Universidad ORT Uruguay
Facultad de Ingeniería

Programación de redes

Preentrega de Obligatorio

Entregado como requisito de la materia Programación de redes

Danilo Biladóniga - 231749
Romina Giaccio Piaggio - 206127

05 de Noviembre de 2022

Resumen

En esta preentrega del obligatorio de la materia “Programación de Redes“ se planteó modificar lo implementado el la entrega anterior, dado un sistema para el manejo de usuarios y perfiles, con tareas implementadas usando paralelismo, concurrencia y sincronización, en este caso usando el patrón asincrónico (async / await) y utilizando las librerías de TCP de alto nivel (TCP Listener / TCP Client) , así como el ajuste de las funcionalidades que tuvieron algún inconveniente en la entrega anterior. Además mantener las dos aplicaciones, una cliente, otra servidor y un protocolo para la comunicación entre las mismas.

Índice

1. Alcance de la aplicación

1.1 Servidor

1.1.1 Requerimientos entregados:

- Se mantienen los requerimientos de la entrega anterior.

1.1.3 Suposiciones

- Cada usuario tiene un único perfil de trabajo.
- El filtro por habilidad se consulta de a una habilidad a la vez.

1.2 Cliente

1.2.1 Requerimientos entregados:

- Se mantienen los requerimientos de la entrega anterior.

2.1 Arquitectura de archivos

Para la utilización de las librerías TCP (TCP Listener / TCP Client) debieron incluirse cambios en la arquitectura de los archivos, así como también en su código. Aquí detallaremos los cambios realizados teniendo en cuenta que se mantienen o mejoran las funcionalidades de la entrega anterior:

LkdinSystem

[Proyecto] ClienteApp

[Clase] Program.cs

- Se modificó su código para adaptarse a las librerías de TCP. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica con el método Main modificado.

[Domain]

[Clase] Messages.cs

- Se mantuvo.

[Clase] User.cs

- Se mantuvo.

[Clase] UserProfile.cs

- Se mantuvo.

[Proyecto] Enum

[Clase] SpecialChars.cs

- Se mantuvo.

[Proyecto] Protocol

[Carpeta] Commands

[Clase] ClientCommands.cs

- Se modificó su código para adaptarse a las librerías de TCP. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Clase] CommandsHandler.cs

- Se mantuvo

[Clase] ServerCommands.cs

- Se modificó su código. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Carpeta] Enum

[Clase] EmptyStatesMessages.cs

- Se mantuvo.

[Clase] States.cs

- Se mantuvo.

[Clase] Constants.cs

- Se mantuvo.

[Clase] FileCommsHandler.cs

- Se modificó su código para adaptarse a las librerías de TCP. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Clase] FileDatabaseManager.cs

- Se modificó su código. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Clase] FileHandler.cs

- Se mantuvo.

[Clase] FileOperations.cs

- Se mantuvo.

[Clase] FileStreamHandler.cs

- Se modificó su código. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Clase] ServerConfig.cs

- Se mantuvo.

[Clase] SocketHelper.cs

- Se modificó su código para adaptarse a las librerías de TCP. Ahora cuenta con el nombre “**TcpHelper**”, conservo las funcionalidades anteriores.

[Clase] TransferSegmentManager.cs

- Se modificó su código para adaptarse a las librerías de TCP. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Proyecto] ServerApp

[Clase] ClientHandler.cs

- Se modificó su código para adaptarse a las librerías de TCP. Ahora cuenta con el nombre “**ClientTcpHandler**”, conservo las funcionalidades anteriores.

[Clase] Program.cs

- Se modificó su código para adaptarse a las librerías de TCP. Conserva su nombre anterior y las funcionalidades anteriores, pero las mismas ahora se realizan de forma asincrónica.

[Proyecto] Utils

[Clase] ConversionHandler.cs
- Se mantuvo.

[Clase] NumberConvertions.cs
- Se mantuvo.

[Clase] RandomData.cs
- Se mantuvo.

[Clase] SettingsManager.cs
- Se mantuvo.

2. Diseño detallado de componentes

3.1 Especificación del protocolo

3.1.1 Descripción

Se mantuvo el protocolo utilizado en la entrega anterior, el mismo 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 tasks en lo que respecta al manejo de concurrencia, correspondientes a la conexión entre los clientes y el servidor, para leer o escribir los distintos archivos que componen nuestro sistema.

Además hemos utilizado async/await para el control automático de tareas a realizar por cada método/acción que realicen los diferentes clientes hacía el servidor

3. Cambios y mejoras en funcionamiento de la aplicación

La modalidad de navegación se conservó desde la entrega anterior, se agregaron cambios al mostrar los datos en pantalla, ante cualquier menú es necesario ingresar el número correspondiente a la opción que queremos ejecutar seguido de un Enter y siguiendo las indicaciones que se muestran en pantalla. Además los mensajes por consola del Servidor se conservaron desde la entrega anterior.

