Universidad ORT Uruguay

Facultad de Ingeniería Bernard Wand Polak

Programación de Redes

Documentación Obligatorio 2

Sebastian Zolkwer - 222841

Alex Piczenik - 231093

Fecha: 26.10.2021
Grupo: M6B

Índice

Link al repositorio

https://github.com/ObligatorioProgramacionRedes2.git

Introducción

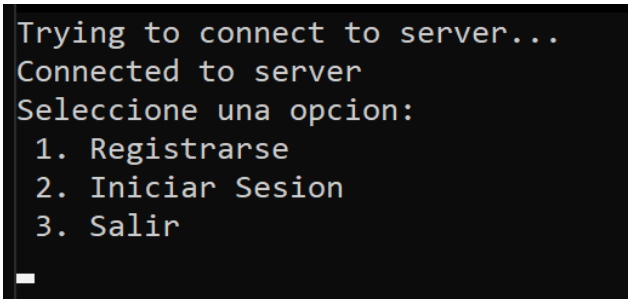
En el siguiente documento se detallan y justifican todos los cambios realizados a nivel de código con respecto a la entrega anterior. El objetivo fue implementar las nuevas funcionalidades requeridas manteniendo o incluso mejorando la calidad de código ya desarrollado, dejando el sistema lo más extensible posible de cara a la tercera entrega.

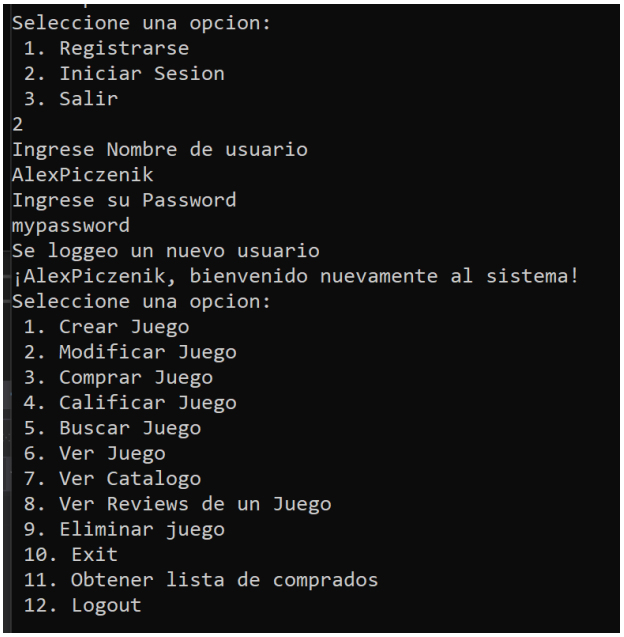
Login/Logout

Se implementó en el sistema la funcionalidad encargada de controlar el manejo de sesiones entre usuarios. De este modo, un mismo usuario podrá iniciar sesión y desloguearse manteniendo sus juegos comprados bajo su nombre y contraseña.

Flujo de funcionalidad:

Al iniciarse una nueva instancia de cliente, este tiene la posibilidad de registrarse desde cero como nuevo usuario en el sistema o directamente loguearse como uno ya creado. Para esta segunda opción se debe ingresar nombre de usuario tal cual este guardado en el sistema y la contraseña correspondiente al mismo.





El sistema se encarga de chequear que efectivamente exista un usuario con ese nombre y esa contraseña, y de ser así lo carga como usuario logueado accediendo automáticamente a la información del mismo.

Con respecto al dominio, para poder implementar la funcionalidad debimos crear una nueva clase llamada Cliente, donde cada cliente tiene los siguientes atributos:

-Name (nombre de usuario utilizado para iniciar sesión, único en el sistema)
-Password (contraseña de seguridad para iniciar sesión)
-Bought games (lista de juegos comprados por el usuario)
-Active (variable booleana que indica si el usuario está o no logueado en el sistema)

```
public class Client
{
    public List<Game> boughtGames;
    public string name;
    public string password;
    public bool active;
```

Una vez definida la clase cliente, creamos una lista estática de clientes en LogicServer para poder realizar correctamente la gestión de los mismos.

