



UAI

Universidad Abierta
Interamericana



Facultad de Tecnología Informática

Ingeniería en Sistemas Informáticos



MODELOS COMPUTACIONES DE GESTIÓN ADMINISTRATIVA

TRABAJO PRACTICO FINAL

“CASINOS DEL NORTE”

Terminales y Servidor de Terminales

Comisión 4º A

Turno y Sede Noche - Norte

Docente Luna, Sebastian

Alumnos:

Gonzalez, Aldo Daniel
Sochan, Guido Alejandro

2023

Tabla de Contenidos

1	INTRODUCCIÓN	3
1.1	OBJETIVO DEL DOCUMENTO.....	3
1.2	ALCANCE DEL SISTEMA	3
1.3	DEFINICIÓN DE TÉRMINOS CLAVE.....	3
2	ASPECTOS DESCRIPTIVOS DE LA SOLUCIÓN TECNOLÓGICA	4
2.1	ARQUITECTURA DEL SISTEMA	4
2.2	LENGUAJES DE PROGRAMACIÓN Y TECNOLOGÍAS UTILIZADAS	5
2.3	MODELO DE DATOS	7
2.4	SEGURIDAD.....	8
3	DESCRIPCIÓN DEL SISTEMA.....	9
3.1	VISIÓN GENERAL DEL SISTEMA	9
3.2	FUNCIONALIDADES PRINCIPALES.....	9
3.3	BENEFICIOS Y VENTAJAS DEL SISTEMA	9
4	LIMITACIONES DE LA SOLUCIÓN	10
4.1	RESTRICCIONES, SUPOSICIONES Y LIMITACIONES TÉCNICAS	10
4.2	FUNCIONALIDADES FUTURAS NO CONTEMPLADAS	11
5	REQUERIMIENTOS	12
5.1	REQUERIMIENTOS FUNCIONALES.....	12
5.2	REQUERIMIENTOS NO FUNCIONALES	12
6	CASOS DE USO.....	14
6.1	DIAGRAMAS DE CASOS DE USO	14
6.2	ESPECIFICACIONES DE CASOS DE USO	15
7	MANUAL DEL SISTEMA.....	20
7.1	INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA	20
7.2	DOCUMENTACIÓN Y FUENTES DEL PROYECTO.....	31
7.3	MANUAL DE USO TERMINALES TRAGAMONEDAS.....	32

1 Introducción

1.1 Objetivo del Documento

Este documento tiene como objetivo proporcionar una visión detallada del sistema de gestión de casinos, centrándose específicamente en los componentes de TERMINALES y SERVIDOR DE TERMINALES.

1.2 Alcance del Sistema

El alcance del sistema abarca la implementación y funcionalidad de las terminales de juego en un casino, así como el servidor encargado de gestionar las conexiones y operaciones de estas terminales y su integración con la capa de Backoffice.

Sin embargo, es importante tener en cuenta que este documento no profundizará en aspectos más complejos, como la implementación de componentes tales como el Servidor Operacional de Backoffice, los Microservicios de depósito, pagos y service bus, así como el Microservicio web core, proxy y Front end.

Estos componentes deberán ser implementados, detallados y tratados en otros documentos.

1.3 Definición de términos clave

A continuación, se proporciona una definición de los términos clave utilizados en el contexto de este sistema:

Terminales: Aplicaciones de escritorio representando juegos de casino.

Servidor de Terminales: Aplicación de consola TCP/IP que nuclea las conexiones de las terminales, gestionando apuestas y premios.

Servicio de Bitacora Transversal: Aplicación API Rest que registra las operaciones funcionales y no funcionales de toda la solución, incluyendo los componentes adicionales no tratados en este documento.

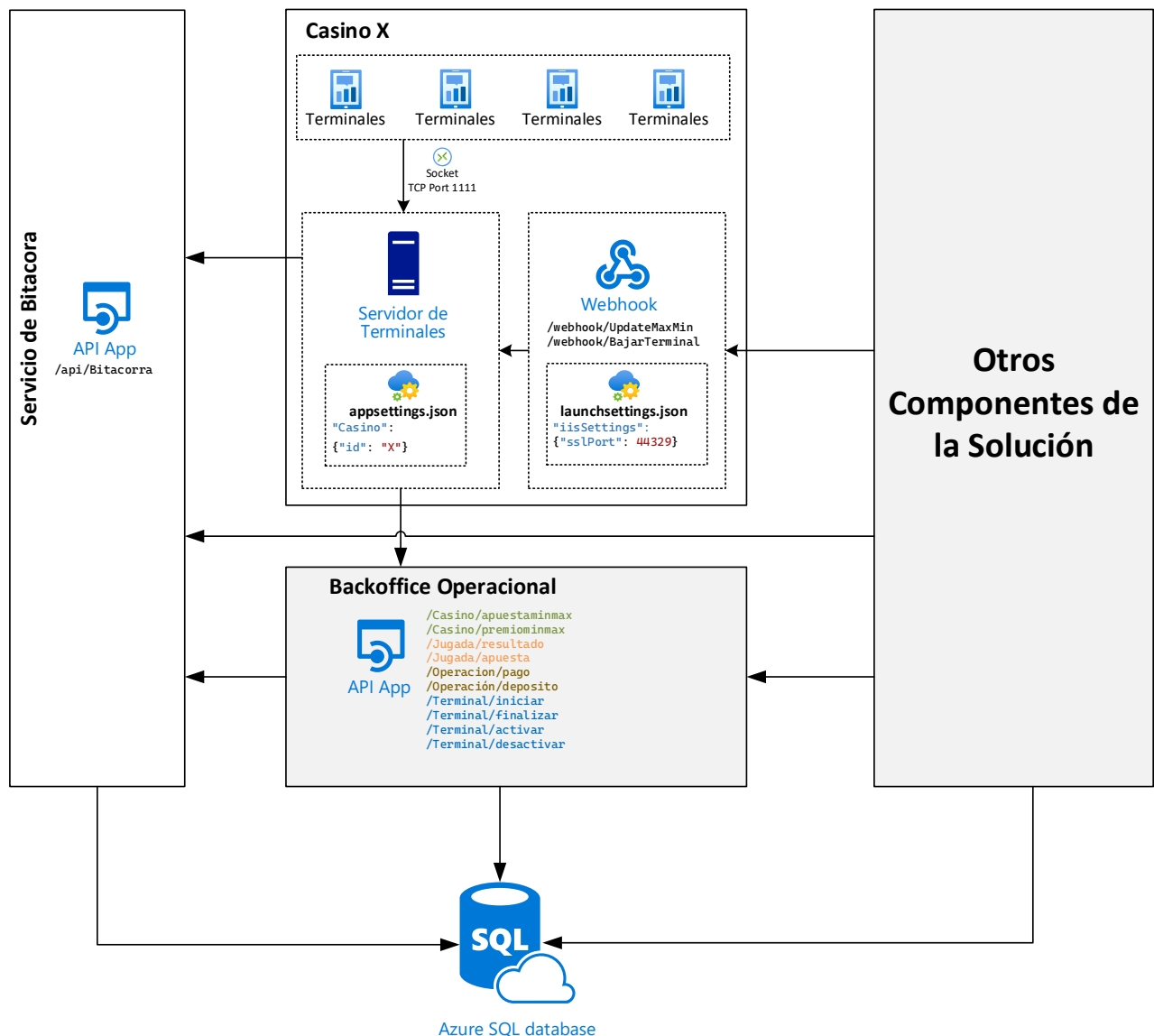
Estas definiciones de términos clave proporcionan una base común de comprensión para el resto de la documentación y ayudan a los lectores a familiarizarse con los conceptos fundamentales del sistema.

Con esta información, se ha establecido una introducción sólida que proporciona una visión general del sistema, sus objetivos, alcance y los términos clave utilizados. A partir de aquí, se puede continuar con el desarrollo de los siguientes apartados de la documentación.

2 Aspectos Descriptivos de la Solución Tecnológica

2.1 Arquitectura del sistema

La arquitectura del sistema es una parte fundamental en el diseño y desarrollo de cualquier solución tecnológica. Define la estructura general del sistema y cómo sus componentes interactúan entre sí para lograr los objetivos establecidos. La arquitectura consta de Terminales y un Servidor de Terminales, comunicándose a través de un servidor TCP/IP. Se utiliza un modelo escalable horizontalmente para manejar aumentos de demanda. Además, se incluye una API Rest que hace de webhook para acciones funcionales inmediatas provistas para ser utilizadas por los demás componentes de la solución.



2.2 Lenguajes de programación y tecnologías utilizadas

Las tecnologías y lenguajes utilizados son los siguientes:

- Aplicación de Terminales: Aplicación WPF C# (Windows Presentation Foundation). Framework: .NET 6.0.
- Servidor de Terminales: Aplicación de Consola C#. Servicio TCP/IP Socket. Framework: .NET 6.0.
- Webhook: Aplicación de Consola C#. API Rest. Framework: .NET 6.0.
- Servicio de Bitacoras: Aplicación de Consola C#. API Rest. Framework: .NET 7.0.
- Sistema de Base de Datos: Azure SQL Database.
- Sistema de control de versiones: GitHub:
 - <https://github.com/AldoDanielGonzalez/MCGA-TP.Final>
- Sistema de despliegue: GitHub Actions.
- Microsoft Azure App Service Plan: Para ejecución de servicios de API de Bitacora y Servicio de Backoffice Operacional (no detallado en este documento).

A continuación, se detallan las librerías utilizadas en la solución, incluyendo versión y una breve descripción de cada una.