Consultar Perfiles

Para esta nueva entrega se modificó la visualización de los perfiles consultados por el usuario, este cambio fue implementado para que el usuario identifique más rápidamente el id de un perfil en caso de querer descargar su imagen y visualizar mejor los detalles del perfil.

Dado el siguiente caso de archivo de perfiles:

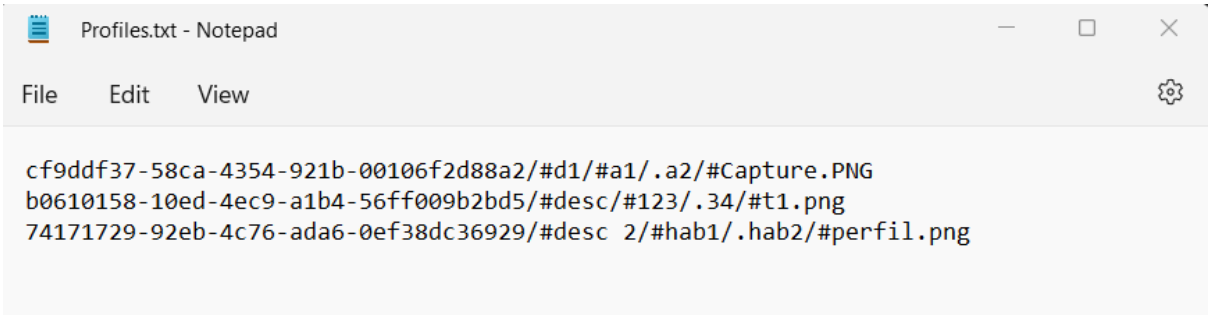

Imagen1 (Archivo Profiles.txt)

Dado el siguiente menú de la aplicación:

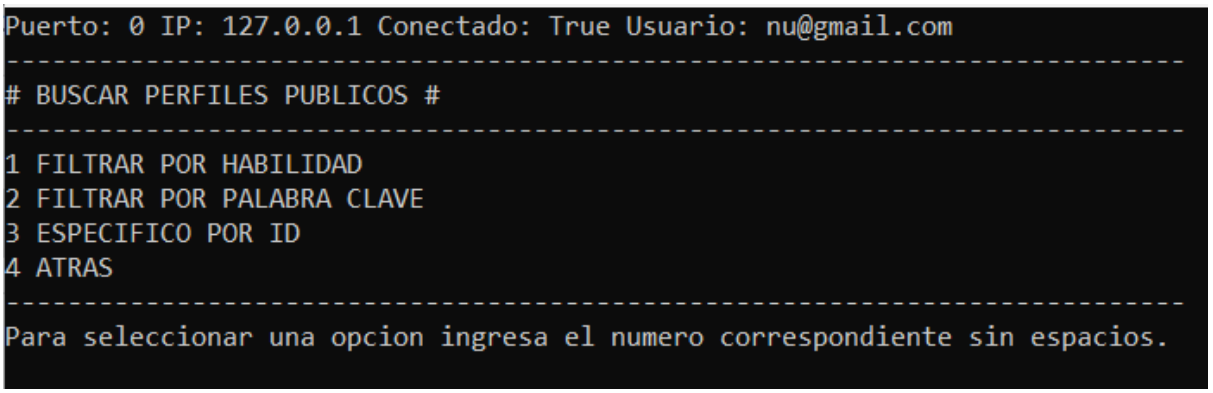

Imagen2 (Cliente)

Filtro por habilidad

Si el usuario decide filtrar el perfil por la habilidad “hab1” el resultado es el siguiente:

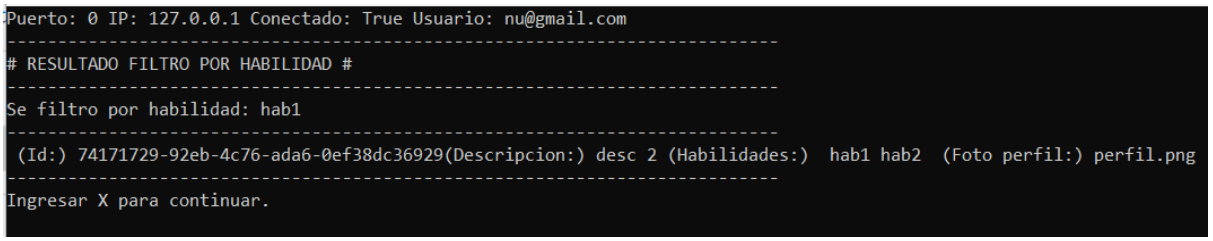

Imagen2 (Cliente)

Mostrándose entre paréntesis los diferentes nombres de los componentes del perfil. Además si introducimos alguna habilidad que no se encuentre dentro de los perfiles existentes, como es el caso de la habilidad “hab3”, se muestra el siguiente mensaje en consola (Este comportamiento es igual para cualquier filtro realizado).

