🔌 Protocolo WebSockets

Este protocolo se construye asumiendo que se usarán **Websockets** para la comunicación entre cliente y servidor ¿Qué son Websockets? Es otro protocolo así como el HTTP, que hemos usado para hacer API para cursos anteriores, que permite una comunicación bidireccional y continua entre un cliente y servidor más info de que son aquí 🔗.

¿Por qué usar WebSockets?

Son un protocolo perfecto para aplicaciones en tiempo real, como: videojuegos online, videollamadas… **y chats!**

Funcionalidades de este protocolo

- Registrar un usuario nuevo (Un usuario es solo un nombre)
- Obtener los usuarios conectados actualmente.
- Obtener la información de un usuario por su nombre (nombre y estatus).
- Cambiar el estatus (ACTIVO, OCUPADO, INACTIVO).
- Chat general y mensajes directos.

✉️ Los mensajes

La unidad mínima de transmisión de información será un “`mensaje`” (no confundir con los mensajes del chat). Un mensaje es realmente un array de unsigned bytes, es decir grupos de 8 bits, que representan números positivos de 0 a 255.

En C/C++ es más fácil programar si se sabe el tamaño de las cosas! Por lo que nuestro protocolo siempre dice primero el tamaño de cada campo que debe parsear. **Cada `mensaje` sigue un formato específico, compuesto de varias partes:**

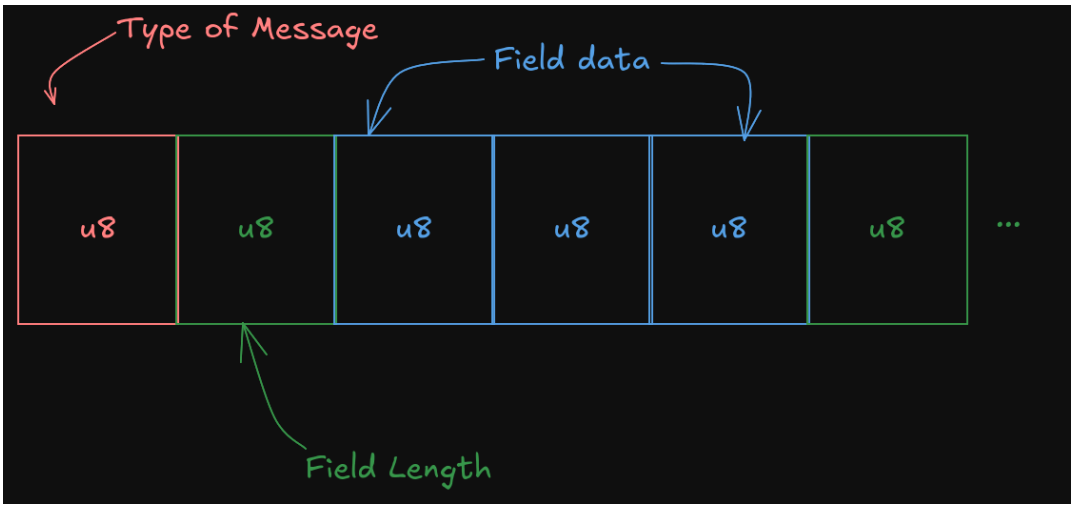
1. **Código de `mensaje` (rojo):**
   Cada evento posible en la app (registrar un usuario, enviar un chat, cambiar tu estado, etc.) será identificado por un ID que puede ir de 1 a 255, y será el primer byte del mensaje. Son como el análogo de las rutas en una REST API normal.

2. **Tamaño del campo (verde):**
   Si el mensaje requiere de argumentos, cada uno deberá especificar el tamaño de la data que se envía, previo a la información en sí, esto permitirá saber de antemano cuánto espacio alocar para la información que se recibe. **Puede especificarse una longitud de 1 a 255 bits (un byte).**

3. **Datos (azul):**
   La información en sí de cada parámetro.

!! NOTA: La información de un mensaje será enviada en formato binario.

❗ **NOTA:** Si los mensajes envían argumentos, **estos deben ser enviados en el ORDEN especificado por este protocolo.**



La idea principal es poder simplemente obtener un número, por ejemplo 50 y crear un array de 50 bytes para guardar la data del Field del mensaje!