Por otro lado, una vez que el cliente inicio sesión, tiene disponible todo el menú de funcionalidades a realizar donde una de ellas es “Logout”.
De seleccionarla, el cliente actual dejará de ser el usuario logueado y el sistema volverá a desplegar el menú inicial solicitando a quien lo utilice registrarse o loguearse con sus credenciales.

A nivel de código, se agregaron dos nuevos casos al switch menu manejado desde LogicServer:

```
case 15:
    request = await Protocol.Protocol.RecieveAndDecodeVariableDataAsync(networkStream, header.GetDataLength());
    client = Logic.Register(request, clients);
    Logic.ActiveUser(client);
    await Protocol.Protocol.SendAndCodeAsync(networkStream, ProtocolMethods.Success,
        "Se creo un nuevo usuario", ProtocolMethods.Response);
    break;
case 16:
    request = await Protocol.Protocol.RecieveAndDecodeVariableDataAsync(networkStream, header.GetDataLength());
    client = Logic.Login(request, clients);
    Logic.ActiveUser(client);
    await Protocol.Protocol.SendAndCodeAsync(networkStream, ProtocolMethods.Success,
        "Se loggeo un nuevo usuario", ProtocolMethods.Response);
```

Para cada uno de los casos se recibe la request correspondiente enviada por el cliente, y se accede a un método de la lógica encargado de realizar la funcionalidad requerida en función de la request recibida por parámetro (Logic.Register, Logic.Login).

Como se puede observar, también se utiliza un método de la lógica para activar el usuario recién logueado y establecerlo como activo en el sistema. De este modo, nos aseguramos una mejor calidad de código además de permitirnos trabajar la mutua exclusión de la lista en un único lugar como explicaremos a continuación.

Cabe destacar que si bien los números en el switch en la entrega anterior se comportan como “números mágicos” difíciles de interpretar para cualquier persona externa al código (Imagen anterior: case 15, case 16…), para esta segunda entrega decidimos cambiarlos por constantes estáticas definidas en ProtocolMethods de forma de clarificar el código lo máximo posible. En este switch no fue necesario establecer el caso Default, ya que para cada una de las request enviadas desde el cliente, somos nosotros mismos quienes establecemos el número de método permitiéndonos evitar la programación defensiva en este aspecto.

```
switch (header.GetMethod())
{
    case ProtocolMethods.Create:
        request = await Protocol.
        response = Logic.Add(requ
        await Protocol.Protocol.S
        break;
    case ProtocolMethods.Update:
        request = await Protocol.
        response = Logic.Update(r
        await Protocol.Protocol.S
        break;
    case ProtocolMethods.Buy:
```

