Documento del Proyecto: Chat Cliente-Servidor

Guía de comandos, instrucciones y funcionamiento del servidor-chat

Versión: 2.5

Participantes del proyecto:

Heredia, Fernando

Oyola, Ruth

Mendoza, Dante

- POO III 2017 -

Introducción

Este documento se dividirá en cuatro secciones:

- En la primera sección se pretende explicar algunas cuestiones de diseño de forma superficial, para dar un vistazo de cómo esta implementado el servidor.
- En la siguiente, se detallarán los comandos que admite el servidor, explicando cómo y en qué caso se usan cada uno.
- Posteriormente se mostrará una conversación tipo como forma de ejemplificar un caso de uso del servidor.
- Finalmente, se hará hincapié en la descripción del cliente, su estructura y sus características.

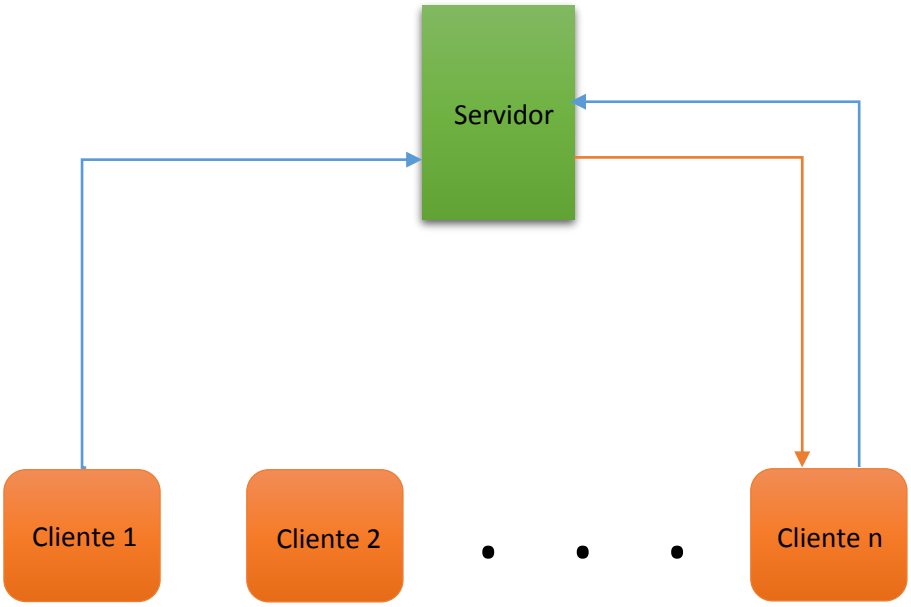
Es importante aclarar que este documento proporciona una mirada informal de conocimiento acerca de la aplicación del servidor-chat.

Características del Servidor

Parte 1: Diseño y funciones del servidor

El servidor fue pensado con la capacidad de atender múltiples clientes mediante el uso de hilos y demás características concurrentes y de sincronización.

Un esquema de diseño básico pensado (para el envío de mensajes y recepción) es el siguiente:



Las peticiones entrantes son las flechas celeste, y las respuestas del servidor son las flechas naranja.

Este esquema no explica en detalle cómo funciona el protocolo, su función es mostrar la idea principal de cómo se envían y se reciben los mensajes.

El Cliente 1 quiere enviarle un mensaje al Cliente n, como se puede observar, el servidor actúa como intermediario obligatorio, es decir, un cliente de ninguna forma puede enviar un mensaje directamente a otro.

Por tanto, el Cliente 1 tiene que enviar una petición/comando al servidor en el que especifica la cadena de texto/mensaje que desea enviar. Acto seguido, el servidor recibe esa petición, y de alguna forma debe saber de antemano, con un procedimiento anterior, a que cliente se destina tal mensaje (eso mismo se verá de forma detallada, posteriormente).

El Servidor tiene la facultad de contar con una base de datos, donde guarda, entre otras cosas, la lista de usuario/clientes registrados en la aplicación, una lista de conversaciones, y la lista de los mensajes que recibió. Es por medio de esto que es posible que el servidor actúe como una especie de directorio de mensajes, donde cada cliente deberá preguntar (enviar una petición/comando) para corroborar si tiene mensajes pendientes. En el caso de que los tenga, el servidor le responderá con aquellos mensajes que tenía en la base de datos, y que tienen como destinatario al cliente solicitante.

