementado para la comunicación entre servidor central, interfaz gráfica y nodo de procesamiento.

El protocolo busca que la semántica de los mensajes sea clara en lo que está haciendo. Los mensajes están compuestos por uno o más campos, en donde el primer campo define en la mayoría de los casos la acción a realizar. Por ejemplo, en el mensaje `startGameWithUser | username | userToConnect | gameName | minutes`, el primer campo `startGameWithUser` indica que se debe iniciar un juego con otro usuario. Los campos subsiguientes proporcionan la información necesaria para ejecutar la acción: `username` especifica el nombre del usuario que solicita la conexión, `userToConnect` indica el usuario con el que se desea conectar, `gameName` identifica el juego a iniciar, y `minutes` define la duración de la sesión de juego en minutos.

En dos casos particulares, a diferencia de lo mencionado, el primer campo puede indicar que se está enviando el Session Description Protocol (SDP). Por ejemplo, en el mensaje `clientSdp | offererUsername | sdp`, el primer campo `clientSdp` indica que se está enviando una descripción de sesión del cliente, `offererUsername` especifica el nombre del usuario que está ofreciendo la conexión, y `sdp` contiene la descripción de la sesión.

La tabla a continuación presenta el protocolo mencionado. Cada fila de la tabla muestra un mensaje específico, indicando su emisor, receptor, contenido y una breve descripción del mismo. Los acrónimos utilizados en la tabla son los siguientes: SC se refiere al Servidor Central, IG corresponde a la Interfaz Gráfica y NP designa el Nodo de Procesamiento. En los casos en que se deba distinguir entre diferentes roles dentro de los nodos de procesamiento, se utilizan los subíndices Offerer y Receiver.

Emisor	Receptor	Mensaje	Descripción
IG	NPofferer	startOffering   offererUsername	Enviado por la interfaz gráfica cuando el Offerer quiere comenzar a ofrecer sus servicios.
IG	NPreceiver	startGameWithUser   clientUsername   offererUsername   gameName   minutes	Enviado por la interfaz gráfica cuando el Receiver quiere comenzar una sesión de juego.
IG	NP	disconnect	Enviado por la interfaz gráfica para interrumpir la conexión de la sesión.
IG	SC	disconnectOfferer   offererUsername	Enviado por la interfaz gráfica indicando que el Offerer no continuara prestando servicios.
IG	SC	subscribe   usernameSubscriber	Enviado por la interfaz gráfica indicando que un usuario se suscribe a las notificaciones del servicio.
NPofferer	SC	initOfferer   offererUsername	Enviado por el nodo de procesamiento indicando al servidor que esta disponible para ofrecer servicios.
NPofferer	SC	offererSdp   clientUsername   sdp	Enviado por el Offerer para transmitir el sdp generado.
NPreceiver	SC	clientSdp   offererUsername   sdp	Enviado por el Receiver para transmitir el sdp generado.
NPreceiver	SC	initClient   clientUsername   offererUsername   gameName   minutes	Enviado por el nodo de procesamiento indicando al servidor que se quiere iniciar una sesión.
NPofferer	SC	startSession   offererUsername   clientUsername   minutes	Enviado por el Offerer indicando el comienzo de la sesión.
NP	SC	forceStopSession   username	Enviado por el nodo de procesamiento indicando el cierre forzoso de la sesión.

Emisor	Receptor	Mensaje	Descripción
SC	NPofferer	sdpRequestFrom   clientUsername   gameName   gamePath   minutes	Enviado por el servidor central indicando al Offerer que llego una solicitud de sesión.
SC	NPofferer	sdpClient   sdp	Enviado por el servidor central para transmitir el sdp del cliente.
SC	NPreceiver	sdpOfferer   sdp	Enviado por el servidor para transmitir el sdp del Offerer.
SC	NP	notifEndSession	Enviado por el servidor indicando que la sesión fue finalizada.
SC	IG	notifConnection   offererUsername   grade   games	Enviado por el servidor indicando que un nuevo Offerer esta prestando servicios.
SC	IG	notifDisconnection   offererUsername	Enviado por el servidor indicando que un Offerer deja de prestar servicios.
SC	IG	notifPayment   usernameSubscriber   quantity	Enviado por el servidor indicando que se concreto un pago en la plataforma.
SC	IG	notifForceStopSession   sessionTerminator   offererUsername   clientUsername   sessionTime	Enviado por el servidor por un cierre forzoso, indicando quien envio el mensaje, cuanto fue el tiempo de sesion y quienes participaron.
SC	IG	notifEndSession   offererUsername   clientUsername   sessionTime	Enviado por el servidor indicando que la sesión fue finalizada, por cuanto tiempo y quienes participaron.
SC	IG	sessionStarted   offererUsername   clientUsername	Enviado por el sevidor cuando comienza una sesión de juego, indicando quienes participaron

## 17.3 Diseño y Accesibilidad

En este anexo, se presentarán capturas de la interfaz gráfica desarrollada, las cuales evidencian la implementación de una interfaz intuitiva y accesible. Las capturas ilustran cómo los usuarios pueden interactuar de manera eficiente con el sistema, facilitando la comprensión y utilización de sus diversas características.

### 17.3.1 Autenticación

En la figura 26 podemos observar la pantalla de inicio de sesión mientras que en la 27 podemos ver la de registro.

