Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Programación de Redes 1

Obligatorio 3

Renzo José Nro. Est. 240272

Nicolas Duarte Nro. Est. 268772

Repositorio: https://github.com/ArqSis-PDR-2024-2/obl-n6a-licenciatura-268772-165010-240272/tree/main

Grupo: N6A

Docentes: Andrés Soria y Guillermo Echichure

TABLA DE CONTENIDO

CONTENIDO

DESCRIPCIÓN DEL PROYECTO:

El proyecto "Sistema Vapor" ha evolucionado significativamente con esta tercera actualización, integrando tecnologías modernas para cumplir con los nuevos requerimientos de monitoreo, administración remota y escalabilidad. El sistema ahora integra tecnologías avanzadas como gRPC, RabbitMQ y REST APIs. Se ha logrado una evolución sustancial respecto a las versiones anteriores, enfocándose en la administración remota, monitoreo en tiempo real y procesamiento de eventos, destacando las siguientes mejoras y comparaciones clave.

TECNOLOGÍAS UTILIZADAS:

- **gRPC:**
  Utilizado para comunicaciones de baja latencia entre el servidor principal y el servidor administrador, optimizando la transmisión de eventos.

  Ventaja: En entregas anteriores, el sistema se limitaba al protocolo TCP, lo que restringía la escalabilidad y la modularidad.

- **RabbitMQ:**
  Maneja la mensajería entre módulos, permitiendo el procesamiento concurrente de eventos y sincronización eficiente.

- **REST-API:**

  Se ha diseñado un nuevo módulo REST para la exposición de estadísticas y generación de reportes. Esto incluye :

  E**ndpoints Definidos:** Consultas avanzadas como filtrado por precio, plataforma y calificación.

  Acceso Remoto: Integración de Swagger UI para facilitar la interacción con los servicios.

  Modernización: Anteriormente, el monitoreo de eventos era limitado. Ahora, el servidor de estadísticas opera en un contenedor Docker junto con RabbitMQ, mejorando portabilidad y despliegue.

IMPACTO DE LAS MEJORAS:

Esta actualización transforma el "Sistema Vapor" en una plataforma aún más robusta, escalable y adaptable a las necesidades del cliente. Los nuevos módulos añaden valor al sistema al integrar monitoreo avanzado y capacidades administrativas remotas, todo mientras mantienen la experiencia de usuario y las funcionalidades clave del sistema base.

Aunque no fue necesario realizar cambios en las reglas del dominio de negocio, la lógica existente se migró a un nuevo servidor gRPC. Este servidor sigue ofreciendo las mismas funcionalidades a través de un puerto TCP, además de exponer un servicio gRPC en un puerto adicional.

Para implementar esta migración, se desarrolló un nuevo proyecto ASP.NET Core gRPC. En el hilo principal de este proyecto, se invoca de manera asincrónica el método Main del servidor anterior en segundo plano, mientras que el servicio gRPC se inicia de forma sincrónica.

El servicio gRPC gestiona las peticiones de los clientes y utiliza el servidor anterior para ejecutar las funcionalidades, adaptando las solicitudes y respuestas según sea necesario. Así, las reglas de negocio permanecen centralizadas en un único lugar, mientras se exponen a través de dos servicios de red distintos.

SUPUESTOS:

- El usuario que utilice el Servidor Administrativo, actuando como un administrador del sistema, podrá dar de alta nuevos juegos y asignarles un usuario específico previamente registrado en el sistema. Inicialmente, se había considerado que los juegos dados de alta se asignan automáticamente al usuario "admin". Sin embargo, optamos por ofrecer mayor flexibilidad al permitir que el administrador seleccione un usuario diferente durante el alta, adaptándose así a distintos escenarios de administración y gestión dentro del sistema.

- No hay un mecanismo de seguridad o solicitud de credenciales al iniciar el cliente del servidor administrativo grcp, simplemente como ya comentamos antes, al tener este ciertos permisos especiales, una vez que se ejecuta y se levanta ya nos despliega el menú para poder realizar las operaciones correspondientes.

