Obligatorio 3 - Programación de redes

Nicolás Rosa - 225609
Gastón Landeira - 238473

Profesor: Luis Barrague

Universidad ORT Uruguay
2021

Gastón Landeira - 238473
Nicolás Rosa - 225609

Índice

Notas:

Todo el proyecto se realizó en Github, se puede encontrar en el siguiente link: https://github.com/nicoRosa18/Obligatorio-2-Programacion-de-redes

Se adjunta una carpeta llamada Diagramas con los archivos de los diagramas completos para una mejor visualización. Estos no se agregan aquí por el tamaño de los mismos. Los archivos .svg se pueden abrir en el navegador.

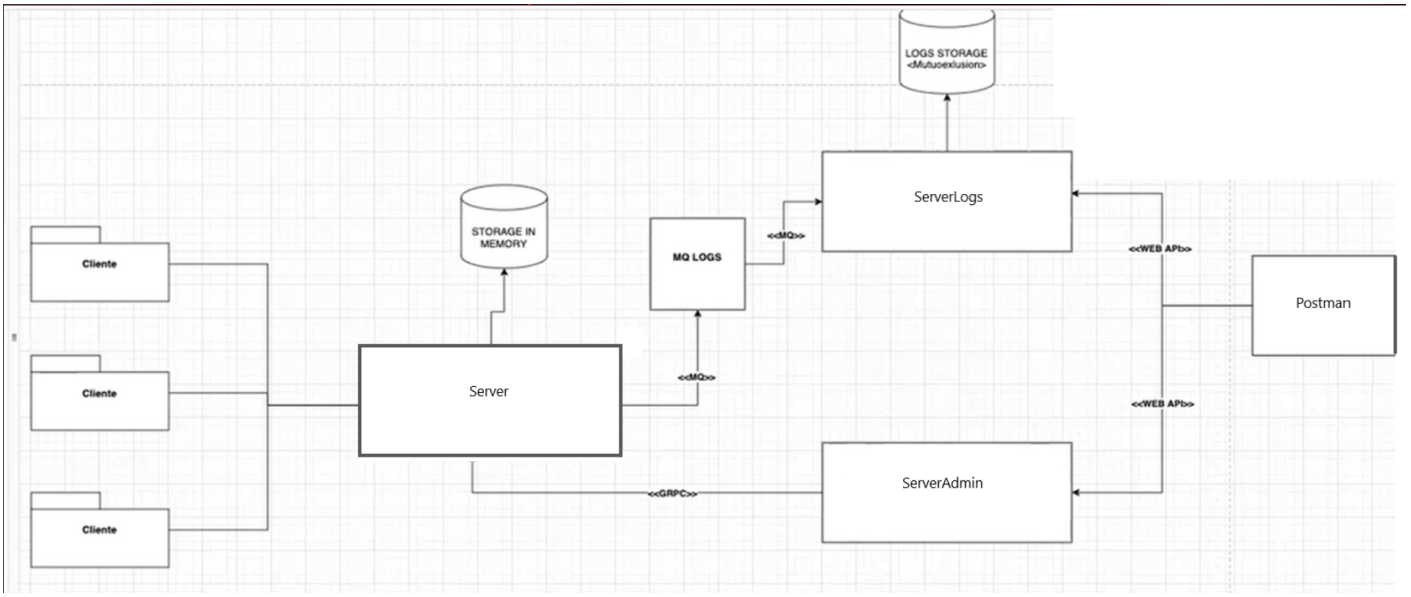
Todo este documento se apoya en que los dos documentos anteriores ya han sido leídos, este solamente incluirá los cambios y características nuevas. Se mantienen los diagramas UML y los diagramas de secuencia que no sufrieron cambios en la carpeta mencionada de Diagramas más los nuevos.

Los endpoints manejados en las Webapis se encuentran documentados en la colección de postman agregada en la carpeta ColeccíondePostman en la Documentacion.

Gastón Landeira - 238473
Nicolás Rosa - 225609

Descripción de la arquitectura.

En esta tercera y final entrega del obligatorio se debió implementar una interfaz de monitoreo y otra de gestión administrativa por fuera de la conexión existente con los clientes (implementados en el primer obligatorio). Para esto se actualizó el proyecto Server y se agregaron tres nuevos proyectos. ServerLogs y CommonLogs para solucionar el primer problema, y ServerAdmin para el segundo.