¡La principal desventaja es que esta arquitectura nos limita a que como máximo los parámetros guardarán 255 bytes de data! **Es decir, que se pueden enviar chats con 255 carácteres máximo, como Twitter.** Aumentar la capacidad a 16bits solamente para el campo mensaje es una opción, pero lamentablemente esto podría llevar a tamaños mayores a los que ciertas memorias stacks soportan (65 Kilobytes). Y no creemos que nadie quiere ponerse a trabajar con buffers :p

♻️ Flujo general de la app

👤 Los Roles

El flujo general de un usuario interactuando con el servicio consta de 2 roles principales: cliente y servidor. **Ambas partes pueden tanto ENVIAR como RECIBIR `mensajes`.**

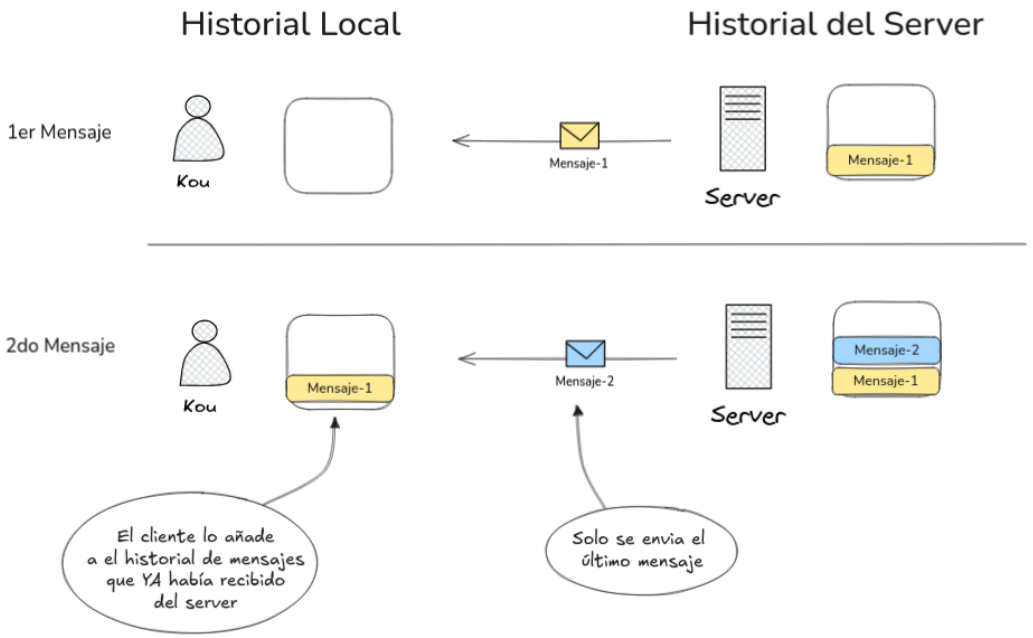
Servidor

Es la "*fuente de verdad*" del programa, **llevará el historial de cada chat entre 2 o más usuarios.** Cada vez que reciba un mensaje de un cliente, este lo agrega al historial y re-enviará a los demás usuarios el cambio.

❗ **NOTA: El servidor se hace responsable de identificar el usuario que lanza cada evento.** En websockets cada conexión es un objeto con su propio state y demás, así que cuando creas la conexión yo estaba pensando guardar de una vez el username, pero todo esto es detalle de implementación.

Cliente

Por otro lado, el cliente se encargará de **mostrar o ocultar los mensajes que recibe del servidor.** Este también deberá llevar su propio historial de los mensajes en un chat, **la idea es construir el historial de mensajes gradualmente** con los mensajes que se recibe del servidor, en vez de enviar el historial completo cada vez. El siguiente diagrama mostraría el flujo:



❗ **NOTA:** El protocolo contiene un "`mensaje`" para poder solicitar TODO el historial de chats, pero SÓLO se recomienda usarlo cuando el cliente, por varias razones, NO cuenta

con el historial previo de mensajes, ejemplo: un usuario es recién registrado y quiere acceder a los mensajes pasados del chat grupal.

🏁 Las Fases

Además de los roles, **el ciclo de vida de un cliente en la app consta de 3 fases:**

1. Creación de conexión
2. Transmisión de mensajes
3. Cierre de sesión

Se explicarán más a profundidad a continuación UwU

1. Creación de conexión

Todo comienza con una simple request HTTP a una URL con la siguiente forma:

```
ws://<host>:<port>?name=<username>
```