Librerías para producción:

- **AutoMapper: 12.0.1** - Biblioteca que facilita el mapeo de objetos entre diferentes capas o modelos, evitando el código repetitivo y aumentando la legibilidad y el mantenimiento.
- **AutoMapper.Extensions.Microsoft.DependencyInjection: 12.0.1** - Extensión de AutoMapper que permite integrarlo con el sistema de inyección de dependencias de Microsoft, simplificando la configuración y el uso de los mapeadores en las clases que los requieren.
- **Microsoft.Extensions.Configuration: 7.0.0** - Biblioteca que proporciona una forma flexible y extensible de cargar y acceder a la configuración de la aplicación desde diferentes fuentes, como archivos json, variables de entorno o argumentos de la línea de comandos.
- **Microsoft.Extensions.Configuration.FileExtensions: 2.2.0** - Biblioteca que provee extensiones de métodos para la configuración basada en archivos, como especificar el nombre del archivo o la recarga opcional cuando cambia el archivo.
- **Microsoft.Extensions.Configuration.Json: 2.20** - Biblioteca que provee el soporte para leer archivos json como fuentes de configuración de la aplicación, permitiendo una sintaxis más expresiva y jerárquica que los archivos ini o xml.
- **NetMQ: 4.0.1.13** - Biblioteca que implementa el protocolo ZeroMQ, que es un framework de mensajería distribuida de alto rendimiento y escalabilidad. NetMQ permite crear aplicaciones que se comuniquen entre sí a través de diferentes patrones de mensajes, como solicitud-respuesta, publicación-suscripción o enrutamiento de negociación.
- **Newtonsoft.Json: 13.0.3** - Biblioteca que facilita el trabajo con datos en formato json, permitiendo serializar y deserializar objetos desde y hacia cadenas json, así como manipularlos a través de clases como JObject o JArray. Newtonsoft.Json es ampliamente usada por otras bibliotecas que requieren de intercambio de datos en formato json.
- **Swashbuckle.AspNetCore: 6.5.0** - Biblioteca que integra la herramienta Swashbuckle con ASP.NET Core, lo que permite generar documentación interactiva

de las APIs REST mediante la especificación OpenAPI. Swashbuckle también provee una interfaz web basada en Swagger UI que permite probar las APIs directamente desde el navegador.

- **Microsoft.Bcl.AsyncInterfaces: 7.0.0** - Biblioteca que provee interfaces y tipos para soportar el uso de programación asincrónica en .NET Standard 2.0. Estas interfaces permiten escribir código que sea compatible con diferentes versiones de .NET y que pueda aprovechar las características del lenguaje C# 8.0, como las expresiones async, los métodos await y las enumeraciones asincrónicas.
- **System.Buffers: 4.5.1** - Biblioteca que contiene clases y estructuras para el manejo eficiente de buffers de memoria, que son regiones de memoria reservadas temporalmente para almacenar datos. Entre estas clases se encuentran ArrayPool, que permite reutilizar arreglos de un tamaño determinado, y MemoryPool, que ofrece una forma de alquilar y devolver bloques de memoria gestionados.
- **System.Memory: 4.5.5** - Biblioteca que proporciona tipos que facilitan el acceso y la manipulación de datos en memoria, como Span, Memory, ReadOnlySpan y ReadOnlyMemory, que representan segmentos contiguos de memoria. Estos tipos permiten operar con datos sin necesidad de copiarlos o asignarlos, lo que mejora el rendimiento y reduce el consumo de recursos.
- **System.Numerics.Vectors: 4.5.0** - Biblioteca que ofrece tipos que representan vectores y matrices numéricas, así como operaciones matemáticas básicas sobre ellos. Estos tipos pueden aprovechar las instrucciones SIMD (Single Instruction Multiple Data) de los procesadores modernos, lo que permite acelerar el procesamiento de datos paralelizables, como en aplicaciones gráficas, científicas o de aprendizaje automático.
- **System.Runtime.CompilerServices.Unsafe: 6.0.0** - Biblioteca que expone métodos estáticos que permiten realizar operaciones consideradas inseguras en el lenguaje C#, es decir, que pueden violar la seguridad de tipos o la verificación de límites de memoria. Estos métodos pueden ser útiles para optimizar el rendimiento de ciertas operaciones de bajo nivel, como el acceso a punteros, la conversión de tipos o la manipulación de bits, pero deben usarse con precaución y solo cuando sea necesario.
- **System.Text.Encodings.Web: 7.0.0** - Biblioteca que contiene clases que ayudan a codificar y decodificar cadenas de texto para su uso en la web, de acuerdo a los estándares definidos por el W3C y la IETF. Estas clases permiten escapar los caracteres que pueden tener un significado especial en HTML, JavaScript, URL u otros contextos web, y evitar así posibles ataques de inyección de código o vulnerabilidades de seguridad.
- **System.Text.Json: 7.0.3** - Biblioteca que provee funcionalidades para leer, escribir y manipular datos en formato json, utilizando el espacio de nombres System.Text.Json. Esta biblioteca ofrece una alternativa a Newtonsoft.Json, con ventajas en rendimiento, compatibilidad con .NET Core y soporte para las características del lenguaje C# 8.0 y .NET 5.0, como las expresiones async, los métodos await y las clases anónimas.
- **System.Threading.Tasks.Extensions: 4.5.4** - Biblioteca que incluye tipos adicionales para el trabajo con tareas asincrónicas, que son objetos que representan operaciones concurrentes o paralelas que pueden devolver un resultado o generar una excepción. Entre estos tipos se encuentran ValueTask y ValueTask, que son estructuras que pueden evitar asignaciones innecesarias de memoria cuando se usan tareas asincrónicas, y que pueden ser convertidas a Task y Task si es necesario.

- **System.ValueTuple: 4.5.0** - Biblioteca que introduce el tipo ValueTuple, que es una estructura que permite crear tuplas, que son colecciones de valores con un número fijo y posiblemente heterogéneo de elementos. Las tuplas pueden ser usadas para agrupar datos sin necesidad de definir una clase o una estructura específica, y pueden ser desempaquetadas mediante la asignación de múltiples variables. El tipo ValueTuple también permite usar la sintaxis de tuplas de C# 7.0 y posteriores, que facilita la creación, el retorno y la descomposición de tuplas.
- **System.Data.SqlClient: 4.8.5** - Biblioteca que permite acceder y manipular datos almacenados en una base de datos SQL Server, utilizando el proveedor de datos .NET Framework para SQL Server. Esta biblioteca ofrece una interfaz de alto nivel para ejecutar comandos, realizar consultas, gestionar transacciones y trabajar con objetos de datos, como SqlDataReader, DataTable, DataSet y SqlDataAdapter. Además, esta biblioteca soporta las características específicas de SQL Server, como los tipos de datos especiales, las notificaciones de consulta, los procedimientos almacenados y las funciones definidas por el usuario.

2.3 Modelo de Datos

La solución de Terminales y Servidor de Terminales, no persisten datos de forma directa, sino que envían sus operaciones al servicio de BACKOFFICE Operacional, por tal motivo no se incluye en la presente documentación un modelo de datos del negocio.

No obstante, el servicio de Bitácora se incluye en la presente documentación con su correspondiente persistencia en la tabla de base de datos que se detalla a continuación.

Bitacora (dbo)			
	Column Name	Data Type	Allow Nulls
?	idBitacora	int	<input type="checkbox"/>
	dateTime	datetime	<input type="checkbox"/>
	operacion	nvarchar(50)	<input type="checkbox"/>
	componente	nvarchar(50)	<input type="checkbox"/>
	casino	int	<input checked="" type="checkbox"/>
	descripcion	nvarchar(MAX)	<input checked="" type="checkbox"/>
	resultado	bit	<input type="checkbox"/>
			<input type="checkbox"/>

2.4 Seguridad

La seguridad en el sistema se enfoca en garantizar la integridad y confidencialidad de la información, así como en proteger las comunicaciones entre los diferentes componentes. Considerando que las terminales son aplicaciones de escritorio basadas en WPF con C# y se conectan a un servidor mediante sockets TCP/IP, y el servidor de webhook es una API Rest en C# en una aplicación de consola, se implementan diversas medidas de seguridad.

Las comunicaciones entre las terminales y el servidor se realizan a través de sockets TCP/IP. Para garantizar la seguridad de estas comunicaciones, se implementa un cifrado de extremo a extremo. Se utiliza el protocolo TLS/SSL para establecer conexiones seguras, asegurando que la información transmitida entre las aplicaciones esté protegida contra posibles amenazas, como la interceptación no autorizada.

3 Descripción del Sistema

3.1 Visión general del sistema

La aplicación de tragamonedas es un sistema diseñado para simular la experiencia de juego de una máquina tragamonedas en un entorno de casino. El sistema consta de un frontend desarrollado en WPF (Windows Presentation Foundation) y se comunica con un backend a través de llamadas HTTP para realizar diversas operaciones, como iniciar sesión, realizar apuestas, y gestionar las ganancias.

El backend es un aplicación de consola que hace de servidor que recibe conexiones por sockets y posee también un webhook que recibe peticiones en formato API Rest para operaciones que deben ser procesadas de forma inmediata.

3.2 Funcionalidades principales

- **Inicio de Sesión:** Permite al casino ingresar el número de terminal e identificador de casino para acceder a la máquina tragamonedas.
- **Agregar Dinero:** Los usuarios pueden ingresar billetes en la terminal que posteriormente se convertirán en apuestas en cada jugada.
- **Realizar Apuestas:** Los usuarios pueden seleccionar la cantidad de créditos a apostar y jugar en la máquina tragamonedas.
- **Retirar Ganancias:** Ofrece la opción de retirar las ganancias acumuladas durante el juego.
- **Actualización Automática:** La aplicación actualiza automáticamente el estado de la terminal, incluyendo los límites de apuesta que son leídos del servidor de terminales de cada casino. Además, se incluye un webhook que atiende peticiones urgentes para procesar bajas de terminales o actualizaciones de rango de apuestas y premios.
- **Reproducción de Audio:** Integra un ambiente sonoro de casino para mejorar la experiencia del usuario durante el juego.

3.3 Beneficios y ventajas del sistema

- **Interfaz Gráfica Atractiva:** La aplicación proporciona una interfaz de usuario visualmente atractiva y fácil de usar.
- **Seguridad:** Utiliza un sistema de inicio de sesión para garantizar que solo terminales de casinos con licencias autorizadas puedan acceder a la máquina tragamonedas.
- **Experiencia de Juego Realista:** Incorpora elementos audiovisuales que emulan la atmósfera de un casino real.

4 Limitaciones de la Solución

El Sistema de Terminales y Servidor de Terminales de Casino tiene un alcance definido que abarca la sesión de juego, apuesta y retiro de dinero de la terminal. Sin embargo, existen restricciones técnicas y consideraciones de alcance que deben tenerse en cuenta. Además, se han identificado algunas funcionalidades futuras que podrían ampliar el sistema y mejorar su capacidad de adaptación y funcionalidad.