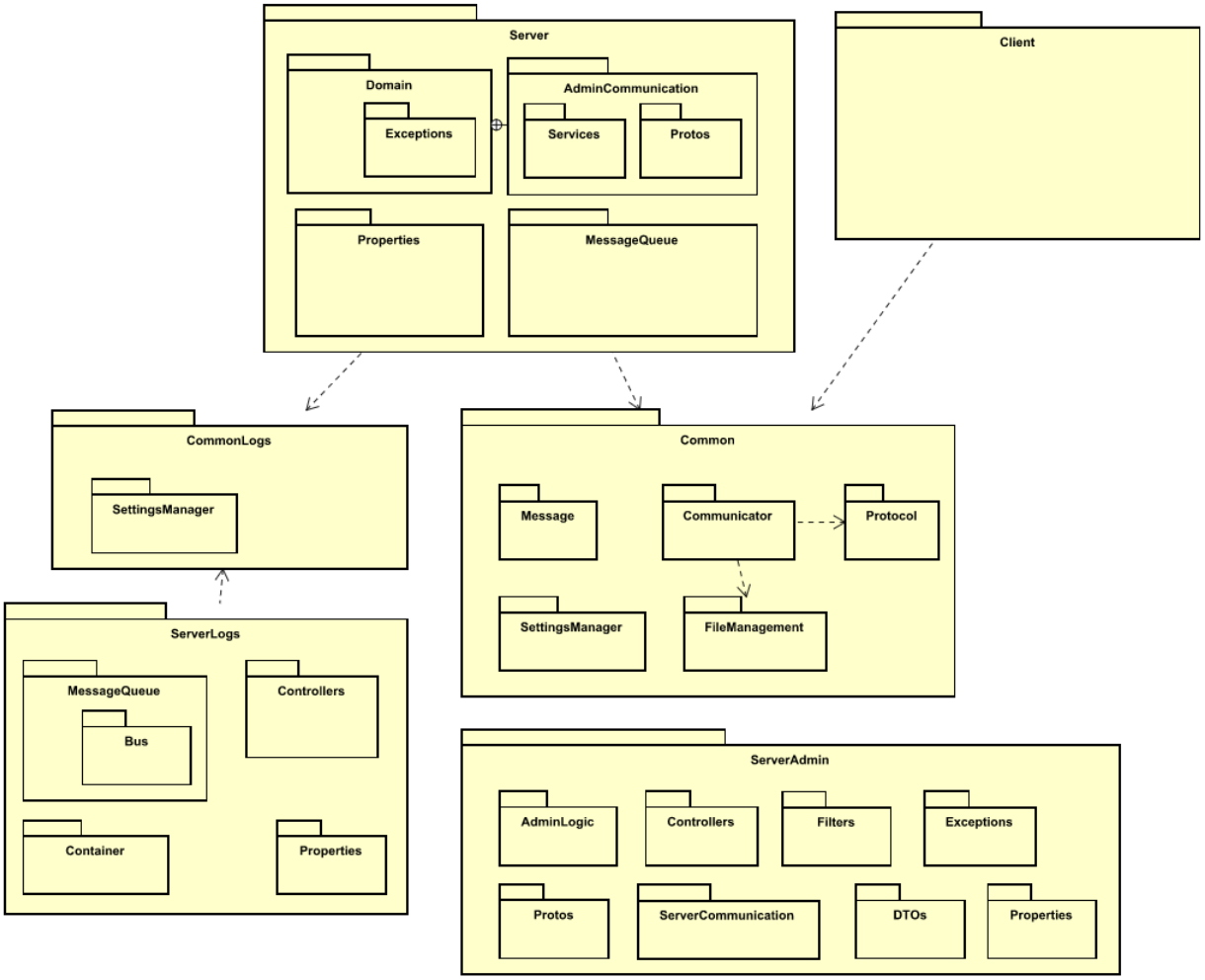
ServerLogs se encarga de guardar un historial de las acciones que realiza todo usuario (cliente) de una manera inteligente, con el propósito de que más tarde sea posible filtrarlos eficientemente. Este filtrado se expone a través de una WebApi, mientras que la conexión con el Server se realiza mediante un MOM. Este middleware es una cola de mensajes simple a la cual Server submite mensajes y cuando ServerLogs se inicializa o se encuentra corriendo este los consume. O sea que es completamente asincrónico, no tienen que estar los dos ejecutándose al mismo tiempo.

CommonLogs cuenta con clases que son de utilidad para la comunicación entre los proyectos que usan la cola de mensajes, brindando el tipo de objeto con el cual trabajarán.

Con el propósito de tener una central administradora se encuentra el servidor administrativo ServerAdmin. Este expone a través de su WebApi métodos de mayor control y autoridad que con los que cuenta un cliente. Este se conecta a Server a través de RPC, específicamente GRPC, utilizando un lenguaje de traducción neutral llamado protocol buffers. Esto permite que tanto el servidor como el cliente manejen diferentes lenguajes de programación pero se comuniquen igualmente como si hubiesen sido programados en el mismo.

Gastón Landeira - 238473
Nicolás Rosa - 225609

Diagrama de paquetes:

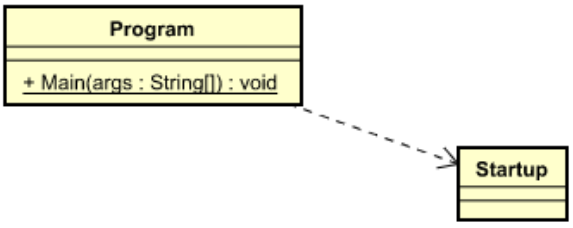


Documentación de diseño detallada de cada componente y justificación del diseño.

Server

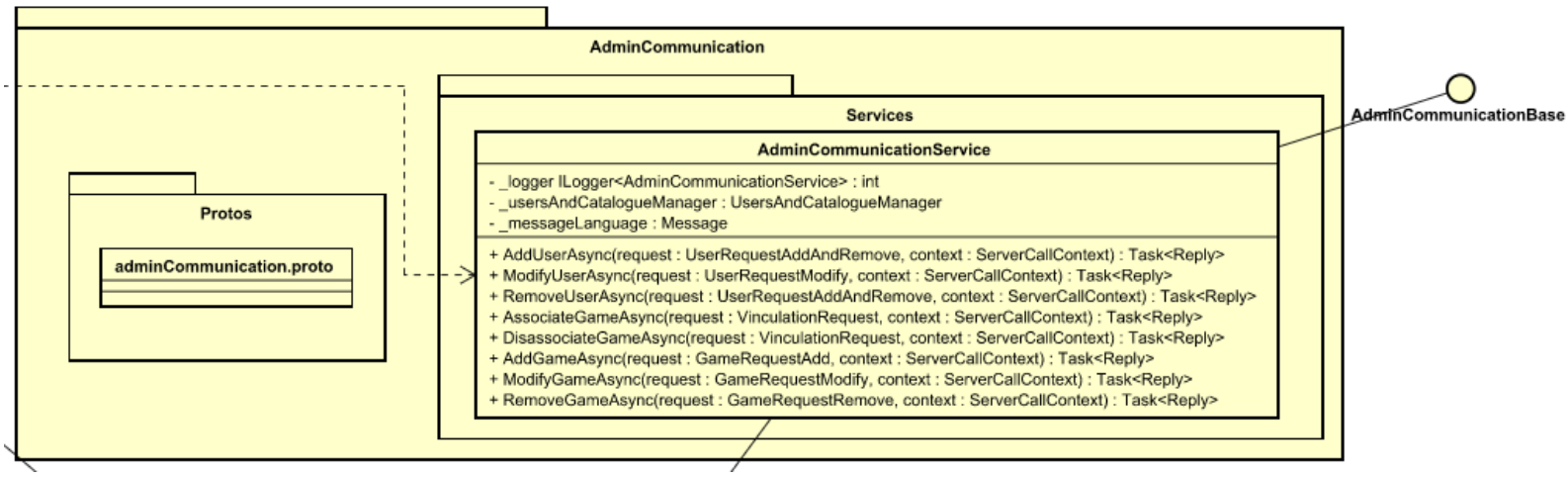
Al proyecto Server se le tuvo que adaptar al ahora encargarse de ser tanto el servidor de GRPC como el proveedor de la cola de mensajes. El cambio más drástico se puede ver con el primero, que al ser un servicio que se inicia aquí, se tuvo que preparar una forma de poder hacer que tanto las funcionalidades anteriores como esta nueva convivan.

