Universidad ORT Uruguay
Facultad de Ingeniería

Documentación Obligatorio 2
Programación de Redes

Federico Iglesias - 244831
Marcelo Pérez - 227362

2021

Índice general

1 Repositorio

Link al repositorio GitHub con el código fuente (rama master):

`https://github.com/mperezjodal/Redes-Obl1-Perez-Iglesias.git`

Como fue solicitado, se agregó a ambos docentes al repositorio.

2 Arquitectura

Nuestra solución cuenta con 7 paquetes, dos de ellos corresponden a las aplicaciones de consola de cliente y servidor, y el resto de ellos son librerías de clase con código común: *Domain*, *ConnectionUtils*, *ProtocolLibrary*, *DisplayUtils* y *FileStreamLibrary*.

Como su nombre pretende denotar, el paquete *Domain* declara las clases que serán usadas por el sistema: GameSystem, User, Game, Review, y algunas clases auxiliares como CustomEncoder que se utiliza para parsear los objetos y convertirlos en strings que serán enviados entre los componentes Cliente y Servidor.

ProtocolLibrary define la clase Header y las constantes que se utilizan para regular el envío de datos, como será explicado detalladamente más adelante. Aquí se definen también los CommandConstants que serán enviados en el header de cada envío de datos y funciones de codificación de información.

ConnectionUtils se utiliza para extraer métodos de envío y recepción de datos.

DisplayUtils contiene métodos para dialogar por consola con los usuarios de las aplicaciones, tanto Client como Server lo utilizan mostrar información y recibir inputs.

FileStreamLibrary es la librería encargada del envió de archivos entre cliente y servidor, utilizado para enviar las carátulas de los juegos.

3 Diseño de los componentes y decisiones de diseño

En cuanto al alcance de la aplicación, se ha cumplido con los requerimientos del sistema, permitiendo tener varios clientes conectados al servidor, que pueden realizar operaciones respetando la mutua exclusión en todo momento.

El dominio de nuestra solución cuenta con la clase User, que utilizan los clientes para conectarse al servidor y mediante la cual podrán adquirir juegos. El login está implementado a modo de permitir manejar concurrencia, y evita que dos usuarios tengan el mismo nombre. Los usuarios no son persistidos, es decir que una vez que se cierre la conexión con el servidor, se deberá crear otro usuario para volver a conectarse.

Luego, la clase Game contiene los atributos de los juegos y su lista de Reviews, que son objetos de otra clase que representa las calificaciones con un rating entre 1 y 10 y un comentario.

La clase GameSystem contiene las listas de usuarios, juegos y juegos en modificación, con más detalles a continuación.

También hay otras clases que tienen el fin de facilitar operaciones como las de codificación y decodificación de información, que permiten pasar los objetos a strings que serán enviados entre los componentes.

La aplicación esta diseñada para soportar la conexión de múltiples clientes y sus solicitudes de forma concurrente, para esto se utiliza la clase *Task* que permite manejar múltiples hilos. El server crea un hilo por cada conexión con un cliente.

En cuanto al manejo de la concurrencia en relación a los juegos del sistema, se decidió guardar en el Server una lista de juegos en modificación. Entonces, cuando un usuario desea modificar los datos de un juego, solo se le permite si el juego no se encuentra en dicha lista. En ese caso, antes de hacer los cambios se ingresa el juego a la lista de juegos en modificación y una vez finalizadas las modificaciones, se elimina el juego de la lista de modo que quede accesible para operaciones nuevamente.

La lista antedicha no solo evita que dos clientes modifiquen un mismo juego a la vez, sino que también evita que otro usuario elimine o ingrese una calificación de los juegos en modificación, ya que estas operaciones se harán sobre objetos que están siendo modificados. Asimismo, si un cliente está realizando ingresando una

calificación de un juego y el mismo es eliminado por otro cliente entre tanto se ingresa esta calificación, la calificación queda sin efecto alguno en el sistema y se retorna el menú principal.

Se utilizaron locks en las partes del código del Server que ejecutan el get, add, delete, modify sobre la lista de juegos, para evitar que estos se ejecuten a la vez desde distintos hilos de clients.

Otros chequeos del sistema incluyen el no poder ingresar dos veces a un mismo juego, el de no ingresar calificación de un juego que haya sido eliminado después empezar pero antes de terminar de ingresar la calificación, el permiso de modificar o eliminar usuarios por parte del Server únicamente cuando estos no tengan una sesión abierta, etc.

Cuando un usuario opta por ver la lista de juegos y una vez visualizándola se elimina uno de sus juegos, el usuario sigue viendo la misma lista, incluyendo los juegos eliminados, pero no se le permitirá hacer ninguna operación sobre ella y apenas vuelva a seleccionar la opción de ver la lista, esta estará actualizada.

4 Mecanismos de comunicación de los componentes

El intercambio de datos entre el cliente y el servidor se da mediante *TcpListener* y *TcpClient* utilizando un protocolo a medida.

Para el envío de los objetos entre el cliente y el servidor se implementaron métodos que los convierten en strings, para que luego nuestro paquete de protocolo pueda enviarlos correctamente a través de bytes. Estos métodos y sus reversiones se encuentran en las clases correspondientes del dominio.

Los clientes envían solicitudes al servidor y el mismo les responde siguiendo el protocolo:

1 byte para el Header que puede ser una request o response.

2 bytes para el Command, que es un código asociado a un pedido o respuesta.

4 bytes para el largo de los datos.

Datos con largo variable identificado anteriormente.

El protocolo antedicho fue el que consideramos que se ajustaba más a los requisitos del sistema.

