

Universidad ORT Uruguay
Facultad de Ingeniería

Programación de redes
Obligatorio

Entregado como requisito de la materia Programación
de redes

Danilo Biladóniga - 231749
Romina Giaccio Piaggio - 206127

29 de Septiembre de 2022

Resumen

En este obligatorio de la materia “comunicación y liderazgo” se planteó construir un sistema que manejara usuarios y perfiles, de esta forma manejando el paralelismo, concurrencia y sincronización de Threads, así como el manejo de archivos. Además de crear dos aplicaciones, una cliente, otra servidor y un protocolo para la comunicación entre las mismas.

Índice

Índice	3
Alcance de la aplicación	4
1.1 Servidor	4
1.1.1 Requerimientos entregados:	4
1.1.2 Funcionalidades adicionales:	4
1.1.3 Suposiciones	4
1.2 Cliente	4
1.2.1 Requerimientos entregados:	4
Descripción de la arquitectura	5
2.1 Arquitectura de archivos	5
Diseño detallado de componentes	8
3.1 Especificación del protocolo	8
3.1.1 Descripción	8
3.1.2 Manejo de errores	8
3.1.3 Mecanismos de concurrencia	9
Funcionamiento de la aplicación	9
Menú principal:	9
Menú del cliente	9
Dar de alta usuario	10
Login Usuario	12
Menú Usuario	13
Dar de alta Perfil	13
Actualizar foto de Perfil	15
Descargar foto de Perfil	16
Consultar Perfiles	17
Filtro por habilidad	17
Filtro por palabra clave	19
Filtrar por id	19
Mensajes	21
Enviar mensaje	21
Consulta histórica	22
Mensajes sin leer	23
Ejecutando varios clientes:	23
Cambio de ips y puertos	25
7. Consideraciones	25

1. Alcance de la aplicación

1.1 Servidor

1.1.1 Requerimientos entregados:

- **SRF1.** Aceptar pedidos de conexión de un cliente. El servidor permite aceptar pedidos de conexión de varios clientes a la vez.
- **SRF2.** Dar de alta a varios usuarios. El sistema permite dar de alta a varios usuarios en el sistema.
- **SRF3.** Crear Perfil de trabajo de usuario. El sistema permite crear un perfil de trabajo para el usuario, el mismo incluye una lista de habilidades y una descripción.
- **SRF4** Asociar una foto al perfil de trabajo. El sistema permite subir una foto y asociarla al perfil de trabajo de un usuario.
- **SRF5.** Consultar perfiles existentes. Los usuarios pueden buscar perfiles existentes, por habilidades, por palabras claves y por id del usuario. También permite descargar la imagen asociada al perfil, en caso de existir la misma.
- **SRF6.** Enviar y recibir mensajes. El sistema permite que un usuario envíe mensajes a otro, y que el usuario receptor chequee sus mensajes sin leer, así como también revisar su historial de mensajes.
- **SRF7.** Configuración. Se pueden modificar los puertos e ip utilizados por el servidor sin necesidad de recompilar el proyecto.

1.1.2 Funcionalidades adicionales:

- El servidor provee un sistema de logs el cual permite ver las “tramas” en la consola enviadas entre el cliente y el servidor.

1.1.3 Suposiciones

- Cada usuario tiene un único perfil de trabajo

1.2 Cliente

1.2.1 Requerimientos entregados:

- **CRF1.** Conectarse y desconectarse al servidor. El cliente es capaz de conectarse y desconectarse del servidor.
- **CRF2.** Alta de usuario. Se permite dar de alta a uno o varios usuarios para que los mismos puedan acceder al sistema.

- **CRF3.** Alta de Perfil de trabajo. Un usuario puede dar de alta a su perfil de trabajo.
- **CRF4.** Asociar foto a perfil de trabajo. El usuario puede actualizar su perfil de trabajo adjuntando una foto.
- **CRF5.** Consultar perfiles existentes. El usuario mediante un menú de opciones puede filtrar perfiles ya sea por habilidades o palabras a buscar dentro de los perfiles.
- **CRF6.** Consultar un perfil específico. El usuario puede solicitar un perfil específico, id del usuario a buscar.
- **CRF7.** Enviar y recibir mensajes. El sistema brinda la posibilidad de enviar mensajes a otros usuarios, así como también consultar los mensajes recibidos.
- **CRF8.** Configuración. Se pueden modificar la ip y puerto utilizados por el cliente sin necesidad de recompilar el proyecto.

2. Descripción de la arquitectura

La arquitectura presentada en este obligatorio es una arquitectura cliente-servidor.

La misma consiste en dos entidades, por un lado el servidor, el cual se encarga de recibir solicitudes de los diferentes clientes y manejar las mismas enviando respuestas independientes a cada solicitud. Y por otro lado las entidades clientes, que pueden ser más de uno, los cuales envían diferentes solicitudes hacia el servidor y esperan una respuesta del mismo.

Por otro lado, la versión de .NET utilizada en el proyecto es la 6.0.