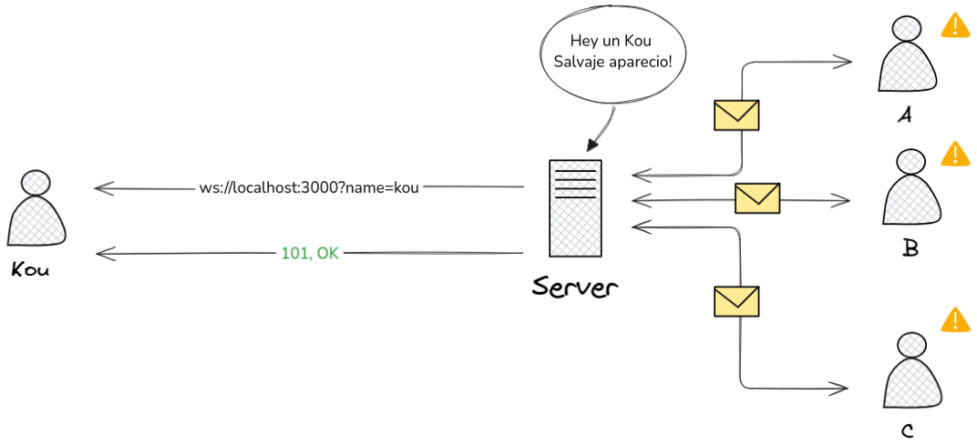
Donde `<username>` es el nombre del usuario con el que te quieres registrar en nuestro servidor de chat.

!! NOTA: El <username> no puede ser “~”. Este nombre está reservado para el chat general.

❗ **NOTA:** Si hay un usuario ya existente con ese nombre EN LÍNEA o el nombre es un string vacío la conexión **debe ser rechazada**!

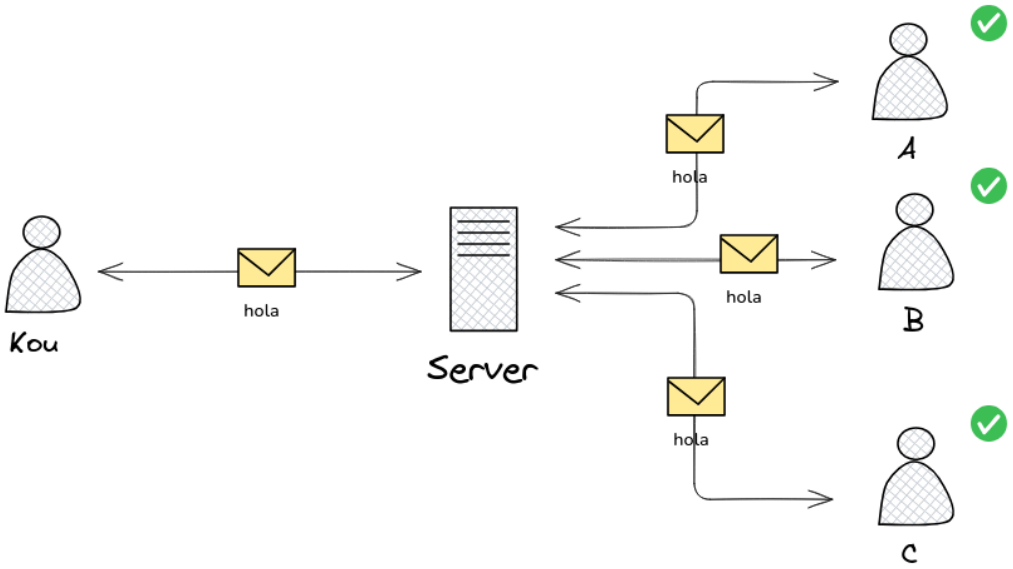
❗ **NOTA:** Si el usuario ya había sido registrado PERO no había un cliente activo con su nombre, la conexión **será aceptada**, **el cliente se hace responsable de solicitar el historial de mensajes pasados para cada chat.**

Al ser una request HTTP normal, se puede rechazar con código 400 en caso sea error de parte del usuario y código 500 si es error interno del servidor. Si todo va bien, la conexión se vuelve tipo `websocket`, y se notifica a los demás usuarios del nuevo integrante (en caso sea nuevo) o se actualiza su estado a “Activo” (en caso ya existía pero se había desconectado).