4.1 Restricciones, Suposiciones y limitaciones técnicas

El Sistema está sujeto a ciertas restricciones y limitaciones técnicas que se detallan a continuación:

- **Plataforma de implementación:** El módulo de terminales esta desarrollado en Windows Presentation Foundation (WPF), por lo tanto se requiere que el entorno de ejecución este basado en plataforma Windows.
- **Tecnologías y lenguajes utilizados:** El sistema utiliza tecnologías y lenguajes de programación modernos y ampliamente utilizados, como C# para el desarrollo del lado del cliente y del servidor. La persistencia de datos se logra mediante una base de datos relacional compatible basada en Azure SQL Database.
- **Entorno de Redes:** Si bien cada casino tiene su propio servidor de terminales y su propio webhook, lo cual garantiza que la terminal tenga una conexión estable, se requiere una conexión de topología WAN (Internet u otras) para establecer la conectividad entre el servidor de terminales / webhook y el servidor de Backoffice Operacional. La solución requiere que los casinos tengan conexión permanente con el servidor de backoffice operacional.
- **Escalabilidad:** Aunque el sistema está diseñado para ser escalable, se debe tener en cuenta que hay ciertos límites y consideraciones en términos de capacidad de manejo de terminales y operaciones. A medida que la cantidad de terminales y operaciones aumente significativamente, puede ser necesario realizar ajustes y optimizaciones en la infraestructura y el diseño del sistema para garantizar un rendimiento óptimo. Estos ajustes pueden incluir tanto planes de mantenimiento a nivel del motor de base de datos como a nivel de las capas de código desarrolladas.
- **Seguridad:** El sistema implementa medidas de seguridad básicas para proteger la autenticación de terminales y casinos. Sin embargo, se recomienda complementar estas medidas con prácticas adicionales de seguridad, como el uso de HTTPS para cifrar las comunicaciones, la implementación de políticas de identificadores cifrados y la protección contra ataques de fuerza bruta. Es recomendable la implementación de una capa de infraestructura de redes que contemple firewalls de nivel 3 y de nivel 7 a fin de lograr una arquitectura más robusta en términos de seguridad perimetral y de backend.
- **Cumplimiento normativo:** El sistema debe tener en cuenta las restricciones y requisitos de seguridad de datos y cumplimiento normativo aplicables. Esto puede incluir medidas como el cifrado de datos confidenciales, la protección contra ataques de seguridad y el cumplimiento de regulaciones como el Reglamento General de Protección de Datos (GDPR) en el caso de datos personales.
- **Mantenimiento y actualizaciones:** El sistema requerirá un mantenimiento regular para asegurar su correcto funcionamiento y aplicar actualizaciones de seguridad y mejoras. Esto implica la necesidad de realizar copias de seguridad periódicas,

monitorear el rendimiento del sistema y aplicar parches y actualizaciones según sea necesario.

- **Integraciones con sistemas existentes:** Si el sistema necesita integrarse con sistemas existentes, como sistemas de gestión de clientes (CRM) o sistemas de inventario, es importante considerar las limitaciones y requisitos técnicos de estas integraciones. Pueden existir limitaciones de compatibilidad o restricciones en cuanto a la disponibilidad y accesibilidad de los datos en otros sistemas, lo que puede requerir un análisis detallado y la implementación de soluciones de integración adecuadas.
- **Disponibilidad y tolerancia a fallos:** El sistema debe tener en cuenta la disponibilidad y la tolerancia a fallos para garantizar que siga funcionando incluso en caso de interrupciones o problemas técnicos. Esto puede implicar la implementación de medidas de redundancia, como la duplicación de servidores o el uso de sistemas de respaldo, así como la planificación y realización de pruebas periódicas de recuperación ante desastres.

4.2 Funcionalidades futuras no contempladas

Aunque el Sistema cubre las funcionalidades principales relacionadas con las terminales y servidor de terminales del casino, existen algunas funcionalidades adicionales que no están contempladas en el alcance actual del sistema. Estas funcionalidades pueden considerarse en futuras versiones o como posibles expansiones del sistema. Algunas de estas funcionalidades futuras identificadas son:

- **Juegos Avanzados:** La solución actual no incluye funcionalidades de juegos avanzados o juegos secundarios con diferentes niveles de bonificaciones. Se podrá analizar en futuras versiones.
- **Integración con sistemas externos:** Si el sistema se utiliza en un entorno que requiere integración con sistemas externos, como sistemas de pago o sistemas de inventario, se podría considerar la implementación de interfaces o API para facilitar la comunicación y la sincronización de datos entre el Sistema de Casinos y otros sistemas.
- **Funcionalidades de importación/exportación de datos:** Si es necesario migrar datos desde o hacia el sistema, se podría considerar la inclusión de funcionalidades de importación y exportación de datos para facilitar este proceso. Esto permitiría cargar datos en el sistema desde fuentes externas o exportar datos del sistema en formatos estándar para su uso en otros sistemas o fines de respaldo.
- **Autenticación y Autenticidad de peticiones API:** Las diferentes peticiones GET/POST hechas a las API del sistema no contemplan el uso de tecnologías que garanticen autenticidad y autenticación de las peticiones. En un futuro se podría implementar una solución de token al estilo JWT (JSON Web Token) que define una forma compacta y autónoma de transmitir información segura entre dos partes.
- **Auto-Registración de Terminales:** El sistema no cuenta con la capacidad de que los casinos autogestionen la registración de terminales, deberán ser dados de alta por un administrador. Se contemplará esta funcionalidad como una mejora futura a ser desarrollada.
- **Bloqueo de terminales ante intentos fallidos:** El sistema no cuenta con bloqueo automático de terminales ante intentos fallidos de ingreso. Se contemplará esta funcionalidad como una mejora futura a ser desarrollada.

5 Requerimientos

5.1 Requerimientos Funcionales

5.1.1 Inicio de Sesión

- El sistema debe permitir a los administradores de cada casino iniciar sesión ingresando el número de terminal y el identificador del casino.
- Se deben validar los campos de entrada y mostrar mensajes de error adecuados en caso de datos incorrectos o faltantes.
- Después del ingreso exitoso, la terminal debe iniciarse y ofrecer las opciones de carga de dinero para jugar.

5.1.2 Agregar Dinero

- Los jugadores deben poder agregar el dinero deseado a la terminal.
- El dinero agregado se acreditará como dinero disponible para jugar.

5.1.3 Realizar Apuestas

- Los jugadores deben poder seleccionar la cantidad de dinero acreditado que deseen apostar en función de los límites mínimos y máximos permitidos para el casino en cuestión.

5.1.4 Realizar Jugada

- Los jugadores deben poder realizar la jugada con la apuesta deseada.
- En caso de ganar, se les ofrecerá retirar el dinero ganado o bien continuar jugando.
- En caso de perder, se les descontará el dinero apostado del crédito disponible.

5.1.5 Retirar Dinero / Ganancias

- Los jugadores deberán poder retirar sus ganancias acumuladas como así también los créditos pendientes no utilizados.

5.2 Requerimientos No Funcionales

5.2.1 Seguridad de acceso

- El sistema debe implementar medidas de seguridad para garantizar el acceso seguro y proteger la autenticación de las terminales.

5.2.2 Política de Identificador de Terminales y Casinos

- Se define como estándar para el ID de Casino y el ID de Terminales, un numero entero válido.

5.2.3 Interfaz Gráfica Atractiva

- La interfaz de jugadores debe ser intuitiva y fácil de usar.
- Se deben proporcionar instrucciones claras y elementos de navegación para guiar a los usuarios a través del sistema.
- La interfaz de usuario debe ser atractiva visualmente y seguir las mejores prácticas de diseño de experiencia de usuario (UX).
- La interfaz debe proporcionar una experiencia de juego realista mediante la reproducción de audio y elementos visuales.

5.2.4 Mantenibilidad y escalabilidad

- El código del sistema debe estar bien estructurado, modularizado y seguir las mejores prácticas del desarrollo.
- Se deben implementar comentarios y documentación adecuada para facilitar el mantenimiento y la comprensión del código.
- El sistema debe estar diseñado para ser escalable y capaz de adaptarse a futuras necesidades de crecimiento y expansión.

