

Console Whatsapp

El objetivo de esta práctica es crear un sistema de mensajería tipo whatsapp por TCP, en modo consola, sin interfaz gráfico, ni medios multimedia pero que permita enviar mensajes de forma asíncrona entre diferentes clientes y desatendida en el servidor (sin intervención de un admin).

Para desarrollar este sistema, manearemos los siguientes conceptos:

- **Cliente:** todo aquel que inicia una conexión para solicitar datos.
- **Servidor:** todo aquel que escucha y atiende nuevas conexiones.
- **Remitente:** todo aquel que envia un mensaje para un destinatario
- **Destinatario:** todo aquel que tiene que recibir un mensaje de un remitente.

Según esta estructura, el sistema se compondrá de un servidor central al que los diferentes clientes se conectarán para enviar y recibir los mensajes de sus conversaciones entre ellos.

Cada remitente, se conectará como cliente al servidor con un usuario y contraseña únicos al inicio de toda nueva conexión, que el servidor deberá evaluar contra un archivo de usuarios y contraseñas válidos.

Esta comunicación inicial, será igual para todas las conexiones y tendrá un formato único de la forma, empezando por una cadena “LOGIN” y separando los campos usuario y contraseña con dos puntos “:” de la manera:

“LOGIN”:USUARIO:CONTRASEÑA

Si el usuario y contraseña son correctos, el servidor devolverá un “OK” y atenderá el resto de comunicaciones según el formato del proyecto. Si el usuario o contraseña no fuesen correctos, el servidor cerrará la conexión y no atenderá más mensajes. Será necesario abrir una nueva conexión para

Si el cliente valida su conexión correctamente, podrá iniciar su trabajo contra el servidor en dos líneas de trabajo separadas: envíos y recepciones.

- **Envíos:** se establecerá una conexión única, mediante el puerto 666, para enviar mensajes de texto al servidor. Los mensajes se escribirán en una consola (Terminal) donde el usuario podrá escribir lo que quiera enviar, en una sola línea, pulsando la tecla “Enter” para que se envíe el mensaje al servidor inmediatamente. El usuario podrá escribir varios mensajes seguidos a un mismo destinatario, sencillamente escribiendo más líneas y pulsando “Enter” con cada línea que quiera enviar. El destinatario al que se enviarán los mensajes aparecerá al inicio de la línea precedido con una @ y un nombre de usuario, tras el que aparecerá el carácter de los dos puntos “:” y tras lo que el nuevo mensaje que se está escribiendo será escrito. Se podrá enviar un mensaje en blanco pulsando “Enter”, sin escribir nada, con la única finalidad de escribir una línea vacía y actualizar el feed de mensajes.

Ejemplo: mensaje del usuario para el destinatario Alfonso

@Alfonso: esto es un ejemplo de escritura

- **Recepciones:** se establecerá una conexión única, mediante el puerto 999, para recibir los mensajes para el cliente que hayan enviado otros remitentes, usando el nombre de usuario del cliente como destinatario. No se podrá mostrar ningún mensaje al usuario hasta que éste abra la conversación correspondiente al remitente que lo envía. La recepción de nuevos mensajes será asíncrona con el envío. Es decir, los mensajes se enviarán en el momento en que el usuario los envíe, pero mientras tanto, se podrán estar recibiendo nuevos mensajes simultáneamente, al estar usándose dos conexiones diferentes. El sistema de recepción revisará con el servidor si hay nuevos mensajes para el cliente, enviándole una solicitud de actualización al servidor cada X segundos (X debe poder ser configurable). Si hay nuevos mensajes de cualquier conversación entre clientes, nos los descargaremos todos juntos y los almacenaremos hasta que el usuario abra la conversación del remitente que sea. Si el usuario pide la lista de usuarios disponibles, se le mostrará al lado del nombre del remitente un paréntesis con los mensajes nuevos que hay de ese remitente. Mientras la consola espera a que el usuario escriba un nuevo mensaje, no se podrá hacer nada por pantalla. Si en ese periodo de tiempo el cliente ha recibido nuevos mensajes como destinatario, se le mostrarán todos los nuevos mensajes recibidos, solamente cuando el usuario envíe un nuevo mensaje en una conversación activa o, en su defecto, cuando abra la conversación con un remitente. Mientras tanto, los nuevos mensajes no podrán aparecer. Para refrescar y actualizar la pantalla, se

puede limpiar la pantalla y escribir toda la conversación de nuevo, o montar un sistema que solamente añada nuevas líneas a lo que se muestra (buffer), indistintamente. Toda conversación mostrará un máximo de los 100 últimos mensajes entre los remitentes o, en caso necesario, el numero de mensajes nuevos sin leer, aunque estos superen los 100 últimos.

