

Universidad de Guadalajara

Centro Universitario de Tonalá



MATERIA: SISTEMAS DISTRIBUIDOS

CARRERA: ING. EN CIENCIAS COMPUTACIONALES

PROYECTO: CHAT GRUPAL CON SOCKET

ALUMNOS: BRYAN ALAN REYES FLORES

MAESTRO: VÍCTOR HUGO VEGA FREGOSO

CICLO ESCOLAR: 2020-A

Tabla de contenido

<u>Introducción</u>	3
<u>Proceso de creación</u>	4
<u>Creación del programa</u>	5
<u>Funcionamiento</u>	5
<u>CODIGO DEL CHAT</u>	8
<u>Clases del proyecto</u>	9
<u>Cliente</u>	9
<u>CLASE PAQUETE ENVIAR</u>	9
<u>Ventana Cliente</u>	9
<u>Servidor</u>	12
<u>clase servidor</u>	12
<u>Clase Hilo Server</u>	14
<u>Conclusión</u>	15

Introducción

La comunicación entre las personas hoy en día es muy importante porque podríamos mandarnos mensajes de una manera muy rápida conectados a la red , con lo cual antes no podríamos hacerlo o tardábamos mucho al comunicarnos ,por ejemplo lo hacíamos con cartas, fax ,etc. Por eso y otras cosas que la comunicación entre todos es importante. Es por eso que realice este proyecto con la finalidad de poder comunicarnos de manera fácil y grupal. El proyecto consta de la creación de un Chat grupal utilizando varias herramientas de programación como socket , thread entro otros, además ,encontrar el funcionamiento y los pasos a seguir en la creación de este chat en el lenguaje de programación Java .

Proceso de creación

Primeramente, para la creación de un chat tenemos que implementar lo que es:

- Socket
- ServerSocket
- ObjectOutputStream
- ObjectInputStream
- Thread

Un socket es un enlace de establecer una comunicación en la red , es decir es por donde pasa la información enviar .

El ServerSocket es quien recibe la información el socket, es decir, sirve para estar atento de la llegada de una información.

Un ObjectOutputStream es quien escribe los tipos de datos primitivos y gráficos de objetos y las convierte en un flujo de salida.

Un ObjectInputStream es lo inverso a un ObjectOutputStream ya que este en ves de escribir este lee los datos de tipo primitivo.

Para utilizar estos dos tipo de datos es necesario implementa una clase llamada Serializable la cual convierte la información a enviar en byte para poder enviarlos por la red de lo contrario los datos no podrán enviarse .

La clase Thread es quien define hilos a ejecutarse. Un Hilo es una forma de programar procesos simultáneamente. por ejemplo, un proceso puede estar haciendo una cosa mientras que el Thread están haciendo otra cosa, pero al mismo tiempo del primer proceso.

Creación del programa

El programa consiste en hacer un pequeño chat grupal con la herramienta de socket.el cual cada cliente tiene que recibir los mensajes de los demás cliente y así poderse comunicar.