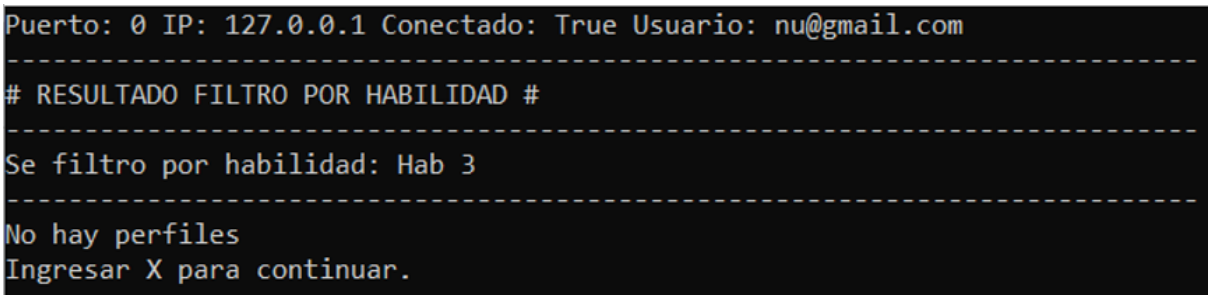

Imagen3 (Cliente)

Filtro por palabra clave

Si el usuario decide filtrar el perfil por la clave “desc” el resultado es el siguiente:

Imagen4 (Cliente)

Todas las coincidencias que incluyan la palabra “desc” en la descripción, mostrándose entre paréntesis los diferentes nombres de los componentes del perfil. A continuación vemos un caso sin coincidencias para la descripción ingresada:

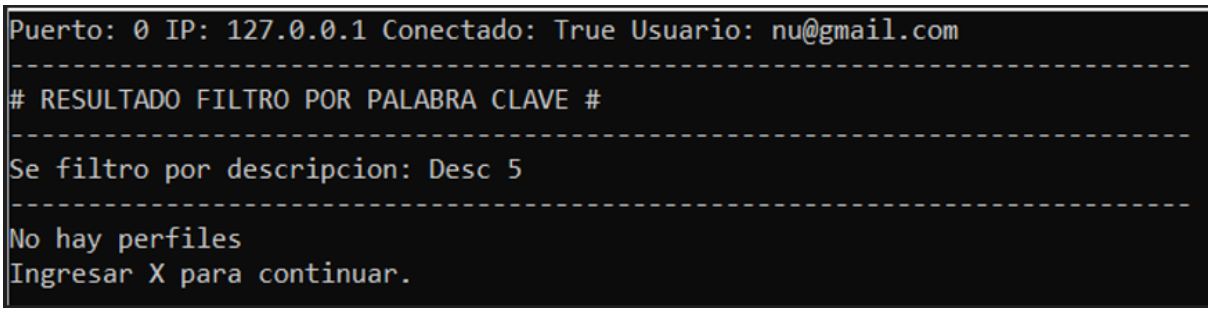

Imagen5 (Cliente)

Filtrar por id

Si el usuario decide filtrar el perfil por la Id “cf9ddf37-58ca-4354-921b-00106f2d88a2” el resultado es el siguiente:

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: nu@gmail.com
--------------------------------------------------------------------------------
# RESULTADO FILTRO POR ID #
--------------------------------------------------------------------------------
Se filtro por Id de usuario: cf9ddf37-58ca-4354-921b-00106f2d88a2
--------------------------------------------------------------------------------
 (Id:) cf9ddf37-58ca-4354-921b-00106f2d88a2(Descripcion:) d1 (Habilidades:)  a1 a2  (Foto perfil:) Capture.PNG
--------------------------------------------------------------------------------
Ingresar X para continuar.
```

Imagen6 (Cliente)

Mostrándose entre paréntesis los diferentes nombres de los componentes del perfil.

Mensajes

Para esta nueva entrega se modificó la visualización de los mensajes consultados por el usuario, este cambio fue implementado para que el usuario visualice mejor los detalles del mensaje e identifique el usuario emisor del mismo.

Dado el siguiente caso de archivo de Mensajes:

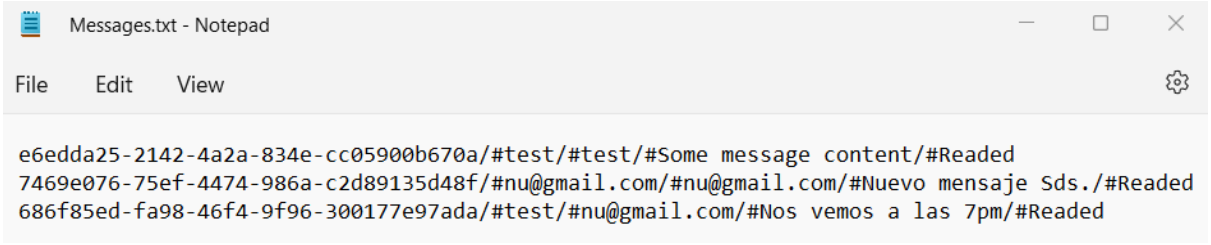

Imagen7 (Cliente)

Dado el siguiente menú de la aplicación:

```
Puerto: 0 IP: 127.0.0.1 Conectado: True Usuario: nu@gmail.com
--------------------------------------------------------------------------------
# MENSAJES #
--------------------------------------------------------------------------------
1 CONSULTA HISTORICA
2 MENSAJES SIN LEER
3 ENVIAR MENSAJE
4 ATRAS
--------------------------------------------------------------------------------
Para seleccionar una opcion ingresa el numero correspondiente sin espacios.
```

Imagen8 (Cliente)

Mensajes sin leer

Si el usuario con correo “nu@gmail.com” decide revisar sus mensajes sin leer, podemos observar lo siguiente:

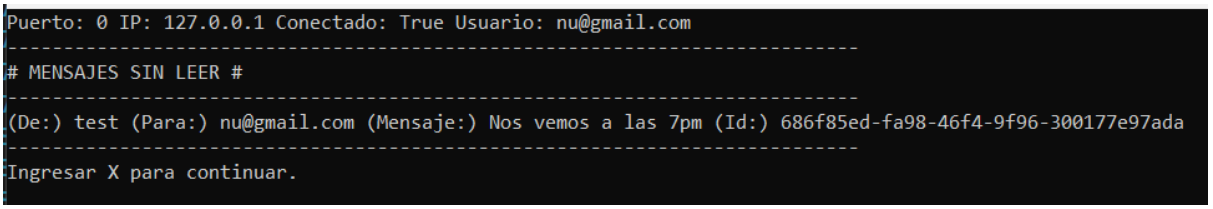

Imagen9 (Cliente)

Mostrándose el mensaje sin leer que ha sido enviado por el usuario de correo “test”, diferenciando el mensaje del identificador.

Consulta histórica

Si el usuario con correo “nu@gmail.com” decide realizar una consulta historica de los mensajes recibidos, podemos observar lo siguiente:

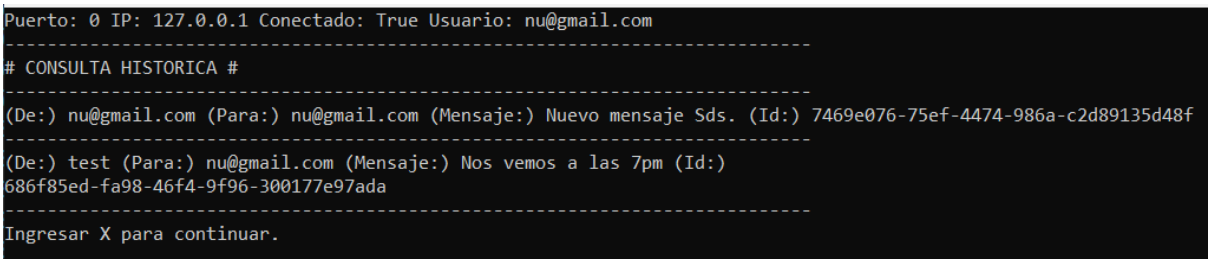

Imagen10 (Cliente)

Mostrándose la lista de mensajes que han sido enviados por el usuario de correo “test”, incluyendo un mensaje a sí mismo, diferenciando el mensaje del identificador y los demás componentes del mensaje.

7. Consideraciones

No se agregan nuevas consideraciones.

8. Corrección de errores

Corrección en la funcionalidad Filtrar perfiles.

La funcionalidad permite introducir una habilidad, palabra clave o id de usuario y visualizar la lista de perfiles que se corresponden a dicho filtro con todos los detalles. En la entrega anterior este estaba presentando ciertos problemas al filtrar, en esta nueva versión estos problemas fueron solucionados. Se encuentra evidencia de este funcionamiento en la sección “Cambios y mejoras en funcionamiento de la aplicación - Consultar Perfiles”.

Cierre de conexión con el servidor amigable.

Se agregaron try catch del lado del cliente para que al momento de que se cierre la conexión con el servidor la interfaz se cierre de forma amigable para el usuario.