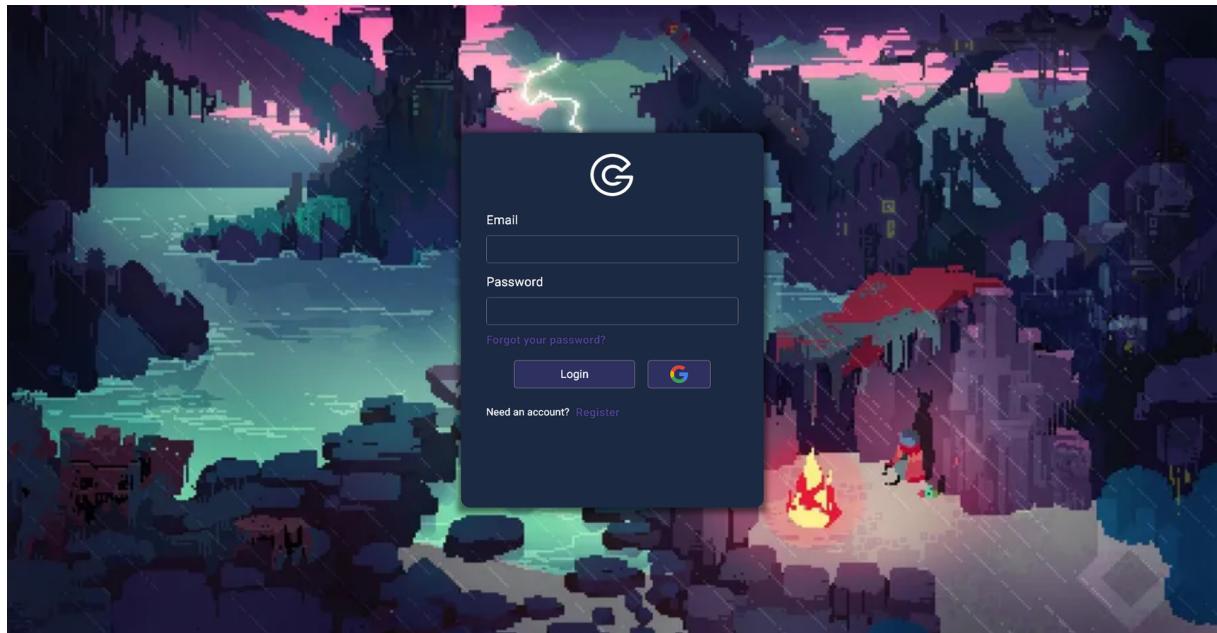


Figura 26: Inicio de sesión

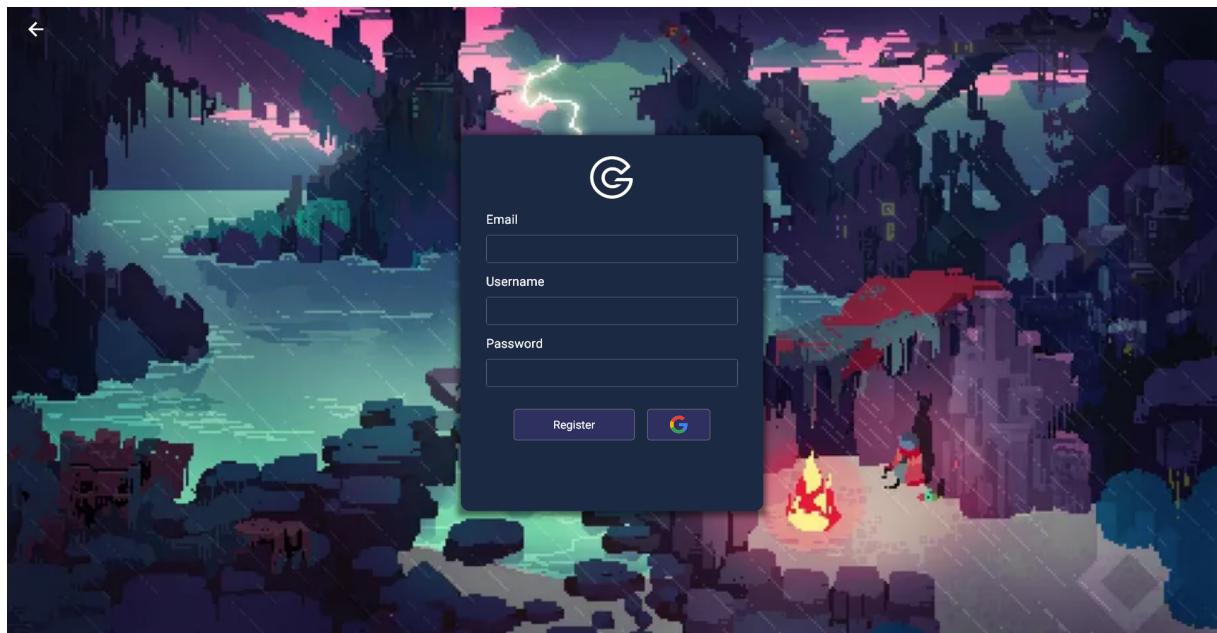


Figura 27: Registro

### 17.3.2 Menú

Desde la pantalla de inicio podemos seleccionar un juego para jugar (figura 28).