Para ahondar en la especificación se tiene en cuenta los siguiente requisitos/validaciones:

- El usuario deberá registrarse con su nombre de usuario de forma que pueda participar en el chat, una vez registrado, se le asignará un ID unívoco e inalterable.
- Un usuario ya registrado en la base de datos deberá loguearse para poder participar del chat, una vez logueado.
- Un usuario logueado/registrado podrá crear conversaciones
- Cada conversación permitirá establecer una conexión entre dos clientes/usuarios
- Sin una conversación creada o vinculada, el usuario no podrá enviar mensajes
- Un usuario no registrado/logueado (se trata como un usuario anónimo) no debe tener la facultad crear conversaciones.
- Se provee una forma para que un usuario averigüe si alguien se desea conectar con él, es decir, si otro usuario ha creado una conversación con el id de este otro usuario. En tal caso, se hace una vinculación, para que el usuario que pregunta si hay conversaciones con su ID, tenga tal conversación referenciada.
- El usuario que cumpla con tales condiciones, podrá enviar mensajes. Estos mensajes los recibirá el servidor y los guardará en la base de datos.
- Se definirá una forma para que un usuario pueda solicitar al servidor si tiene mensajes pendientes. En tal caso, el servidor responderá con esos mensajes.
- El servidor proveerá un comando para mostrar los usuarios conectados.
- El servidor también proporcionará un mecanismo de desconexión y desvinculación de una conversación para los clientes que lo deseen.
- Cuando ocurra una desconexión de una conversación, se borrará esta de la base de datos del servidor y se borraran también los mensajes que pertenezcan a esta conversación.
- Un usuario podrá desconectarse completamente del servidor mediante un comando.

Otras características:

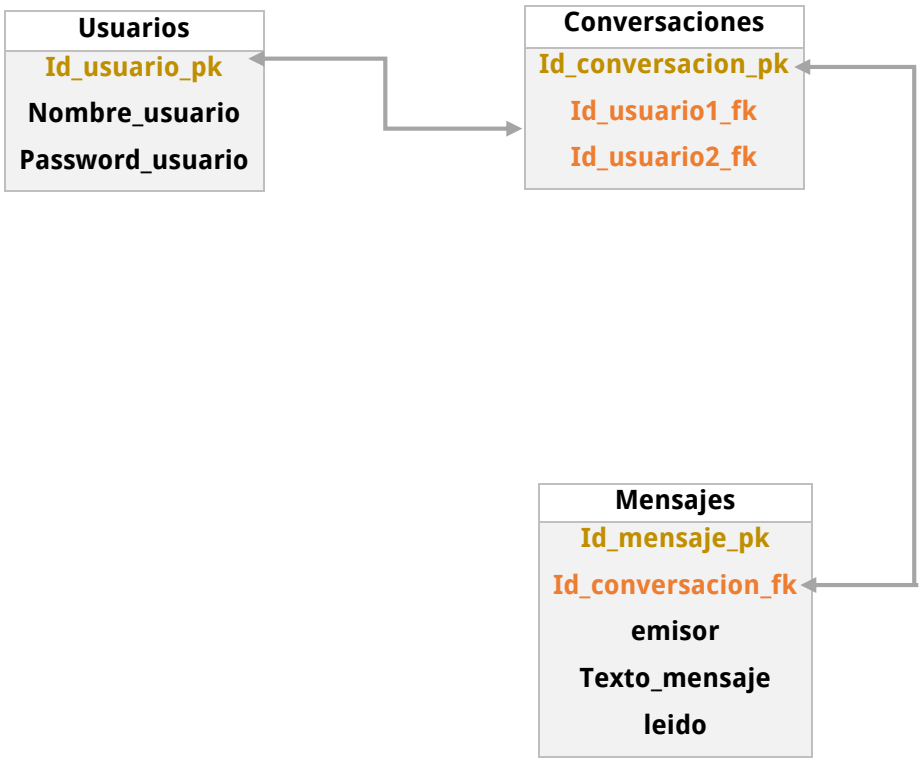
El servidor fue programado utilizando el lenguaje Java, utiliza el patrón de diseño singleton para evitar crear más de una instancia de determinados tipos de objetos.