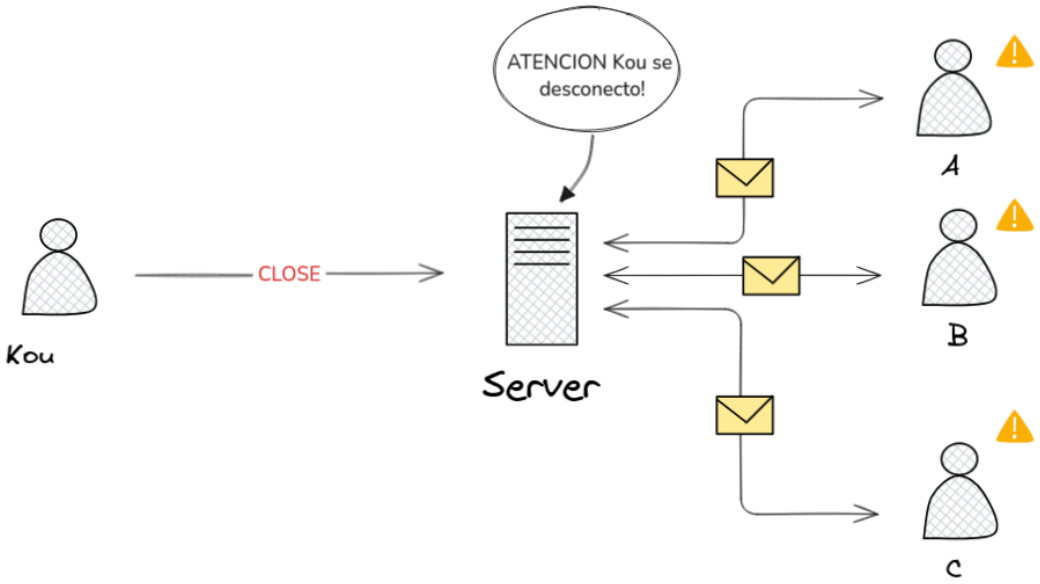
2. Transmisión de `mensajes`

Durante esta fase el cliente y servidores enviarán `mensajes` de todo tipo. No todos los mensajes son iguales, dependiendo del tipo se envia una respuesta al que envio el mensaje original o todos los usuarios **incluso el que inició la petición**. Por favor leer bien a quien se envia que tipo de mensaje.



3. Cierre de sesión

Por último, si un cliente cierra su conexión al servidor, s**u usuario y chats se mantienen dentro del servidor**, pero su estado cambia a `desconectado` . Y se notifica a los demás usuarios del cambio de estado.

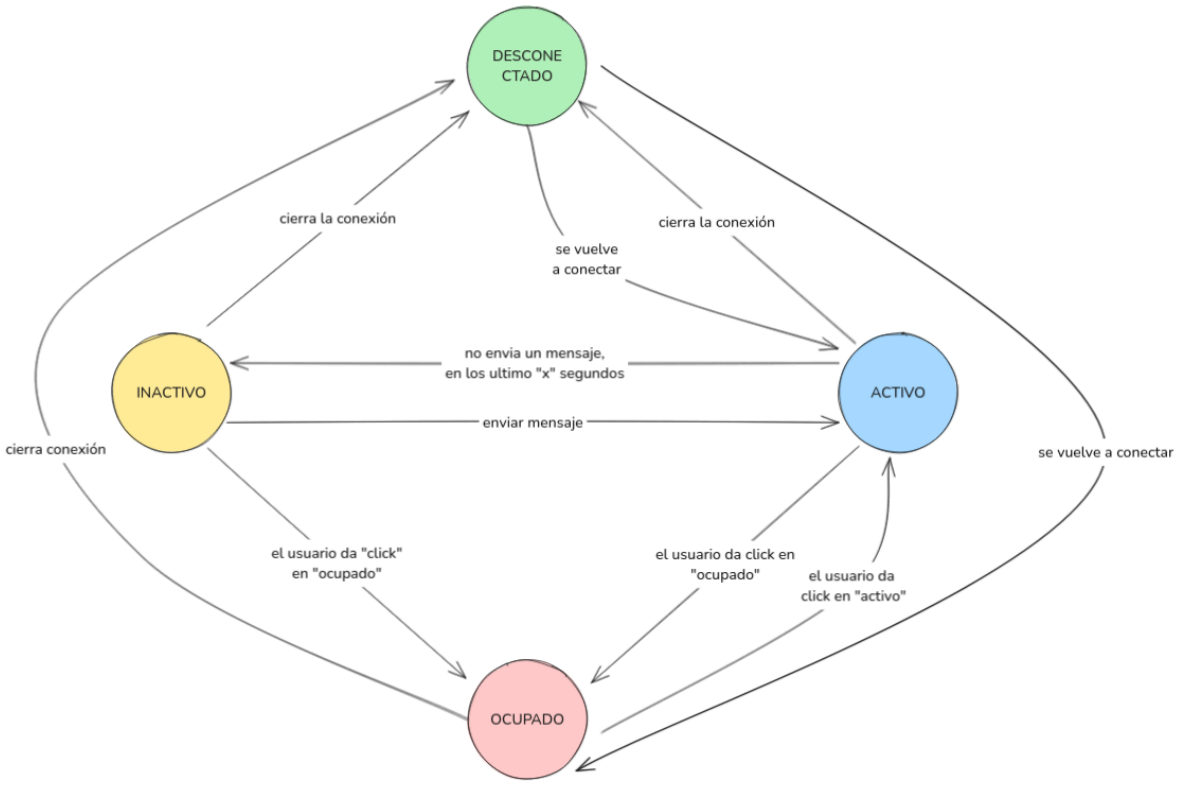


🚦 Estados de un Usuario

Un usuario puede estar en 4 estados, el servidor administra y guarda los estados de cada usuario.