ARQUITECTURA Y NUEVOS MÓDULOS:

El sistema consta de cuatro módulos principales:

1. **Servidor Principal**: Continúa gestionando la publicación, compra y búsqueda de videojuegos, ahora con mejoras para integrar eventos en tiempo real hacia otros módulos. De manera simultánea corriendo el servidor gRPC para el cliente administrador y el ServerAPP antiguo para manejar los clientes TCP. Actúa como productor de notificaciones hacia RabbitMQ
2. **Cliente**: Aplicación de línea de comandos. Mantiene las funcionalidades de interacción de usuarios finales, alineándose con los requerimientos iniciales.
3. **Servidor de Estadísticas**: Proporciona un API para el monitoreo y generación de reportes basados en eventos de usuarios en el ServerAPP.
4. **Servidor Administrativo**: Aplicación de línea de comandos. Permite la gestión remota del sistema, incluyendo la administración de videojuegos y la visualización en tiempo real de compras. Este servidor administrativo tendrá la potestad de acceder directamente a funciones especiales y con permisos de usuario administrador, dando de alta juegos para determinados usuarios, eliminándolos, modificándose, etc. Mantiene una conexión con socket gRCP hacia el servidor principal

MEJORAS CLAVE Y FUNCIONALIDADES NUEVAS

1. Servidor de Estadísticas:

Se implementa un API para la funcionalidad de un servidor de estadísticas de la aplicación STEAM. Actualmente solo se han diseñado los endpoints necesarios para generar las consultas especificadas, en un futuro se pretende implementar una interfaz web con el fin de favorecer al usuario.

Monitoreo de Eventos:

- *SERF1*: Recibe eventos en tiempo real desde el servidor principal relacionados con videojuegos (creación, modificación, bajas, compras).
- *SERF3*: Registra eventos de inicio de sesión de usuarios.

Filtrado y Estadísticas Avanzadas:

- *SERF2*: Filtrado de videojuegos por los siguientes criterios combinables, siendo precio mínimo/máximo, plataforma y mínima/máxima valorización.
- *SERF4*: Generación de estadísticas sobre la cantidad de usuarios que han iniciado sesión.

Reportes Detallados:

- *SERF5*: Genera reportes sobre las ventas de videojuegos por editor. El proceso es asíncrono, permitiendo realizar otras operaciones mientras se completan las tareas de generación, a su vez como aportando un endpoint para verificar su estado.

Acceso Remoto:

- *SERF6*: Exposición de las funcionalidades de filtrado (*SERF2*), estadísticas de usuarios (*SERF4*) y (SERF5) mediante una REST API, habilitando consultas desde aplicaciones externas.

Despliegue Moderno:

- *SERF7*: Desplegado como un contenedor Docker, facilitando su portabilidad y configuración en diferentes entornos.

2. Servidor Administrativo:

La clase AdminGrpc representa el núcleo del Servidor Administrativo, diseñado para gestionar la administración remota de los juegos en el sistema. Este componente fue desarrollado utilizando la tecnología gRPC, seleccionada por su eficiencia en la comunicación entre aplicaciones y su capacidad para soportar múltiples usuarios administrativos simultáneamente.

La implementación del **GameManagementService** dentro del servidor administrativo (AdminGrpc) destaca por su capacidad de administrar los juegos en el sistema de manera eficiente y escalable mediante el uso de gRPC. Este servicio actúa como el núcleo funcional del servidor, ofreciendo operaciones avanzadas para la gestión de juegos.

Gestión Completa de Juegos:

- *SARF1*: Alta, baja y modificación de videojuegos, manteniendo las reglas del sistema.

Consultas y Monitoreo en Tiempo Real:

- *SARF2*: Permite a los administradores consultar calificaciones de juegos específicos.
- *SARF3*: Implementa un sistema de notificaciones en vivo que imprime las próximas N compras directamente en consola.

Interfaz Concurrente:

- *SARF4*: La consola soporta múltiples administradores conectados simultáneamente, asegurando una experiencia fluida.

Comunicación entre servicios:

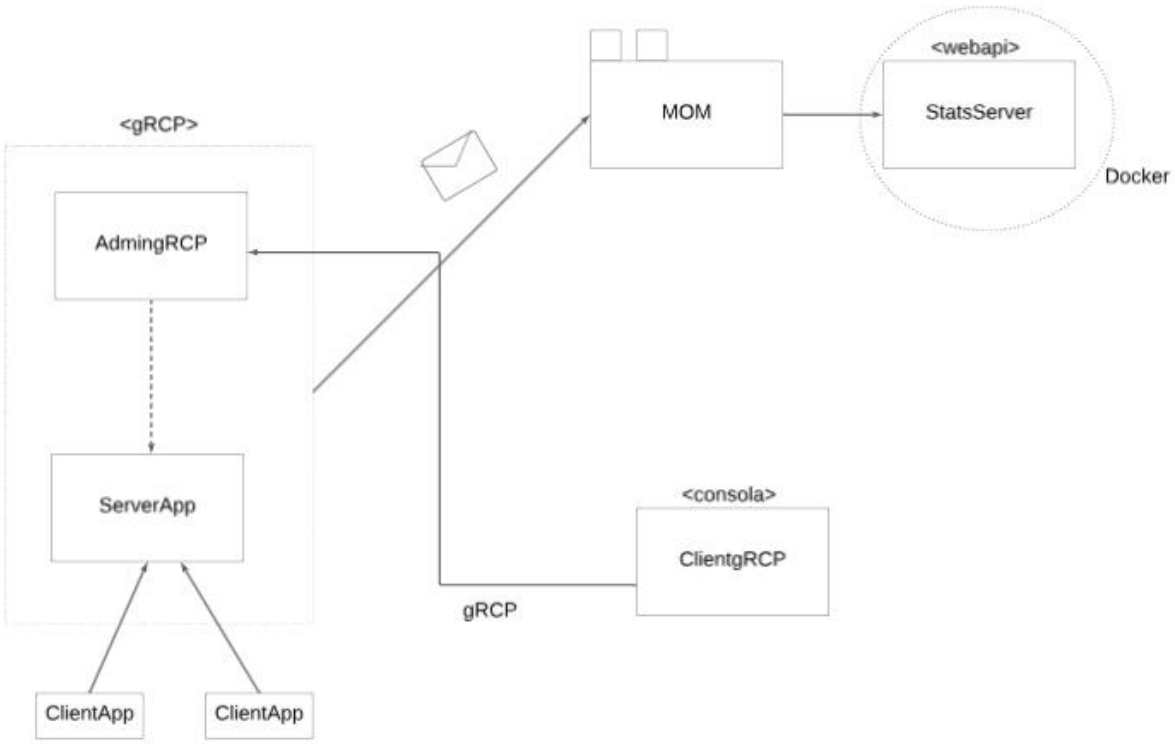
La arquitectura de la aplicación está diseñada para maximizar la eficiencia, escalabilidad y confiabilidad en la comunicación entre los diversos servicios. Aquí analizamos las razones y beneficios de la estrategia de comunicación adoptada:

- **Uso de gRPC y Sockets en ServerApp:**
    - gRPC: Proporciona una comunicación eficiente y de alto rendimiento entre servicios, ideal para las operaciones que requieren llamadas remotas frecuentes y datos estructurados, como las interacciones entre el ServerApp y el AdmingRCP.
    - Sockets: Permite una comunicación directa y de bajo nivel, adecuada para casos de uso que requieren un control detallado de la conexión y un flujo de datos personalizado, como la interacción entre el **ClientApp** y el **ServerApp**.
- **RabbitMQ para la Mensajería Asíncrona:**
    - Desacoplamiento: RabbitMQ actúa como un intermediario entre servicios, permitiendo que estos se comuniquen sin depender directamente unos de otros. Esto reduce el acoplamiento y aumenta la flexibilidad del sistema.
    - Fiabilidad y Consistencia: RabbitMQ garantiza que los mensajes no se pierdan en caso de fallos temporales en alguno de los servicios, lo que es crucial para la consistencia de las operaciones de compra y notificaciones por correo.
- **REST API en StatsServer**:
- **Colas de Mensajes y Eventos de Compras**:
    - Los servidores de compras y correo se suscriben a colas de mensajes en RabbitMQ para recibir eventos de compra, lo que permite un procesamiento asincrónico y eficiente de estas operaciones. Disponible en los clientes administrador gRPC.

Beneficios de la Estrategia de Comunicación:

- **Escalabilidad**: La arquitectura basada en mensajes y gRPC/TCP permite escalar los servicios de manera independiente según la demanda.
- **Resiliencia**: Al utilizar colas de mensajes, el sistema puede manejar mejor los picos de carga y recuperarse de fallos temporales.
- **Flexibilidad**: La combinación de diferentes métodos de comunicación (gRPC, TCP, RabbitMQ, REST API) ofrece flexibilidad para adaptarse a diferentes necesidades y contextos de uso.
- **Mantenimiento y Desarrollo**: Esta estrategia facilita el mantenimiento y desarrollo continuo del sistema, ya que los cambios en un servicio pueden implementarse con un impacto mínimo en otros servicios.