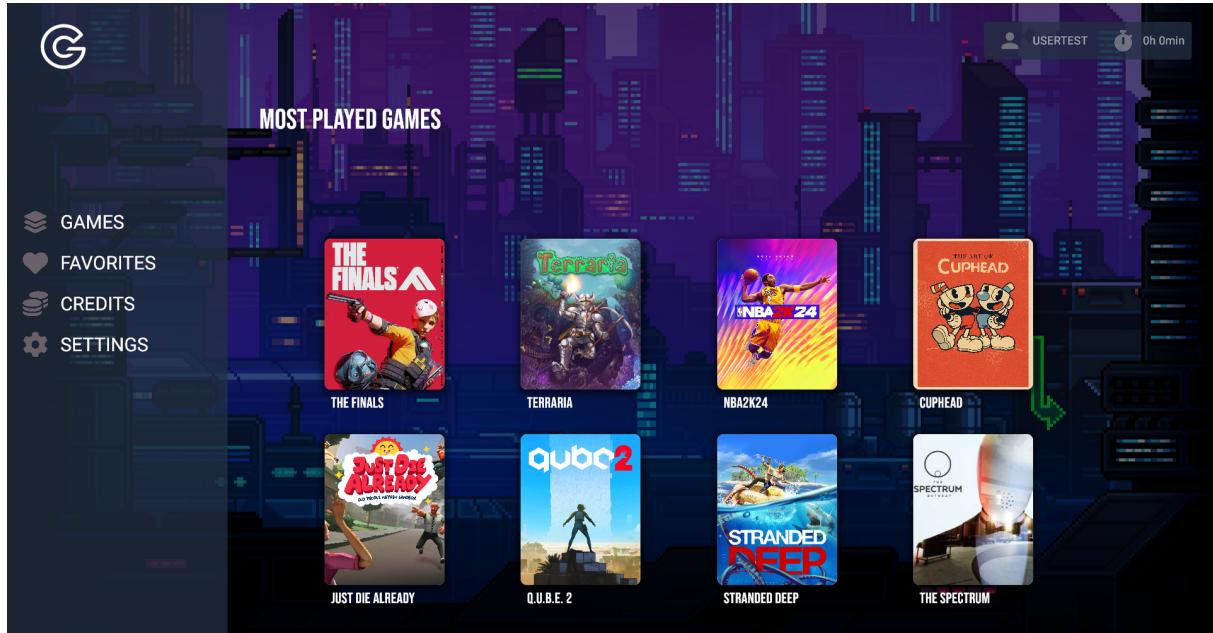


Figura 28: Menú principal

### 17.3.3 Sesión de juego

Una vez seleccionado un juego, podremos observar una lista de aquellos usuarios a los que nos podemos conectar para jugar (figura 29). Cuando ya hayamos decidido a qué usuario conectarnos, podremos seleccionar la cantidad de horas que queremos jugar (figura 30) y luego, comenzará la sesión (figura 31).