2.1 Arquitectura de archivos

LkdinSystem

[Proyecto] ClienteApp

[Archivo de configuración] App.config

- Permite cambiar ip y puerto de cliente y servidor sin necesidad de recompilar el proyecto

[Clase] Program.cs

- Contiene todos los menús que se presentan a los clientes, así como las llamadas a cada método/opción que el cliente seleccione

[Domain]

[Clase] Messages.cs

- Contiene los atributos que componen a los mensajes dentro de nuestro sistema
 - Id, identificador de cada mensaje
 - SenderEmail, email del usuario que envía el mensaje
 - ReceiverEmail, email del usuario destino del mensaje

- Text, mensaje que ha sido enviado
- CurrentState, que guarda strings "Readed" para mensajes leídos o "NotReaded" para mensajes no leídos
- Contiene los métodos "ToString", para convertir un objeto mensaje a un string el cual a ser guardado dentro de nuestro sistema de archivos y "ToEntity", para convertir un string a un objeto Message

[Clase] User.cs

- Contiene los atributos que componen a los usuarios dentro de nuestro sistema
 - Id, identificador de cada usuario
 - Name, nombre del usuario
 - Email, email del usuario
 - CurrentState, que guarda strings "Logged" para usuarios logeados o "NotLogged" usuarios no logeados
- Contiene los métodos "ToString", para convertir un objeto usuario a un string el cual a ser guardado dentro de nuestro sistema de archivos y "ToEntity", para convertir un string a un objeto User

[Clase] UserProfile.cs

- Contiene los atributos que componen a los perfiles dentro de nuestro sistema
 - UserId, identificador de cada usuario
 - Description, descripción
 - Abilities, lista de habilidades del perfil
 - Image, ruta de la imagen del perfil
- Contiene los métodos "ToString", para convertir un objeto perfil a un string el cual a ser guardado dentro de nuestro sistema de archivos y "ToEntity", para convertir un string a un objeto UserProfile

[Proyecto] Enum

[Clase] SpecialChars.cs

- Contiene caracteres usados para la separación de mensajes

[Proyecto] Protocol

[Carpeta] Commands

[Clase] ClientCommands.cs

- Contiene los comandos de comunicación entre sockets del lado del cliente

[Clase] CommandsHandler.cs

- Contiene un map de key-value, donde las keys son los comandos y el value es un Delegate, el cual nos permite ejecutar diferentes métodos según el comando que se reciba

[Clase] ServerCommands.cs

- Contiene los comandos de acceso a archivos y operaciones del lado del servidor

[Carpeta] Enum

[Clase] EmptyStatesMessages.cs

- Contiene mensajes para cuando no hay perfiles o mensajes

[Clase] States.cs

- Contiene los estados 200 y 400 guardados en un enum

[Clase] Constants.cs

- Contiene variables estáticas para usar posteriormente en las funcionalidades de los sockets, por ejemplo los tamaños de la parte fija de las tramas y para transferencia de archivos

[Clase] FileCommsHandler.cs

- Contiene métodos para el manejo de streams

[Clase] FileDatabaseManager.cs

- Contiene operaciones relacionadas con la lectura o escritura archivos, que luego son usadas en los comandos del servidor

[Clase] FileHandler.cs

- Contiene operaciones para el manejo de archivos, validar su existencia, nombre y tamaño

[Clase] FileOperations.cs

- Posee un método para calcular las partes del archivo

[Clase] FileStreamHandler.cs

- Contiene métodos para manejar el FileStream, leyendo, escribiendo o vaciar archivos

[Clase] ServerConfig.cs

- Contiene variables estáticas para referenciar a las configuraciones de los archivos App.config

[Clase] SocketHelper.cs

- Clase para dividir un poco más la estructura y facilitar enviar o recibir datos mediante los sockets

[Clase] TransferSegmentManager.cs

- Posee el objeto SegmentDataObject que nos facilita leer el comando, estado, largo de datos o datos de una trama, y principalmente manejo de tramas

[Proyecto] ServerApp

[Clase] ClientHandler.cs

- Contiene el método Handler para manejar el comportamiento de enviar y recibir datos de y hacia los clientes dentro de nuestro sistema

[Clase] Program.cs

- Contiene el método Main para especificar las configuraciones iniciales para inicializar el socket del servidor y aceptar nuevos clientes

[Proyecto] Utils

[Clase] ConversionHandler.cs

- Contiene métodos para hacer conversiones entre tipos de datos

[Clase] NumberConversions.cs

- Contiene un método para agregar ceros a la izquierda de un número y devolverlo en forma de string

[Clase] RandomData.cs

- Contiene un método para generar string randoms, en nuestro caso lo hemos usado para generar IDs

[Clase] SettingsManager.cs

- Contiene un método para leer la configuración de nuestros archivos en App.config

3. Diseño detallado de componentes

3.1 Especificación del protocolo

3.1.1 Descripción

Nuestro protocolo consta de 2 partes principales, las mismas de tipo string, una parte fija de 13 bytes y otra dinámica. La parte fija se divide en 3 partes, comandos/instrucciones, estado, largo del mensaje a enviar.

Los comandos/instrucciones se definen con 2 bytes de largo, siendo números entre 0 y 99 con acarreo de ceros a la izquierda, el mismo nos permite identificar la instrucción que debe ejecutar el servidor y mandar la correspondiente respuesta al cliente.

La parte del estado consiste en 3 bytes de los cuales actualmente solo se mandan dos valores, “200” indicando que la instrucción se ejecutó correctamente y “400” indicando que ocurrió un error en la misma.

Por último la parte que indica el largo del mensaje contiene 8 bytes y decidimos usar esta cantidad debido que al momento de consultar perfiles, en caso de haber muchos no se podía consultar más de 400 sin que se tuviera que mandar en mensaje en varias “tramas”.