A continuación, presentamos un diagrama de la solución para esta tercera entrega, con el objetivo de facilitar una mejor comprensión de esta:



Por un lado, tendremos el servidor TCP (ServerApp) junto con sus clientes TCP antiguos (ClientApp), los cuales se conectan utilizando esta tecnología. Además, contaremos con un servidor administrativo (Cliente gRPC) y, por otra parte, un servidor gRPC encargado de proporcionar servicios al Cliente gRPC mediante la tecnología gRPC. El servidor gRPC (Admin gRPC) será importado al ServerApp como una biblioteca de clases, lo que permite visualizar tanto al Admin gRPC como al ServerApp como un único servidor principal.

También dispondremos del StatsServer, que será un servidor de WebApi encargado de brindar ciertas operaciones. Este obtendrá los datos desde el servidor WebApi mediante la tecnología MOM (Message-Oriented Middleware). Contaremos con un servidor de cola de mensajes que notificará eventos del servidor principal al MOM. Posteriormente, el servidor WebApi consumirá esos mensajes, los procesará y obtendrá los datos necesarios para persistirlos en su sistema.

La configuración incluye dos tipos de exchanges en MOM:

-**Básico**, que se utiliza para la notificación de eventos estándar declarado “steam_logs”.

-Fanout, diseñado para implementar la notificación de múltiples eventos futuros, en este caso las próximas compras, declarado “next_purchases”.

StatsServer Endpoints:

Debido a la simpleza y facilidad que brinda Swagger UI para la utilización de los endpoints lo hemos implementado a nuestra solución de forma tal que se ejecute a la vez que el servidor de estadísticas pudiendo así ejecutar y observar cada endpoint. Para esto es necesario ingresar a la siguiente url http://localhost:8080/swagger/index.html.

A continuación, se detallan los endpoints definidos:

FUNCIONALIDAD: Obtener cantidad de usuarios que iniciaron sesión.		
URL: GET /stats/users		
RESPUESTA	**CÓDIGO**	**CUERPO DE LA RESPUESTA**

Solicitud exitosa.	200: OK	{
		2
		}

Solicitud exitosa, pero nadie inició sesión.	200: NotFound	{
		“message”: "No one logged in."
		}

Error en solicitud.	500: IActionResult	{
		"Internal Server Error: " + ex.Message
		}

FUNCIONALIDAD: Obtener lista de productos filtrados		
URL: GET /stats/filtered		
RESPUESTA	**CÓDIGO**	**CUERPO DE LA RESPUESTA**
Solicitud exitosa	200: OK	[{ "name": "Fornite", "genre": "Action", "releaseDate": "2020-01-01T00:00:00", "publisher": "admin", "platform": "PC", "price": 10, "imageName": null, "unitsAvailable": 9, "valoration": 8, "reviews": [] }, { "name": "Minecraft", "genre": "Adventure", "releaseDate": "2010-03-03T00:00:00", "publisher": "Mojang", "platform": "PC", "price": 20, "imageName": null, "unitsAvailable": 150, "valoration": 10, "reviews": [] },

Solicitud exitosa, no existen juegos con criterio.	200: OK	{
		"message": "No games found with the specified criteria."
		}

Error en solicitud.	500: IActionResult	{
		"Internal Server Error: " + ex.Message
		}

FUNCIONALIDAD: Obtener el reporte de ventas.		
URL: GET /stats/reports		
RESPUESTA	**CÓDIGO**	**CUERPO DE LA RESPUESTA**

Solicitud exitosa	200: OK	{
		"Mojang": 1
		}

Solicitud con reporte aún no realizado.	404: Not Found	{
		"message": "No sales report available."
		}

Error en solicitud.	500: IActionResult	{
		"Internal Server Error: " + ex.Message
		}

FUNCIONALIDAD: Generar reporte de ventas.		
URL: POST /stats/reports		
RESPUESTA	**CÓDIGO**	**CUERPO DE LA RESPUESTA**

Solicitud exitosa	200: OK	{
		"message": `Report is being generated.`
		}

Error en solicitud.	500: IActionResult	{
		`"Internal Server Error: " + ex.Message`
		}

FUNCIONALIDAD: Verificar estado del reporte.		
URL: GET /stats/reports/status		
RESPUESTA	**CÓDIGO**	**CUERPO DE LA RESPUESTA**

Solicitud exitosa	200: OK	{
		"message": `The report is ready.`
		}

Solicitud exitosa	200: OK	{
		"message": `The report isnt ready.`
		}

Error en solicitud.	500: IActionResult	{
		`"Internal Server Error: " + ex.Message`
		}

Bugs conocidos:

Por alguna razón desconocida, los juegos precargados no se pueden editar desde el administrador GRPC. Sin embargo, esto no afecta en absoluto el funcionamiento general.