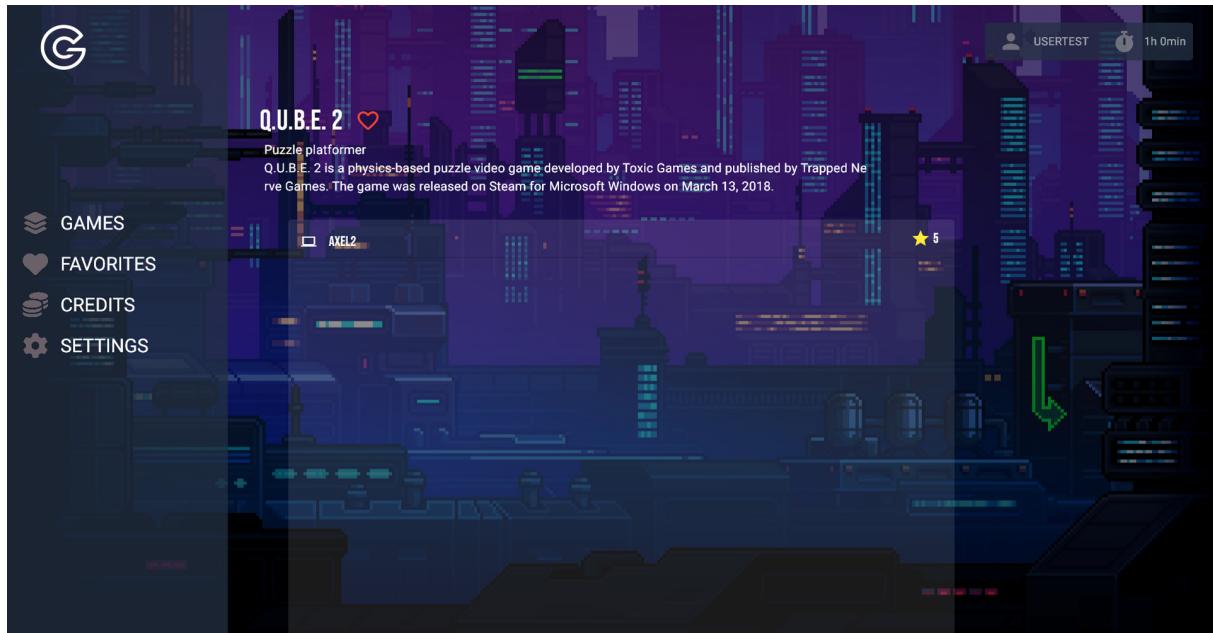


Figura 29: Listado de oferentes

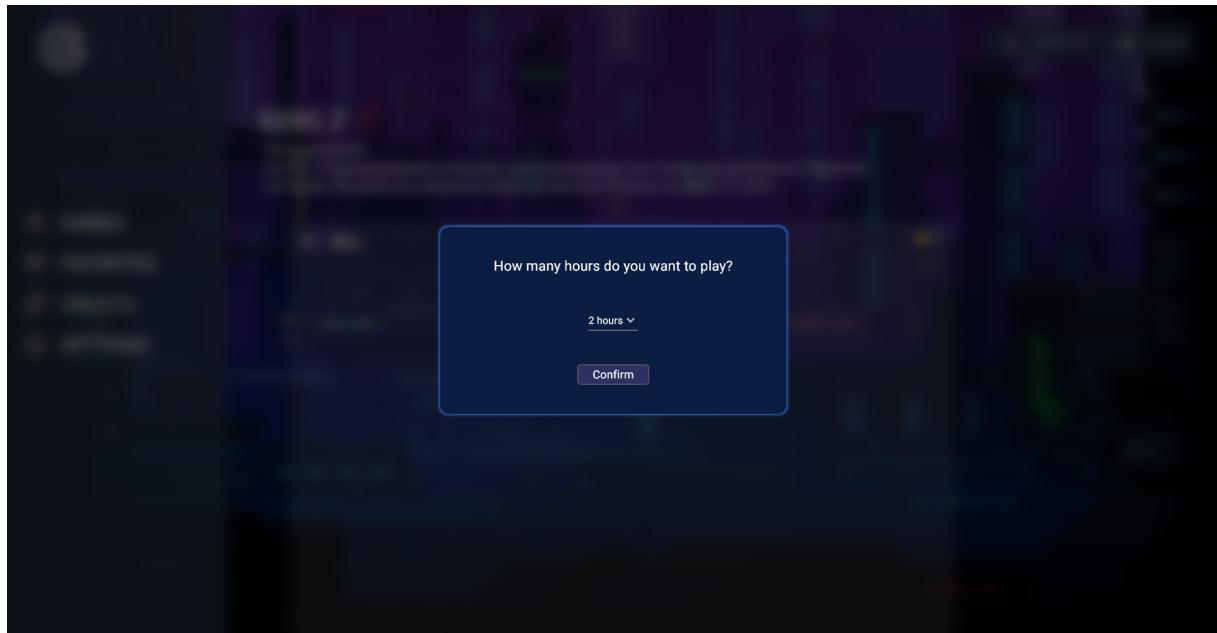


Figura 30: Tiempo de juego

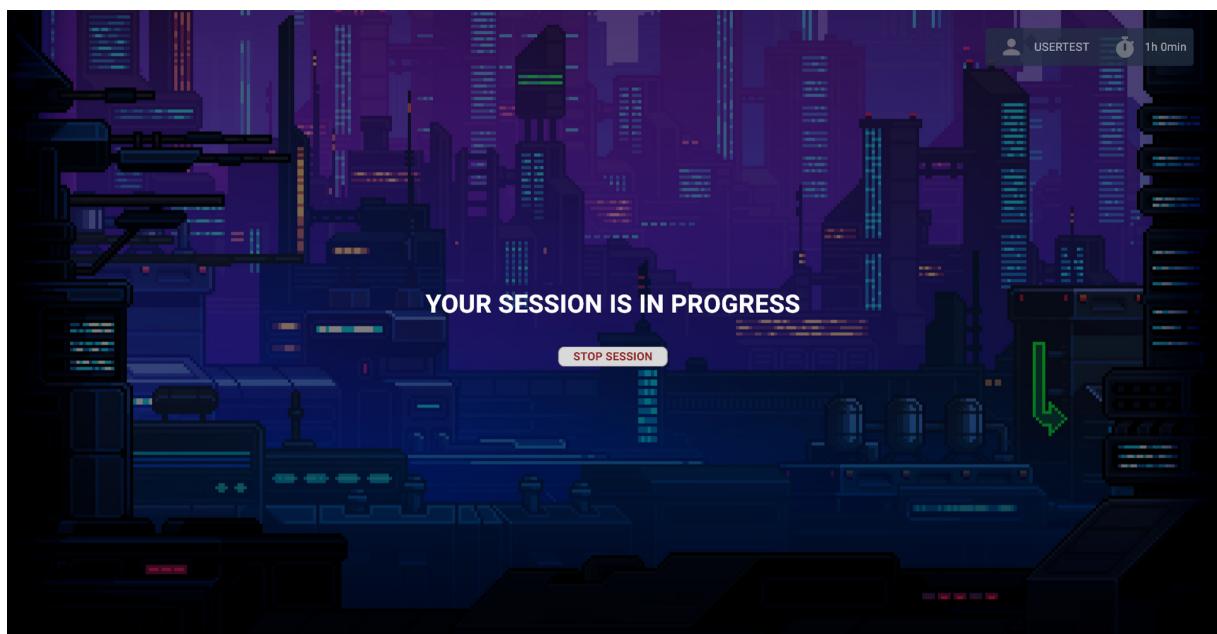


Figura 31: Juego en progreso

#### 17.3.4 Agregar Juego

Desde pantalla "Games"(figura 36) podremos agregar aquellos juegos que querramos ofrecer, seleccionando la ruta del ejecutable como se lo muestra en las figuras 36, 33, 34 y, 35.