5.2.5 Conexión a Internet

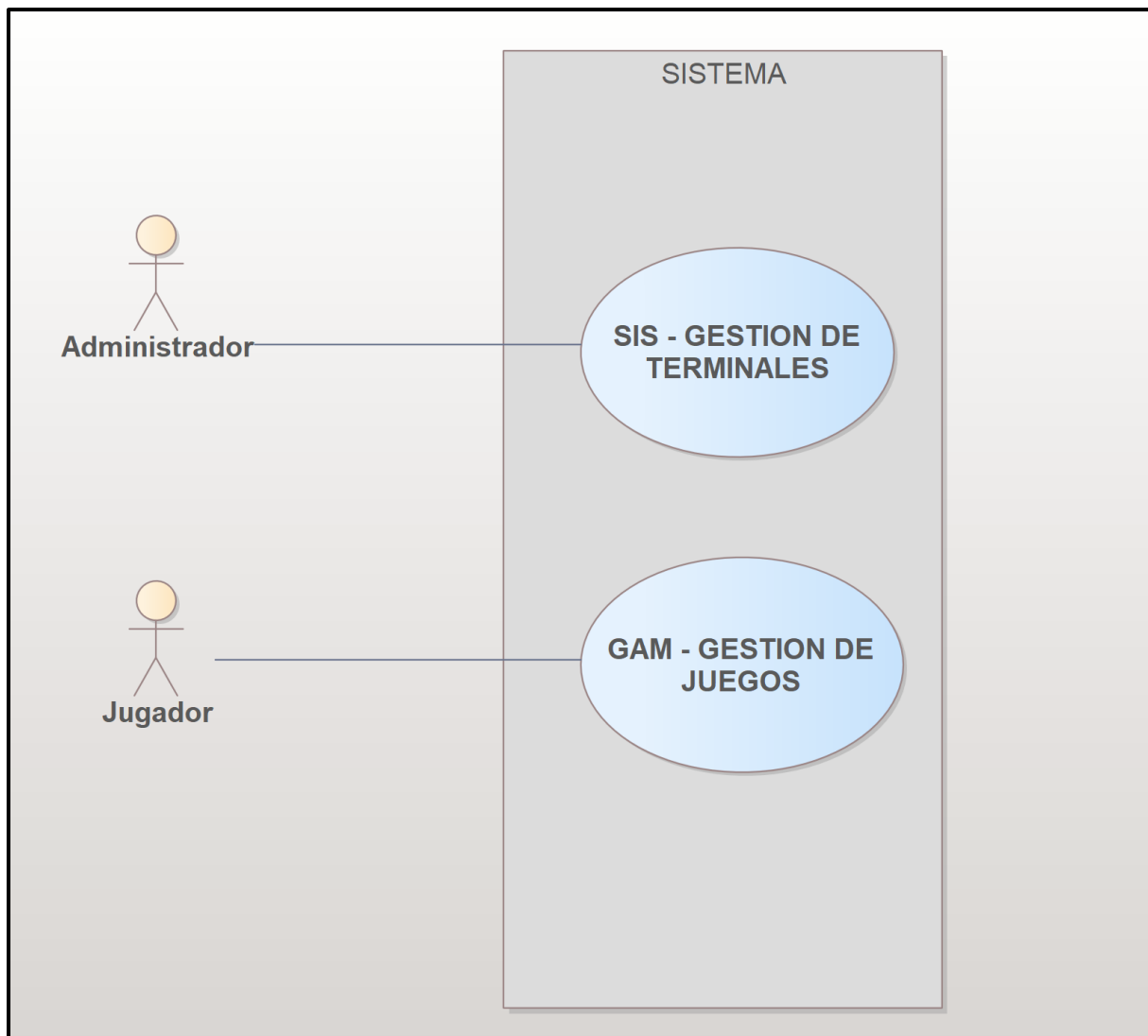
- Los casinos requieren una conexión estable a internet para transaccionar con el servidor de backoffice operacional que se encuentra en un punto central de la topología de la solución. Eventualmente podrá establecerse algún tipo de red privada segura, pero el acceso siempre será por medio de una WAN, lo que supone un nivel de servicio requerido de cierta estabilidad con los diferentes proveedores.

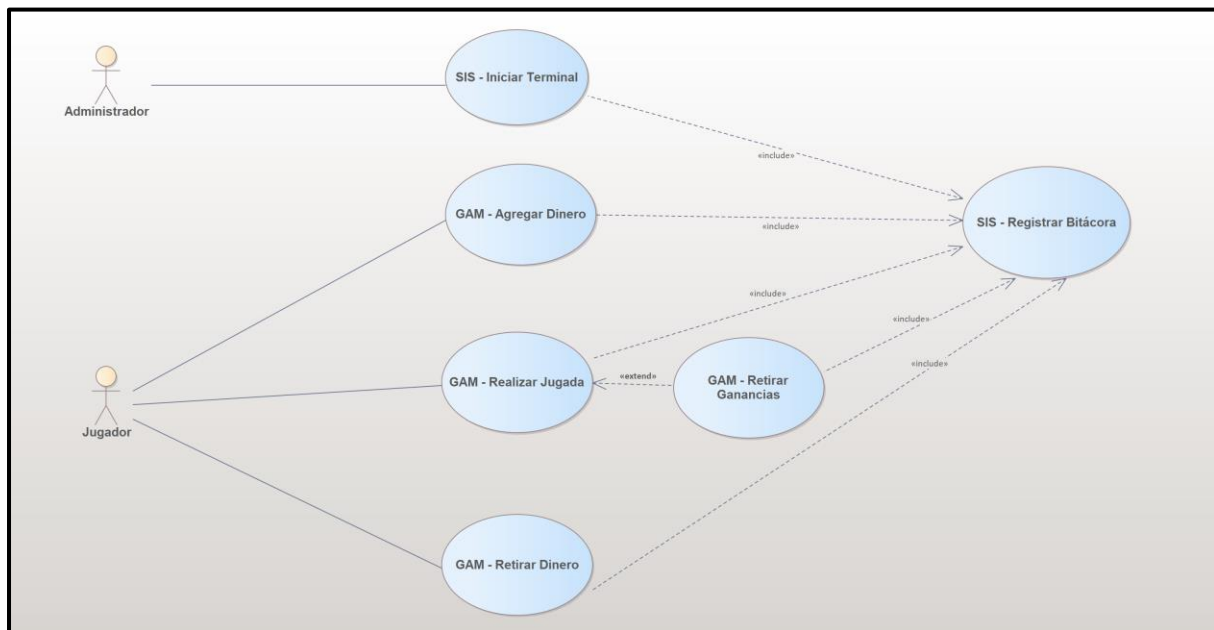
6 Casos de Uso

Los casos de uso contemplan dos (2) tipos de actores que a su vez son roles de usuarios que pueden acceder a determinadas funcionalidades (casos de uso) del sistema.

1. **Administrador:** Accede a todas las funcionalidades del sistema a nivel de Casino que permite iniciar las terminales de juegos.
2. **Jugador:** Accede a las terminales de juegos ya iniciadas listas para jugar.

6.1 Diagramas de Casos de Uso





6.2 Especificaciones de Casos de Uso

6.2.1 SIS – Iniciar Terminal

Objetivo: Iniciar una terminal de tragamonedas.

Precondiciones:

- El servidor de casino este operativo y esperando conexiones.
- El sistema está en un estado operativo.
- El usuario tiene permisos para iniciar la terminal con el correspondiente identificador de casino y de terminal.

Flujo principal:

1. El usuario inicia la aplicación de tragamonedas.
2. Se presenta la pantalla de inicio de sesión.
3. El usuario ingresa el número de terminal y el número de casino.
4. El sistema valida la información ingresada con el servidor de casino.
5. Si la validación es exitosa, se abre la ventana principal de la terminal.
6. Se inicia la sesión de juego.
7. Se ejecuta el caso de uso [SIS – Registrar Bitácora](#).

Postcondición: La terminal se inicia y el usuario puede realizar jugadas.

Flujo Alternativo:

4.1 Si el número de casino y/o terminal no se valida correctamente por el servidor, se muestra un mensaje de error y se devuelve al usuario a la pantalla de inicio de sesión nuevamente.

Puntos de Extensión: Este caso de uso incluye a [SIS – Registrar Bitácora.](#)

6.2.2 GAM - Agregar Dinero

Objetivo: Permitir a los jugadores agregar créditos a la terminal.

Precondiciones:

- La terminal está en un estado operativo autorizada por el servidor de casino.

Flujo principal:

1. El usuario selecciona el valor del billete que desea agregar y presiona la opción “Agregar Dinero” tantas veces como lo desee.
2. El sistema verifica que la cantidad sea válida.
3. Se registra la transacción y se acumula el dinero como disponible para realizar apuestas.
4. Se actualiza el saldo de la terminal.
5. Se ejecuta el caso de uso [SIS – Registrar Bitácora.](#)

Postcondición: El saldo de la terminal se incrementa según la cantidad ingresada.

Flujo Alternativo:

2.1 Si la cantidad no es válida, se retorna a la pantalla de la terminal para seleccionar el valor del billete correcto.

Puntos de Extensión: Este caso de uso incluye a [SIS – Registrar Bitácora.](#)

6.2.3 GAM - Realizar Jugada

Objetivo: Permitir a los jugadores realizar una jugada en la máquina tragamonedas.

Precondiciones:

- La terminal está en un estado operativo con la sesión iniciada.
- El jugador tiene créditos suficientes.

Flujo principal:

1. El jugador selecciona la apuesta a realizar.
2. El jugador presiona el botón principal para realizar la jugada.
3. El sistema verifica el resultado de la jugada.
4. En caso de que el resultado de la jugada sea ganadora, el sistema ofrece al usuario la posibilidad de retirar el dinero de la ganancia y se ejecuta el caso de uso **GAM – Retirar Ganancias**.
5. Se actualiza el saldo de la terminal según el resultado.
6. Se ejecuta el caso de uso [SIS – Registrar Bitácora](#).

Postcondición: El saldo de la terminal se ajusta según el resultado de la jugada.

Flujo Alternativo:

2.2.1 Si el usuario no tiene créditos suficientes, el sistema no ejecutará la jugada.

2.2.2 Si la apuesta seleccionada esta fuera del rango mínimo y máximo permitido por el casino, el sistema mostrará un error y el jugador deberá corregir el monto de la apuesta a realizar.

Puntos de Extensión: Este caso de uso incluye a [SIS – Registrar Bitácora](#). Este caso de uso es extendido por [GAM – Retirar Ganancias](#).

6.2.4 GAM – Retirar Dinero

Objetivo: Permitir a los jugadores retirar dinero de la terminal generando una orden de pago.

Precondiciones:

- La terminal está en un estado operativo con la sesión iniciada.
- El jugador tiene dinero disponible para retirar.

Flujo principal:

1. El jugador selecciona la opción de retirar dinero.
2. Se verifica que el monto a retirar sea válido.
3. Se registra la transacción en el sistema.
4. Se actualiza el saldo de la terminal.
5. Se ejecuta el caso de uso [SIS – Registrar Bitácora](#).

Postcondición: El saldo de la terminal se reestablece a \$0.

Flujo Alternativo:

2.2 Si el monto a retirar no es válido, se muestra un mensaje de error.

Puntos de Extensión: Este caso de uso incluye a [SIS – Registrar Bitácora](#).

6.2.5 GAM – Retirar Ganancias

Objetivo: Permitir a los jugadores retirar las ganancias obtenidas después de una jugada exitosa.

Precondiciones:

- La terminal está en un estado operativo con la sesión iniciada.
- La jugada resultó ganadora.

Flujo principal:

1. Ante el resultado de la jugada ganadora, el jugador confirma que desea retirar la ganancia generada.
2. Se verifica que las ganancias a retirar sean válidas.
3. Se registra la transacción en el sistema.
4. Se actualiza el saldo de la terminal.
5. Se ejecuta el caso de uso [SIS – Registrar Bitácora](#).