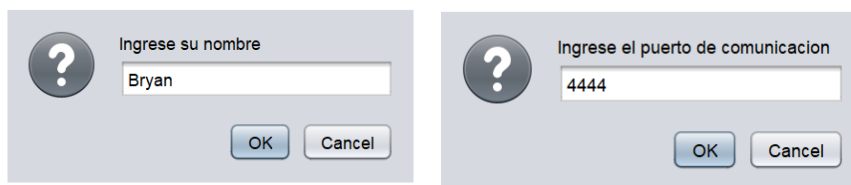
Funcionamiento

1-. Corremos nuestro servidor

```
run:  
Server Connected
```

2-. Iniciamos nuestros clientes

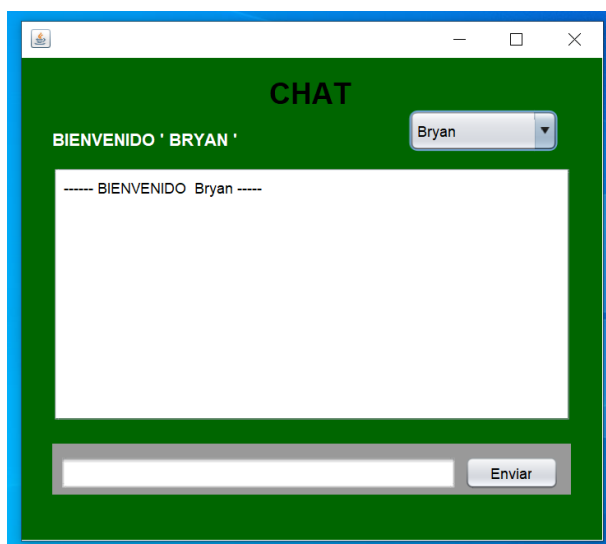
Nos mostrara una ventana para ingresar el nombre y otra para el puerto.



The image shows two separate input windows. The first window is titled 'Ingrese su nombre' (Enter your name) and contains a text field with the name 'Bryan' and 'OK' and 'Cancel' buttons. The second window is titled 'Ingrese el puerto de comunicacion' (Enter the communication port) and contains a text field with the number '4444' and 'OK' and 'Cancel' buttons.

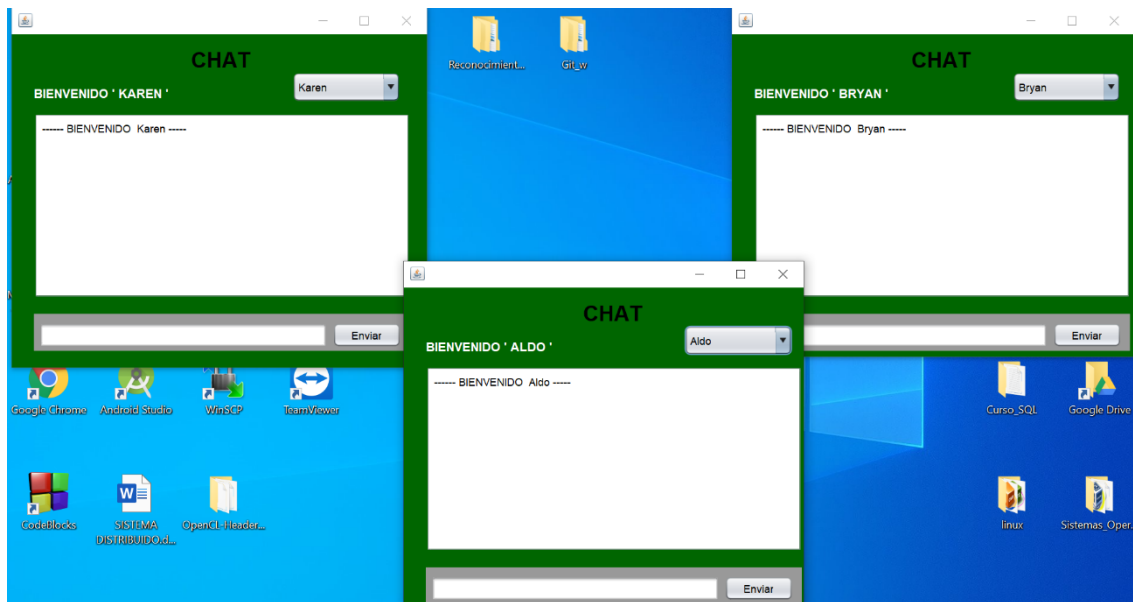
3-. Después de ingresar la información nos saldrá una venta como esta.

Nos mostrara nuestro nombre.



The image shows the main chat window. It has a green background with a blue border. At the top, it says 'CHAT'. Below that, it says 'BIENVENIDO ' BRYAN '' and there is a dropdown menu showing 'Bryan'. In the center, there is a large white text area with the message '----- BIENVENIDO Bryan -----'. At the bottom, there is a text input field and an 'Enviar' (Send) button.

4-. Conectamos a los demás clientes .