Nombre del campo	CMD	ESTADO	LARGO	DATOS
Valores	0-99	200 o 400	Entero	Variable
Largo	2	3	8	Variable

La parte dinámica contiene el mensaje a enviar, dentro de la misma se utilizan caracteres especiales para identificar divisiones entre los datos a enviar, en nuestro caso por ejemplo, para dividir entidades dentro de nuestro dominio usamos “\n” indicando el final de cada fila, “/#” entre cada propiedad de la misma y “/.” para dividir valores en caso de que una propiedad sea un array.

3.1.2 Manejo de errores

Para el manejo de errores utilizamos estructuras try catch, tanto dentro del servidor como dentro del cliente.

Principalmente los mismos son utilizados en los menús del cliente, así como también en los métodos correspondientes a cada comando que ejecuta el servidor, capturando de esta forma errores al acceder a archivos o errores hechos por nosotros al validar la existencia de las distintas entidades del dominio.

3.1.3 Mecanismos de concurrencia

Hemos utilizado la concurrencia entre los diferentes hilos, correspondientes a la conexión entre los clientes y el servidor, para leer o escribir los distintos archivos que componen nuestro sistema.

En específico hemos usado la sentencia de bloqueo “lock”, la cual nos permite garantizar mutua exclusión mediante el bloqueo de una instancia y liberando la misma una vez se haya completado la tarea. Mientras el bloqueo está activo, el hilo bloqueante puede liberar el lock o volverlo a adquirir, por otro lado, los demás hilos permanecerán en espera hasta que el mismo sea liberado.

4. Funcionamiento de la aplicación

Menú principal:

Se espera observar el menú de la Imagen1 al iniciar un cliente. Donde al ingresar (1+Enter), se establece la conexión cliente-servidor con la configuración correspondiente y al ingresar (2+Enter) se finalizan las acciones del mismo:

```
Puerto: 0 IP: 127.0.0.1 Conectado: False
-----
# START MENU #
-----
1 CONECTAR A SERVIDOR
2 SALIR DEL CLIENTE
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.
```

Imagen1 (Cliente)

Menú del cliente

Al establecer la conexión cliente-servidor aparece el menú del cliente, en el mismo se tiene al ingresar (1+Enter) para loguearse a un usuario ya registrado o al ingresar (2+Enter) para crear un nuevo usuario (registro). También es posible volver al menú anterior al ingresar (3+Enter).

Se puede observar que al establecer la conexión cliente-servidor el indicador “Conectado” en la parte superior de la ventana cambia de False a True, además se muestra el puerto e ip seleccionados para la conexión (Imagen2). También se muestra en la consola del servidor la conexión establecida (Imagen3).

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# MENU CLIENTE #
-----
1 LOGIN
2 DAR DE ALTA USUARIO
3 EXIT
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.
```

Imagen2 (Cliente)

```
Server initialize...
New client 1 accepted
Attention Routine
```

Imagen3 (Cliente)

Dar de alta usuario

Para dar de alta un usuario se solicitará ingresar: nombre, apellido y email de forma consecutiva, no se aceptarán textos vacíos (no se valida formato email).

(Imagen4,5 y 6)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# CREAR USUARIO #
-----
Ingresar nombre de usuario:
```

Imagen4 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# CREAR USUARIO #
-----
Ingresar nombre de usuario:
Pamela
Ingresar apellido:
```

Imagen5 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# CREAR USUARIO #
-----
Ingresar nombre de usuario:
Pamela
Ingresar apellido:
Lebrin
Ingresar email:
```

Imagen6 (Cliente)

Al final se consultará si se desea dar de alta el usuario, pudiendo ingresar “A” para confirmar la acción o “X” (cualquier otra letra) en caso de que no se desee realizar el registro. Si se ingresó (A+Enter) aparecerá la confirmación del registro de usuario. (Imagen7)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# CREAR USUARIO #
-----
Ingresar nombre de usuario:
Pamela
Ingresar apellido:
Lebrin
Ingresar email:
pl@gmail.com
Ingresar A para aceptar o C para cancelar.
```

Imagen7 (Cliente)

Si se ingresó (A+Enter) aparecerá la confirmación del registro de usuario, se debe ingresar (X+Enter) para volver al Menú de Cliente.(Imagen8)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# Usuario se creo exitosamente. #
-----
Ingresar X para continuar.
```

Imagen8 (Cliente)

En la consola del servidor veremos como se confirma la creación del usuario: (Imagen9)

```
<- Client: 1 - Instruction: 01 - Status: 200 - Message: bdaf538a-f85c-4d10-9c85-5b0ef1e58e63/#Pamela/#pl@gmail.com/#NotL
ogged
-> Client: 1 - Instruction: 01 - Status: 200 - Message: Usuario guardado
```

Imagen9 (Servidor)

Login Usuario

El login de usuario solicitar ingresar un email , se validará ingresar un texto no vacío pero no el formato del mismo, debe ser un texto. (Imagen10)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# LOGIN #
-----
Ingresar email de usuario:
```

Imagen10 (Cliente)

Si se intenta loguear con un email no registrado se mostrará un mensaje de error y se dará la opción de ingresar (X+Enter) para cancelar la acción y volver al menú del cliente o (Y+Enter) para volver a intentar, esta última acción puede realizarse ingresando cualquier letra diferente de X. (Imagen11)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# LOGIN #
-----
Ingresar email de usuario:
romina@gmail.com
ERROR: El usuario es incorrecto o no existe, intente nuevamente.
Ingresar X para cancelar, Y para reintentar:
```

Imagen11 (Cliente)