Base de datos

Es necesario que el servidor guarde:

- Los datos de los usuarios (IDUsuario, Nombre, contraseña)
- El registro de las conversaciones creadas por los usuarios (IDConversacion, IDusuario1, IDusuario2)
- Los mensajes (IDMensaje, IDConversacion, Emisor, TextoMsj, Leido)

Por lo que el esquema de tablas y relaciones es el siguiente:



Detalles de los campos:

Usuarios:

- Id_usuario_pk: un identificador numérico entero que comienza desde 1000.
- Nombre_usuario: texto en formato carácter
- Password_usuario: por defecto es 1234, pero puede ser alfanumérico

Conversaciones:

- Id_conversacion_pk: identificador numérico entero que comienza desde 2000.
- Id_usuario1_fk: ID de usuario 1 de la conversación.
- Id_usuario2_fk: ID de usuario 2 de la conversación.

Mensajes:

- Id_mensaje_pk: Identificador único numérico entero que comienza desde 3000.
- Id_conversacion_fk: identificador de la conversación a la que pertenece el mensaje.
- Emisor: ID del usuario que envió el mensaje.
- Texto_mensaje: contenido del mensaje.
- Leido: campo booleano que indica si el mensaje ha sido leido.

Parte 2: Comandos del servidor

Los comandos que entiende el servidor son los siguientes:

- UN: Con este comando el usuario puede registrarse en el chat.

  UN –*nombreUsuario*

  Ejecución correcta: devuelve un OK + ID de usuario asignado

  Efectos de la ejecución correcta:

  -agrega un registro en la tabla usuarios de la BD.

  -crea un objeto Usuarios en memoria con los datos ingresados, que vincula al hilo.

  -añade el objeto Usuario creado a la lista de usuarios conectados del servidor.

- LO: Permite loguear un usuario ya registrado.

  LO –*nombreUsuario* –*contraseña*

  Ejecución correcta: devuelve un OK + ID de usuario asignado

  Efectos de la ejecución correcta:

  -crea un objeto Usuarios en memoria con los datos ingresados, que vincula al hilo.

  -añade el objeto Usuario creado a la lista de usuarios conectados del servidor.

- CN: El usuario conectado puede crear una conversación con otro usuario.

  CN -*IdUsuarioDestino*

  Ejecución correcta: devuelve un OK + el ID de la conversación creada.

  Efectos de la ejecución correcta:

  -agrega un registro a la tabla conversaciones de la BD.

  -crea un objeto Conversaciones y lo añade a la lista de conversaciones del servidor

  -crea una referencia a la conversación, *en esta versión, solamente al hilo invocador del comando, para crear la referencia a la conversación en el hilo que maneja al otro usuario participante, éste debe usar el comando PC (para preguntar si alguien ha creado una conversación con su ID como destino).*

- UC: Muestra una lista de usuarios conectados

  UC

  Ejecución correcta: devuelve OK + la lista con los campos ID y nombre de usuario

  Efectos de la ejecución correcta:

  -ninguno

- TX: Permite enviar mensajes a otro usuario.

  TX –*TextoDelMensaje*

  Ejecución correcta: devuelve un OK

  Efectos de la ejecución correcta:

  -agrega un registro en la tabla mensajes de la BD, con el campo "leído" establecido en "NO".

  -el servidor sincroniza con la base de datos recuperando los mensajes, posteriormente crea objetos Mensajes con ellos y los guarda en una lista.

- PC: Al usuario que llame a este comando estará "preguntando" si hay una conversación cuyo ID de destino sea el propio, es decir, si algún otro usuario ha creado una conversación con él (*Visto de otra forma, sirve para "sincronizar" las referencias a la conversación de aquel hilo/usuario que tenga un comportamiento "pasivo", o sea, que no haya creado la conversación ni la haya eliminado. Complementando así al comando CN cuando se crea una conversación, y al comando DS cuando se la elimina*).

  PC

  Ejecución correcta: devuelve OK + ID del otro usuario participante de la conversación + id de la conversación pendiente (en caso de que la hubiera).

  Efectos de la ejecución correcta:

  -le asigna una referencia a la conversación al hilo que controla al usuario invocador del comando, en caso de que tenga una conversación pendiente. (*caso de uso en CN*)

  -elimina la referencia a la conversación del hilo/usuario, en caso de que el otro participante la haya eliminado previamente (*caso de uso en DS*).

- GM: El usuario que invoque este comando recibirá uno de los mensajes pendientes, si los tuviese.