5-. Revisamos nuestro servidor para saber si mostro la información de cada cliente

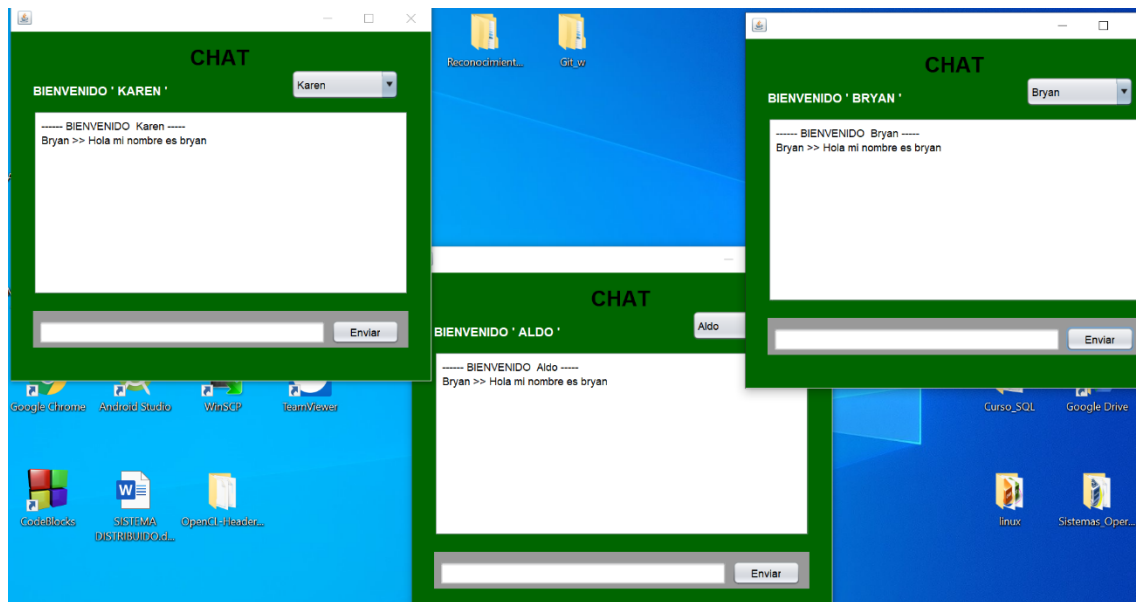
```
paquete recibido
ip : 127.0.0.1 puerto : 4444 nombre : Bryan mensaje : online
usuario agregado a la lista
Hilo comenzo servidor
paquete recibido
ip : 127.0.0.1 puerto : 5555 nombre : Aldo mensaje : online
usuario agregado a la lista
Hilo comenzo servidor
paquete recibido
ip : 127.0.0.1 puerto : 3333 nombre : Karen mensaje : online
usuario agregado a la lista
Hilo comenzo servidor
```

6-. Para enviar un mensaje solo tenemos que escribir en el recuadro gris de la parte inferior de la UI y presionar el botón enviar.