En Program se llama a Startup y este se encarga de inicializar ServerManager y AdminCommunicationService (GRPC). Del segundo se hablará más tarde. Ambos conviven en la misma consola, y se sigue pudiendo observar cuando se conectan clientes o hasta se puede cerrar ServerManager y solo dejar corriendo GRPC.

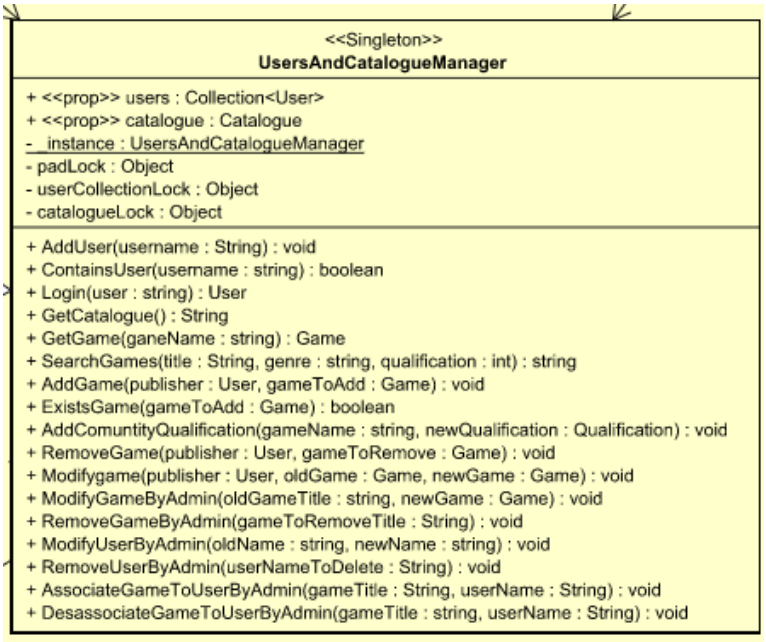


Gastón Landeira - 238473
Nicolás Rosa - 225609

Paquete AdminCommunication



Este paquete encapsula la comunicación con serverAdmin. Mediante clases generadas a través de estos, se recibe la request y se llama al método que expone AdminCommunicationService. Todos estos métodos realizan algún tipo de cambio en la base de datos en memoria del Server. Para llegar a estos métodos se utiliza el singleton de UsersAndCatalogueManager y, para poder cumplir sus requerimientos, también se lo tuvo que modificar. Varios fueron reusados, pero los métodos propios de Admin siguen la nomenclatura "...ByAdmin" como firma.



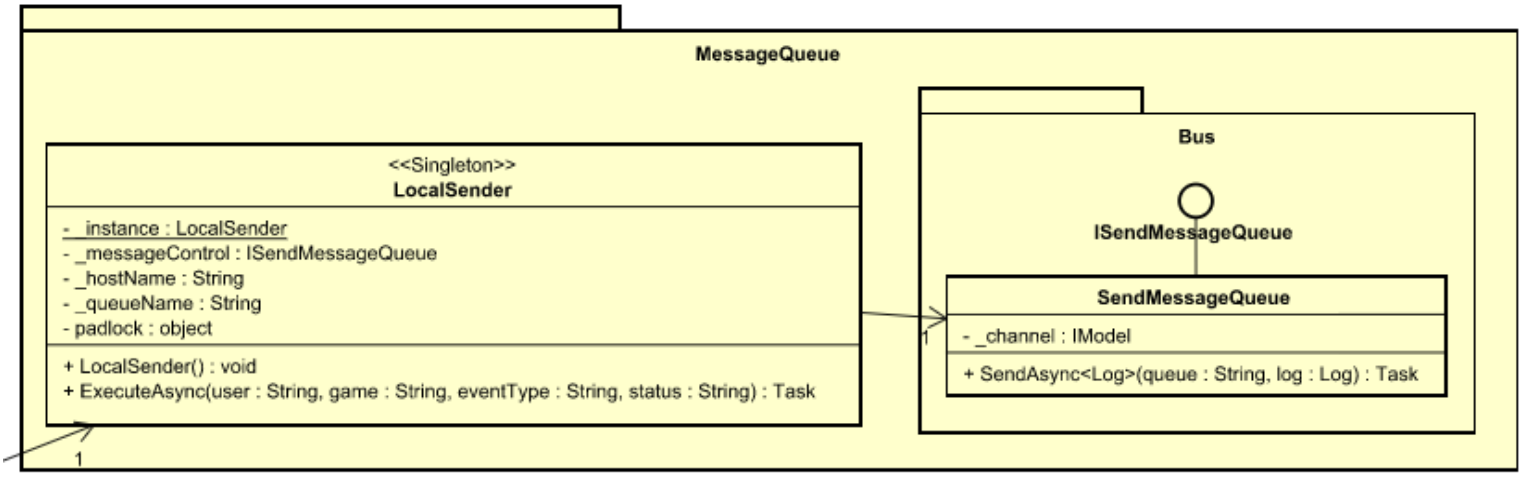
Se destaca el manejo del archivo .proto en el paquete Protos. Este es igual al archivo .proto de ServerAdmin. Es el lenguaje universal que se utiliza para su comunicación. De este archivo es que se generan las clases que permiten la comunicación con sus respectivos objetos de request y reply. En el caso de este obligatorio, el objeto reply es el mismo para todos los métodos, porque lo único que se responde es si fue satisfactoria la acción con un booleano, y el mensaje de respuesta del servidor en un string. Estos últimos son los mismos que están almacenados en SpanishMessages en el proyecto Common de las entregas pasadas.

Una curiosidad de cómo se realizó la conversión de proyecto simple ejecutable a GRPC Server es que se implementó GRPC en el. No se tuvo que crear otro proyecto GRPC Server e introducir todo Server allí.

Nota: Un juego agregado que no figura como haber sido publicado por un usuario, fue publicado o su usuario fue removido por un administrador desde ServerAdmin.