Postcondición: El saldo de la terminal se actualiza según las ganancias retiradas o acumuladas.

Flujo Alternativo:

2.1 Si el jugador no confirma, el sistema acumula la ganancia como dinero disponible para continuar jugando.

2.2 Si las ganancias a retirar no son válidas, se muestra un mensaje de error.

Puntos de Extensión: Este caso de uso incluye a [SIS – Registrar Bitácora](#). Este caso de uso extiende al caso de uso [GAM – Realizar Jugada](#).

6.2.6 SIS – Registrar Bitácora

Objetivo: Registrar eventos importantes en la bitácora del sistema para mantener un historial de operaciones relevantes.

Precondiciones:

- La terminal está en un estado operativo.

Flujo principal:

1. El sistema genera un evento significativo que requiere ser registrado en la bitácora.
2. La terminal por medio del servidor de casino envía una solicitud al servicio de Bitácora para registrar el evento.
3. El servicio de Bitácora recibe la solicitud y extrae la información necesaria del evento, como la operación realizada, el componente afectado, el casino asociado, la descripción de la operación y el resultado obtenido.
4. Se establece una conexión con la base de datos utilizando la cadena de conexión proporcionada en la configuración.
5. Se crea la consulta SQL de inserción con los parámetros necesarios para el registro en la tabla "Bitacora".
6. Se ejecuta la consulta SQL utilizando un comando SQL en una transacción abierta.
7. Se verifica si la inserción fue exitosa mediante el número de filas afectadas.
8. Si la inserción es exitosa, se cierra la transacción y se devuelve una respuesta positiva al cliente.
9. Si la inserción falla, se revierte la transacción y se devuelve una respuesta de error al cliente.

Postcondición: El evento queda registrado en la bitácora del sistema para su posterior consulta y seguimiento.

Flujo Alternativo:

4.1 Si la conexión con la base de datos no se puede establecer, se devuelve una respuesta de error al cliente.

6.1 Si la consulta SQL no se ejecuta correctamente, se revierte la transacción y se devuelve una respuesta de error al cliente.

Puntos de Extensión: Este caso de uso es incluido por todos los casos de uso funcionales detallados en el presente documento.

7 Manual del Sistema

7.1 Instalación y Configuración del Sistema

La aplicación de Terminal funciona en forma local en el dispositivo de tragamonedas basado en Windows, conectado al servidor de Casino que está basado en tecnología Windows Server.

El servidor de casino se conecta a un servicio transversal de bitácora que, además, es utilizado por los restantes componentes de la solución, no cubiertos en el presente documento (Backoffice Operacional, Microservicios, Backend, Frontend, etc.)

A fin de hacer extensible el servicio de bitácora a los demás componentes de la solución, el mismo se despliega en un servicio de cloud Computing en formato de API App Services sobre Microsoft Azure.

7.1.1 Configuración Servidor de Casino y Webhook

Para el despliegue local de ambos servicios, será necesario realizar los siguientes pasos:

- 1- Abrir la solución **/Servidor-Webhook/Server-webhook.sln**

Esta solución ejecuta tanto el servidor de Casino como el servidor webhook.

El servidor de casino mantiene su configuración de puerto donde está activo el socket de escucha esperando la conexión de las terminales tragamonedas de forma predeterminada en el puerto TCP 1111. Se puede realizar el cambio de esta configuración editando el archivo ubicado en el proyecto **ConsoleApp1/Program.cs**, modificando el parámetro del número de puerto por el deseado:

```
//Variables de configuración del endpoint local que utilizaremos en el socket.  
// Dns.GetHostName retorna el nombre del host que está ejecutando la aplicación server  
IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());  
IPAddress ipAddr = ipHost.AddressList[0];  
IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 1111);
```

- 2- El servidor de Casino mantiene la configuración del ID de casino al que pertenece, el mismo se configura en el archivo **appsettings.json** ubicado en el proyecto ConsoleApp1 de la solución "Server-webhook":

```
{  
  "Casino": {  
    "id": "2"  
  }  
}
```

- 3- El servidor de Casino mantiene la configuración de referencia hacia los servicios de Backoffice y de Bitácora, los cuales son configurados en el archivo **appsettings.json** ubicado en el proyecto ConsoleApp1 de la solución "Server-webhook":

```
{
  "ConnectionString": {
    "backofficeAzure": "https://mcgabackoffice.azurewebsites.net/",
    "backofficeLocal": "http://localhost:5214/",
    "bitacora": "https://mcgabitaacora.azurewebsites.net/api/Bitacora"
  }
}
```

De ser necesario, podrán ser modificadas las referencias hacia las URLs correspondientes. De forma predeterminada, la solución utilizada “backofficeAzure” en el desarrollo, pero si eventualmente se requiere el uso de un backoffice alojado onpremises, puede utilizarse la variable “backofficeLocal” haciendo referencia al servidor y puerto que corresponda. No obstante cabe aclarar que deberá cambiarse en **Program.cs** del Proyecto **ConsoleApp1** de la solución “**Server-webhook**” la siguiente referencia:

```
private static void loadURLs()
{
    var builder = new ConfigurationBuilder();
    builder.SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);

    IConfiguration config = builder.Build();

    //config["ConnectionString:backofficeLocal"];
    cas = new Casino(Int32.Parse(config["Casino:id"]),
        config["ConnectionString:backofficeAzure"],
        config["ConnectionString:bitacora"]);
}
```

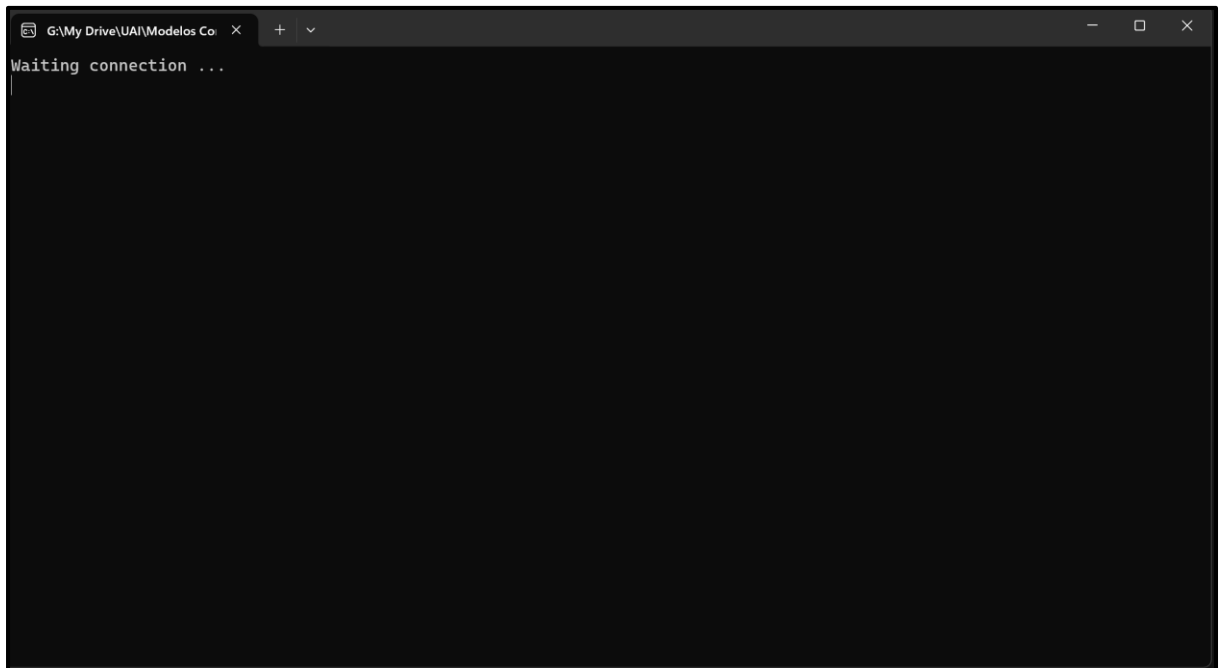
De igual manera, se podrá hacer el cambio a la referencia de los servicios de bitácora.

- 4- El webhook mantiene su configuración escuchando de forma predeterminada en el puerto TCP 44329, y podrá ser cambiada su configuración modificando el archivo launchsettings.json de las propiedades del proyecto webhook de la solución “Server-webhook”:

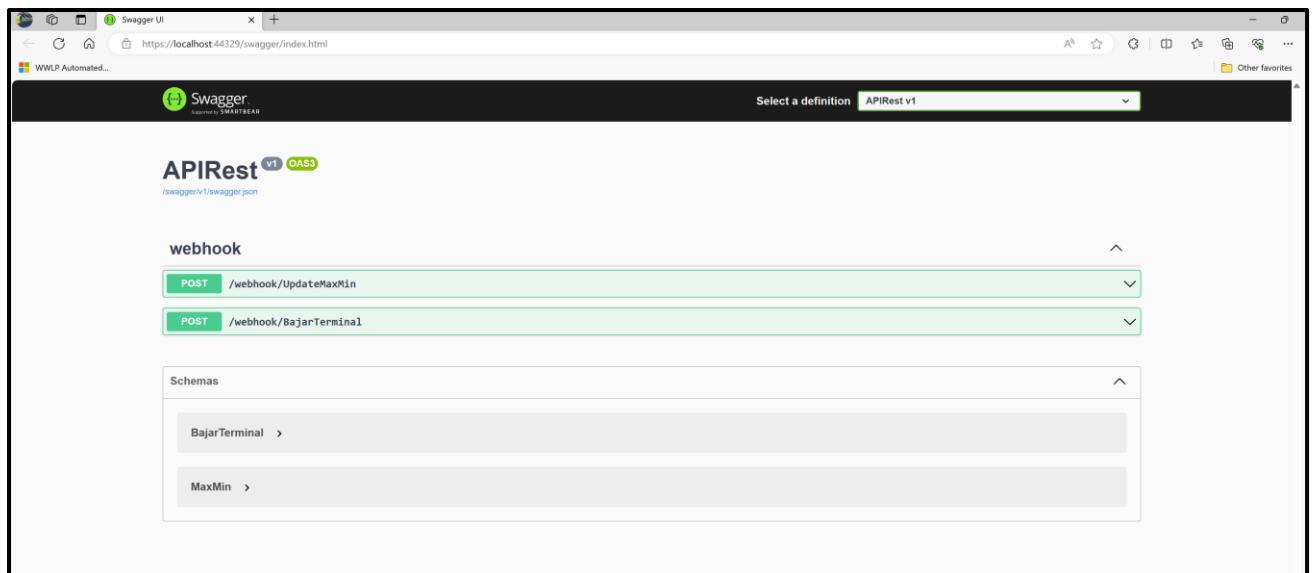
```
//Cada casino deberia tener un puerto distinto
"sslPort": 44329
```

