Manuel Julian Esteban Protocol -- MJEP/v.1

Resumen

Este documento describe el *Manuel Julian Esteban Protocol* (MJEP), un protocolo de nivel de aplicación diseñado para la comunicación cliente-servidor mediante el intercambio de mensajes simples y cifrados. Se explican detalladamente las funciones del protocolo, como el envío y recepción de mensajes, las acciones que ejecutan tanto el cliente como el servidor, y las reglas y notaciones que deben seguirse para su implementación. MJEP facilita la conexión y el intercambio de datos, priorizando la fiabilidad y el correcto orden de los mensajes. El protocolo opera sobre TCP/IP en el puerto 8080/TCP, permitiendo una comunicación eficiente y segura para múltiples clientes conectados a un servidor.

Índice

Introducción

El protocolo **Manuel Julian Esteban Protocol (MJEP)** es un protocolo de nivel de aplicación (capa 5 según TCP/IP) para el intercambio de mensajes simples. En la primera versión del protocolo encontramos que se encarga de realizar dicho intercambio de

mensajes con un cifrado a base de texto con un encabezado con el cual se toman acciones que se toman desde las dos perspectivas que el mismo protocolo define, el *cliente* y el *servidor*.

Este protocolo define describe características para poder generar un intercambio de mensajes por medio de la red de manera consistente y cómoda para quienes lo utilicen. Se definen como tal dos entidades para su implementación, el cliente y el servidor, donde más adelante las conoceremos más a fondo, aunque como tal para su correcto funcionamiento es necesario la presencia de la entidad servidor como de dos entidades clientes.

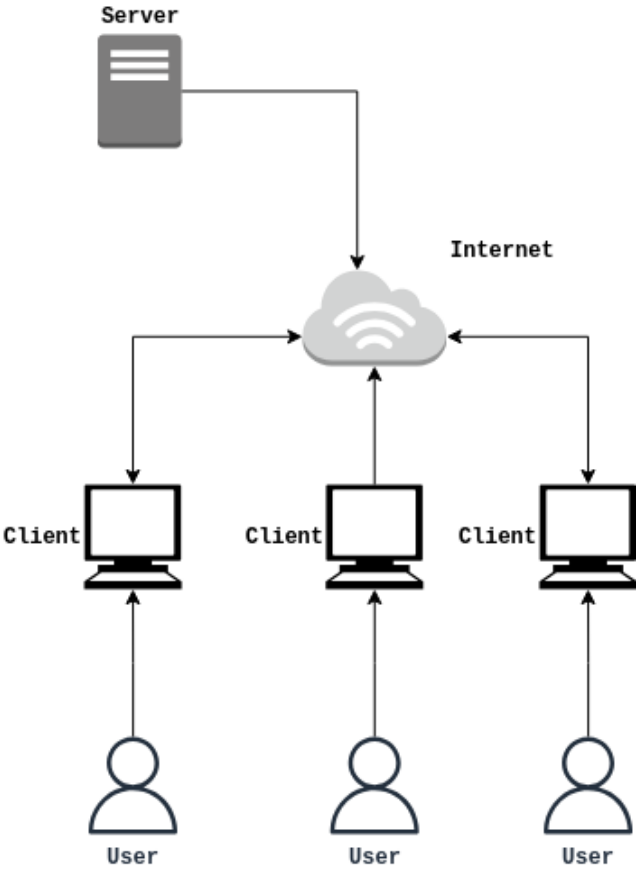
Este protocolo pretende facilitar la comunicación de todo tipo de personas a través del internet, y con la posibilidad de poder configurar su uso para distintos niveles, tanto como una perspectiva global como un local, priorizando la comodidad y la fiabilidad de la información.

Definición General

Este protocolo se encarga de trabajar a un nivel de capa de aplicación soportado en el protocolo de capa de transporte TCP, y a su vez sobre los protocolos IP (capa de red) y Ethernet (capa de enlace). Este protocolo se soporta sobre TCP, ya que se prioriza la fiabilidad de la información para la comunicación de los dos usuarios, los mensajes debe de llegar en el orden que son mandados al destino y no se permite que información pueda quedar extraviada en este intercambio.