Paquete MessageQueue

El paquete MessageQueue se encarga de brindar la posibilidad a Server de subir sus mensajes a la cola de mensajes llamada “log_queue” de la cual se consumirá más tarde en ServerLogs. Bus es la clase que envía los mensajes y LocalSender la cual rellena los parámetros de sus métodos al ser utilizada. Es importante destacar que se decidió hacer de LocalSender una clase singleton. Esto es porque será llamada una infinidad de veces por el resto de Server y no tiene propósito crearla cada vez que se utilizara. Si se recuerda como estaba implementado Server en la primer entrega, este crea n sesiones para n clientes. Tener solo una clase que se encarga de subir los mensajes de todas las sesiones permite un mejor control en la subida de mensajes.



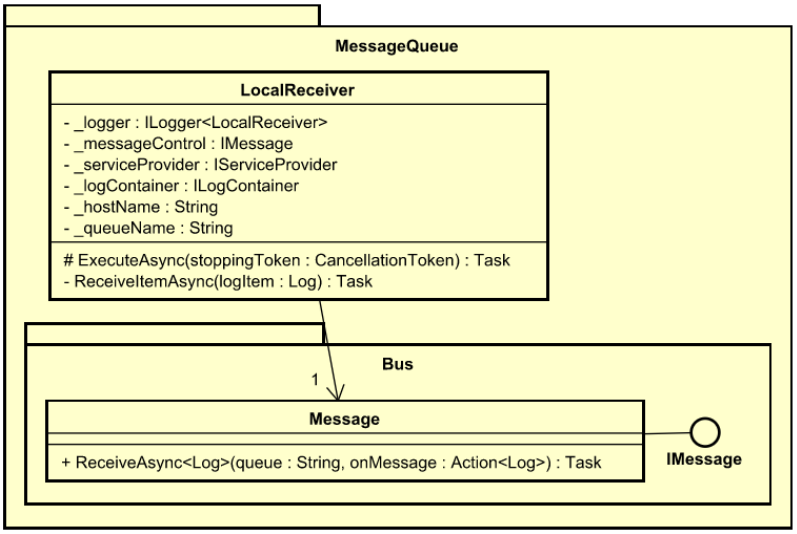
LocalSender como se menciona es utilizado en la clase Session y registra cada acción entre usuario y servidor. No está incluido el registro del trato entre un administrador y el servidor debido a que él es consciente de lo que hace, pero no lo es de un cliente.

ServerLogs

En resumen ServerLogs es un proyecto WebApi que implementa el consumo de colas RabbitMQ, en este caso “log_queue”, y almacena lo consumido en memoria. Del tipo de objeto consumido se mencionara más tarde en CommonLogs.

Paquete MessageQueue

Funciona a imagen y semejanza del paquete de mismo nombre en Server pero viceversa, su función es consumir de la cola. Consume asincrónicamente cada mensaje y lo deposita en Logic, el container de los mensajes.



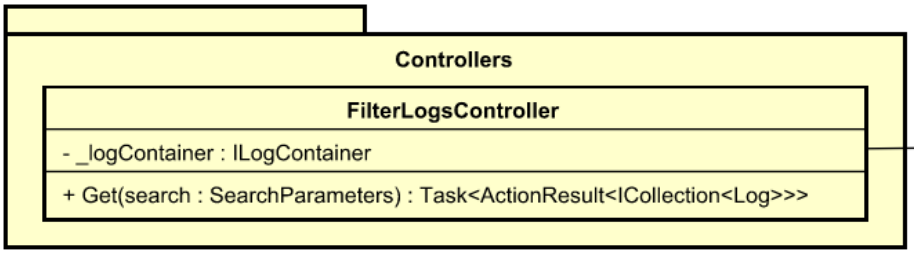
Descripción de la cola de mensajes utilizada

Se utiliza RabbitMQ para guardar los logs del servidor TCP. La comunicación será de la siguiente manera mediante una cola de mensajes:

Con un exchange entre el productor(P) y la queue (rectángulo rojo). Donde, el productor de los mensajes es el servidor Tcp Server que envía los logs a la cola y asincrónicamente cuando el servidor de Logs se levante, este recibirá los mensajes de la cola como cliente de esta.

Paquete Controllers

Su única clase FilterLogsController expone el endpoint del filtrado de los mensajes de la cola. Este retira de Logic una copia de los mensajes, filtrados por juego(string), usuario(string) o fecha(formato mm/dd/aaaa), y los envía como respuesta a una llamada.



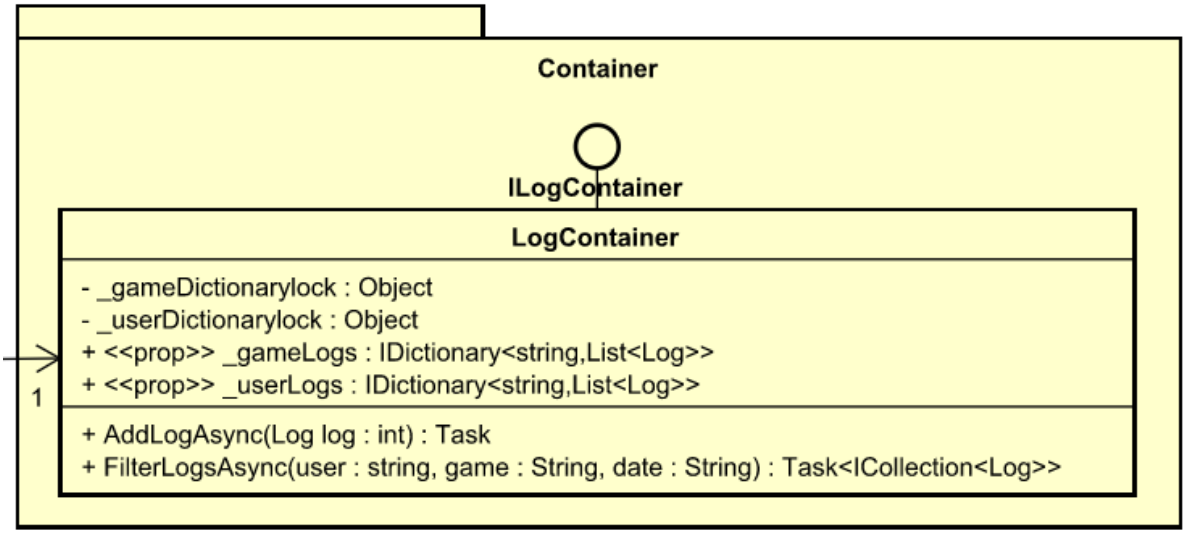
Los parámetros de búsqueda están definidos por fuera de este paquete.