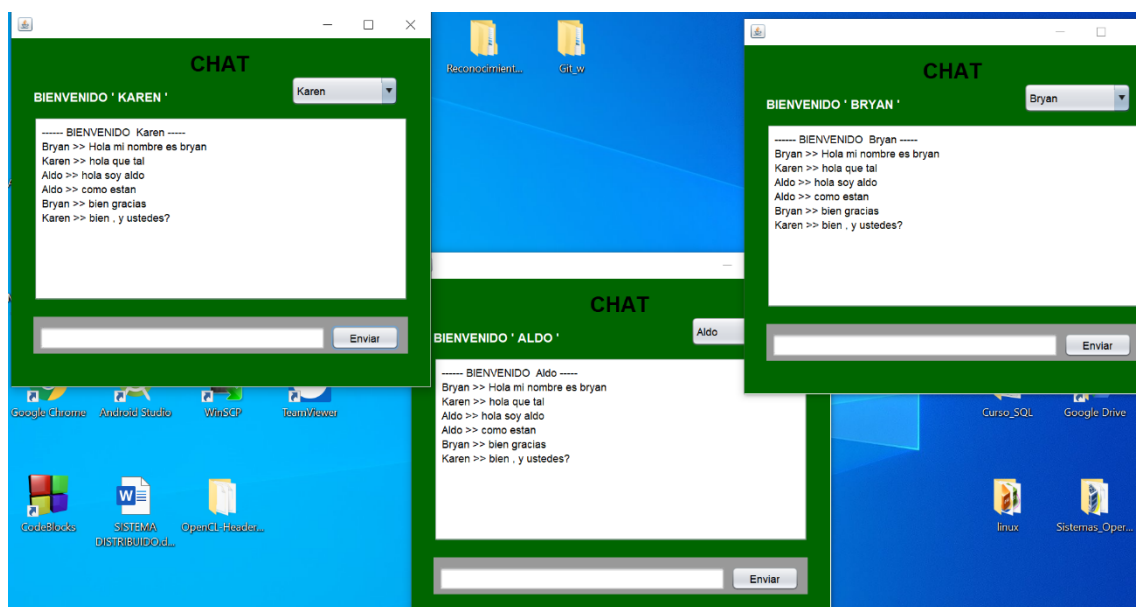


7-. Enviamos el mensaje y nos aparecerá el mensaje en nuestra ventana y en los demás clientes.

Nota: creo que es necesario enviarlo a uno mismo, ya que podríamos olvidar lo que escribimos y no sabríamos de que estábamos hablando.



8-. Ingresamos mas mensajes con los demás clientes.



9-. Para saber que el servidor este funcionando y este enviando los mensajes , lo imprimiremos en consola :

```
put
Proyecto_Final_Chat_Paquete (run) x Proyecto_
-----
direccion127.0.0.1
paquete recibido del hilo servidor
Karen >>> hola que tal
mensajes enviados a Karen
mensajes enviados a Bryan
mensajes enviados a Aldo
-----
direccion127.0.0.1
paquete recibido del hilo servidor
Aldo >>> hola soy aldo
mensajes enviados a Karen
mensajes enviados a Bryan
mensajes enviados a Aldo
-----
direccion127.0.0.1
paquete recibido del hilo servidor
Aldo >>> como estan
mensajes enviados a Karen
mensajes enviados a Bryan
mensajes enviados a Aldo
-----
direccion127.0.0.1
paquete recibido del hilo servidor
Bryan >>> bien gracias
mensajes enviados a Karen
mensajes enviados a Bryan
mensajes enviados a Aldo
-----
direccion127.0.0.1
paquete recibido del hilo servidor
```

9-. Listo ese es el funcionamiento del programa

CODIGO DEL CHAT

Se encuentra en GITHUB

https://github.com/Furia1799/Chat_Socket.git

Clases del proyecto

Cliente

Para la creación de la ventana del cliente tuve que implementar un hilo el cual iba a estar a la escucha de recibir información.

Creamos una clase llamada “Paquete_Enviar.java” la cual contiene todos los datos que vamos a enviar , además es implementable de Serializable para poder enviar los datos de porfa de byte y contiene todos los metodos get y los set para poder obtener su información.

CLASE PAQUETE_ENVIAR

```
public class Paquete_enviar implements Serializable{

    private String ip;
    private int puerto;
    private String nombre;
    private String mensaje;

    public Paquete_enviar(String ip, int puerto, String nombre, String mensaje) {
        this.ip = ip;
        this.puerto = puerto;
        this.nombre = nombre;
        this.mensaje = mensaje;
    }

    public Paquete_enviar() {
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }
}
```

Ventana Cliente

Primeramente, creamos nuestras variables a utilizar .

Socket para comunicarnos.

ObjectOutputStream para enviar el paquete de mensaje.

Nuestra clase Paquete_enviar.

```

public class Client extends javax.swing.JFrame implements Runnable {

    Socket conexion = null;
    ObjectOutputStream out_paquete;

    String ip, nombre;
    static int puerto_cliente;
    Paquete_enviar datos;

    public Client() {
        //preguntamos como se llama y que puerto comunicarse
        nombre = JOptionPane.showInputDialog("Ingrese su nombre ");
        String puerto = JOptionPane.showInputDialog(null, "Ingrese el puerto de comunicacion");
        puerto_cliente = Integer.parseInt(puerto);
    }
}

```

Después obtenemos el nombre y numero de puerto del cliente, se abrirá una ventana pequeña para poder ingresar los valores , esto con la cuestión que programa se vea mejor .