  GM

  Ejecución correcta: devuelve OK + el ID del emisor de los mensajes + una lista de los mensajes pendientes para este usuario separados por />*<

  Efectos de la Ejecución correcta:

  -accede a la base de datos, si encuentra mensajes cuyo ID de usuario destino coincida con el ID del hilo que maneja al usuario invocador del comando, recupera esos mensajes, los tilda como leídos y muestra la lista en pantalla.
  -a continuación, borra los mensajes recibidos de ese hilo en memoria.

- DS: Desconecta al usuario de la conversación activa.

  DS –*IdUsuario*

  Ejecución correcta: devuelve OK + una cadena compuesta con el nombre de los participantes de la conversación eliminada.
  Efectos de la ejecución correcta:

  -elimina los mensajes de la tabla mensajes en la BD cuyo ID de conversación coincida con el ID de la conversación a la cual pertenece el ID de usuario especificado.
  -a continuación elimina el registro en la tabla conversaciones de la BD.
  -elimina el objeto Conversaciones en la lista de conversaciones del servidor.
  -elimina la referencia a la conversación en el hilo que maneja al usuario que invoca al comando (*necesidad de que el otro hilo/usuario participante de la conversación elimine su referencia a ésta también, usando el comando PC*).

- QC: Muestra una lista de los usuarios registrados en la base de datos.

  QC

  Ejecución correcta: devuelve OK + la lista de usuarios con sus nombres e IDs.
  Efectos de la ejecución correcta:

  -ninguno.

- LU: Cierra la sesión activa.

  LU

  Ejecución correcta: devuelve un OK

Efectos de la ejecución correcta:

-reinicializa todas las variables e instancias de objetos del hilo, que previamente tenían la información del usuario antes de ejecutar el comando.

- EX: Desconecta el cliente del servidor.

  EX

  Ejecución correcta: devuelve OK + el ID del usuario que invoca el comando.

  Efectos de la ejecución correcta:

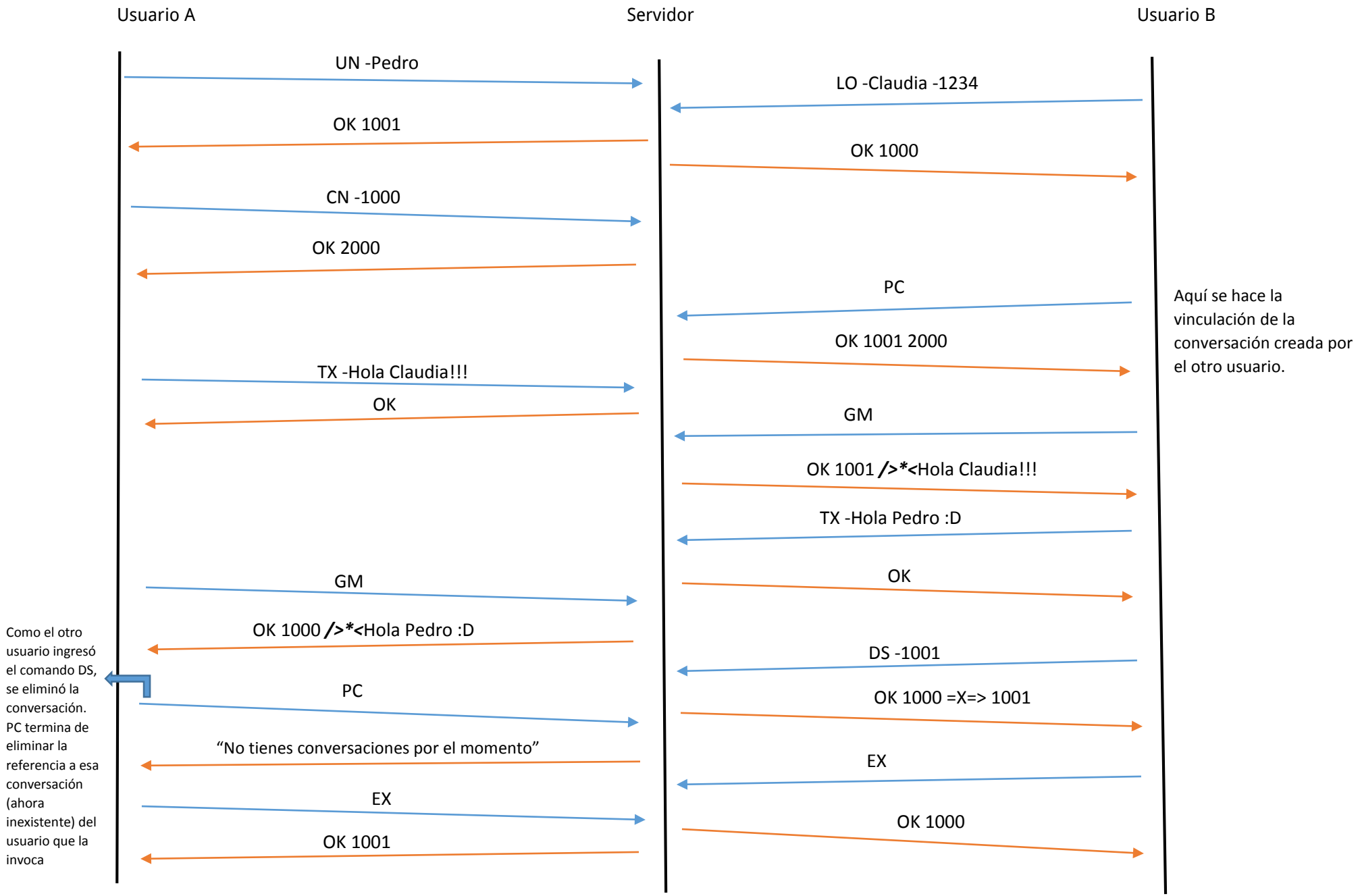
  -elimina el objeto Usuarios correspondiente de la lista de usuarios conectados del servidor