En la consola del servidor podremos corroborar que el usuario no existe y no está logueado. (Imagen12)

```
<- Client: 1 - Instruction: 10 - Status: 200 - Message: /#/#romina/#NotLogged
-> Client: 1 - Instruction: 10 - Status: 400 - Message: No existe el usuario
```

Imagen12 (Servidor)

Si se ingresa un correo de un usuario registrado, el usuario se loguea y puede trabajar con el “Menú Usuario”. Además podremos ver el correo del usuario logueado en la parte superior derecha de la consola.(Imagen13)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# MENU USUARIO #
-----
1 DAR DE ALTA PERFIL
2 ACTUALIZAR FOTO DE PERFIL
3 DESCARGAR FOTO DE PERFIL
4 CONSULTAR PERFILES
5 MENSAJES
6 LOGOUT
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.

```

Imagen13 (Cliente)

En la consola del servidor podremos corroborar que el usuario se logueo correctamente viendo el estado #Logged. (Imagen14)

```

<- Client: 1 - Instruction: 10 - Status: 200 - Message: /#/#pl@gmail.com/#NotLogged
-> Client: 1 - Instruction: 10 - Status: 200 - Message: bdaf538a-f85c-4d10-9c85-5b0ef1e58e63/#Pamela/#pl@gmail.com/#Logged

```

Imagen14 (Servidor)

Menú Usuario

El menú usuario permite crear un perfil para el usuario logueado ingresando (1+Enter), actualizar la foto del perfil del usuario logueado ingresando (2+Enter), descargar la foto de perfil actual ingresando (3+Enter), ingresar al menú de consulta de perfiles ingresando (4+Enter) e ingresar al menú de consulta de mensajes ingresando (5+Enter). Por último permite volver al menú de Cliente ingresando (6+Enter) y se desloguea.(Imagen15)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# MENU USUARIO #
-----
1 DAR DE ALTA PERFIL
2 ACTUALIZAR FOTO DE PERFIL
3 DESCARGAR FOTO DE PERFIL
4 CONSULTAR PERFILES
5 MENSAJES
6 LOGOUT
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.

```

Imagen15 (Cliente)

Dar de alta Perfil

Para dar de alta un perfil se solicitará ingresar: descripción, y una habilidad como mínimo, luego de ingresar la primera habilidad se puede ingresar una lista de habilidades ingresando una a la vez. (Imagen16, 17 y 18)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
```

```
-----  
# CREAR PERFIL #  
-----
```

```
Ingresa descripción de usuario:
```

Imagen16 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
```

```
-----  
# CREAR PERFIL #  
-----
```

```
Ingresa descripción de usuario:
```

```
Desc1.
```

```
Ingresa habilidad al perfil:
```

Imagen17 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
```

```
-----  
# CREAR PERFIL #  
-----
```

```
Ingresa descripción de usuario:
```

```
Desc1.
```

```
Ingresa habilidad al perfil:
```

```
Hab1.
```

```
Ingresa Y para agregar más habilidades o N para continuar:
```

Imagen18 (Cliente)

Luego de ingresar la habilidad se consulta si se desea ingresar más habilidades, para aceptar debe ingresar (Y+Enter), ingresar cualquier otra letra se considerará como cancelar y se pasará a enviar el perfil al servidor para ser creado. (Imagen 19)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
```

```
-----  
# CREAR PERFIL #  
-----
```

```
Ingresa descripción de usuario:
```

```
Desc1.
```

```
Ingresa habilidad al perfil:
```

```
Hab1.
```

```
Ingresa Y para agregar más habilidades o N para continuar:
```

```
Y
```

```
Ingresa habilidad al perfil:
```

```
Hab2.
```

```
Ingresa Y para agregar más habilidades o N para continuar:
```

Imagen19 (Cliente)

Una vez aceptado el paso anterior se mostrará un mensaje que indica la creación correcta del perfil. (Imagen 20)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# Perfil creado correctamente. #
-----
Ingresar X para continuar.
```

Imagen20 (Cliente)

En la consola del servidor veremos como el perfil y sus datos fueron guardados correctamente: (Imagen21)

```
<- Client: 1 - Instruction: 02 - Status: 200 - Message: bdaf538a-f85c-4d10-9c85-5b0ef1e58e63/#Desc1./#Hab1./#Hab2./#
-> Client: 1 - Instruction: 02 - Status: 200 - Message: Perfil guardado correctamente
```

Imagen21 (Servidor)

Actualizar foto de Perfil

Para actualizar la foto de perfil se solicitará introducir el path de la misma conteniendo la dirección de la imagen en la máquina y luego debe dar a Enter. (Imagen22 y 23)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# AGREAR FOTO DE PERFIL #
-----
Ingresar foto de perfil.
```

Imagen22 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# AGREAR FOTO DE PERFIL #
-----
Ingresar foto de perfil.
C:\Users\Public\Images\fp1.png
```

Imagen23 (Cliente)

Si el path es correcto (es una imagen real en la máquina) aparecerá un mensaje indicando que la foto se guardó correctamente, la misma podrá ser chequeada en la carpeta del proyecto : "...ServerApp\bin\Debug\net6.0". (Imagen24)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# Foto guardada. #
-----
Ingresar X para continuar.
```

Imagen24 (Cliente)

En la consola del servidor se podrá ver como este recibe el archivo y actualiza el perfil: (Imagen25)

```
Reading file
Profile image received
-> Client: 1 - Instruction: 03 - Status: 200 - Message: Perfil actualizado
```

Imagen25 (Servidor)