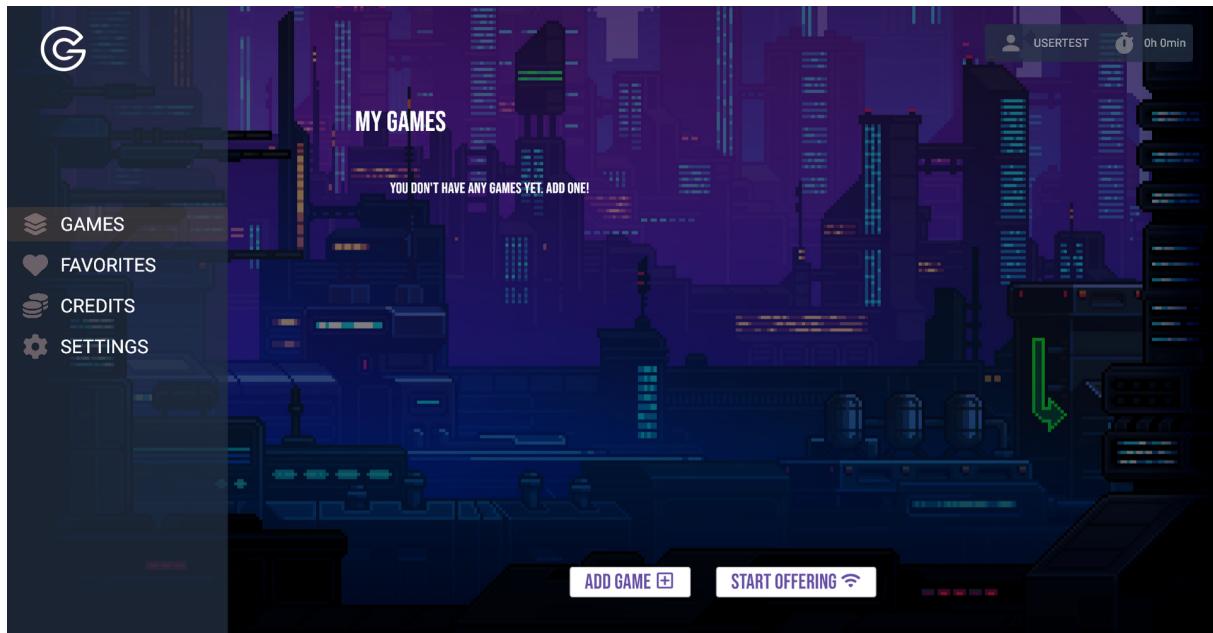


Figura 32: Mis juegos

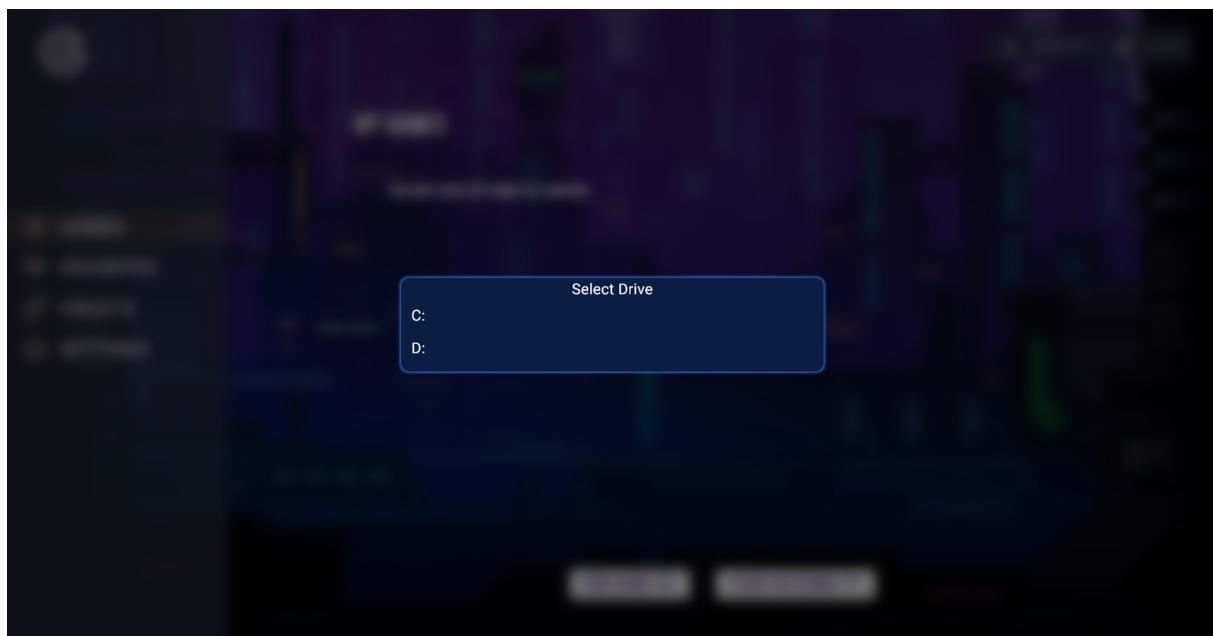


Figura 33: Selección de disco



Figura 34: Seleccion de path

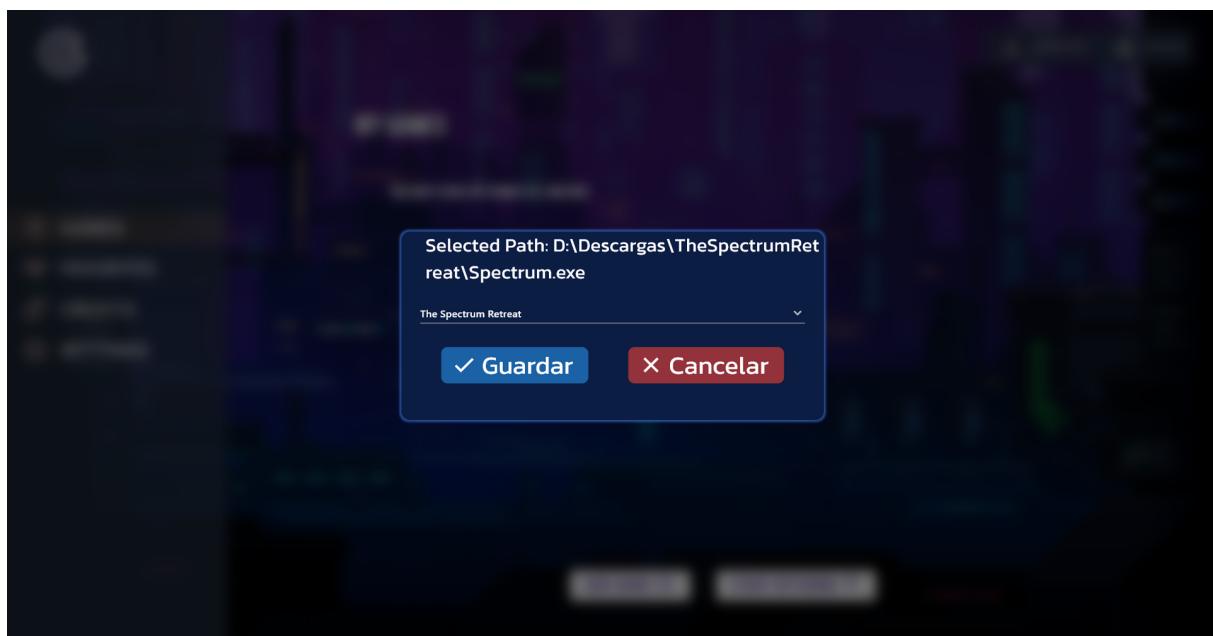


Figura 35: Guardar juego

### 17.3.5 Ofrecer Juego

Una vez cargados aquellos juegos que deseamos ofrecer, podremos comenzar a ofrecer desde el botón start offering (figura 36) Y comenzaremos a esperar hasta que un usuario se conecte (figuras 37 y 38).