- 5- Una vez realizada estas configuraciones en el servidor de casino y el webhook, se podrá iniciar la solución y el servidor iniciará tanto la escucha de terminales por sockets como el webhook para operaciones urgentes:

Servidor de Casino Sockets Esperando Terminales:



Webhook con sus endpoints correspondientes:



7.1.2 Configuración Terminales Tragamonedas

Para configurar una terminal de tragamonedas, realizar los siguientes pasos desde la terminal:

1- Abrir la solución **/Tragamonedas/Tragamonedas.sln**

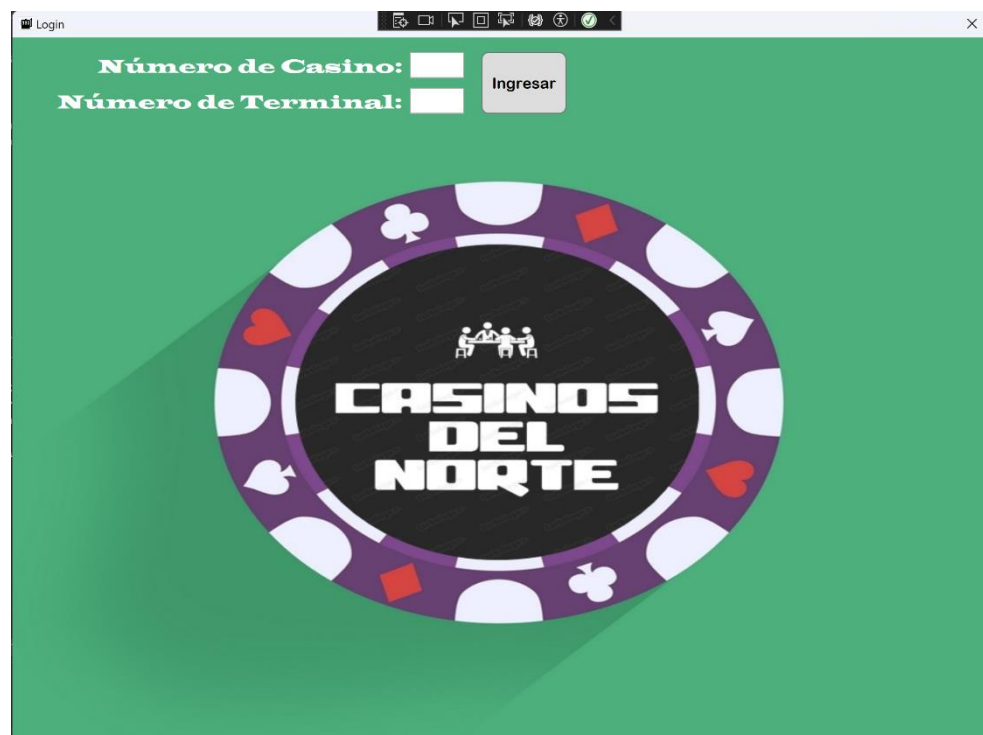
Esta solución ejecuta el software de Terminal de Tragamonedas del casino.

La terminal se encuentra configurada de forma predeterminada para conectarse a un servidor de casino que sea ejecutado en el mismo equipo ("localhost" o 127.0.0.1) en el puerto remoto TCP 1111. Se puede realizar el cambio de esta configuración editando el archivo ubicado en el proyecto **Tragamonedas/Helpers/callers.cs**, modificando el parámetro de IPAddress y número de puerto por el deseado.

Tanto la dirección IP como el numero de puerto deberán corresponder al servidor configurado en el paso 1 del apartado anterior [Configuración Servidor de Casino y Webhook](#).

```
//Variables de configuración del endpoint remoto que utilizaremos en el socket.  
//Este ejemplo usa el puerto 1111 en la computadora local.  
IPHostEntry ipHost = Dns.GetHostEntry(Dns.GetHostName());  
//IPAddress ipAddr = IPAddress.Parse("172.16.1.200"); //Servidor de Casino se ejecuta en host remoto  
IPAddress ipAddr = ipHost.AddressList[0]; //Servidor de Casino Se ejecuta en localhost  
IPEndPoint localEndPoint = new IPEndPoint(ipAddr, 1111); //Puerto
```

2- Una vez realizada estas configuraciones en la terminal de casino, se podrá iniciar la solución y la terminal iniciará solicitando el número de casino y el número de terminal:



7.1.3 Configuración Servicio de Bitácora

Para el despliegue del servicio de Bitácora en Microsoft Azure, se deberá contar con una suscripción con créditos suficientes, en función de la carga y componentes de alta disponibilidad que se desee.

Para crear una suscripción en Microsoft Azure: <https://azure.microsoft.com/en-us/free/>

- 1- Desde la consola de Microsoft Azure, crear un Servidor de Base de Datos para alojar la base de datos Azure SQL Database, utilizando los siguientes criterios:

The screenshot shows the 'Create SQL Database Server' page in the Microsoft Azure portal. The breadcrumb navigation at the top reads 'Dashboard > Create SQL Database >'. The main heading is 'Create SQL Database Server' with a three-dot menu icon to its right. Below the heading is the Microsoft logo. The 'Server details' section contains instructions: 'Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.' It features two input fields: 'Server name *' with the value 'mcga' and a '.database.windows.net' suffix, and 'Location *' with a dropdown menu set to '(US) East US 2'. The 'Authentication' section provides instructions on selecting authentication methods. It includes three radio button options: 'Use only Azure Active Directory (Azure AD) authentication', 'Use both SQL and Azure AD authentication', and 'Use SQL authentication', which is currently selected. Below these are three more input fields: 'Server admin login *' with the value 'adminmcga', 'Password *' with masked characters and a green checkmark, and 'Confirm password *' also with masked characters and a green checkmark.

- 2- Desde la consola de Microsoft Azure, crear una base de datos que se aloje en el servidor creado en el paso anterior:

Dashboard >

Create SQL Database

Microsoft

BasicsNetworkingSecurityAdditional settingsTagsReview + create

Product details

SQL database
by Microsoft
[Terms of use](#) | [Privacy policy](#)

Estimated cost
Storage cost 23.75 ARS / month + Compute cost 0.023025 ARS / vCore second

Terms

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. For additional details see [Azure Marketplace Terms](#).

Basics

Subscription	DanielG Visual Studio Enterprise Subscription
Resource group	UAI-MCGA
Region	East US 2
Database name	CASINOS
Server	(new) mcga
Authentication method	SQL authentication
Server admin login	adminmcga
Compute + storage	General Purpose - Serverless: Standard-series (Gen5), 1 vCore, 1 GB storage, zone redundant disabled
Backup storage redundancy	Locally-redundant backup storage

Networking

Allow Azure services and resources to access this server	Yes
Add current client IP address 181.46.71.133	Yes
Private endpoint	None
Minimum TLS version	1.2
Connection Policy	Default


Security

Identity	Not enabled
Transparent data encryption	Service-managed key selected
Advanced data security	Not now
Sql Ledger(Database)	Disabled
Digest Storage	Disabled

Additional settings

Use existing data	Blank
Collation	SQL_Latin1_General_CP1_CI_AI
Maintenance window	System default (5pm to 8am)

Tags



Cost summary

General Purpose (GP_S_Gen5_1)	
Cost per GB (in ARS)	18.27
Max storage selected (in GB)	x 1.3
ESTIMATED STORAGE COST / MONTH	23.75 ARS
COMPUTE COST / VCORE SECOND	0.023025 ARS

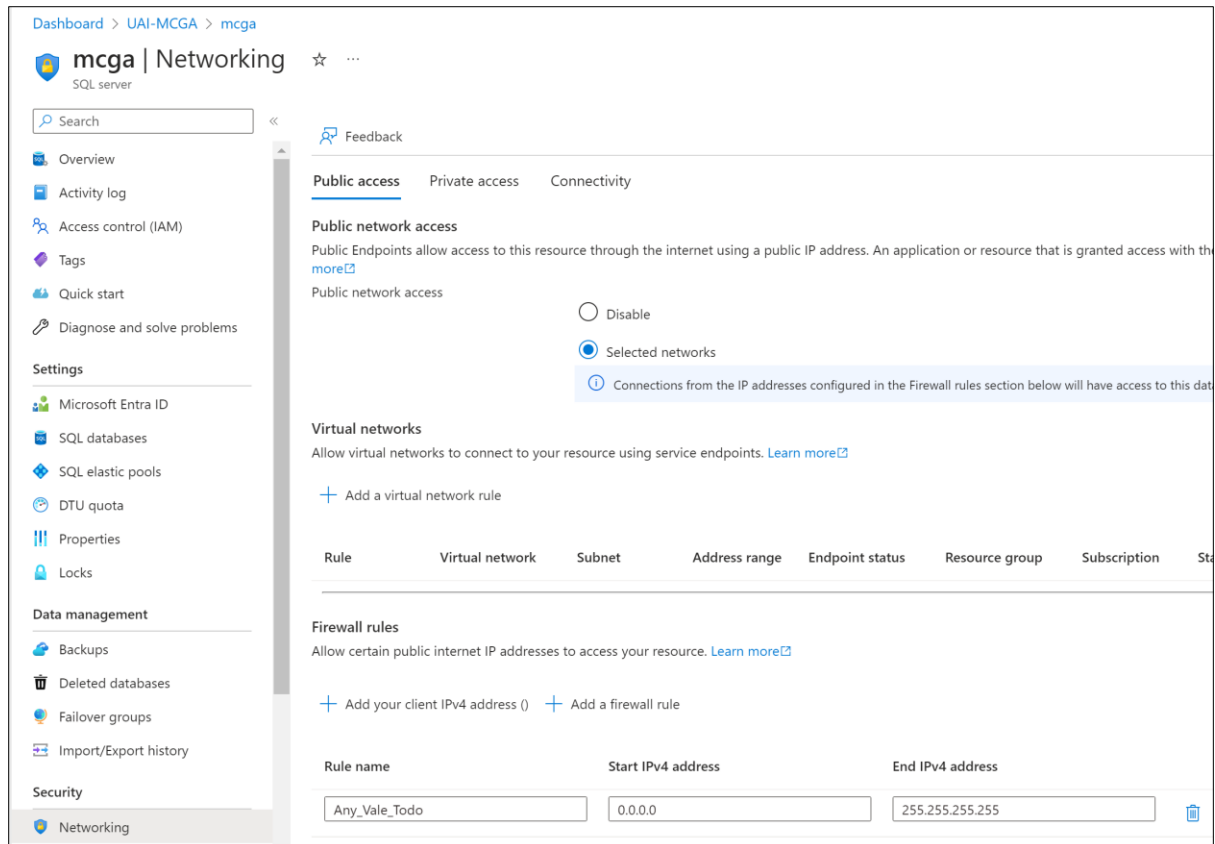
NOTES
¹ Serverless databases are billed in vCore seconds based on a combination of CPU and memory utilization. [Learn more about serverless billing](#)