En el constructor de la ventana

Creamos a nuestro socket y le ingresamos la dirección del servidor y puerto para comunicarnos. Después creamos un objeto de a clase Paquete enviar y obtenemos nuestra IP .

```

System.out.println("cliente inicializado " + nombre);
conexion = new Socket("127.0.0.1", 6666); //18.232.165.3
datos = new Paquete_enviar();
ip = conexion.getInetAddress().getHostName();

```

Para poder enviar el paquete al servidor tenemos que ingresar los datos en el objeto creado, ingresamos la IP , puerto_cliente , nombre y el mensaje .

Creamos nuestro ObjectOutputStream y finalmente enviamos nuestro objeto con .writeObject() le cual tenemos que pasamos como parámetro nuestro objeto creado.

Este primer paquete lo enviamos al servidor para que sepa que estamos conectados y enviarle nuestra información de quien somos, para que después pueda enviarnos mensajes.

```
//ingresar datos
datos.setIp(ip);
datos.setPuerto(puerto_cliente);
datos.setNombre(nombre);
datos.setMensaje("online");

out_paquete = new ObjectOutputStream(conexion.getOutputStream());
out_paquete.writeObject(datos);

System.out.println("paquete enviado cliente");
```

Iniciamos nuestro Thread

Creamos este hilo para que fuera como un servidor (que este siempre a la escucha) y poder obtener los mensajes de los demás clientes.

```
Thread mihilo = new Thread(this);
mihilo.start();
```

La misma ventana esta implementando la clase Runnable por lo cual podemos alquiere hilos .En el método Run(), implementamos el siguiente código .

```
try {
    System.out.println("Hilo cliente inicializado");

    ServerSocket servidor_cliente = new ServerSocket(puerto_cliente);
    Socket socket_cliente;
    Paquete_enviar paquete_recibido;

    while (true) {
        socket_cliente = servidor_cliente.accept();

        ObjectInputStream out_paquete_2 = new ObjectInputStream(socket_cliente.getInputStream());
        paquete_recibido = (Paquete_enviar) out_paquete_2.readObject();

        System.out.println("mensajes recibidos de hilo cliente");

        if (paquete_recibido.getMensaje().equals("#")) {
            break;
        }

        System.out.println(paquete_recibido.getNombre() + " >> " + paquete_recibido.getMensaje());
        txt_area.append("\n" + paquete_recibido.getNombre() + " >> " + paquete_recibido.getMensaje());
    }

    socket_cliente.close();
}
```

Lo que hacemos es creamos un ServerSocket para poder estar a la escucha en cualquier momento, también creamos un Socket e instanciamos la clase Paquete_enviar . Creamos un ciclo while infinito para que siempre este a la espera . Aceptamos la conexión del servidor y la guardamos en nuestro socket ,

para poder recoger la información recibida tenemos que crear un objeto de tipo `ObjectInputStream` y para leerla es necesario implementar el método `.readObject()` el cual igualamos a nuestra clase `Paquete_enviar`. Para terminar la comunicación es necesario enviar como mensaje el "#". Finalmente obtenemos la información con el método `get` de la clase y lo mostramos en la interfaz de usuario con `.append` y el texto.

Servidor

Para la creación del servidor tuve que implementar una clase llamada "Usuarios" la cual tienen que guardar todos los clientes con su información completa, creando así objetos de la clase y guardándolos en un array para después enviarles un mensaje a todos.

```
* @author Bryan
*/
public class Usuarios {

    private String ip;
    private String nombre;
    private int puerto;
    private Usuarios lista_usuarios[] = new Usuarios[10];

    public Usuarios(String nombre, String ip, int puerto) {
        this.nombre = nombre;
        this.ip = ip;
        this.puerto = puerto;
    }

    public Usuarios() {
    }

    public int getPuerto() {
        return puerto;
    }

    public void setPuerto(int puerto) {
```