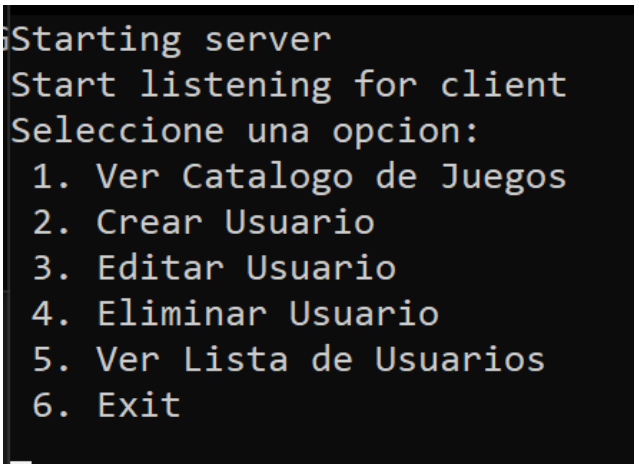
Lista de juegos comprados

Como lo solicitaba la letra, se agregó como funcionalidad la posibilidad de obtener lista de juegos comprados de un usuario. Los juegos mostrados son únicamente aquellos pertenecientes al sistema por lo que de existir un juego comprado que fue borrado del sistema posteriormente, el usuario no lo tendrá en su lista resultante.

```
Seleccione una opcion:
 1. Crear Juego
 2. Modificar Juego
 3. Comprar Juego
 4. Calificar Juego
 5. Buscar Juego
 6. Ver Juego
 7. Ver Catalogo
 8. Ver Reviews de un Juego
 9. Eliminar juego
 10. Exit
 11. Obtener lista de comprados
 12. Logout
11
Lista de juegos:
 -Titulo:Fifa Genero:Futbol
 -Titulo:Catan Genero:Estrategia
 -Titulo:Truco uruguayo Genero:Fluidez
```

ABM de usuarios desde el servidor

En el sistema se implementó la posibilidad de que un usuario supuesto “Admin” pueda realizar altas, bajas y modificaciones en los usuarios del mismo.



Como se mencionó anteriormente, cada usuario cuenta con un nombre y una contraseña por lo que para cualquiera sea la opción a realizar, se deben ingresar ambos campos requeridos. En caso de querer editar o eliminar, el sistema se encarga de verificar que el usuario pre existente efectivamente esté instanciado en el sistema.

Análogo a lo realizado en la funcionalidad Login/Logout se agregaron nuevos posibles casos al switch menú correspondiente:

```
case 2:
     CreateUser();
    break;
case 3:
    UpdateUser();
    break;
case 4:
     DeleteUser();
    break;
```

Desde cada uno de estos métodos auxiliares, se llama a un método de la lógica encargado de realizar la correspondiente funcionalidad:

```
private void DeleteUser()
{
    string name;
    string password;
    Console.WriteLine("Ingrese Nombre de usuario");
    name = Console.ReadLine();
    Console.WriteLine("Ingrese Password");
    password = Console.ReadLine();
    Console.WriteLine(LogicServer.DeleteUser(name, password));
}
```

Aclaraciones específicas:

- Al momento de eliminar un usuario, el sistema se asegura que el mismo no esté logueado en otra ventana de cliente, evitando la caída del sistema cuando el mismo desee realizar una funcionalidad. En el caso de que el Admin desee eliminar un usuario activo, el sistema no lo permite y lanza el siguiente mensaje:

```
if (client.active)
{
    throw new Exception("No es posible eliminar usuarios activos.\n");
}
```

- Para poder modificar los datos registrados de un usuario, se deben ingresar tanto el nombre de usuario como la contraseña anteriores y el nombre de usuario y contraseña actualizados. De querer modificar únicamente 1 de los atributos, se puede volver a ingresar el valor del atributo original como nuevo.

```
Seleccione una opcion:
 1. Ver Catalogo de Juegos
 2. Crear Usuario
 3. Editar Usuario
 4. Eliminar Usuario
 5. Ver Lista de Usuarios
 6. Exit
3
Ingrese Nombre de usuario anterior
Alex
Ingrese Password anterior
password
Ingrese Nombre de usuario nuevo
Alex
Ingrese Password nueva
nuevaPassword
Se ha modificado el usuario.
```

Mutua exclusión

En la primera entrega el sistema contaba con una única lista de games que podía ser modificada por las distintas funcionalidades a cargo del cliente. Con el fin de evitar

problemas en el manejo del recurso, se implementó mutua exclusión con locks tal como lo explicamos en la documentación anterior. En este caso, se implementó una nueva lista de Clientes, donde se mantienen todos los usuarios alguna vez inicializados en el sistema. Análogamente al caso anterior, implementamos locks de forma que en cada instancia que el código accede a la lista, esta se bloquea sin permitir que ningún otro cliente pueda modificarla hasta que se libere el recurso.

```
lock (clientsLock)
{
    if (clients.Any(c => c.name == name))
    {
        throw new Exception("Ya existe un usuario con ese nombre.\n");
    }
    Client client = new Client(name, password);
    clients.Add(client);
    return client;
}
```