En una descripción del protocolo encontramos que un cliente envía un mensaje al servidor, el protocolo realiza su respectivo trabajo de encapsular la información dadas ciertas circunstancias (momento de ejecución, palabras clave, etc.) cuando el servidor recibe dicho mensaje analiza el encabezado y de acuerdo a su análisis toma diferentes acciones, como lo puede ser el enviar el mensaje a un segundo cliente que también esté conectado con el servidor. Este proceso lo explicaremos más a detalle en el transcurso de este documento.

El mismo protocolo se encarga de realizar la conexión entre el servidor y los clientes, se plantea la existencia de un único servidor para múltiples clientes, además es fundamental primero que el servidor se encuentre ejecutándose para que los clientes se puedan conectar a él. Se soportan tantos clientes como el servidor pueda soportar.



El protocolo se encuentra definido para ejecutar su funcionalidad en el puerto **8080/TCP**.

Terminología

- **Cliente:** Es la entidad encargada de iniciar una comunicación con el servidor. Es responsable de enviar solicitudes de datos o servicios y espera una respuesta del servidor.

- **Servidor:** Es la entidad encargada de recibir y responder a las solicitudes del cliente. Su función es procesar esas solicitudes y devolver, y/o redirigir, los datos o servicios solicitados.

- **Mensaje:** Es la unidad de datos *(PDU)* que se intercambia entre el cliente y el servidor. Está compuesto de un encabezado (metadatos) y un cuerpo (contenido de datos).

- **Socket:** Es un punto final de la comunicación entre las dos entidades. Incluye una dirección IP y un puerto, permitiendo la transferencia de los datos correspondientes.

- **Encabezado:** Es la parte de un mensaje que contiene metadatos necesarios para la correcta interpretación y manejo del mensaje.

- **Encapsular:** Proceso de añadir datos (encabezados) a un mensaje para ser transmitido adecuadamente a través de la red.

- **Des encapsular:** Proceso inverso a la encapsulación. Implica eliminar los encabezados añadidos para realizar la correcta extracción para la interpretación respectiva del mensaje.

Notación y Gramática

Sintaxis y semántica:

El protocolo MJEP utiliza un encoding tipo texto que sigue la siguiente estructura, tanto para el cliente como para el servidor:

HEADER BODY → Ejemplo: REGISTER Esteban Muriel
header body

Headers servidor:

REGISTER: Por parte del servidor, se encarga de tomar el valor encapsulado en el body y lo almacena en una estructura donde se guarde y registre este “username” dentro del servidor. Además, se encarga de encapsular otro mensaje tipo REGISTER el cual contiene el nombre de los otros usuarios previamente registrados, y se lo manda al cliente.

CONNECT: Se encarga de meter a un chat privado a 2 clientes si es que ambos lo desean. Además, si el cliente lo requiere, reenvía los usuarios conectados al cliente que lo solicite.

EXIT: Una vez un cliente este conectado en un chat, EXIT se encarga de desconectar de este al cliente que lo desee. Cuando el primer cliente se desconecte del chat, le notifica al segundo cliente que el otro usuario se ha desconectado del chat.

DISCONNECT: Se encarga de desconectar a un cliente del servidor, sacándolo de la estructura donde se almacenan los usuarios registrados en el servidor. A su vez, si la desconexión se hace dentro de un chat, se le notifica al otro usuario de la salida del desconectado.

Headers cliente:

REGISTER: Le notifica al servidor el nombre con el que este cliente se desea registrar en la aplicación.

NACK: La llegada de un NACK al cliente, significa que algo salió mal en algo relacionado con este usuario, por lo que se encarga de finalizar con la ejecución de este cliente, sacándolo del servidor.

MESSAGE: Se encarga del envío y de la recepción de los mensajes que se intercambian cuando 2 clientes están conectados en un mismo chat.

EXIT: Le notifica al servidor que el cliente se desea salir del chat en el que está actualmente.

DISCONNECT: Le notifica al servidor que el cliente se desea desconectar de la aplicación y finaliza la ejecución del cliente que ha solicitado la desconexión.