Paquete Container

Se le debe su nombre a su funcionalidad de guardar los mensajes de forma inteligente. Para esto cuenta con dos diccionarios, el primero de juegos y el segundo de usuarios.

Por lo tanto, al recibir un mensaje se lo deposita en alguno de los dos diccionarios, o en los dos si es el caso de que es la acción de un usuario sobre un juego, aunque no resulte del todo eficiente guardar algo dos veces. Todo protegido con mutua exclusión mediante dos locks (uno por diccionario) que aseguran que son el único método utilizando el diccionario en cuestión.

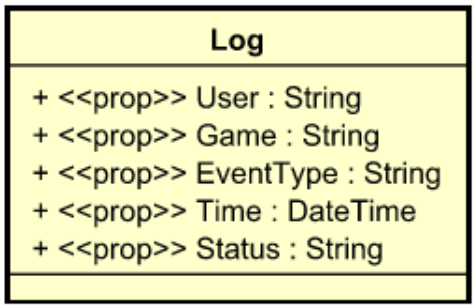
En cuanto a la búsqueda y retiro de mensajes, estos son de forma eficiente gracias a lo anteriormente mencionado. Se realiza el filtro en cada diccionario, obteniendo una copia de lo guardado en ellos, y finalmente se los filtra por fecha para su retorno al controlador.

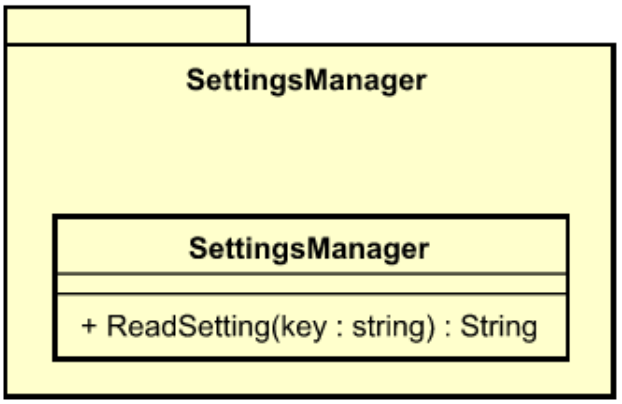


CommonLogs

La única funcionalidad de este proyecto es tener el objeto de la cola de mensajes compartido entre Server y ServerLogs. Este objeto es un log que contiene los datos más importantes de la comunicacion entre cliente y servidor, tanto los casos de éxito como de error.

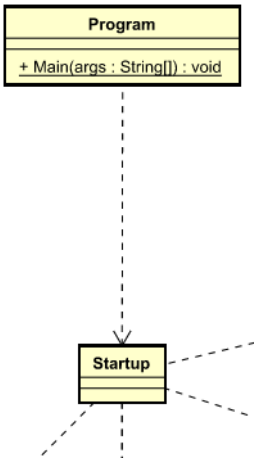
Además se tiene el paquete SettingsManager que permite a ServerLogs leer de su AppConfig el nombre de la cola y del host.





ServerAdmin

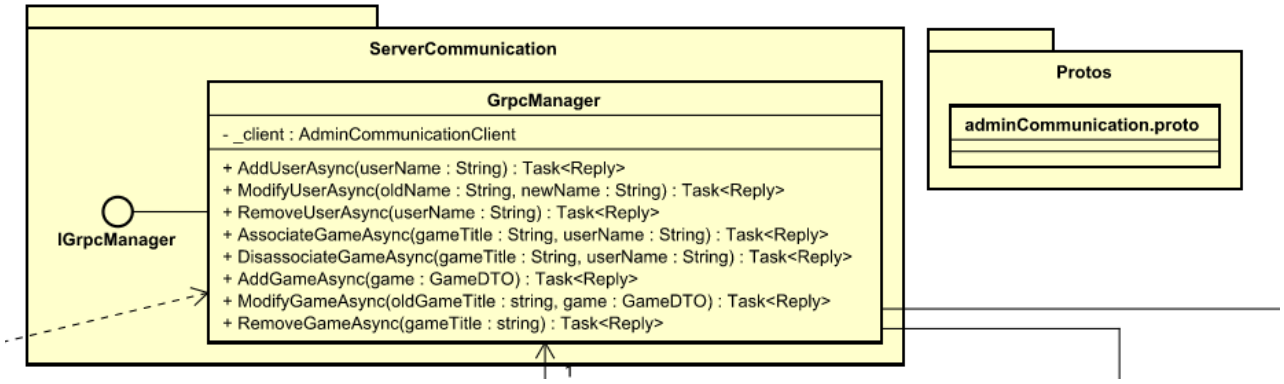
Es un proyecto WebApi que implementa GRPC Client.



Al iniciar el proyecto se inicializan los controladores, el filtro de excepciones, la comunicacion GRPC y la lógica como servicio scoped, o sea que se crea una instancia por llamada a los endpoints.