Descargar foto de Perfil

Para descargar una foto de perfil, se solicitará el id del perfil del cual se desea descargar la foto. (Imagen26 y 27)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: test
-----
# DESCARGAR FOTO DE PERFIL #
-----
Ingresar el id del perfil que posee la foto a descargar.
```

Imagen26 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: test
-----
# DESCARGAR FOTO DE PERFIL #
-----
Ingresar el id del perfil que posee la foto a descargar.
cf9ddf37-58ca-4354-921b-00106f2d88a2_
```

Imagen27 (Cliente)

Si el path a descargar es correcto se indicará que la foto fue descargada correctamente, la misma podrá ser chequeada en la carpeta del proyecto: "...\\ClientApp\\bin\\Debug\\net6.0". (Imagen28)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# Foto descargada. #
-----
Ingresar X para continuar.
```

Imagen28 (Cliente)

Una vez terminada la transferencia se podrá ver en la consola del servidor que la imagen fue enviada. (Imagen29)


```
<- Client: 1 - Instruction: 09 - Status: 200 - Message: /#/#C:\Users\Public\Images\fp1.png
Sending image...
Image sended
```

Imagen29 (Server)

Si se ingresa un path para descargar una foto de perfil que no existe se obtendrá el siguiente aviso: (Imagen30)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: test
-----
# DESCARGAR FOTO DE PERFIL #
-----
Ingresar el id del perfil que posee la foto a descargar.
cf9ddf
No existe la imagen
No es posible descargar la foto con el id ingresado.
Ingresar X para continuar.
```

Imagen30 (Server)

Consultar Perfiles

El menú de consulta de perfiles permite filtrar los perfiles registrados por una habilidad ingresando (1+Enter), una palabra clave ingresando (2+Enter) o un id ingresando (3+Enter). También permite volver al menú anterior ingresando (4+Enter), menú de usuario. (Imagen30)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# BUSCAR PERFILES PUBLICOS #
-----
1 FILTRAR POR HABILIDAD
2 FILTRAR POR PALABRA CLAVE
3 ESPECIFICO POR ID
4 ATRAS
-----
Para seleccionar una opcion ingresa el numero correspondiente sin espacios.
```

Imagen30 (Cliente)

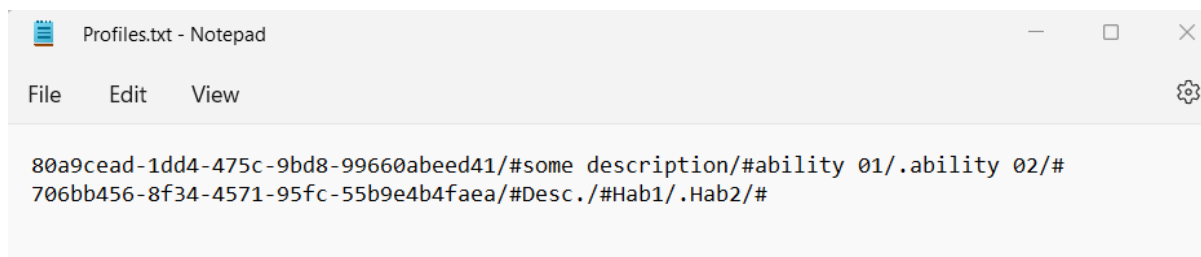
Filtro por habilidad

Para filtrar por la habilidad se solicitará ingresar una habilidad, la misma debe coincidir de forma exacta con la habilidad ingresada al momento de crear el perfil de los usuarios.(Imagen31)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# FILTRO POR HABILIDAD #
-----
Ingrese la habilidad por la cual desea filtrar:
```

Imagen30 (Cliente)

Ejemplo de perfiles existentes, supongamos este caso en el cual existen dos perfiles registrados con anterioridad, podemos verlos en el archivo Profiles.txt: (Imagen32)



```
Profiles.txt - Notepad
File Edit View
80a9cead-1dd4-475c-9bd8-99660abeed41/#some description/#ability 01/.ability 02/#
706bb456-8f34-4571-95fc-55b9e4b4faea/#Desc./#Hab1/.Hab2/#
```

Imagen32 (Archivo Profiles.txt)

Si se envía una habilidad no registrada “H4”, se indicará en consola que no se encuentran perfiles con la misma: (Imagen33)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# RESULTADO FILTRO POR HABILIDAD #
-----
No hay perfiles
Ingresar X para continuar.
```

Imagen33 (Cliente)

Podremos ver en la consola del servidor, la misma respuesta para la consulta: (Imagen34)

```
<- Client: 1 - Instruction: 04 - Status: 200 - Message: /#/#H4
-> Client: 1 - Instruction: 04 - Status: 200 - Message: No hay perfiles
```

Imagen34 (Servidor)

Si se envía una habilidad registrada “Hab1”, se mostrará en pantalla todos los perfiles que coincidan con dicha habilidad: (Imagen35)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# RESULTADO FILTRO POR HABILIDAD #
-----
UserId/#Description/#Abilities/#Image
706bb456-8f34-4571-95fc-55b9e4b4faea/#Desc./#Hab1/.Hab2/#
Ingresar X para continuar.
```

Imagen35 (Cliente)

Entonces podremos ver en la consola del servidor como se transfiere el perfil que corresponde a dicho filtro: (Imagen36)

```
<- Client: 1 - Instruction: 04 - Status: 200 - Message: /#/#Hab1  
-> Client: 1 - Instruction: 04 - Status: 200 - Message: 706bb456-8f34-4571-95fc-55b9e4b4faea/#Desc./#Hab1/.Hab2/#
```