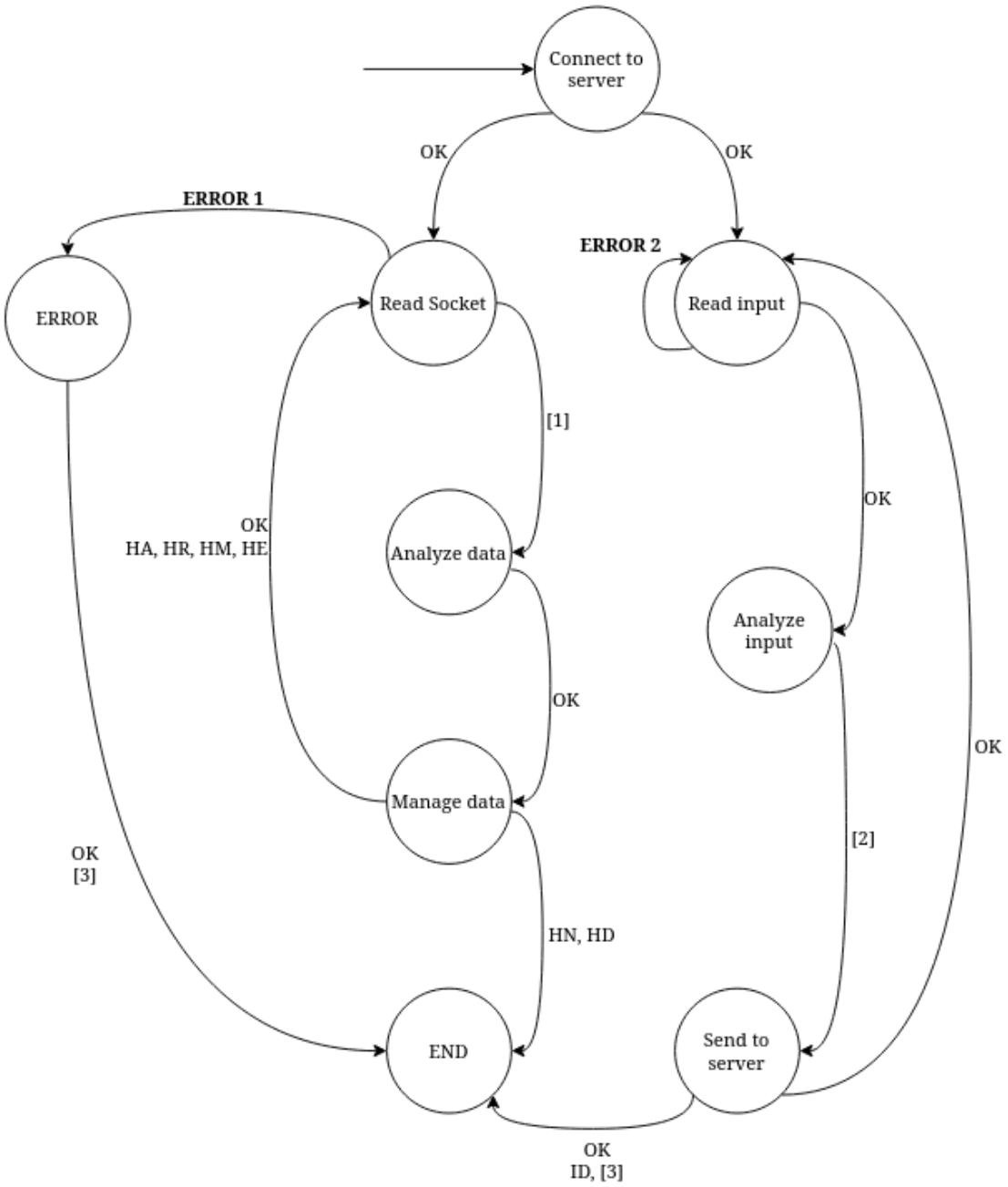
Funcionamiento

Cliente:

Tabla

STATUS	Input											
	Header ack	Header nack	Header register	Header messaje	Header exit	Header disconnect	[1] Uncapsulate action	[2] Encapsulate action	[3] Close connection action	ERROR 1 recv error	ERROR 2 input null	OK
Read socket	-	-	-	-	-	-	analyze DATA	-	-	Error	-	-
Analyze data	-	-	--	-	-	-	-	-	-	-	-	MANAGE DATA
Manage data	Read Socket	END	Read Socket	Read Socket	Read Socket	END	-	-	-	-	--	Read Socket
Read input	-	-	-	-	-	-	-	-	-	--	Read input	Analyze input
Analyze input	-	-	-	-	-	-	-	Sent to server	-	-	-	-
Send server	-	-	-	-	-	-	-	-	END	-	-	--
Error	-	-	-	-	-	-	-	-	END	-	-	END
End	-	-		-	-	-	-	-	-	-	-	-

M.E.F


Connect to server
OK
OK
ERROR 1
ERROR 2
ERROR
Read Socket
Read input
[1]
OK
OK
HA, HR, HM, HE
Analyze data
Analyze input
OK
OK
Manage data
[2]
OK
[3]
HN, HD
END
Send to server
OK
ID, [3]

Servidor

Tabla

STATUS	Input											
	Header connect	Header register	Header messaje	Header exit	Header disconnect	[1] Uncapsulate action	[2] Encapsulate action	[3] Close connection action	[4] Time start action	ERROR 1 recv error	ERROR 2 TIMEOUT, INVALID BODY	OK
listening connections	-	-	-	-	-		-	-			-	Read socket
read socket	-	--	-	-	-	Analyze data	-	-	Analyze data	Error	-	MANAGE DATA
Manage data	--	-	-	-	END	-	Send client	END	-	-	--	END
Analyze data	Manage data	Manage data	Manage data	Manage data	Manage data	-	Sent to server	-	-	-	Error	Manage data
Send client	-	-	-	-	-	-	-	-	-	-	-	Read socket-
Error	-	-	-	-	-	-	-	END		-	-	END
End	-		-	-	-	-	-	-		-	-	-