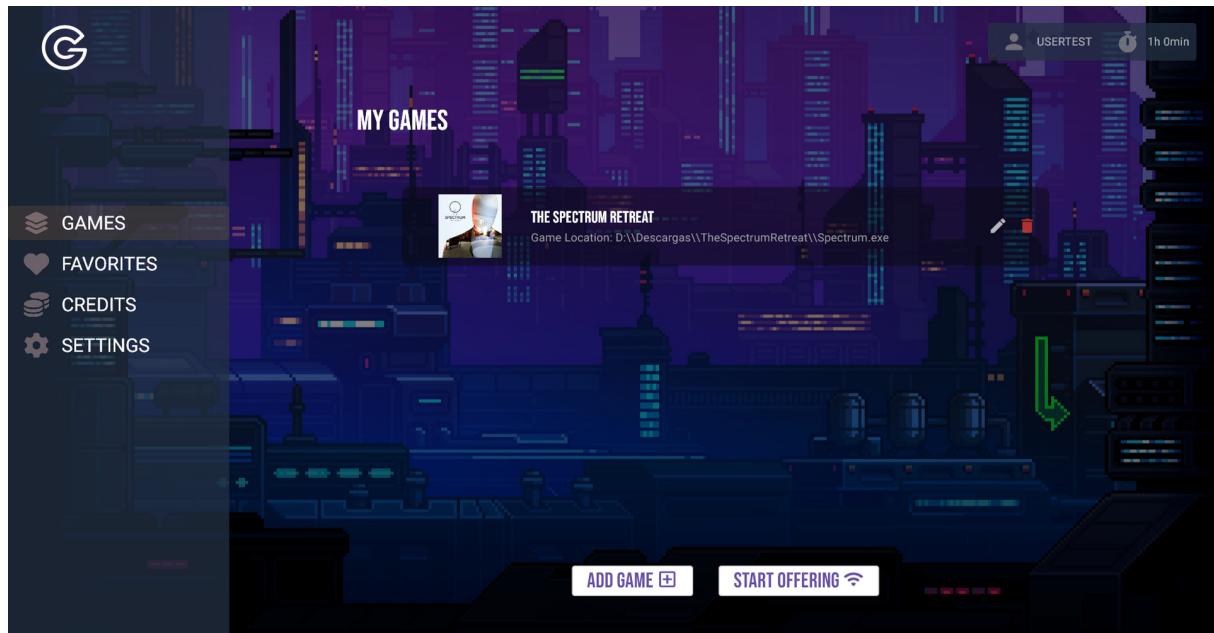


Figura 36: Mis juegos

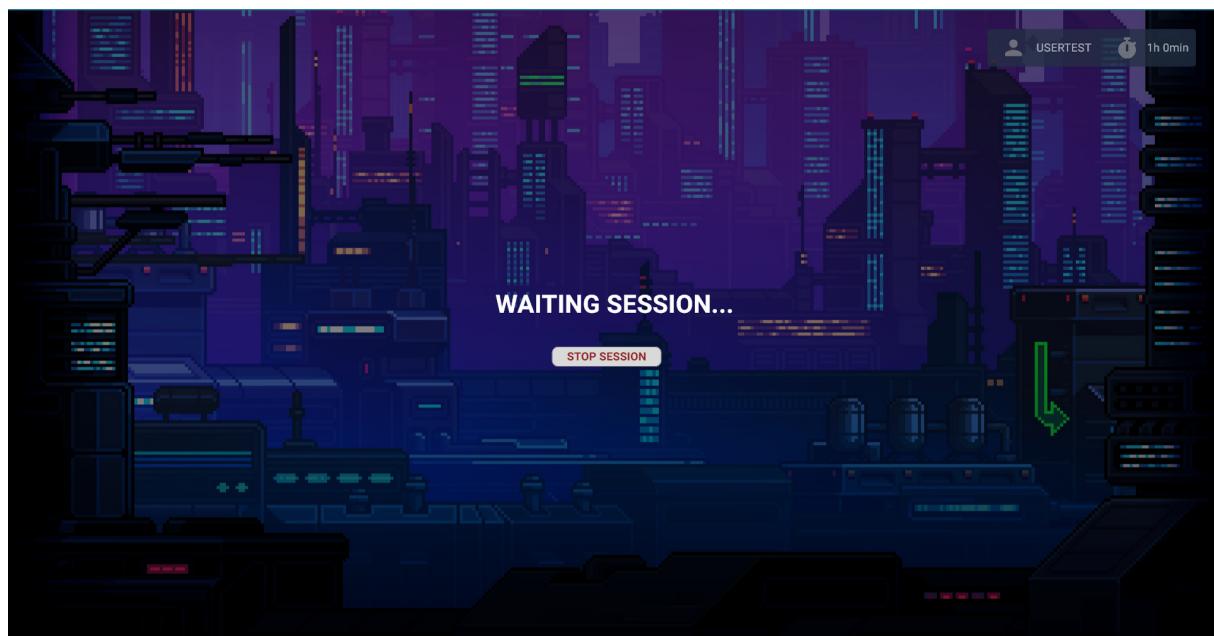


Figura 37: Esperando sesión

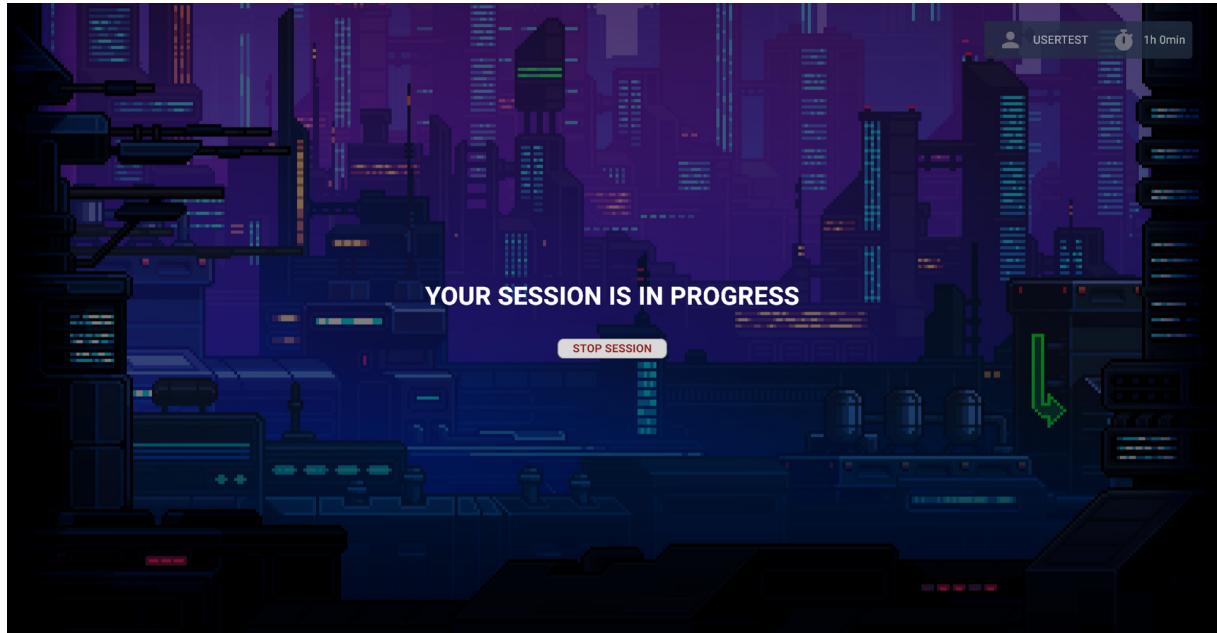


Figura 38: Sesion en progreso

#### 17.3.6 Cargar creditos

Desde la pantalla Credits podremos cargar una cantidad créditos predeterminada (figura 39). Una vez cargados se verá reflejado en la misma pantalla así como tambien en la parte superior derecha de la aplicación (figura 40).