Create

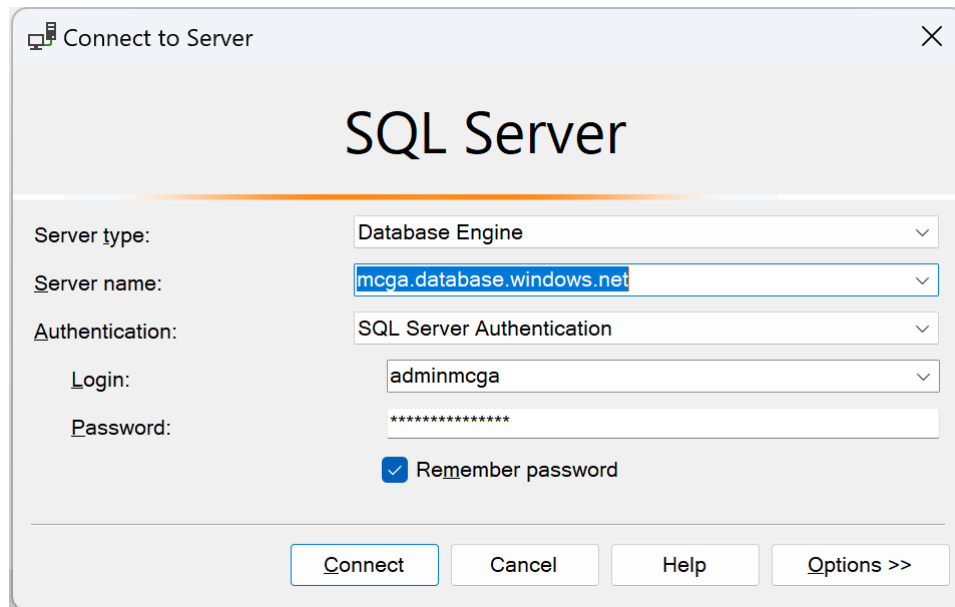
< Previous

[Download a template for automation](#)

- 3- Si se realizará la conexión a la base de datos desde otros sitios diferentes al que creo el servicio, es posible que se requiera agregar una regla de firewall a nivel de servicio de Azure SQL Database. No se recomienda realizar esto para entornos de producción, ya que el servidor quedará expuesto sin protección. Se detalla este paso con fines de pruebas y desarrollo de la solución. Para el ambiente de producción final, se recomienda utilizar Private links que garanticen la conectividad de forma interna y segura entre los diferentes componentes de Microsoft Azure.



- 4- Instalar la consola de SQL Server Management Studio desde: <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>
- 5- Desde la consola de SQL Server Management Studio, instalada en el paso anterior, conectarse a **mcga.database.windows.net**, con el usuario y contraseña configurados en el paso 1 donde se creó la base de datos en el servicio Azure SQL Database.



- 6- Una vez conectado al servidor de Base de Datos, desde la consola de SQL Server Management Studio, ejecutar el siguiente script de creación de la tabla BITACORA sobre la base de datos CASINOS:

https://github.com/AldoDanielGonzalez/MCGA-TP.Final/blob/master/scripts/Create_Bitacora_Table.sql

Esto creara la tabla necesaria para soportar la persistencia del servicio transversal de bitácora para fines de auditoría.

- 7- Crear un App Service API Apps para ejecutar la RESTful API del servicio de Bitacora. Utilizar los siguientes criterios en la consola de Microsoft Azure. Tener en cuenta que, para conectarse al repositorio de GitHub, será requisito clonar el repositorio en otro repositorio propio que este integrado con la suscripción de Azure. Este proceso de vinculación entre la cuenta GitHub y la suscripción de Azure, se realizará de forma automática cuando se le indique al siguiente formulario el nombre de organización y repositorio de la cuenta GitHub. En ese momento solicitara credenciales para realizar la vinculación. Esta configuración generará un workflow de Despliegue Continuo en GitHub Actions, que desplegará de forma automática la solución en la Azure Static Web App. El despliegue se realizará cada vez que se detecte un "Push" en la rama principal del repositorio GitHub que contenga el código de la solución de Bitacora. Para mas información de ubicación de repositorio y códigos fuentes ver el siguiente capítulo [7.2 Documentación y Fuentes del Proyecto.](#)

[Dashboard](#) > [Create a resource](#) > [Marketplace](#) > [API App](#) >

Create API App ...

[Basics](#) [Deployment](#) [Networking](#) [Monitoring](#) [Tags](#) [Review + create](#)

Microsoft Azure App Service API Apps offers secure and flexible development, deployment, and scaling options for any sized RESTful API application. Use frameworks and templates to create RESTful APIs in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your RESTful API with .NET, Java, PHP, Node.js or Python. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

DanielG Visual Studio Enterprise Subscription

Resource Group * ⓘ

UAI-MCGA

[Create new](#)

Instance Details

Name *

mcgabitacora

.azurewebsites.net

Publish *

☒ Code ☐ Docker Container ☐ Static Web App

Runtime stack *

.NET 7 (STS)

Operating System *

☐ Linux ☒ Windows

Region *

East US 2

i Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Windows Plan (East US 2) * ⓘ

mcga-AppService (F1)

[Create new](#)

Pricing plan

Free F1 (Shared infrastructure)

Zone redundancy

An App Service plan can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make an App Service plan zone redundant after it has been deployed [Learn more](#)

Zone redundancy

- ☐ **Enabled:** Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.
- ☒ **Disabled:** Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

[Dashboard](#) > [Create a resource](#) > [Marketplace](#) > [API App](#) >

Create API App ...

[Basics](#) [Deployment](#) [Networking](#) [Monitoring](#) [Tags](#) [Review + create](#)

Enable GitHub Actions to continuously deploy your app. GitHub Actions is an automation framework that can build, test, and deploy your app whenever a new commit is made in your repository. If your code is in GitHub, choose your repository here and we will add a workflow file to automatically deploy your app to App Service. If your code is not in GitHub, go to the Deployment Center once the web app is created to set up your deployment. [Learn more](#)

GitHub Actions settings

Continuous deployment ☐ Disable ☒ Enable

GitHub Actions details

Select your GitHub details, so Azure Web Apps can access your repository. You must have write access to your chosen repository to deploy with GitHub Actions.

GitHub account AldoDanielGonzalez [Change account](#) ⓘ

Organization * AldoDanielGonzalez ▼

Repository * MCGA-TP.Final ▼

Branch * master ▼

Workflow configuration

File with the GitHub Actions workflow configuration.

[Preview file](#)

[Review + create](#) [< Previous](#) [Next : Networking >](#)


Dashboard > Create a resource > Marketplace > API App >

Create API App ...

Basics Deployment **Networking** Monitoring Tags Review + create

Web Apps can be provisioned with the inbound address being public to the internet or isolated to an Azure virtual network. Web Apps can also be provisioned with outbound traffic able to reach endpoints in a virtual network, be governed by network security groups or affected by virtual network routes. By default, your app is open to the internet and cannot reach into a virtual network. These aspects can also be changed after the app is provisioned. [Learn more](#)

Enable public access * ☒ On ☐ Off

 Network injection is only available in Basic, Standard, Premium, Premium V2, and Premium V3 Dedicated App Service plans.

Enable network injection ☐ On ☒ Off

Dashboard > Create a resource > Marketplace > API App >

Create API App ...