Imagen36 (Servidor)

Filtro por palabra clave

Para el filtro de la clave se requiere ingresar una palabra dentro de la descripción de algún perfil de usuario existente. Por ejemplo entre las descripciones de los perfiles del archivo Profiles.txt podemos ver que existe la descripción “desc hola”:(Imagen 37)

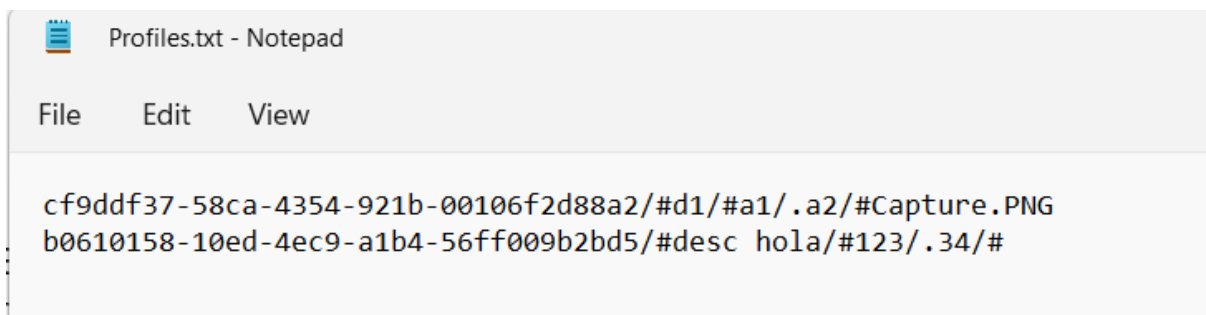


Imagen37 (Archivo Profiles.txt)

Si ingresamos la palabra “hola” en el filtro:(Imagen38)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: test  
-----  
# FILTRO POR PALABRA CLAVE #  
-----  
Ingrese la palabra clave por la cual desea filtrar:  
hola
```

Imagen38 (Cliente)

Vemos que al hacer Enter se obtiene como resultado el perfil que tiene esta palabra en su descripción:(Imagen39)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: test  
-----  
# RESULTADO FILTRO POR PALABRA CLAVE #  
-----  
UserId/#Description/#Abilities/#Image  
b0610158-10ed-4ec9-a1b4-56ff009b2bd5/#desc hola/#123/.34/#  
Ingresar X para continuar.
```

Imagen39 (Cliente)

También podemos ver la consulta en la consola del server: (Imagen40)

```
<- Client: 1 - Instruction: 04 - Status: 200 - Message: hola/#hola/#hola
-> Client: 1 - Instruction: 04 - Status: 200 - Message: b0610158-10ed-4ec9-a1b4-56ff009b2bd5/#desc hola/#123/.34/#
```

Imagen40 (Servidor)

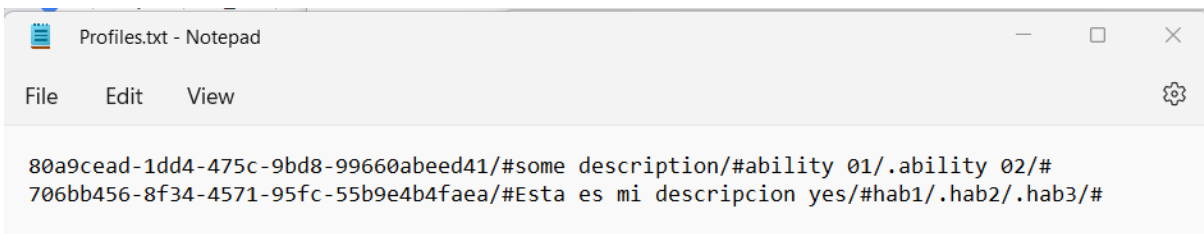
Filtrar por id

Para filtrar por la id se solicitará ingresar una id de un perfil existente, la misma debe coincidir de forma exacta con las id que se encuentren en el archivo Profiles.txt.(Imagen42)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# FILTRO POR ID #
-----
Ingrese el ID por la cual desea filtrar:
```

Imagen42 (Cliente)

Por ejemplo, dado el siguiente archivo de perfiles: (Imagen43)



```
Profiles.txt - Notepad
File Edit View
80a9cead-1dd4-475c-9bd8-99660abeed41/#some description/#ability 01/.ability 02/#
706bb456-8f34-4571-95fc-55b9e4b4faea/#Esta es mi descripcion yes/#hab1/.hab2/.hab3/#
```

Imagen43 (Archivo Profiles.txt)

Si ingresamos una id que se encuentra dentro del archivo: (Imagen44)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# FILTRO POR ID #
-----
Ingrese el ID por la cual desea filtrar:
706bb456-8f34-4571-95fc-55b9e4b4faea
```

Imagen44 (Cliente)

Se muestra en pantalla el perfil asociado a dicha id:(Imagen45)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# RESULTADO FILTRO POR ID #
-----
UserId/#Description/#Abilities/#Image
706bb456-8f34-4571-95fc-55b9e4b4faea/#Esta es mi descripcion yes/#hab1/.hab2/.hab3/#
Ingresar X para continuar.
```

Imagen45 (Cliente)

Pero si se introduce un id que no figura en el archivo de perfiles, como el siguiente: (Imagen46)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# FILTRO POR ID #
-----
Ingrese el ID por la cual desea filtrar:
706bb456-8f34-4571-95fc-55b9e4b4fae0
```