Interfaz de usuario cliente:

Para cambiar de destinatario y de conversación, el usuario deberá escribir primero una @ y luego el nombre de usuario del destinatario del cual quiere ver una conversación. Si el destinatario existe, se cambiará automáticamente el nombre tras la @ inicial. Si el destinatario no tiene aún ninguna conversación con el usuario, pero existe, se iniciará una nueva conversación vacía con él. Si el usuario no existe, se mostrará un mensaje de error.

Para ver una lista de todos los destinatarios disponibles, el usuario escribirá una @ con la palabra “lista” y se le imprimirá por pantalla la lista de todas las conversaciones existentes en ese momento en el lado del cliente (no de todos los usuarios existentes en el servidor. No es un listín telefónico).

Para salir del programa, se escribirá también una @ y la palabra “salir”, cerrándose el programa con ello.

Si el nombre de un remitente está mal escrito o el comando no se reconoce, sencillamente se escribirá un error por pantalla y se seguirá en la última conversación de usuario en la que se estaba.

Ejemplo de uso: mensaje para el usuario Alfonso, listado de conversaciones, nueva conversación y fin.

@Alfonso: esto es un ejemplo de escritura para Alfonso

@Alfonso: **@lista**

- 1-** @Alfonso
- 2-** @Manuel (4)
- 3-** @Alvaro (75)
- 4-** @Maria (10)

@Alfonso: **@Manuel**

@Manuel: me voy ya a casa Manuel. Te veo mañana.

@Manuel: **@salir**

FIN DEL PROGRAMA!

Para saber si un usuario existe y se puede hablar con él, será necesario primero contactar con el servidor, utilizando la conexión de envío de mensajes, y confirmación de que ese usuario existe en el servidor mediante la verificación de que el usuario que se quiere contactar está en la lista que devuelve el servidor.

Cada cliente tiene que tener un archivo de sus conversaciones mantenidas completo. Las ya iniciadas, solamente. En este archivo se guardarán todos los mensajes enviados y recibidos, en el orden en que se hayan emitido desde remitente y destinatario. Es decir, cada vez que se inicia el cliente, se cargará cada archivo de conversación en memoria como una lista, la cual deberá ser posible ordenar cronológicamente, independientemente de cuando se hayan recibido los mensajes (si hemos recibido un bloque de varias respuestas de un destinatario, mientras nosotros estábamos enviando, deberán intercalarse los mensajes según la fecha+hora de cada mensaje, para que el orden sea cronológico).

Cada vez que el cliente se ejecute, intentará conectar con el servidor. Pero si la conexión no está disponible, tendrá que guardar todos los mensajes que intente enviar en un archivo de texto por nombre de usuario de destino y el sufijo “_tmp”. Al estar de nuevo disponible la conexión, lo primero que se hará es enviar todos los mensajes pendientes de envío, con la confirmación del servidor, y vaciar ese archivo por completo.

Con cada nuevo mensaje, se deberá ordenar su posición en el archivo y guardarse en ambos archivos de texto, para que quede una copia de seguridad siempre, en caso de que el programa se cerrase solo.

El nombre de archivo local para cada cliente, comenzará siempre con el nombre del usuario de cliente, seguido del nombre de usuario de cada destinatario de los mensajes que haya enviado, separados por “_”.

Interfaz de usuario servidor:

El servidor no tiene interfaz de usuario. Opcionalmente, se podrá mostrar por pantalla el registro de las conexiones y desconexiones de los diferentes clientes, pero no es necesario.