Instalación y ejecución

Archivos de Configuración

Los archivos App.config son utilizados en ServerApp, ClientApp y AdmingRCP para almacenar y gestionar configuraciones relevantes. Estas configuraciones cubren diversos aspectos, como la configuración de red, parámetros del entorno de ejecución, y detalles de la integración con RabbitMQ.

Definición de Servicios

- Cada servicio (como AdmingRCP, ClientgRCP, StatsServer) se define de manera individual
- El servidor de estadística estará contenido dentro de una imagen docker con RabbitMQ, ambos se ejecutarán al mismo tiempo.
- Todos los servicios están desarrollados utilizando .NET Core 8

Exposición de Puertos

- Se exponen puertos específicos para cada servicio, lo que permite la comunicación entre el contenedor y el host externo

RabbitMQ

- Se utiliza la imagen rabbitmq:4.0.3-management, que incluye la interfaz de administración de RabbitMQ.
- Expuesto en el puerto 15672
- Se monta un volumen para persistir los datos de RabbitMQ.

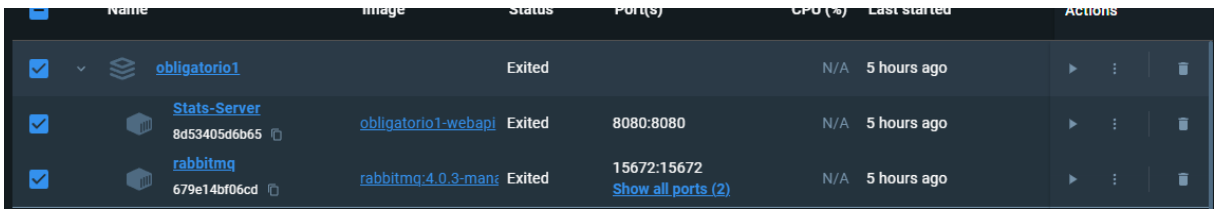
Ejecución:

Como prerrequisito es necesario contar docker engine corriendo localmente, luego en la raíz del proyecto ejecutar los siguientes comandos:

> docker-compose build

> docker-compose up

Esperar unos segundos hasta que el log aparezca, que ya está construido. Luego podremos ir a Docker y veremos los siguientes contenedores.

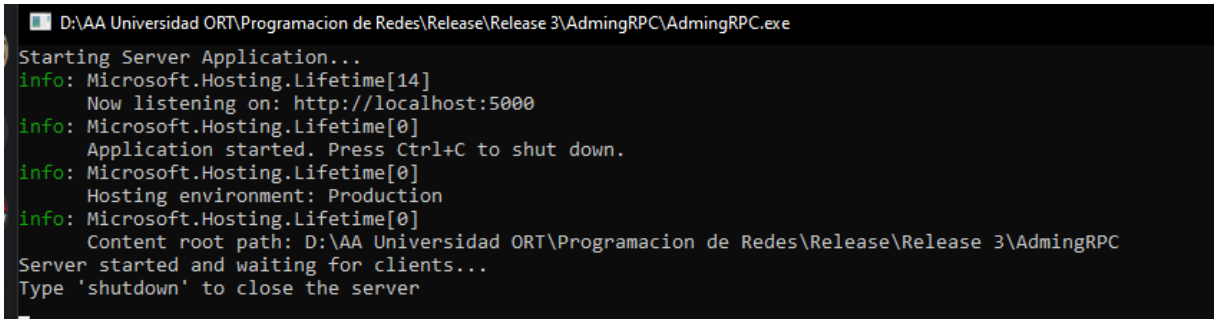


Los cuales nos permitirán ver en qué puertos están nuestras aplicaciones desplegadas, en este caso el stats server en el puerto 8080, útil información para utilizar luego el swagger y el RabbitMQ en el 15672.