- **Activo**: El usuario ha mandado chats recientemente.
- **Inactivo**: El usuario está conectado pero no ha tenido actividad en “x” segundos.
- **Ocupado**: El usuario voluntariamente decidió moverse a este estado. Aquí, el usuario seguirá recibiendo nuevos chats, PERO el cliente será responsable de ocultarlos, hasta que cambie de estado.
- **Desactivado**: Se cortó la conexión entre el cliente y servidor, la información del usuario no será borrada.

El dibujito de abajo explica como se transiciona de un estado a otro.


DESCONE
CTADO
cierra la conexión
se vuelve
a conectar
cierra la conexión
INACTIVO
no envia un mensaje,
en los ultimo "x" segundos
ACTIVO
enviar mensaje
cierra conexión
se vuelve a conectar
el usuario da "click"
en "ocupado"
el usuario da click en
"ocupado"
el usuario da
click en "activo"
OCUPADO

✉️ Mensajes en el Protocolo

Y por último LO MÁS IMPORTANTE ¿Qué mensajes existen?! Aquí la lista está dividida por cada rol.

❗ **NOTA:** Si los mensajes envían argumentos, **estos deben ser enviados en el ORDEN especificado por este protocolo.**

Cliente

Mensajes que envía el cliente al servidor:

<table>
<tr><th>ID Tipo</th><th>Descripción</th><th>Campos</th></tr>
<tr><td>1</td><td>Listar usuarios conectados actualmente</td><td>No tiene campos.</td></tr>
<tr><td>2</td><td>Obtener un usuario por su nombre</td><td>Solamente tiene un campo, indica el nombre del usuario que se desea obtener.
1) Username
<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido campo anterior</td></tr></table></td></tr>
<tr><td>3</td><td>Cambiar estatus de un usuario</td><td>1) Username: Usuario al que se quiere cambiar el status.
2) Status: El estatus debe ser uno de los siguientes 4 números

0: DISCONNECTED
1: ACTIVE
2: BUSY
3: INACTIVE
<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Status</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido campo anterior</td><td>1 byte</td></tr></table></td></tr>
</table>

<table>
<tr><td>4</td><td>Mandar un mensaje</td><td>1) Username Dest : nombre del usuario al que se envía el mensaje. Usar “~” para referirse a Chat General. En caso de enviarse un mensaje a un usuario desconectado el server retornará un error.
2) Mensaje: Contenido del mensaje

<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Len Mensaje</th><th>Mensaje</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td><td>1 byte</td><td>Definido antes</td></tr></table></td></tr>
<tr><td>5</td><td>Obtener Mensajes</td><td>Mensaje que puede usar el cliente para obtener el historial del chat que quiere visualizar.

1) Chat : es decir, el nombre del usuario con el que entabla conversación y se quiere saber su chat.

<table><tr><th>Tipo</th><th>Len Chat</th><th>Chat</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td></tr></table></td></tr>
</table>

Servidor

Mensajes que envía el servidor al/los clientes:

<table>
<tr><th>ID Tipo</th><th>Descripción</th><th>Campos</th></tr>
<tr><td>50</td><td>ERROR</td><td>Solamente se envía al cliente que hizo la petición. No tiene ningún campo, el siguiente u8 en el array indica el error. Más adelante en el documento puedes encontrar una tabla con el listado de errores y el número asociado a ellos.

<table><tr><th>Tipo</th><th>Tipo error</th></tr><tr><td>1 byte</td><td>1 byte</td></tr></table></td></tr>
</table>

<table>
<tr><td>51</td><td>Respuesta a: Listar usuarios registrados.</td><td>Solamente se envía al cliente que hizo la petición.

1) Número de usuarios: Número de usuarios que se enviaron.
2) Usuarios : Después, se tiene un listado de usuarios, cada usuario tiene dos campos, el primero es su nombre y el segundo es su estatus.