  -a continuación, cierra la conexión del hilo con el socket.

Parte 3: Ejemplo de conversación

Se muestra un ejemplo de conversación mediante un diagrama de interacción/secuencia entre dos clientes y un servidor.

- Se asume que el usuario A, no está registrado, mientras que el usuario B sí lo está y tan solo debe loguearse.
- Los usuarios se conectarán, se enviarán un mensaje entre sí y a continuación se desconectan.


Usuario A
Servidor
Usuario B
UN -Pedro
LO -Claudia -1234
OK 1001
OK 1000
CN -1000
OK 2000
PC
Aquí se hace la vinculación de la conversación creada por el otro usuario.
OK 1001 2000
TX -Hola Claudia!!!
OK
GM
OK 1001 />*<Hola Claudia!!!
TX -Hola Pedro :D
OK
GM
OK 1000 />*<Hola Pedro :D
DS -1001
Como el otro usuario ingresó el comando DS, se eliminó la conversación. PC termina de eliminar la referencia a esa conversación (ahora inexistente) del usuario que la invoca
PC
OK 1000 =X=> 1001
“No tienes conversaciones por el momento”
EX
EX
OK 1000
OK 1001

Limitaciones del diseño de la versión actual

Existen algunas restricciones en el servidor que pueden ser eliminadas en el futuro:

- Las conversaciones no pueden ser de más de dos usuarios.
- El usuario debe tomar la iniciativa de enviar un comando para averiguar si tiene mensajes, no existe actualmente un mecanismo de eventos para que el mensaje sea mostrado de forma inmediata.
- El campo contraseña se ha agregado pero permanece invariante, en esta versión no existe comandos para editar la contraseña, así que ésta va a ser siempre '1234'.
- Un usuario no puede tener más de una conversación, debe desconectarse de esa conversación para iniciar otra.

Parte 4: Funcionamiento del Cliente

Estructura

El Cliente está compuesto por dos capas: una capa de presentación y una capa de negocio.

Mediante la capa de presentación, brindamos al usuario una interfaz mediante la cual se podrá acceder al sistema e iniciar conversaciones con los demás usuarios.

Mediante la capa de negocio, se reciben las peticiones provenientes de la capa de presentación, para su procesamiento. Se denomina capa de negocio porque en ella residen todas las reglas que deben cumplirse para el correcto funcionamiento del sistema.

Si bien estas reglas de negocio aparentan ser cumplidas desde el cliente, es el servidor el encargado de definirlas y validarlas *(ver parte 1/requisitos y validaciones)*. Más bien lo que el cliente hace es: dadas las acciones del usuario, enviar solicitudes al servidor y mostrar al usuario el resultado devuelto.

Relativo a la capa de negocio

La conexión con el servidor se hace mediante un *Socket,* el cual permite al Cliente comunicarse con el servidor.