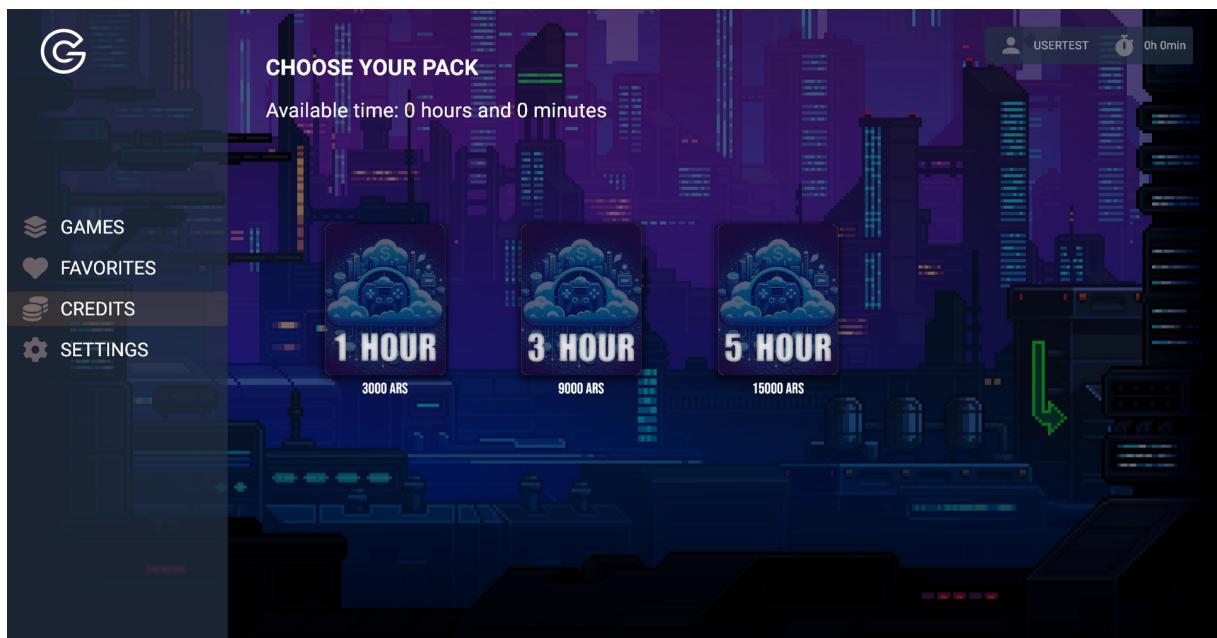


Figura 39: Menu de creditos

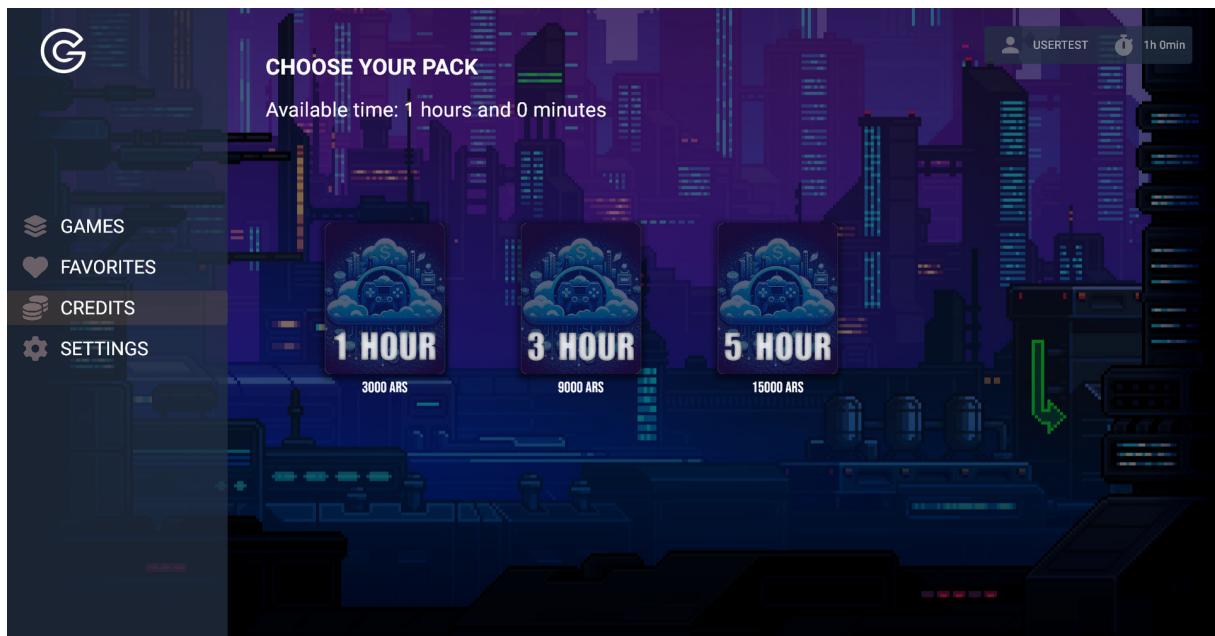


Figura 40: Carga realizada