M.E.F

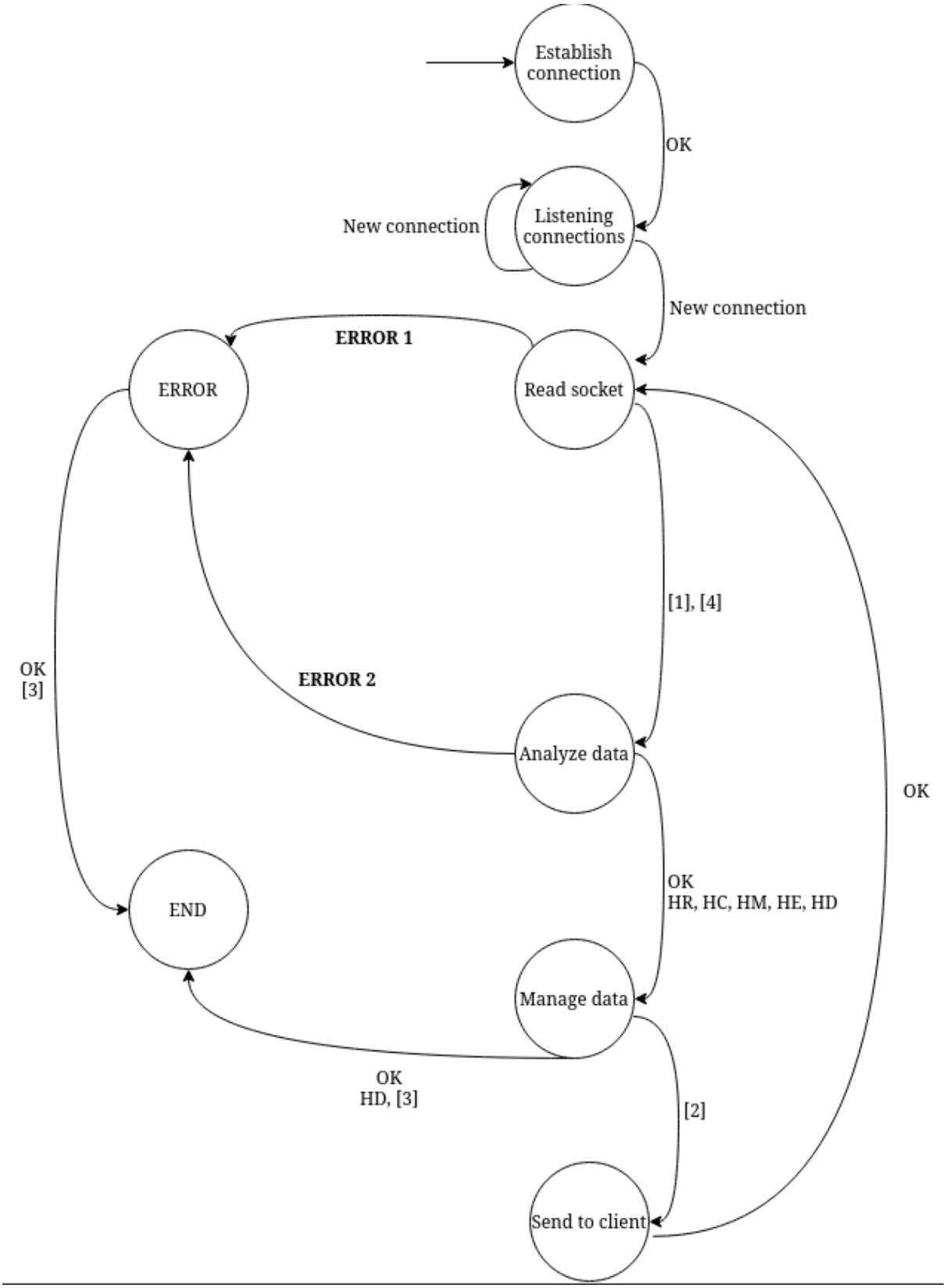
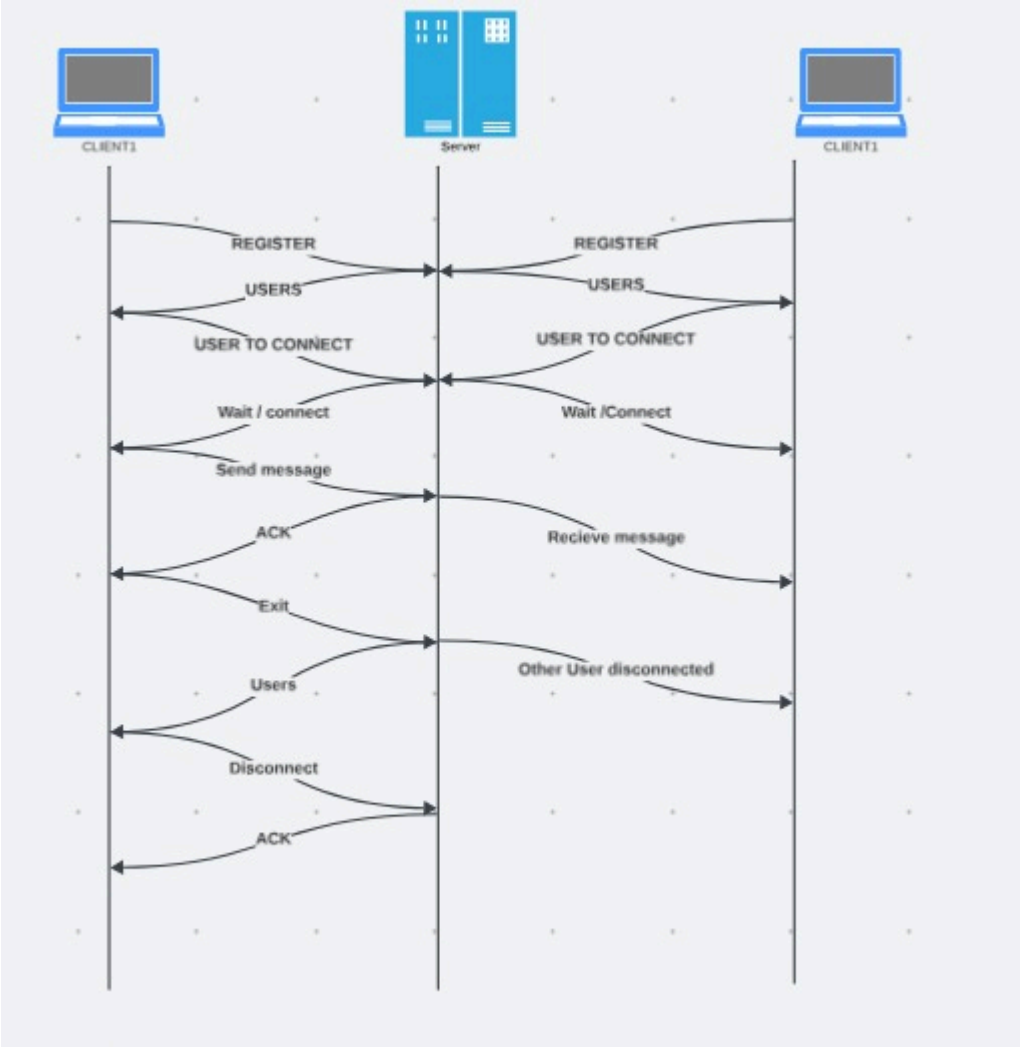