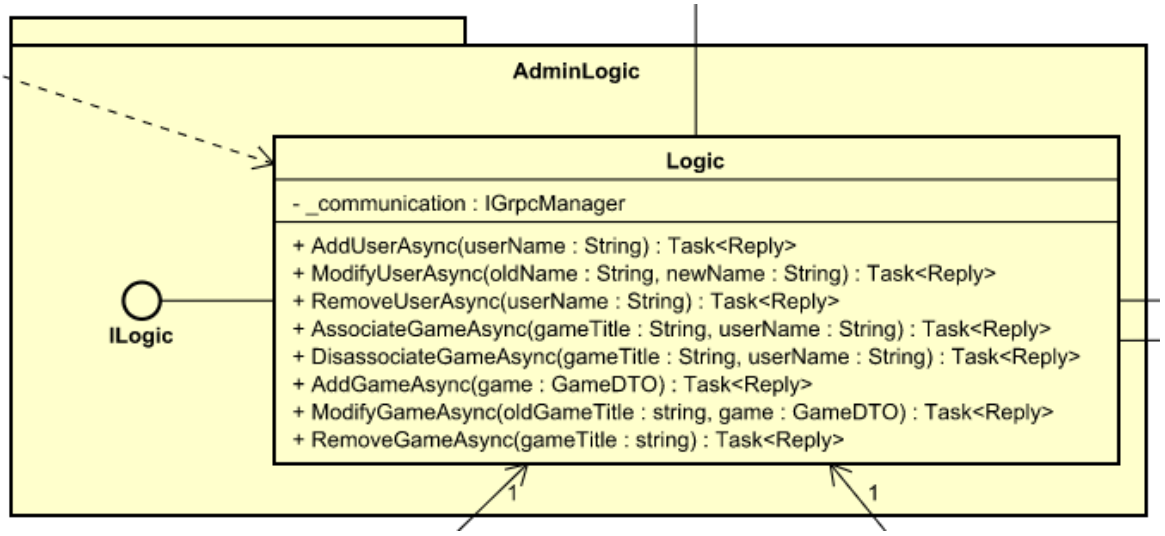
Punto a destacar es que para la comunicación GRPC se utiliza el puerto 31700 para la conexión. Esto fácilmente puede ser hecho una variable e insertada en un AppConfig.

Proyecto ServerCommunication



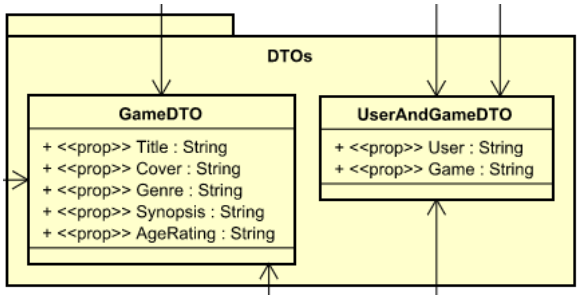
Funciona a imagen y semejanza del paquete de nombre AdminCommunication en Server pero viceversa, su función es enviar pedidos y esperar por su respuesta. GrpcManager contiene los métodos que se usan como intermediarios entre la comunicación y la lógica de ServerAdmin.

Proyecto AdminLogic



Cuenta con la lógica de crear las excepciones cuando el mensaje de respuesta no es satisfactorio, si no solo funciona como intermediario entre los controllers y GRPC.

Los tres proyectos utilizan tipos de objetos de traslado llamados DTOs. Estos son llamados así porque su contenido no es de importancia a ServerAdmin, solo están de paso para que sus propiedades sean enviadas.

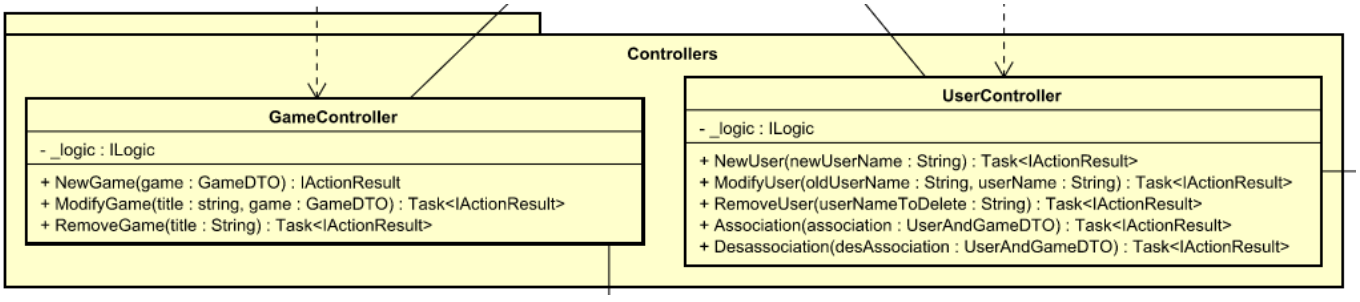


Proyecto Controllers

.

Se cuenta con dos controllers, uno dedicado a exponer la funcionalidad de los juegos y otro la del usuario.

Debido a que la transmisión de un archivo por GRPC requiere el uso de streaming de datos, no fue implementada la funcionalidad. Se asume que la carátula ya se encuentra en el Server y solo se le pasa su path.



Gastón Landeira - 238473
Nicolás Rosa - 225609

Documentación de mecanismos de comunicación de los componentes de la solución

Diagrama de secuencia de enviado de un request GRPC de ServerAdmin

En el diagrama se puede apreciar de mejor manera como los componentes de la solución trabajan en conjunto para enviar un pedido GRPC en ServerAdmin, y posteriormente recibir su respuesta. Comienza desde una llamada para crear un nuevo usuario en la WebApi, el llamado pasa por la lógica y finalmente por GrpcManager el cual utiliza los métodos instanciados en el archivo .protos para enviar el request a Server.

Más tarde recibe una respuesta que, en caso de ser satisfactoria, la WebApi responde a la llamada con el nuevo usuario agregado. En caso de que la respuesta contenga un error, esta será utilizada como respuesta del controller.