Imagen46 (Cliente)

Se indica que no se encuentra ningún perfil con dicho perfil: (Imagen47)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# RESULTADO FILTRO POR ID #
-----
No hay perfiles
Ingresar X para continuar.
```

Imagen47 (Cliente)

En la consola del server podemos ver la misma respuesta a la consulta: (Imagen48)

```
<- Client: 1 - Instruction: 04 - Status: 200 - Message: 706bb456-8f34-4571-95fc-55b9e4b4fae0/#/#
-> Client: 1 - Instruction: 04 - Status: 200 - Message: No hay perfiles
```

Imagen48 (Servidor)

Mensajes

Enviar mensaje

Para enviar un mensaje se solicitará el email del receptor, seguido del texto a enviar y una confirmación del mismo, si se decide no continuar, el mensaje no será enviado: (Imagen49 y 50)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail
-----
# ENVIAR MENSAJE #
-----
Ingresa email del usuario al que desea enviar el mensaje:
```

Imagen49 (Cliente)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail
-----
# ENVIAR MENSAJE #
-----
Ingresa email del usuario al que desea enviar el mensaje:
pl@gmail
Ingresar texto del mensaje a enviar:
Nuevo mensaje
El mensjae que desea enviar es: Nuevo mensaje
Ingresa A para enviar el mensaje o X para cancelar:
```

Imagen50 (Cliente)

Si se acepta enviar el mensaje entonces aparecerá un aviso de mensaje enviado en la consola y solicitará ingresar (X+Enter) para continuar. (Imagen51)

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail
-----
# Mensaje enviado. #
-----
Ingresar X para continuar.
```

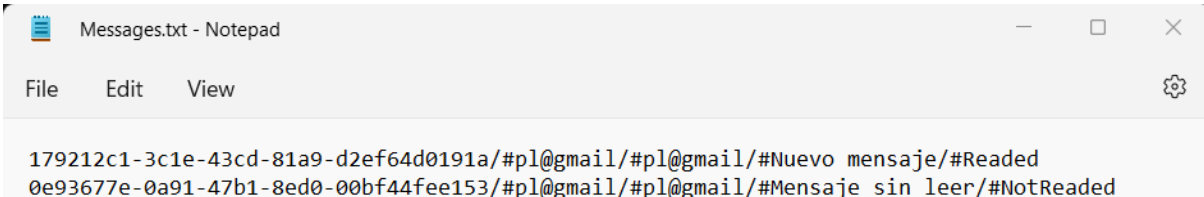
Imagen51 (Cliente)

En la consola del servidor se podrá ver la transferencia del mensaje y un aviso indicando que el mensaje se guardó correctamente: (Imagen52)

```
<- Client: 1 - Instruction: 05 - Status: 200 - Message: 179212c1-3c1e-43cd-81a9-d2ef64d0191a/#pl@gmail/#pl@gmail/#Nuevo
mensaje/#NotReaded
-> Client: 1 - Instruction: 05 - Status: 200 - Message: Mensaje guardado correctamente
```

Imagen52 (Servidor)

El mensaje se enviará con un estado inicialmente NotReaded, los mensajes también podrán verse manualmente en el archivo de mensajes: (Imagen53)



```
179212c1-3c1e-43cd-81a9-d2ef64d0191a/#pl@gmail/#pl@gmail/#Nuevo mensaje/#Readed
0e93677e-0a91-47b1-8ed0-00bf44fee153/#pl@gmail/#pl@gmail/#Mensaje sin leer/#NotReaded
```

Imagen53 (Archivo Messages.txt)

Consulta histórica

La consulta histórica despliega todos los mensajes recibidos, tanto los que han sido leídos como los que no. (Imagen57)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail
-----
# CONSULTA HISTORICA #
-----
179212c1-3c1e-43cd-81a9-d2ef64d0191a/#pl@gmail/#pl@gmail/#Nuevo mensaje/#Readed
0e93677e-0a91-47b1-8ed0-00bf44fee153/#pl@gmail/#pl@gmail/#Mensaje sin leer/#NotReaded
Ingresar X para continuar.

```

Imagen57 (Cliente)

En la consola del servidor veremos la consulta realizada, donde se obtienen los dos mensajes del ejemplo. (Imagen58)

```

<- Client: 1 - Instruction: 07 - Status: 200 - Message: /#/#pl@gmail/#NotLogged
-> Client: 1 - Instruction: 07 - Status: 200 - Message: 179212c1-3c1e-43cd-81a9-d2ef64d0191a/#pl@gmail/#pl@gmail/#Nuevo
mensaje/#Readed
0e93677e-0a91-47b1-8ed0-00bf44fee153/#pl@gmail/#pl@gmail/#Mensaje sin leer/#NotReaded

```

Imagen58 (Servidor)

Si al usuario logueado nunca se le ha enviado un mensaje (no tiene mensajes en el archivo Messages.txt con él como receptor) la consulta histórica será vacía: (Imagen59)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# CONSULTA HISTORICA #
-----
No hay mensajes
Ingresar X para continuar.