<table><tr><th>Tipo</th><th>Num Usuarios</th><th>Len User 1</th><th>User 1</th><th>Status User 1</th><th>Demás usuarios</th></tr><tr><td>1 byte</td><td>1 byte</td><td>1 byte</td><td>Definido antes.</td><td>1 byte</td><td>…</td></tr></table></td></tr>
<tr><td>52</td><td>Respuesta a: Obtener un usuario por su nombre</td><td>Solamente se envía al cliente que hizo la petición.
1) Usuario: Usuario del que se obtiene información
2) Status:

<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Status</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td><td>1 byte</td></tr></table></td></tr>
<tr><td>53</td><td>Usuario se acaba de registrar</td><td>Este mensaje se envía a todos los clientes ya conectados cuando un nuevo usuario es registrado en la app!
1) Username
2) Status

<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Status</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td><td>1 byte</td></tr></table></td></tr>
<tr><td>54</td><td>Usuario cambió estatus</td><td>Este mensaje se envía a todos los clientes (incluido el que hizo la petición de cambiar estatus) cuando un usuario cambia su estatus!
1) Username
2) Status

<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Status</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td><td>1 byte</td></tr></table></td></tr>
</table>

<table>
<tr><td>55</td><td>Recibió mensaje</td><td>Este mensaje se envía a los clientes interesados cuando reciben un mensaje. SE ENVIA TAMBIEN al usuario que mandó el mensaje original

1) Origen: representa el chat (usuario) que envió el mensaje
2) Mensaje: Contenido del mensaje.

<table><tr><th>Tipo</th><th>Len Username</th><th>Username</th><th>Len Mensaje</th><th>Mensaje</th></tr><tr><td>1 byte</td><td>1 byte</td><td>Definido antes</td><td>1 byte</td><td>Definido antes</td></tr></table></td></tr>
<tr><td>56</td><td>Respuesta a: Obtener historial de mensajes</td><td>Se envía solamente al cliente que hizo la petición.

1) Num Mensajes: Número de mensajes que se están enviando.
2) Mensajes: Lista de mensajes uno a la par del otro. Donde la primera sección es el usuario que lo escribió y la segunda es el mensaje en sí.

<table><tr><th>Tipo</th><th>Num mensajes</th><th>Len User1</th><th>User1 Mensaje 1</th><th>Len mensaje 1</th><th>Mensaje 1</th><th>Demás mensajes</th></tr><tr><td>1 byte</td><td>1 byte</td><td>1 byte</td><td>Def. antes</td><td>1 byte</td><td>Def. antes</td><td>…</td></tr></table></td></tr>
</table>

Errores

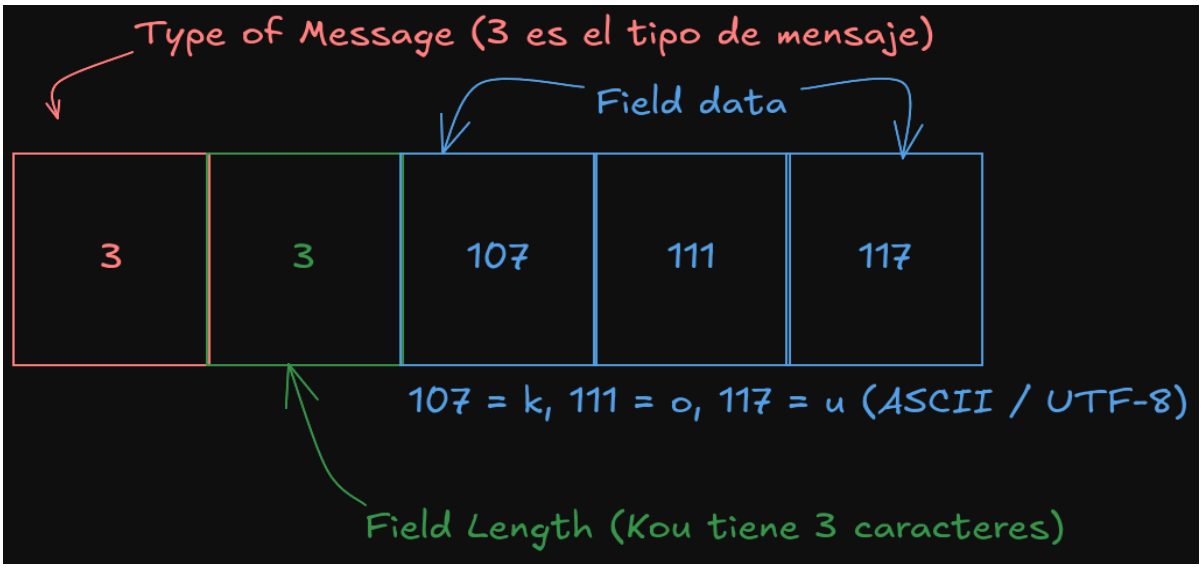
Número	Descripción
1	El usuario que intentas obtener no existe.
2	El estatus enviado es inválido.
3	¡El mensaje está vacío!
4	El mensaje fue enviado a un usuario con estatus desconectado