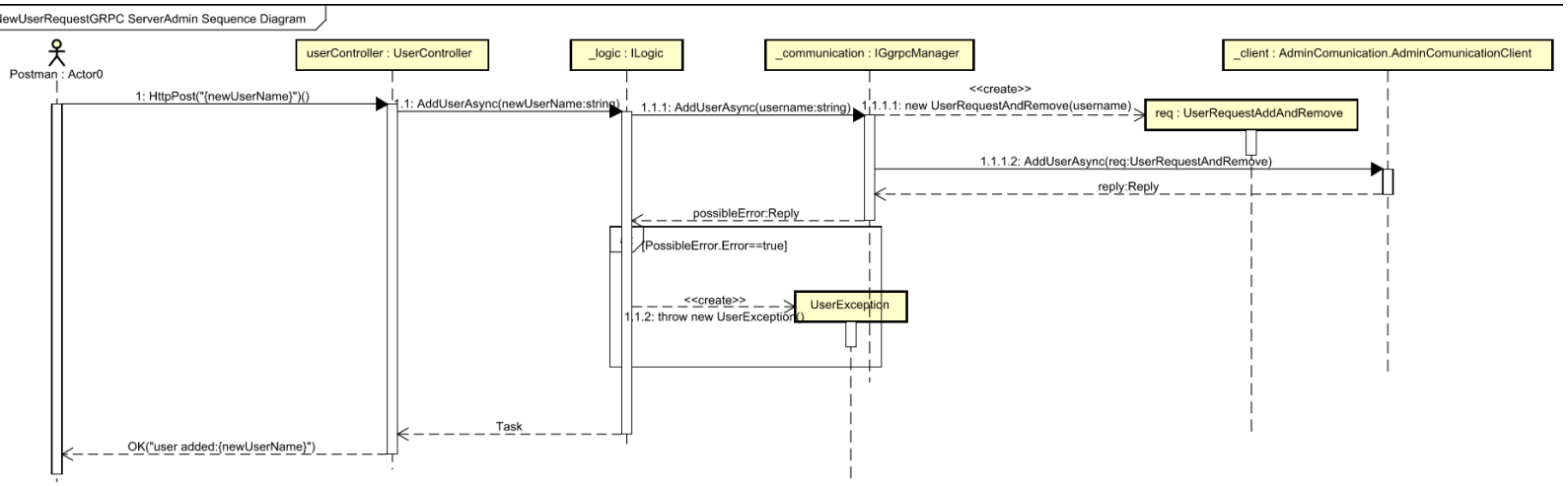


Diagrama de secuencia de recibida de un request GRPC de Server

En contraparte al envío de la request GRPC, se diagrama la recibida de este envío en Server.

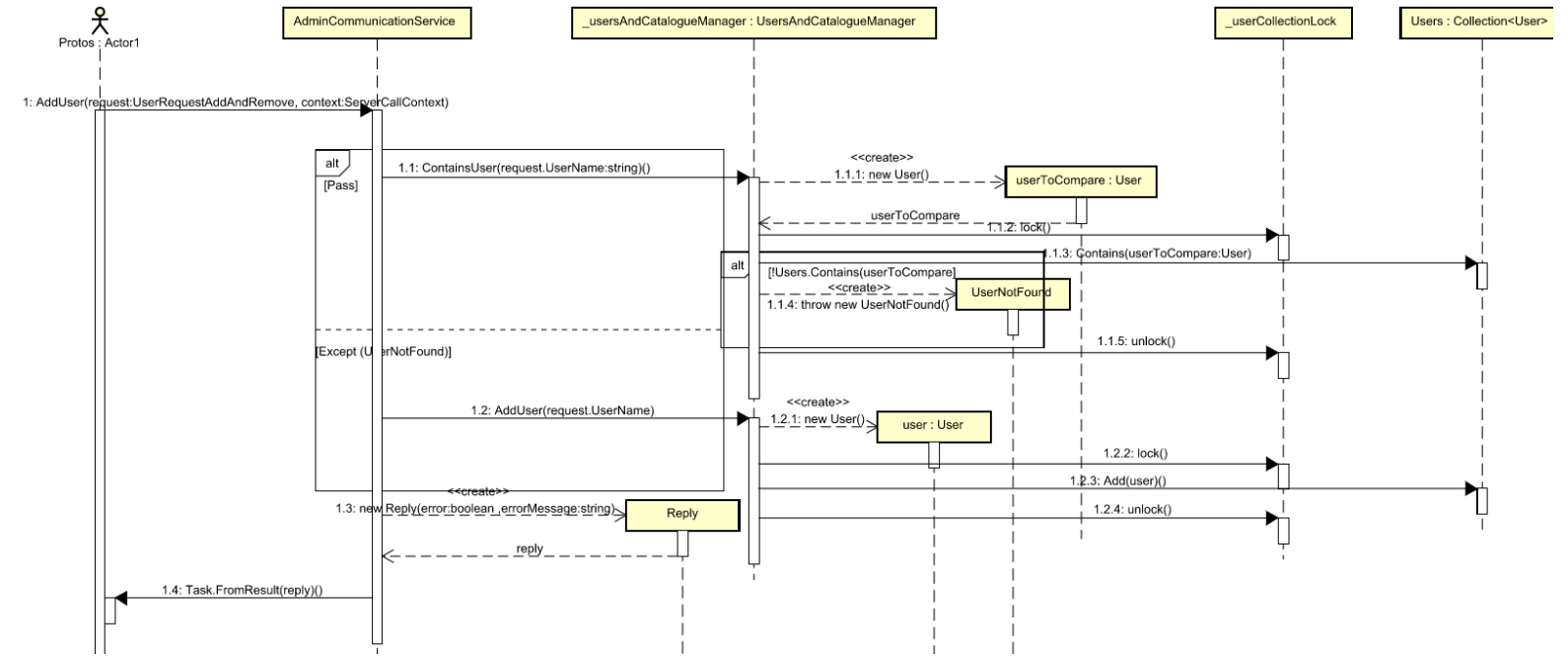
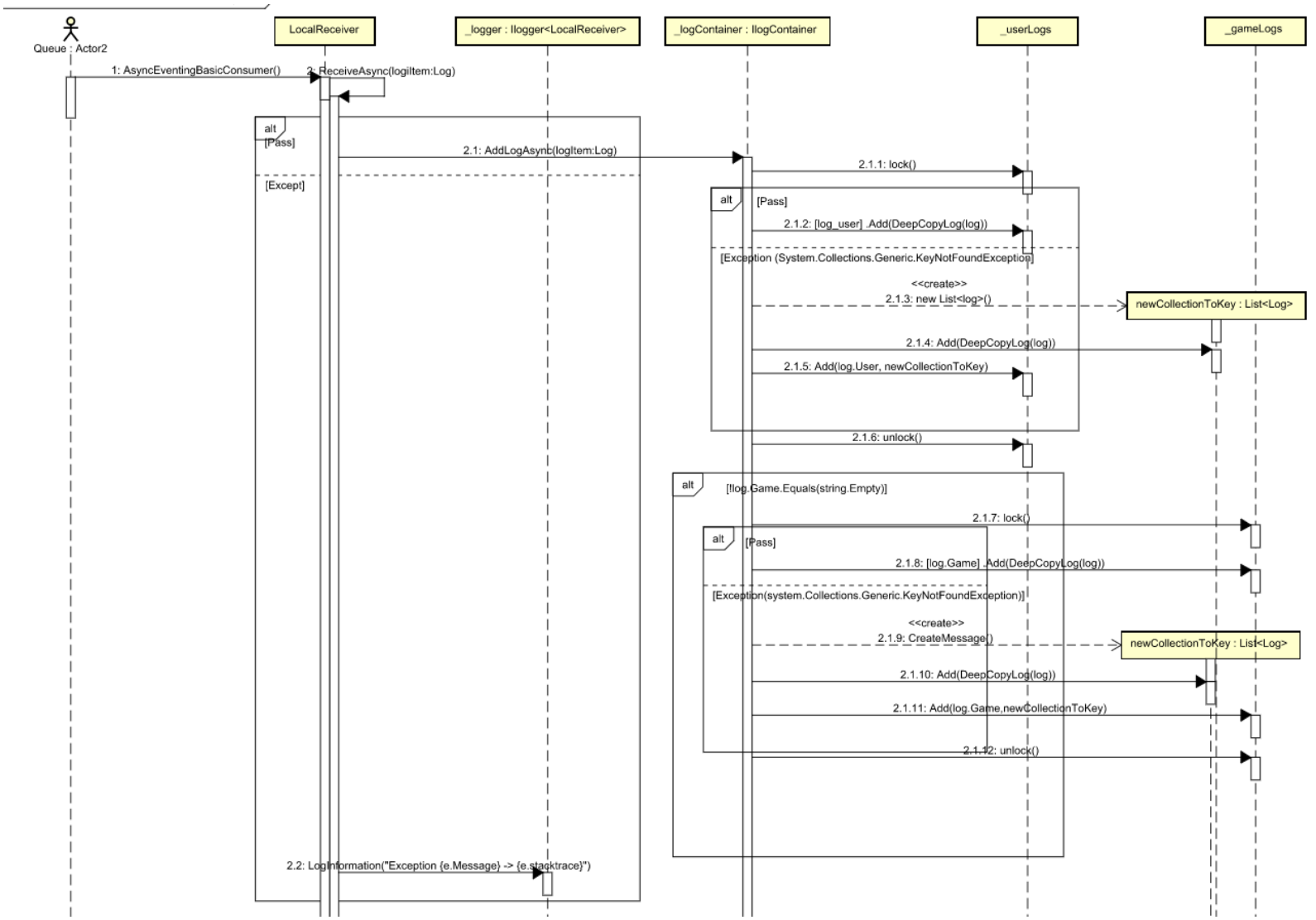
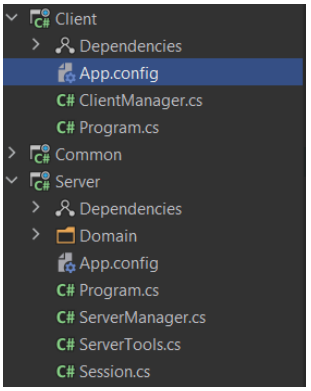