Establish connection
OK
Listening connections
New connection
New connection
ERROR 1
ERROR
Read socket
[1], [4]
OK
[3]
ERROR 2
Analyze data
OK
END
OK
HR, HC, HM, HE, HD
Manage data
OK
HD, [3]
[2]
Send to client

Diagrama de secuencia (cliente - servidor - cliente)



Reglas

El protocolo define ciertas reglas que toda aplicación que lo utilice debe de tener en cuenta al momento de realizar su correcta implementación, estas reglas son un conjunto de aspectos a tener en cuenta para todo aquel que lo vaya a utilizar.

- **buffer_size_msg:** define el máximo tamaño en bytes de body de los mensajes → **500**.
- **buffer_size_header:** define el tamaño máximo los headers →**15**.
- **port:** define el puerto que se usara para la comunicación cliente-servidor → **8080/tcp**.
- **max_len_username:** establece el máximo tamaño de los username de los clientes → **15**.

- **timeout_conn:** define cuanto tiempo espera el cliente para desconectarse una vez solicita conectarse con otro cliente y no recibe respuesta (no logra conectarse) → **30000 milisegundos**.

- **buffer_size:** establece el tamaño máximo del mensaje completo → **buffer_size_header+ buffer_size_msg**.

Interfaz del protocolo

Esta es la interface que le permite al servidor y al cliente comunicarse con el protocolo. Esta define diferentes estructuras y variables que pueden ser usadas tanto por cliente como por servidor (los nombres son definidos por quien implemente la interface):

- se define una estructura que contiene lo siguiente por cada cliente que se conecta con el servidor:
  - ❖ username → nombre con el que se identifica cada cliente
  - ❖ socket → almacena el número del socket correspondiente al cliente.
  - ❖ chatting → guarda con cuál usuario se desea conectar el cliente correspondiente.

La interface también define las siguientes funciones:

- **connect_to_server:** Lo usa el cliente para establecer la comunicación con el servidor. Recibe el socket de dicho cliente.

- **initialize_connection:** Lo usa el servidor para activar el socket con el que los clientes se conectaran con el server (crea el socket, bind y listen). Recibe un socket únicamente inicializado por el servidor.
- **accept_connection:** lo usa el servidor para aceptar las conexiones provenientes por los clientes. Recibe tanto el socket del server como del cliente

- **encapsulate_{header}:** hay un encapsulate para cada header del protocolo. Es usado por el servidor y los clientes para encapsular los diferentes bodies que se intercambiaran. Recibe el body que se desea encapsular.
  Ej.: Esteban →encapsulate_register(Esteban) → REGISTER Esteban.

- **uncapsulete_server:** lo usa el servidor para des encapsular los mensajes provenientes de los clientes, separando los headers y el body. Además, dependiendo del header realiza la operación/acción requerida.

- **uncapsulate_client:** Lo usan los clientes para des encapsular los mensajes provenientes del servidor. Además, analiza el header y hace la operación/acción correspondiente a este.

- **read_socket:** lo usan los clientes para escuchar los mensajes que llegan del servidor (recv).