Protocolo

COMUNICACIÓN CLIENTE => SERVIDOR

Mensaje al servidor	Respuesta del servidor	Descripción

{
request:
INIT_CONEX,
body: [
connect_time,
"user_id"
]
}
	{
	response:
	INIT_CONEX,
	code: 200
	}
		Para establecer una conexión con el servidor, se debe de agregar el tiempo en que se conecta y el nombre de usuario.
		
		La ip se agrega del lado del servidor al recibir la conexión.El status está por defecto en "Activo" al establecer la conexión. El id en este caso será el nombre y se manda en un array.

{
request:
END_CONEX
}
	{
	response:
	END_CONEX,
	code: 200
	}
		Para finalizar una conexión con el servidor, se debe de remover el nombre de usuario

{
request:
GET_CHAT,
body: 'all'
}
	{
	response:
	GET_CHAT,
	code: 200,
	body: [
	[
	message,
	from,
	delivered_at,
	]
	]
	}
		Retorna un array de arrays de mensajes del chat global.

{
request:
GET_CHAT,
body:
'username'
}
	{
	response:
	GET_CHAT,
	code: 200,
	body: [
	[
	message,
	from,
	delivered_at,
	]
	]
	}
		Retorna un array de los arrays de mensajes del chat con cierto usuario

{ request: POST_CHAT	{ response: POST_CHAT,	Crea un nuevo mensaje para el grupo completo

body: [message, from, delivered_at: “22:40”, to: “all”,] }	code: 200 }	

{
request: POST_CHAT
body: [
message,
from,
delivered_at: “22:40”,
to: “username”,
]
}
	{
	response: POST_CHAT,
	code: 200
	}
		Crea un nuevo mensaje para algún chat

{
request: GET_USER
body: ‘all’
}
	{
	response: GET_USER,
	code: 200,
	body: [
	“ [“usuario1”, “0”],
	[“usuario2”, “1”]
	]
	}
		Obtiene la lista de los usuarios conectados.
		Cada item es una lista [username, status]
		
		Importante: Castear el status a int del lado del cliente

{
request: GET_USER
body: ‘username’
}
	{
	response: GET_USER,
	code: 200,
	body: [“127.0.0.1”, “0”]
	}
		Array con la ip de un usuario en específico, su status actual
		
		Importante: Castear el status a int del lado del cliente

{
request: PUT_STATUS
body: “0”
}
	{
	response: PUT_STATUS,
	code: 200
	}
		Cambia el estado del cliente:
		0 => Activo
		1 => Inactivo
		2 => Ocupado

COMUNICACIÓN SERVIDOR => CLIENTE

Mensaje al cliente	**Descripción**

{
response: NEW_MESSAGE,
body: [
message,
from,
delivered_at,
to: “username”,
]
}
	Le avisa al usuario que tiene un nuevo mensaje de un usuario

{
response: NEW_MESSAGE,
body: [
message,
from,
delivered_at,
to: “all”,
]
}
	Le avisa al usuario que
	tiene un nuevo mensaje del chat global

TABLA DE ERRORES

Mensaje que envía el server ante un error	**Descripción**

{
code: 101
}
	Usuario ya registrado

{
code: 102
}
	Usuario no conectado

{
code: 103
}
	No hay usuarios conectados

{
code: 104
}
	El status no se pudo modificar

{
code: 105
}
	Error inesperado del server

Librería necesaria para compilar el programa:

sudo apt-get install libjson-c-dev

Para compilar, se debe de ejecutar:

- gcc -o client client.c -lpthread -ljson-c
- gcc -o server server.c -lpthread -ljson-c

Los links que se visitaron para poder llevar a cabo el proyecto fue:

Moon, Silver (2020). How to code a server and client in C with sockets on Linux. BinaryTides: https://www.binarytides.com/server-client-example-c-sockets-linux/

Loading…animating dots in C. Code review: https://codereview.stackexchange.com/questions/139440/loading-animating-dots-in-c

Mintz, R. (2010). Json-c/Libjson Tutorial: Programming in Linux: https://linuxprograms.wordpress.com/2010/05/20/json-c-libjson-tutorial/

Package: libjason-c-dev: https://packages.debian.org/sid/libjson-c-dev

Parsing JSON array in C: https://stackoverflow.com/questions/31836167/parsing-json-array-in-c

Using The JSON C API: https://realtimelogic.com/downloads/Using-The-JSON-C-API.pdf