```

Imagen59 (Cliente)

Mensajes sin leer

La bandeja de mensajes sin leer, muestra aquellos mensajes que el usuario logueado recibió y que no han sido abiertos o vistos por el mismo. Si el usuario no tiene ningún mensaje nuevo, se indicará en pantalla que no se tiene mensajes: (Imagen50)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail.com
-----
# MENSAJES SIN LEER #
-----
No hay mensajes
Ingresar X para continuar.

```

Imagen60 (Cliente)

Si enviamos un nuevo mensaje y el mismo aparece en el listado de mensajes sin leer, se considerará leído: (Imagen61)

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: pl@gmail
-----
# MENSAJES SIN LEER #
-----
8b06bd3c-4b95-4c29-a8ba-2d09545b0fd4/#pl@gmail/#pl@gmail/#Mensaje sin leer/#Readed
Ingresar X para continuar.

```

Imagen61 (Cliente)

Podemos ver como en la consola del servidor se muestra el nuevo mensaje enviado y el cambio de estado a leído de este al realizarse la consulta de mensajes sin leer: (Imagen62)

```

<- Client: 1 - Instruction: 05 - Status: 200 - Message: 8b06bd3c-4b95-4c29-a8ba-2d09545b0fd4/#pl@gmail/#pl@gmail/#Mensaje sin leer/#NotReaded
-> Client: 1 - Instruction: 05 - Status: 200 - Message: Mensaje guardado correctamente
<- Client: 1 - Instruction: 06 - Status: 200 - Message: /#/#pl@gmail/#NotLogged
-> Client: 1 - Instruction: 06 - Status: 200 - Message: 8b06bd3c-4b95-4c29-a8ba-2d09545b0fd4/#pl@gmail/#pl@gmail/#Mensaje sin leer/#Readed

```

Imagen62 (Servidor)

Ejecutando varios clientes:

Al intentar realizar dos login con el mismo usuario “user1” previamente registrado en dos clientes diferentes vemos que el primer cliente logra loguearse pero el segundo cliente obtiene un error:

Cliente1:

```

Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: user1
-----
# MENU USUARIO #
-----
1 DAR DE ALTA PERFIL
2 ACTUALIZAR FOTO DE PERFIL
3 DESCARGAR FOTO DE PERFIL
4 CONSULTAR PERFILES
5 MENSAJES
6 LOGOUT
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.

```

Cliente2:

```

Puerto: 0 IP: 127.0.0.1 Conectado: True
-----
# LOGIN #
-----
Ingresar email de usuario:
user1
ERROR: El usuario es incorrecto o no existe, intente nuevamente.
Ingresar X para cancelar, Y para reintentar:

```


Servidor:

```
Server initialize...
New client 1 accepted
Attention Routine
New client 2 accepted
Attention Routine
<- Client: 2 - Instruction: 10 - Status: 200 - Message: /#/user1/#NotLogged
-> Client: 2 - Instruction: 10 - Status: 200 - Message: fb6eca26-654a-41af-bb5a-e8bfb33213d9/#user1/#user1/#Logged
<- Client: 1 - Instruction: 10 - Status: 200 - Message: /#/user1/#NotLogged
-> Client: 1 - Instruction: 10 - Status: 400 - Message: Usuario ya logeado
```

Si cambiamos el email del cliente 2 vemos que este es capaz de loguearse al tratarse de un usuario distinto:

Cliente2:

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: user2
-----
# MENU USUARIO #
-----
1 DAR DE ALTA PERFIL
2 ACTUALIZAR FOTO DE PERFIL
3 DESCARGAR FOTO DE PERFIL
4 CONSULTAR PERFILES
5 MENSAJES
6 LOGOUT
-----
Para seleccionar una opción ingresa el número correspondiente sin espacios.
```

Servidor:

```
Server initialize...
New client 1 accepted
Attention Routine
New client 2 accepted
Attention Routine
<- Client: 2 - Instruction: 10 - Status: 200 - Message: /#/user1/#NotLogged
-> Client: 2 - Instruction: 10 - Status: 200 - Message: fb6eca26-654a-41af-bb5a-e8bfb33213d9/#user1/#user1/#Logged
<- Client: 1 - Instruction: 10 - Status: 200 - Message: /#/user1/#NotLogged
-> Client: 1 - Instruction: 10 - Status: 400 - Message: Usuario ya logeado
<- Client: 1 - Instruction: 10 - Status: 200 - Message: /#/user2/#NotLogged
-> Client: 1 - Instruction: 10 - Status: 200 - Message: bdf36afa-f6a0-4fe0-8843-29c093d019de/#user2/#user2/#Logged
```

Cambio de ips y puertos

Para realizar los mismos, se debe ir a los archivos App.config dentro del proyecto ServerApp y cambiar en la parte de value, del mismo modo se debe cambiar en el archivo App.config dentro del proyecto ClientApp.

7. Consideraciones

1. Se requieren los siguientes archivos, los mismos se encuentran en “...\ServerApp\bin\Debug\net6.0”:
 - a. Archivo para el registro de usuarios: Users.txt
 - b. Archivo para el registro de perfiles: Profiles.txt
 - c. Archivo para el registro de perfiles: Messages.txt
2. Para eliminar un usuario es necesario eliminarlo del archivo Users.txt manualmente, lo mismo ocurre para eliminar mensajes o perfiles.
3. Se incrementó el tamaño de datos a 8 bytes con la intención de poder transferir archivos considerablemente pesados.
4. Por defecto al ejecutar el programa este abrirá un cliente. Para abrir más clientes es necesario hacer click derecho en el proyecto de AppClient.
5. Todos los menús y ventanas validan el ingreso de respuesta diferente de vacío.
6. En caso de error en la consola del cliente el usuario que estaba logueado al instante del error, se considerará logueado para la siguiente vez en que se intente hacer el login, por esto es recomendable ir al archivo Users.txt y cambiar su estado a #NotLogged.