Ejemplos

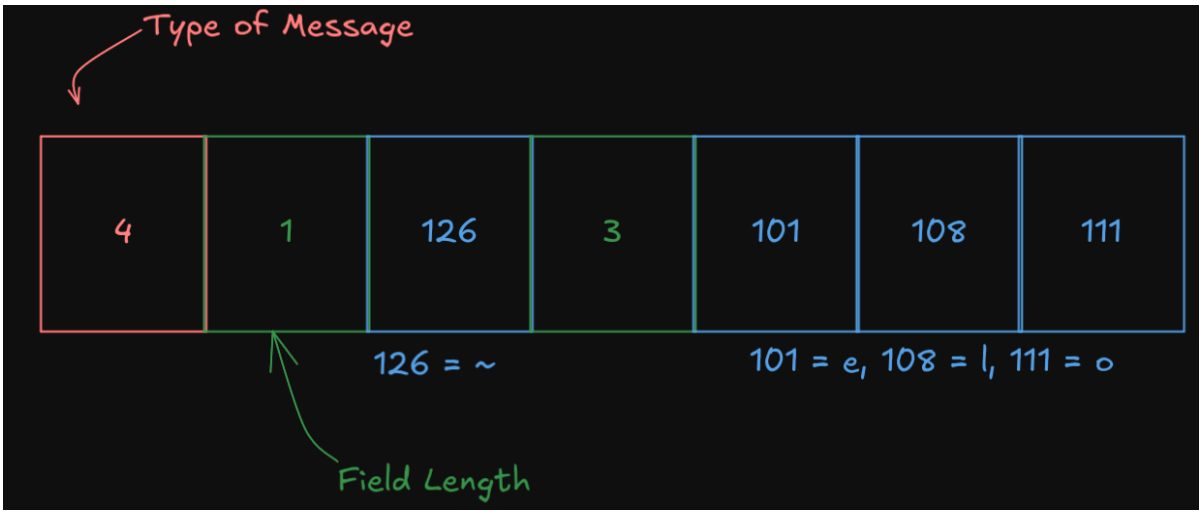
Sé que solo leyendo la descripción es muy probable que no se entienda, así que ¡Aquí hay algunos mensajes de ejemplo explicados con dibujitos!

Mensajes que el cliente envía al servidor

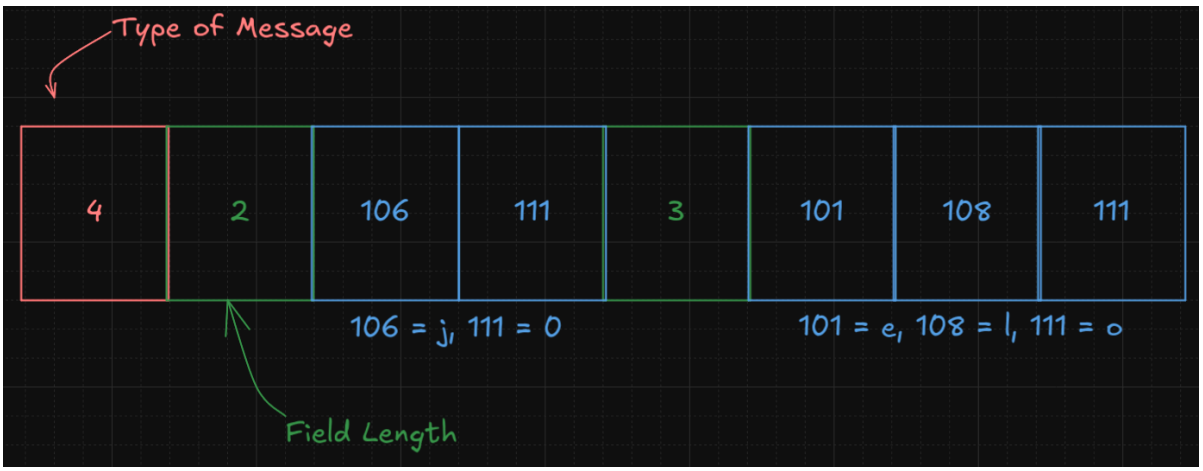
Obtener información de un usuario registrado como kou por su nombre:



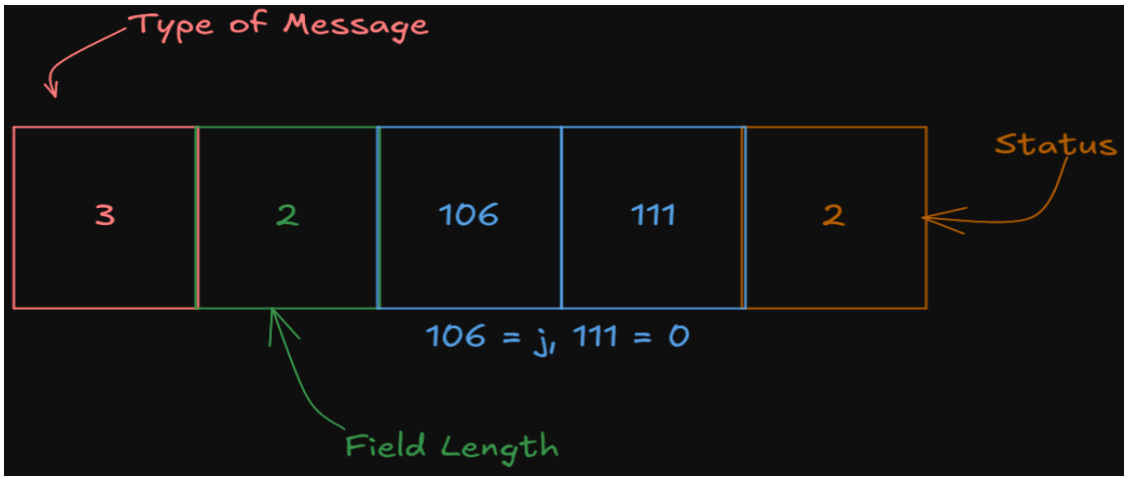
- Mandar el mensaje “elo” al chat general:



- Mandar el mensaje "elo" al chat de un usuario llamado Jo:

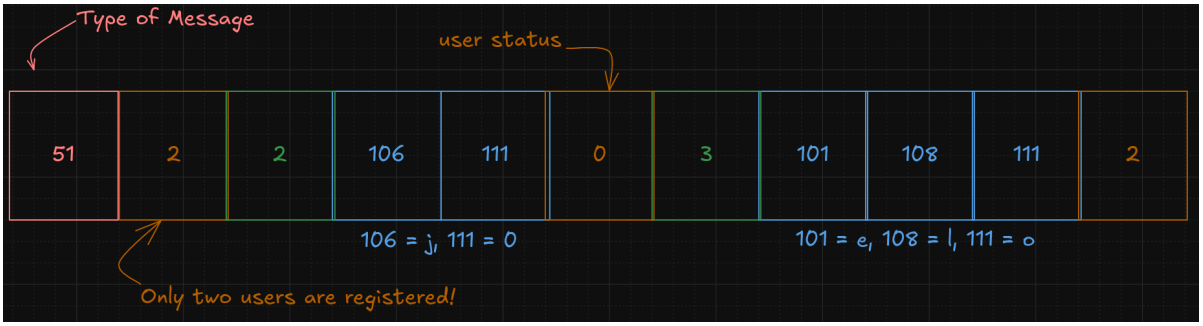


- Cambiar estatus del usuario jo a “OCUPADO”:

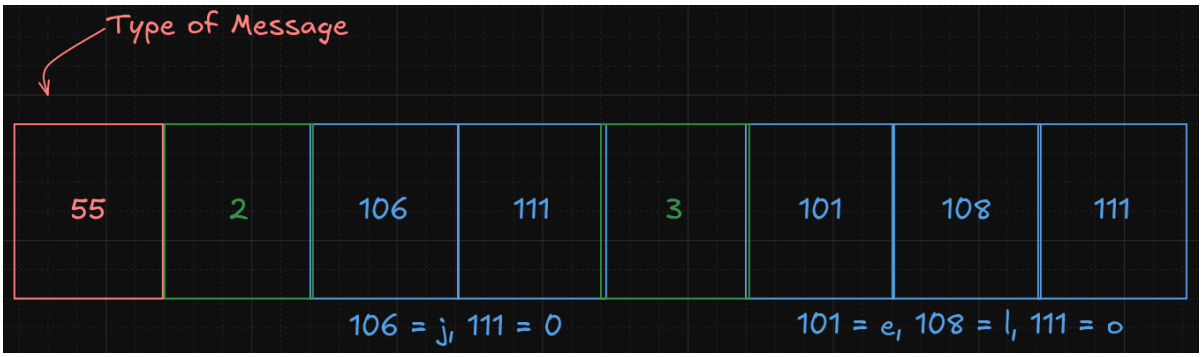


Mensajes que el servidor envía al cliente

- Listar usuarios registrados en el servidor. En el ejemplo tenemos dos usuarios: “Jo” y “Elo”



- El usuario “kou” recibió un mensaje de “elo” de parte del usuario “jo”.



Nótese que el mensaje no incluye a “kou”, puesto que este mensaje solamente le debería de llegar a la conexión asociada al usuario “kou” y a la conexión asociada a “jo”.