Para poder realizar esto, el cliente conoce el nombre del host de la máquina que está ejecutando el servidor y el número de puerto en el que el servidor está escuchando. Para realizar una solicitud de conexión, el cliente intenta reunirse con el servidor en la máquina y el puerto del servidor. El cliente también necesita identificarse con el servidor para que se vincule con un número de puerto local que usará durante esta conexión.

Para poder enviar solicitudes, el cliente enviará los comandos correspondientes a través del Socket.

Relativo a la capa de presentación

La interfaz de usuario está realizada en WPF(Windows Presentation Foundation), una tecnología de Microsoft basada en la arquitectura Modelo-Vista-Controlador para el desarrollo de aplicaciones.

WPF utiliza el lenguaje declarativo XAML para el desarrollo de la interfaz de usuario, y el lenguaje C# para el manejo de la lógica del sistema.

Funcionamiento: Hilos y concurrencia

Como explicamos anteriormente, dada una acción del usuario, el cliente es el encargado de comunicar esto al servidor.

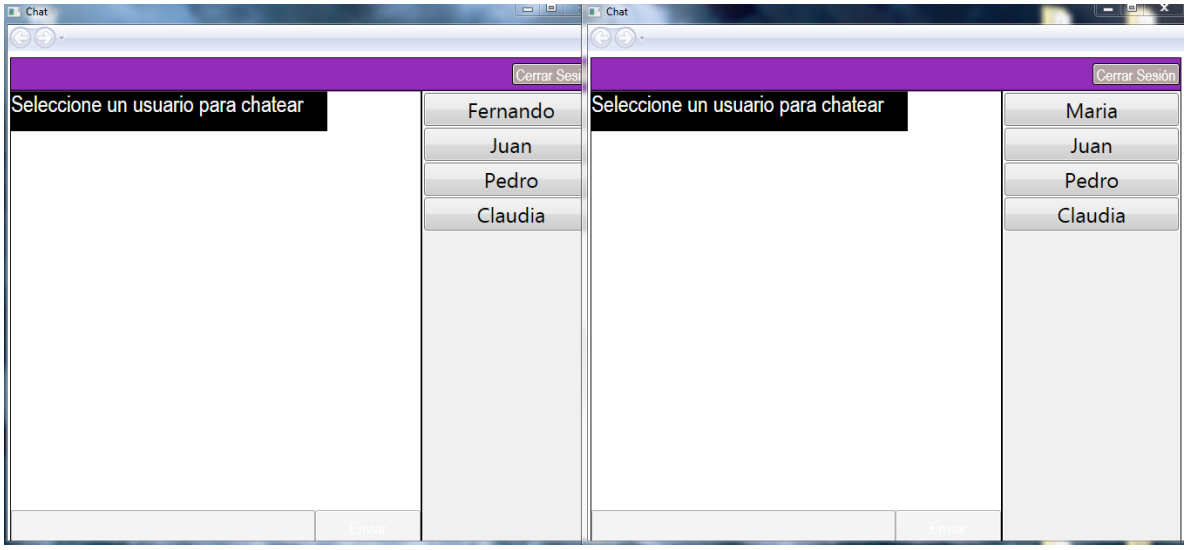
Pero es necesario tener en cuenta lo siguiente: existen procedimientos que por su naturaleza no se disparan por una acción del usuario, sino más bien son disparados debido a un cambio de estado, casos como éste son:

- Mientras un usuario este logueado, el cliente debe consultar al servidor si existe un usuario que haya iniciado una conversación con el. Caso que se da cuando el usuario logueado es el receptor de la conversación y no el emisor.
- Mientras un usuario tenga una conversación activa, el cliente debe consultar al servidor si tiene mensajes entrantes que leer.
- Mientras un usuario tenga una conversación activa, el cliente debe consultar al servidor si la conversación sigue activa o si el otro usuario la ha finalizado.

Dicho esto, destacamos que como parte del funcionamiento interno del cliente, creamos y mantenemos dos hilos de ejecución: el principal, asociado a las acciones del usuario y un hilo secundario, que es el que se comunica con el servidor para dar soporte a estas situaciones de cambios de estado.