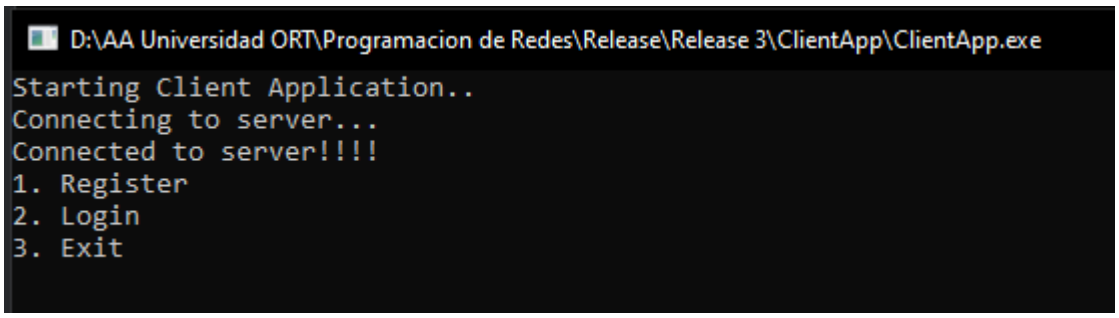
Una vez que tengamos nuestro Docker corriendo ya podremos ejecutar el resto de las aplicaciones y servidores de forma normal a través de los ejecutables otorgados, comenzando por el AdmingRPC, el cual contendrá el servidor nuevo de consultas de administrador gRPC y a su vez simultáneamente el servidor antiguo TCP para los clientes normales. Luego ya podremos levantar la cantidad que sea necesaria de Clientes TCP y Clientes gRPC.

Luego si se quiere utilizar la API y sus funcionalidades, se accede a http://localhost:8080/swagger/index.html donde estarán todos los endpoints establecidos de una manera fácil y simple de interpretar gracias a nuestra especificación de que hace cada endpoint.

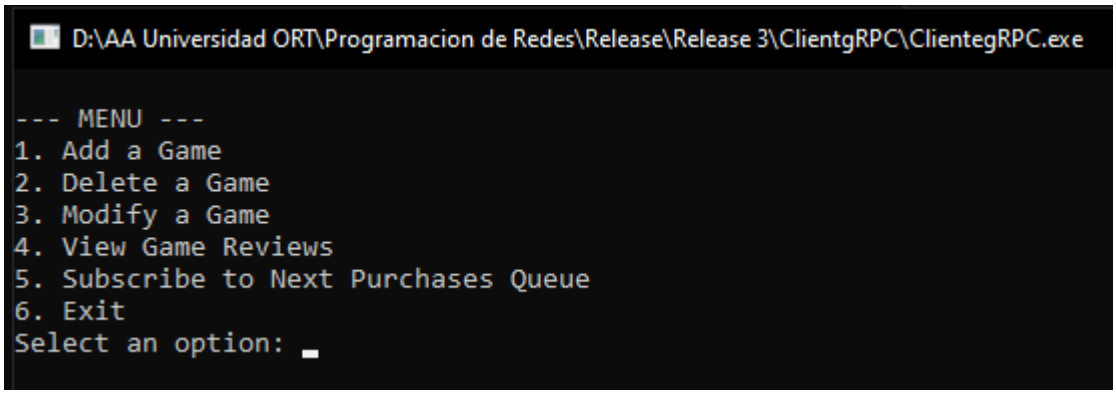
AdmingRPC:



ClientApp:



ClientegRPC:



Datos Pre-Cargados:

Juegos:

```
1 usage   RenzoJse +1
private void PreLoadedGames()
{
    Game fortnite = new Game
    {
        Name = "Fornite",
        Genre = "Action",
        Publisher = "admin",
        ReleaseDate = new DateTime( year: 2020,  month: 1,  day: 1),
        UnitsAvailable = 10,
        Valoration = 8,
        Price = 10,
        Platform = "PC"
    };
    _games.Add(fortnite);

    Game roblox = new Game
    {
        Name = "Roblox",
        Genre = "Adventure",
        Publisher = "Roblox Corporation",
        ReleaseDate = new DateTime( year: 2009,  month: 2,  day: 2),
        UnitsAvailable = 150000,
        Valoration = 6,
        Price = 5,
        Platform = "PC"
    };
```

Usuarios:

```
1 usage   Nicolás Duarte +1
private void PreLoadedUsers()
{
    var admin = new User
    {
        Username = "admin",
        Password = "admin",
        PublishedGames = new List<Game>(),
        PurchasedGames = new List<Game>()
    };
    _users.Add(admin);

    var user1 = new User
    {
        Username = "nicolasduarte",
        Password = "password",
        PublishedGames = new List<Game>(),
        PurchasedGames = new List<Game>()
    };
    _users.Add(user1);
```