Migración de Threads al modelo asincrónico de .NET

En el proceso de la migración se sustituye el uso de Threads (implementado en la primera entrega) por Tasks incluyendo un modelado del proyecto en base a los conceptos de async y await.
Los conceptos async y await permiten convertir los métodos en asíncronos, creando una máquina de estados logrando que los métodos en lugar de bloquear la ejecución, se mantengan en espera.
Para migrar el cambio en el código definimos los métodos con async y la llamada a los mismos con await, recordando que todo método async debe tener un await.
El tipo de retorno de cada uno de los métodos también fue modificado como consecuencia de lo anterior, para todos los casos donde se definió un método async, el retorno es de tipo Task<T> siendo T el tipo de retorno anterior. En caso de ser void se retorna únicamente Task. Además en lugar de utilizar las funciones del socket para enviar datos, se utilizaron las funciones del NetworkStream.

A continuación dejamos ejemplos de código antes y después de la modificación de Thread a método asíncrono:

Antes:

```
public static string RecieveAndDecodeVariableData(Socket socket, int length)
{
    byte[] value = ReceiveData(socket, length);
    string response = Encoding.UTF8.GetString(value);
    return response;
}
```

Ahora:

```
public async static Task<string> RecieveAndDecodeVariableDataAsync(NetworkStream networkStream, int length)
{
    byte[] value = await ReceiveDataAsync(networkStream, length);
    string response = Encoding.UTF8.GetString(value);
    return response;
}
```

Otro ejemplo es el siguiente:
En el metodo SendAndCodeAsync

Antes:

```
public static void SendAndCode(Socket socket, int method, String message, string direction)
{

    Header header = new Header(direction, method, message.Length);
    byte[] data = header.GetRequest();

    byte[] values = Encoding.UTF8.GetBytes(message);


    SendSpecificMessage(socket, data);
    SendSpecificMessage(socket, values);
}
```

Ahora

```
31 referencias | Sebastian Zolkwer, Hace 4 días | 1 autor, 1 cambio
public async static Task SendAndCodeAsync(NetworkStream networkStream, int method, String message, string direction)
{

    Header header = new Header(direction, method, message.Length);
    byte[] data = header.GetRequest();

    byte[] values = Encoding.UTF8.GetBytes(message);


    await SendSpecificMessageAsync(networkStream, data);
    await SendSpecificMessageAsync(networkStream, values);
}
```

El proceso de cambios para async/await lo comenzamos en la función ReadData y WriteData del FileStreamHandler y esto se propaga por nuestro código.

Es importante aclarar que no se cambiaron todos los métodos a asíncronos, solamente los que implican input/output y alto procesamiento de cpu, además de los métodos que estos arrastran.

Cambio de thread a task:

```
while (runServer)
{
    TcpClient tcpClient = tcpListener.AcceptTcpClient();
    NetworkStream _networkStream = tcpClient.GetStream();
    Task.Run(async () => await LogicServer.ClientHandlerAsync(_networkStream));
}
```

```
Thread acceptClients = new Thread(() => AcceptClients(_tcpListener));
acceptClients.IsBackground = true;
acceptClients.Start();
```

Migración de Sockets a Tcp Listener y Tcp Client

Del lado del servidor realizamos los siguientes cambios
Servidor queda escuchando y acepta conexiones

Antes:

```
Socket serverSocket = new Socket(
    AddressFamily.InterNetwork,
    SocketType.Stream,
    ProtocolType.Tcp);
IPEndPoint serverIpEndPoint = new IPEndPoint(
    IPAddress.Parse(GetIPAddress()),
    GetPort());
serverSocket.Bind(serverIpEndPoint);
serverSocket.Listen(GetBackLog());

private void AcceptClients(Socket socket)
{
    while (runServer)
    {
        Socket clientSocket = socket.Accept();
        Thread client = new Thread(() => LogicServer.ClientHandler(clientSocket));
        client.IsBackground = true;
        client.Start();
    }
}
```