Deberá escuchar conexiones por dos puertos diferentes, 666 y 999, cada uno gestionado por separado para los respectivos servicios de envío y recepción de mensajes.

Los envíos que reciba por el puerto 666 deberá entregarlos por el puerto 999 a aquellos destinatarios que hayan conectado previamente al 999 con peticiones de actualización de su bandeja de entrada.

Todos los mensajes deberán quedar registrados en archivos de texto locales a modo de log de conversaciones. Habrá un archivo de texto para cada pareja de interlocutores que existan. El nombre del archivo tendrá siempre primero el nombre del usuario del cliente que haya enviado el primer mensaje (quien haya iniciado la conversación), seguido del destinatario de ese primer mensaje, separados por “_”. Para localizar una conversación entre remitentes, será por tanto necesario buscar tanto los archivos de Usuario1_Usuario2 como de Usuario2_Usuario1, para poder localizar el archivo correcto, ya que estará solamente en una de esas dos opciones. Si no existe ninguno de los dos posibles nombres de archivo, será porque la conversación es nueva y habrá que crearlo nuevo, a partir del nombre de usuario del primer mensaje.

En los archivos de log del servidor y los clientes, todos los mensajes deben guardarse con todos los datos disponibles. Pero para mostrarle a los usuarios de los clientes las conversaciones, solo se les mostrará la fecha y hora de cada mensaje y el nombre del remitente del mismo.

Estos archivos guardarán las conversaciones en un formato separado por “;” (punto y coma) que permita trazar el estado de cada mensaje según los siguientes parámetros:

- **Origen:** nombre de usuario remitente de un mensaje
- **Destino:** nombre de usuario receptor de un mensaje
- **Timestamp:** fecha del mensaje en el equipo del remitente en formato SQL Date (YYYYMMDDhhmmss).
- **Estado:** string que define el estado del mensaje. Solo puede tomar los valores:
 - “ENVIADO” -> solo para el cliente, cuando esté en un archivo “_tmp”, SIN enviar al servidor.
 - “RECIBIDO” -> cuando lo ha recibido el servidor, pero no lo ha entregado al destinatario.
 - “ENTREGADO” -> cuando el destinatario ha conectado como cliente y se ha descargado los mensajes que tenía en estado “RECIBIDO”.
 - “LEIDO” -> solo para el cliente, cuando el usuario del destinatario haya abierto la conversación con los mensajes en estado “ENTREGADO” que hubiera.
 - “ERROR” -> mensaje de respuesta del servidor indicando en el texto qué error hay con el mensaje que se pretende enviar.

- **TiempoEstado:** es la marca de tiempo (como el timestamp) que tiene el estado actual de cada mensaje en formato SQL Date también.
- **Mensaje:** el contenido del mensaje en un string dentro de comillas dobles

De esta manera, con los estados de los mensajes, se podrá en todo momento trazar qué hay que hacer con cada mensaje.

Ejemplo de registro de conversación en el servidor de conversación entre dos remitentes, @Alfonso y @Manuel:

@Alfonso;@Manuel; 20251218145533;LEIDO;20251218145534;"Tio, vas a venir a clase o qué?"

@Alfonso;@Manuel; 20251218145536;LEIDO;20251218145537;"El profesor está pasando lista."

@Manuel;@Alfonso; 20251218145723;ENTREGADO;20251218145724;"Bro, me quedé dormido."

@Manuel;@Alfonso; 20251218145729;ENTREGADO;20251218145730;"Dile que estoy malo."

@Manuel;@Alfonso; 20251218151011;RECIBIDO; 20251218151012;"Bro, has leído mi mensaje?"

Un mensaje en estado ENVIADO solo debería estar en el cliente remitente del mensaje, mientras éste no haya podido conectar aún al servidor para entregarlo. Una vez recibido por el servidor, tanto en el archivo del servidor como en el archivo del cliente remitente, debería actualizarse el estado y pasar a estado RECIBIDO en ambos archivos.

En cambio, un mensaje marcado como estado LEIDO, deberá terminar marcado como tal en los archivos del destinatario que lo ha leído, en todos los archivos (la copia local del destinatario, la copia de log del servidor y en el archivo local del remitente) una vez haya conectado al servidor y actualice sus conversaciones.