Basics Deployment Networking **Monitoring** Tags Review + create

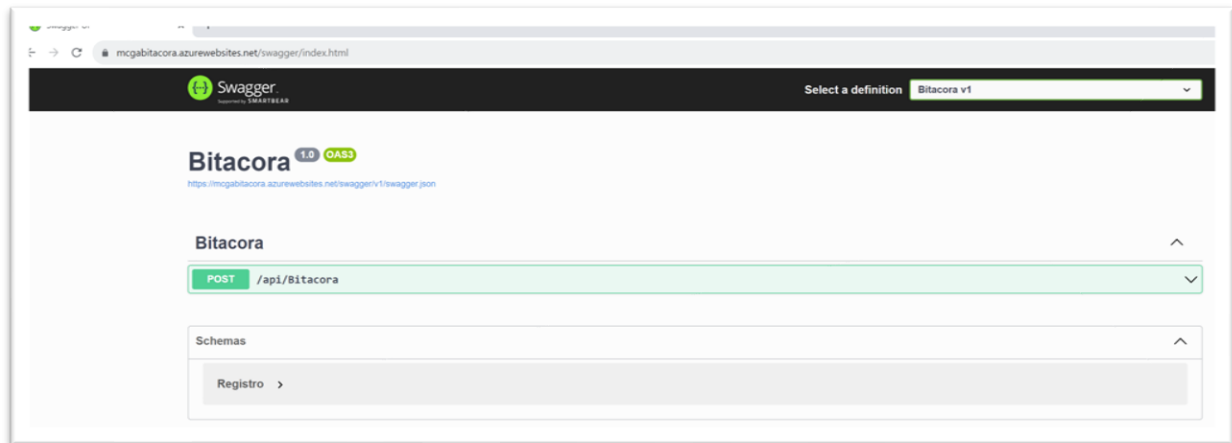
Azure Monitor application insights is an Application Performance Management (APM) service for developers and DevOps professionals. Enable it below to automatically monitor your application. It will detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. Your bill is based on amount of data used by Application Insights and your data retention settings. [Learn more](#)

[App Insights pricing](#)

Application Insights

Enable Application Insights * ☒ No ☐ Yes

- 8- Al finalizar la creación de la API App en Azure, se generará la URL pública bajo el nombre de <https://mcgabitacora.azurewebsites.net>.
- 9- Para probar que el despliegue ha sido satisfactorio, ingresar a la URL de Swagger de la API creada en el paso anterior:
<https://mcgabitacora.azurewebsites.net/swagger/index.html>



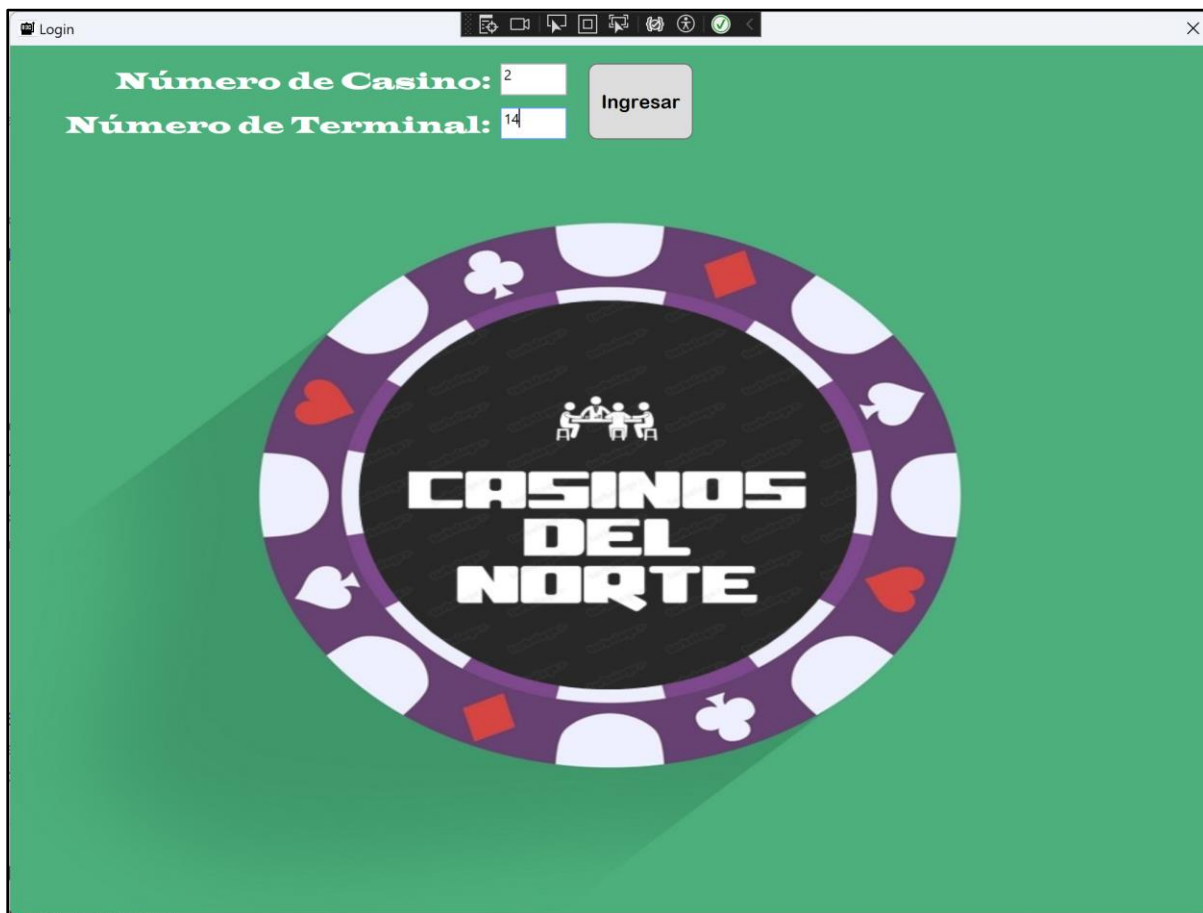
7.2 Documentación y Fuentes del Proyecto

El proyecto tiene la documentación completa y el código fuente de la solución en un repositorio privado de GitHub que se detalla a continuación:

- **Repositorio Principal:** <https://github.com/AldoDanielGonzalez/MCGA-TP.Final>
- **Documentación Completa:**
<https://github.com/AldoDanielGonzalez/MCGA-TP.Final/tree/master/docs>
- **Código Fuente Servidor de Casino y Webhook:**
<https://github.com/AldoDanielGonzalez/MCGA-TP.Final/tree/master/Servidor-Webhook>
- **Código Fuente Terminales de Casino:**
<https://github.com/AldoDanielGonzalez/MCGA-TP.Final/tree/master/Tragamonedas>
- **Código Fuente Servicio de Bitácora:**
<https://github.com/AldoDanielGonzalez/MCGA-TP.Final/tree/master/Bitacora>
- **Scripts de Bases de Datos:**
<https://github.com/AldoDanielGonzalez/MCGA-TP.Final/tree/master/scripts>

7.3 Manual de Uso Terminales Tragamonedas

- 1- Una vez iniciado la terminal, se deberá ingresar el numero de CASINO y TERMINAL válido provisto por el administrador:



- 2- Al ingresar se deberá seleccionar el valor del billete para agregar dinero disponible para jugar y presionar en el botón AGREGAR DINERO, tantas veces como sea necesario a fin de tener dinero disponible para múltiples jugadas o bien se podrá ir agregando dinero a lo largo de la sesión de juego:

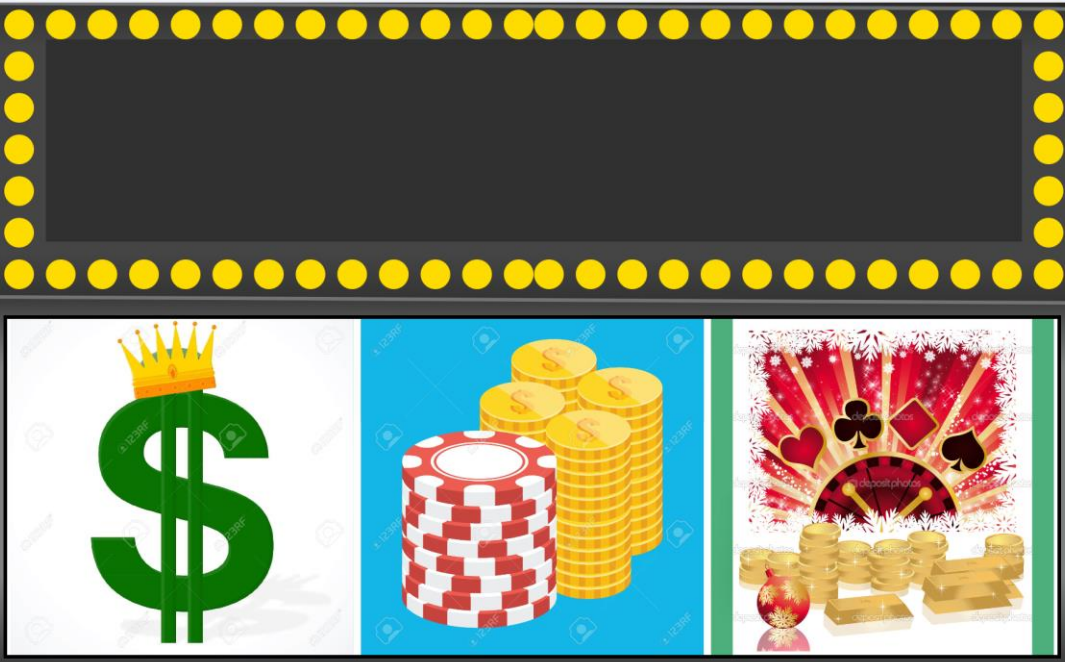
Disponible U\$S	0	Retirar
Billete U\$S	▼	Agregar Dinero
Apuesta	1	
	5	
	10	
	20	
	50	
	100	


- 3- Una vez ingresado el dinero, aparecerá como DISPONIBLE. A partir de ahora se deberá seleccionar la apuesta que se desea realizar, contemplando que la apuesta deberá estar entre el rango de Apuesta mínima y máxima permitida para el casino donde estamos jugando. Dichos valores se encuentran detallados a la izquierda del botón principal de la terminal:

Apuesta Mínima U\$S	200		Disponible U\$S	3,000	Retirar
Apuesta Máxima U\$S	2,000		Billete U\$S	100	Agregar Dinero
			Apuesta	100 200 300 400 500	

- 4- Configurada la apuesta deseada, el jugador deberá presionar el botón principal de la terminal a fin de realizar la jugada y esperar el resultado:

Terminal



Apuesta Mínima U\$S	200		Disponible U\$S	3,000	Retirar
Apuesta Máxima U\$S	2,000		Billete U\$S	100	Agregar Dinero
			Apuesta	200	

- 5- En el caso de que la jugada no resulte ganadora, se mostrará la leyenda "GAME OVER" y se actualizará el dinero disponible restando el importe de la apuesta realizada. El usuario tendrá la opción de continuar jugando con el dinero restante o presionar la opción "RETIRAR DINERO" para retirarse y finalizar la sesión de juego:



- 6- En el caso de una jugada ganadora, se visualizará la leyenda "GANASTE". Posteriormente, se presentará un mensaje detallando el monto de la ganancia obtenida. En este punto, el usuario deberá tomar la decisión de retirar la ganancia acumulada o continuar jugando con su dinero disponible y la ganancia obtenida hasta el momento. En ambas elecciones, se procederá a actualizar el saldo disponible en la terminal.