Diagrama de secuencia de consumo de un log a la cola de mensajes RabbitMQ

En este diagrama se ve todo lo que desata un mensaje al ser recibido por ServerLogs. Desde el evento consumir asincrónico hasta el guardado del log en Logic.



Guía de Uso (actualizada):



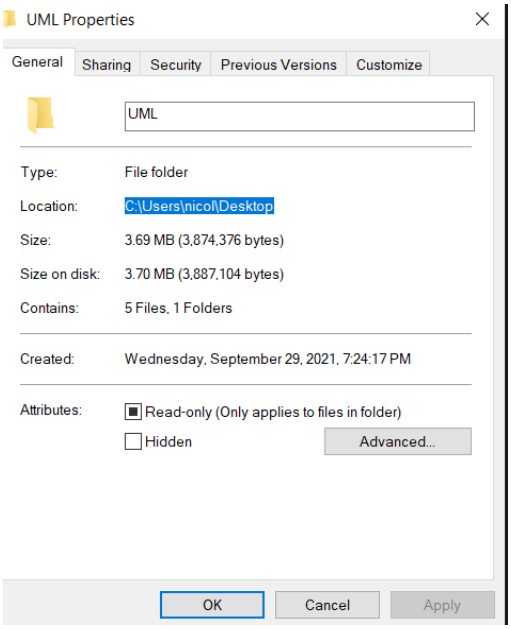
Lo primero que debemos hacer es localizar los archivos App.config tanto en el package Client, como en Server.

Buscamos en los archivos dentro del tag App.Config la key ServerIpAdress.

Cambiamos el valor de la key mencionada en ambos archivos por la ip de nuestra PC.

En windows podemos conocer la ip accediendo a la command Prompt e ingresando el comando ipconfig.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="ServerIpAddress" value="192.168.1.5"/>
        <add key="ServerPort" value="30000"/>
        <add key="DefaultPath" value="defaultImage.png"/>
        <add key="CoversPath"  value="C:\Users\nicol\Desktop\coversServer\"/>
    </appSettings>
</configuration>
```



Paso siguiente debemos cambiar el valor de la key “CoversPath” para establecer dónde queremos que se almacenen las carátulas de los juegos. Para esto vamos a cualquier elemento que tengamos en el escritorio, le damos click derecho y propiedades.
Se nos abrirá una pequeña pestaña con información acerca del archivo y la ruta en la que se encuentra, la copiamos y reemplazamos en el valor de la key. Una vez pegada agregamos ***\coversServer*** **y** ***\coversClient*** respectivamente.

Por último se debe levantar el servidor de colas de mensajes y añadir el valor del host y el nombre de la cola de mensajes a utilizar en los archivos App.config tanto de Server como de ServerLogs.

```
<add key="HostName" value="localhost"/>
<add key="QueueName" value="log_queue"/>
```

Una vez realizados los cambios, lo guardamos y simplemente ejecutamos primero el servidor, y luego el cliente. Luego de esto se guiará intuitivamente gracias a los mensajes desplegados en la consola.

Si se desea salir del sistema sin cerrar la consola se tiene la posibilidad de escribir exit en el cliente dentro del menú principal, o si se desean cerrar todas las conecciones, exit en la consola del servidor.

Anexo

Para poder visualizar la cola de mensajes, se debe levantar el servidor de RabbitMQ y acceder a http://localhost:15672/ con nombre de usuario guest y contraseña guest.