Cada vez que un cliente se conecte, intentará actualizar sus conversaciones activas mediante una lista de tuplas de remitente-destinatario y el último timestamp que tiene de cada conversación. La lista de tuplas utilizará el mismo formato de un mensaje, pero alterando la información de los campos

El servidor le devolverá primero un mensaje de una línea, con el número de líneas que va a enviar después en mensajes individuales. Luego enviará una lista con todas las líneas de las diferentes conversaciones que con un timestamp igual o posterior al indicado por el cliente.

- **Origen:** será siempre el nombre de usuario que solicita actualizarse
- **Destino:** nombre de usuario de cada conversación mantenida. En el caso de solicitar un listado de conversaciones, este campo llevará un @ solamente.

- **Timestamp:** fecha del mensaje en el equipo del remitente (para el log) en formato SQL Date (YYYYMMDDhhmmss).
- **Estado:** string que solicita una acción en el servidor. Se reemplazará los estados de las conversaciones por los siguientes valores:
 - “*UPDATE*” -> solicita la actualización de todos los mensajes posteriores al timestamp provisto para los dos remitentes aportados. El timeStamp será 0 (0000000000000000 como YYYYMMDDhhmmss) para aquellas conversaciones nuevas que no se hayan descargado aun nunca. Se devolverá un primer mensaje con todos los campos igual que el recibido, pero con un número entero en el campo de texto indicando el numero de conversaciones que se enviarán, una a una, independientemente, tras recibir un OK al número.
 - “*LIST*” -> solicita el listado de todas las conversaciones que tiene el servidor para el cliente que lo solicita. Tendrá que ser el cliente el que revise su listado local con el listado que le da el servidor, para detectar qué conversaciones no tiene en local (las nuevas) y solicitarle al actualizacion al servior. La lista de de contactos se devolverá como una serie de mensajes individuales con los campos de los usuarios de origen y destinatario en el orden normal, todos los campos timestamp a 0 (0000000000000000 como YYYYMMDDhhmmss), el campo estado como LIST igual que se recibió y el campo de texto con el numero de menajes pendiente de entregarse al remitente para cada conversación.
- **TiempoEstado:** es la marca de tiempo (como el timestamp) correspondiente al ultimo timestamp que se tiene de una conversación entre remitente y destinatario. Para el modo LIST se reemplazará el timestamp con todos los valores de fecha y hora a 0 (0000000000000000 como YYYYMMDDhhmmss)
- **Mensaje:** para los mensajes con el servidor, este campo puede ir vacío (con “” solamente) cuando no haga falta. Pero para actualizaciones con el servidor, este campo contendrá un umero para informar a los clientes del número de datos que se van a devolver desde el servidor (una lista de conversaciones, por ejemplo).

Para cada mensaje de coordinación entre un cliente y el servidor, se establecerá una conexión TCP que, en toda comunicación, verificará que un mensaje ha llegado con un “OK” o un “KO” de vuelta. Todas las conexiones tendrán un timeout de 10 segundos (configurable), de manera que si un mensaje no devuelve un “OK” o un “KO” pasado el timeout, la conexión deberá cerrarse y el proceso que estuviera en curso deberá empezar de nuevo desde el inicio.

Ejemplo de petición, solicitada el día 18/12/2025 a las 12:15:33, de actualización de los mensajes entre @Alfonso y @Manuel (o @Manuel y @Alfonso indistintamente) que hayan ocurrido desde el 17/12/2025 a las 04:10:00.

@Alfonso;@Manuel; 20251218141533;UPDATE;20251217041000;””

Ejemplo de petición, solicitada el día 18/12/2025 a las 15:15:15, de actualización la lista de mensajes existentes para el cliente @Alfonso, con cualquier destinatario.

@Alfonso;@; 20251218151515;LIST;0000000000000000;””

Ejemplo de comunicación completa de un cliente con el servidor, por el puerto 999, para recibir 3 mensajes de 1 conversación existente y una nueva.