Ahora:

```
IPEndPoint serverIpEndPoint = new IPEndPoint(
   IPAddress.Parse(GetIPAddress()),
   GetPort());
TcpListener _tcpListener = new TcpListener(serverIpEndPoint);
_tcpListener.Start(GetBackLog());
```

```
private void AcceptClients(TcpListener tcpListener)
{
    while (runServer)
    {
        TcpClient tcpClient = tcpListener.AcceptTcpClient();
        NetworkStream _networkStream = tcpClient.GetStream();
        Task.Run(async () => await LogicServer.ClientHandlerAsync(_networkStream));
    }
}
```

Del lado de cliente:
Conectarse
Antes:

```
Socket clientSocket = new Socket(
    AddressFamily.InterNetwork,
    SocketType.Stream,
    ProtocolType.Tcp);
IPEndPoint clientEndPoint = new IPEndPoint(
    IPAddress.Parse(GetIPAddressClient()),
    GetPortClient());
clientSocket.Bind(clientEndPoint);
Console.WriteLine("Trying to connect to server...");
IPEndPoint serverEndPoint = new IPEndPoint(
    IPAddress.Parse(GetIPAddressServer()),
    GetPortServer());
if (Directory.Exists(Directory.GetCurrentDirectory() + @"\CaratulasClient"))
{
    Directory.Delete("CaratulasClient", true);

}
Directory.CreateDirectory(Directory.GetCurrentDirectory() + @"\CaratulasClient");
clientSocket.Connect(serverEndPoint);
```

Ahora:

```
IPEndPoint clientEndPoint = new IPEndPoint(
    IPAddress.Parse(GetIPAddressClient()),
    GetPortClient());
TcpClient tcpClient = new TcpClient(clientEndPoint);
Console.WriteLine("Trying to connect to server...");
if (Directory.Exists(Directory.GetCurrentDirectory() + @"\CaratulasClient"))
{
    Directory.Delete("CaratulasClient", true);

}
Directory.CreateDirectory(Directory.GetCurrentDirectory() + @"\CaratulasClient");
await tcpClient.ConnectAsync(IPAddress.Parse(GetIPAddressServer()), GetPortServer());
```

Cerrar la conexión:

Antes:

```
socket.Shutdown(SocketShutdown.Both);
socket.Close();
```

Ahora:

```
networkStream.Close();
tcpClient.Close();
```

Enviar datos

Antes:

```
int totalDataSent = 0;

while (totalDataSent < dataLength.Length)
{
    try
    {
        int sent = socket.Send(dataLength, offset: totalDataSent, size: dataLength.Length - totalDataSent, SocketFlags.None);
        if (sent == 0)
        {
            throw new SocketException();
        }
        totalDataSent += sent;
    } catch(Exception ex) { }
}
```

Ahora:

```
await networkStream.WriteAsync(dataLength, 0, dataLength.Length);
```

Recibir datos

Antes:

```
int offset = 0;
byte[] response = new byte[length];
while (offset < length)
{
    int received = socket.Receive(response, offset, length - offset, SocketFlags.None);
    if (received == 0)
    {
        throw new SocketException();
    }
    offset += received;
}
return response;
```

Ahora:

```
int offset = 0;
byte[] response = new byte[length];
while (offset < length)
{
    int received = await networkStream.ReadAsync(response, offset, length - offset);
    if (received == 0)
    {
        throw new SocketException();
    }
    offset += received;
}
return response;
```

Por último, en el momento que el servidor se desconecte y el cliente quiera realizar alguna funcionalidad, se va a lanzar la IOException que va a ser capturada y se va a ser la conexión.

```
catch (IOException)
{
    Console.WriteLine("Se cerro la conexion del servidor");
}
networkStream.Close();
tcpClient.Close();
```

Aclaración: Algunas de las capturas de código se encuentran con fondo blanco y otras con fondo oscuro debido a que ambos miembros del equipo tienen diferente formato de VS, al haber participado los dos por igual en el desarrollo del proyecto y de la documentación, nos pareció adecuado agregar capturas de ambos casos.