A su vez, los juegos pueden tener una carátula, y para implementar el intercambio de archivos entre las partes se utilizaron los *NetworkStreams* de la clase *TcpListener* y *TcpClient*. Las carátulas pueden ser ingresadas cuando el server publica un juego, o cuando un cliente publica o modifica uno, y estas se almacenan en la carpeta del Server. Cuando el Client solicita ver el detalle de un juego se le envía la carátula desde el Server a través de un *NetworkStreams* y estas se almacenan en la carpeta Client.

Las carátulas pueden ser modificadas como cualquier otro atributo de los juegos, y también pueden haber varios juegos con la misma carátula sin que se generen conflictos.

En cuanto al manejo de errores, tanto cuando se hace *exit* desde el server como desde el Client, el sistema maneja diferentes exceptions. Si el Server ejecuta `exit` sin ningún cliente conectado, simplemente se cierra el *TcpListener*. En el caso de existir clientes conectados, cuando el cliente quiera intercambiar datos se lanza una SocketException que el sistema maneja y se le avisa al Client que el Server cerró la conexión.

5 Funcionamiento de la aplicación

El sistema cuenta con dos aplicaciones, un **Client** y un **Server**.

Por el lado del **Server** se despliega el Menú Principal que brinda **siete** opciones, junto a la posibilidad de cerrar la conexión mediante el ingreso de la palabra *exit*:

1. Ver juegos y detalles.
2. Publicar juego
3. Publicar calificación de un juego
4. Buscar juegos
5. Insertar usuario
6. Modificar usuario
7. Eliminar usuario

Para seleccionar una funcionalidad, el usuario ingresa el número correspondiente a la misma, de ingresar una opción incorrecta se retorna el Menú Principal.

En *Ver juegos y detalles*, se despliega la lista de juegos y se da la opción de ver el detalle de uno de ellos ingresando el nombre del mismo.

En *Publicar juego*, se permite publicar un juego con todos sus datos correspondientes (el nombre del juego debe ser único) en la lista de juegos del sistema. Si el usuario desea puede insertar una carátula para el juego, para esto, cuando la consola lo pida se debe ingresar una ruta de la foto que se quiere asociar.

A su vez, en *Publicar calificación de un juego* se despliega la lista de juegos y se selecciona el nombre del juego que se quiere calificar.

Además, el sistema permite una búsqueda filtrada sobre los juegos es la lista del sistema, los filtros puede ser los siguientes: Título, Calificación y Categoría. Se selecciona la opción por la cual se quiere filtrar y se despliegan los juegos filtrados.

En *Insertar usuario*, se permite realizar la alta de un usuario, siempre y cuando y ya no exista en el sistema.

En *Modificar usuario*, se permite modificar el nombre de usuario, mientras este no tenga una sesión abierta.

En *Eliminar usuario*, se permite realizar la baja de un usuario, siempre y cuando este no tenga una sesión abierta en el sistema.

Además, el servidor soporta la conexión de varios clientes en forma concurrente, que tienen permitido ejecutar acciones en el servidor mediante la aplicación del Client.

La **aplicación del Client** cuenta con un Menú Principal que brinda **ocho** opciones, como también la posibilidad de cerrar la conexión mediante el ingreso de la palabra *logout*:

1. Publicar juego
2. Modificar juego
3. Eliminar juego
4. Buscar juego
5. Calificar juego
6. Adquirir juego
7. Ver juegos adquiridos
8. Ver juegos y detalles

Para seleccionar una, el usuario ingresa el numero correspondiente a la funcionalidad, de ingresar una opción incorrecta se retorna el Menú Principal.

Modificar juego: despliega la lista de juegos y el usuario selecciona el nombre de aquel juego a modificar, luego se ingresan los nuevos datos o ENTER en los campos que no se quieran modificar.

Eliminar Juego: despliega la lista de juegos y el usuario selecciona el nombre de aquel juego a eliminar del sistema.

Adquirir juego: el usuario cliente puede adquirir un juego que se almacenará en su propia lista de juegos que luego podrá ver en *Ver juegos adquiridos*.

Las funcionalidades no explicadas en detalle tienen el mismo funcionamiento que sus pares en la aplicación del Servidor.

Una consideración para ambas aplicaciones es que una vez dentro de cualquier funcionalidad, se permite retornar al menú mediante la tecla ENTER.

6 Modificaciones Obligatorio 2

6.1 Tasks

Motivados por la eficiencia, se reemplazó el uso de Threads por el de Tasks. Este cambio impacta fuertemente en la eficiencia del sistema, ya que la creación de Threads en general es muy costosa para el sistema operativo y es mejor evitar su uso. En el caso del Server, que previamente generaba una nueva Thread por cada cliente que aceptaba, se sustituyó esta herramienta por Tasks para mejorar la implementación de paralelismo utilizando `async` y `await`.

6.2 Tcp Client/Listener

La comunicación entre Client y Server, previamente implementada con *Sockets* para el Obligatorio 1, fue reemplazada por *TcpClient* y *TcpListener*, utilizando un envío de datos asíncrono mediante *network streams*. Si bien en cuanto a eficiencia no existe diferencia entre estas clases, el cambio permitió que el código sea mas corto y entendible.

6.3 Async y await

La implementación de envío de datos fue modificada para cumplir con el patrón asincrónico, y consecuentemente los métodos async se propagaron hasta llegar al Main en el caso de Client, y HandleClient en el caso de Server. Asimismo, se invocan con await todas los métodos asincrónicos, cuyas firmas son siempre `async Task`.

6.4 Otros

Otros cambios incluyen el uso de un mensaje más adecuado para el Cliente cuando el Server cierra su conexión, la optimización del código y la implementación de alta, baja y modificación de usuarios del lado del Server.

En cuanto al protocolo propietario para la comunicación entre partes, se decidió no realizarle modificaciones.