CLIENTE		SERVIDOR	Comentarios
LOGIN:Alfonso:123456	→		Intento fallido de login
	←	KO	Mensaje de KO porque el login es incorrecto (sin explicaciones adicionales).
LOGIN:Alfonso:12345678	→		Intento correcto de login
	←	OK	Mensaje de OK. Se inicia comunicación.
@Alfonso;@Manuel; 20251218141533;UPDATE; 20251217041000;””	→		A las 14:15:33 del 18/12/2025 el cliente (@Alfonso) pide actualización de los mensajes nuevos para su conversación con @Manuel, desde la última vez que conectó alas 4:10:00 del 17/12/2025
	←	OK	Mensaje recibido OK del servidor.
	←	@Alfonso;@Manuel; 20251218141534;UPDATE; 20251217041000;”3”	Mensaje a las 14:15:34 del 18/12/2025 de respuesta, indicando que hay 3 mensajes nuevos, posteriores a las 4:10:00 del 17/12/2025
OK	→		Mensaje recibido OK del cliente
	←	@Alfonso;@Manuel; 20251218145535;LEIDO; 20251217210512; ”Tio, vas a venir a clase o qué?”	Primer mensaje de los 3 que se van enviar: Manuel leyó a las 21:05:12 del 17/12/2025 un mensaje que estaba pendiente de leerse.
OK	→		Mensaje recibido OK
	←	@Alfonso;@Manuel; 20251218145536;LEIDO; 20251217210513; ”El profesor está pasando lista.”	Segundo mensaje de los 3 que se van enviar: Manuel leyó a las 21:05:13 del 17/12/2025 otro mensaje que estaba pendiente de leerse.
OK	→		Mensaje recibido OK

	←	@Manuel;@Alfonso; 20251218145538;ENTREGADO; 20251218093307; "Bro, me quedé dormido."	Tercer mensaje de los 3 que se van enviar: Manuel escribió de vuelta a las 09:33:07 del 18/12/2025 otro mensaje que estaba pendiente de entregarse a @Alfonso y que no se ha hecho hasta ahora a las 14:55:38 deñ 18/12/2025, razón por la cual se recibe ya en ese estado ENTREGADO.
OK	→		Mensaje recibido OK
@Alfonso;@; 20251218151515;LIST; 0000000000000000;"	→		El cliente solicita la lista de remitentes (conversaciones existentes en el servidor para él)
	←	OK	Mensaje recibido OK
	←	@Alfonso;@; 20251218151515;LIST; 0000000000000000;"2"	El servidor contesta que tiene una lista de 2 conversaciones existentes en las que está @Alfonso presente
OK	→		Mensaje recibido OK
	←	@Manuel;@Alfonso; 0000000000000000;LIST; 0000000000000000;"0"	El servidor informa que la conversación entre @Alfonso y @Manuel tiene 0 mensajes sin entregar
OK	→		Mensaje recibido OK
	←	@Carlos;@Alfonso; 0000000000000000;LIST; 0000000000000000;"5"	El servidor informa de que hay una conversación entre @Carlos y @Alfonso con 5 mensajes a la espera de ser entregados a @Alfonso.
OK	→		Mensaje recibido OK
@Alfonso;@Maria; 20251218141533;ENVIADO; 20251217041000;"Hola María. Tu vas a ir hoy a clase?"	→		Alfonso intenta iniciar una conversación con María, sin saber si está registrada en este servidor de whatsapp.
	←	KO	Mensaje recibido KO (la comunicación ha sido correcta, pero María no existe en el servidor. No es un usuario registrado, porque no usa este whatsapp)
	←	@Alfonso;@Maria; 20251218141533;ERROR; 0000000000000000;"Usuario no registrado."	El servidor informa de que el usuario María no existe. Con el anterior KO el cliente deberá esperar un mensaje con el error y no enviar ninguna nueva comunicación hasta recibirlo.
	→	OK	Mensaje recibido OK

Tras cada flujo de comunicación, se puede dejar la conexión abierta o cerrarla, avisando con un último mensaje "FIN", desde el cliente o desde el servidor, que será respondido con un "OK" desde el otro lado. En cualquier caso, debe estar siempre contemplada la gestión del cierre o corte de una conexión para que el sistema no falle y siga permitiendo nuevas conexiones entrantes de cualquier cliente.