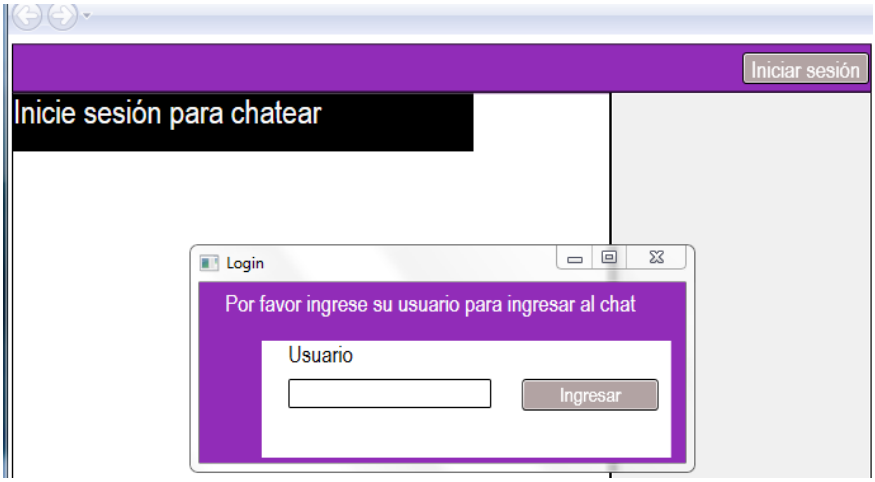
Al correr estos dos hilos en paralelo, contamos con elementos de la interfaz de usuario que fueron declarados como concurrentes, para mantener la consistencia a la vista del usuario, de lo que sucede por detrás.

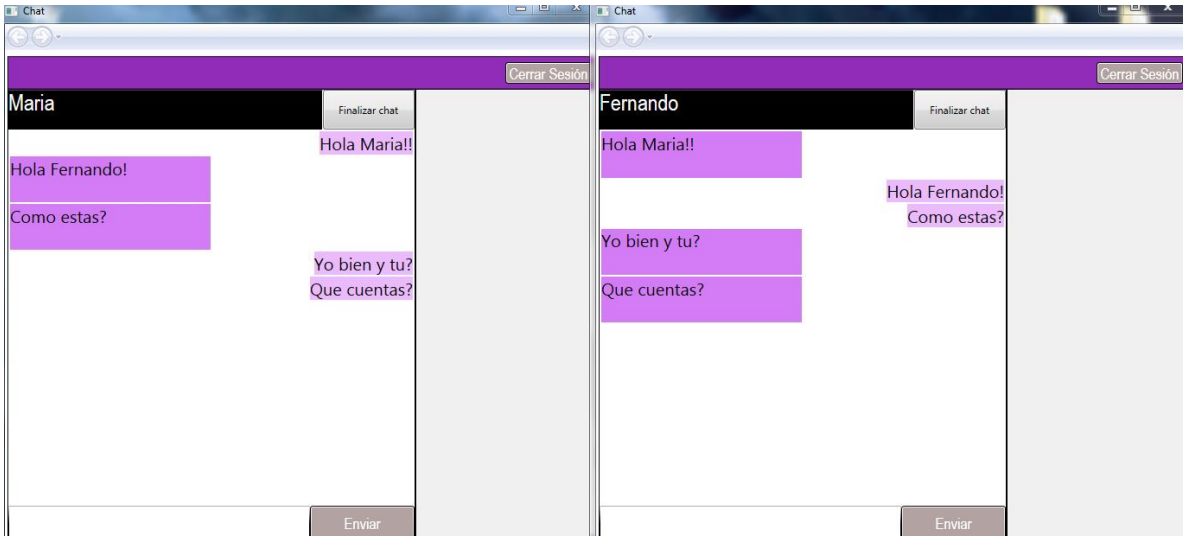
Algunas capturas del chat funcionando:



En esta captura se inician múltiples clientes, cada uno de ellos, excepto mi propio usuario, aparece en la lista de usuarios conectados ubicada en la parte derecha de cada ventana.

Como en todo chat, es necesario iniciar sesión para participar, cada uno de los usuarios conectados hicieron clic en el botón iniciar sesión, donde introdujeron su Nick.





Una vez logueados, cada usuario es libre de hacer clic en uno de los usuarios que aparecen conectados, de este modo, se iniciará una conversación con tal usuario.

Cuando alguno de los usuarios quiere terminar la conversación, simplemente debe hacer clic en el botón “Finalizar chat” que aparece en la imagen, a continuación volverán a poder elegir otro usuario para chatear o cerrar su sesión.

El código está alojado en el siguiente repositorio en GitHub:

https://github.com/DanteMendoza/POO3---TP1---2017