clase servidor

La clase servidor tiene su `main`, creamos un número de puerto fijo, como es servidor obviamente su `ServerSocket` y creamos nuestro objeto de `Usuarios` para poder acceder al array, un contador para saber cuando clientes tenemos, implementamos la clase `Paquete_enviar` para poder acceder a la información de los usuarios y finalmente un `Socket` para la comunicación.

```

* @author Bryan
*/
public class Server {

    public static void main(String[] args) {
        final int PUERTO = 6666;
        ServerSocket socket_server = null; //server
        //socket
        Usuarios lista_usuarios = new Usuarios();
        int con = 0;
        Paquete_enviar paquete_recibido;

        try {
            socket_server = new ServerSocket(PUERTO);
            System.out.println("Server Connected");

```

Después de declarar las variables, creamos un ciclo while infinito para que siempre este a la escucha , aceptamos la comunicación del cliente y la guardamos en nuestro socket, creamos nuestro objeto de ObjectInputStream para recibir la comunicación del cliente y además implementamos el método .readObject(); para obtener la información. Finalmente imprimimos en consola la información del cliente, creamos un objeto Usuario para poderlo aguardar en el array e implementamos un Thread de servidor, le pasmos como parámetro el socket, la lista y el ObjectInputStream .

```

while (true) {
    Socket client_socket = socket_server.accept();

    ObjectInputStream in_paquete = new ObjectInputStream(client_socket.getInputStream());
    paquete_recibido = (Paquete_enviar) in_paquete.readObject();

    System.out.println("paquete recibido");

    System.out.println("ip : " + paquete_recibido.getIp() + " puerto : " + paquete_recibido.getPuerto()
        + " nombre : " + paquete_recibido.getNombre() + " mensaje : " + paquete_recibido.getMensaje());

    Usuarios usuario = new Usuarios(paquete_recibido.getNombre(),
        paquete_recibido.getIp(), paquete_recibido.getPuerto());

    lista_usuarios.agregar(con, usuario);
    System.out.println("usuario agregado a la lista");

    Hilo_Server hilo = new Hilo_Server(client_socket, lista_usuarios, in_paquete); //,lista_usuarios
    hilo.start();
    con++;
}

```

Clase Hilo_Server

La clase debe estar extendiendo de Thread para poderla utilizar como hilo, creamos nuestras variables y el constructor.

```
* @author Bryan
*/
public class Hilo_Server extends Thread {

    private Socket socket_cliente;
    private Usuarios usuario = null;
    ObjectInputStream in_paquete;

    public Hilo_Server(Socket cliente, Usuarios usuario, ObjectInputStream in_paquete) {
        this.socket_cliente = cliente;
        this.usuario = usuario;
        this.in_paquete = in_paquete;
    }
}
```

En el método run ()

Declaramos nuestro ObjectOutputStream y la clase Paquete_enviar, de igual manera creamos un ciclo while infinito para que a la escucha de los clientes. Recibimos nuestro paquete con .readObject() y obtenemos la información de nuestro cliente, para salir tenemos que enviar nuestro “ # “ como mensaje.

```
@Override
public void run() {
    System.out.println("Hilo comenzo servidor");

    try {
        ObjectOutputStream out_paquete;
        Paquete_enviar datos_recibidos;

        while (true) {

            datos_recibidos = (Paquete_enviar) in_paquete.readObject();

            String direccion = datos_recibidos.getIp();
            System.out.println("direccion" + direccion);

            System.out.println("paquete recibido del hilo servidor");
            System.out.println(datos_recibidos.getNombre() + " >>> " + datos_recibidos.getMensaje());

            if (datos_recibidos.getMensaje().equals("#")) {
                break;
            }
        }
    }
}
```

Para poder enviar el mensaje que envió el cliente y replicarlo en los demás clientes, primero como ya sabes todos los clientes están guardados en un array, entonces tenemos que acceder a ese array y obtener la información de su IP y puerto.

El for es para recorrer el array, hacemos un if para saber si en una posición esta varia, al verificar todo eso , tenemos que crear un socket con la dirección IP del cliente y el puerto ,después creamos un ObjectOutputStream para convertir la información en objeto y finalmente lo enviamos con el método .writeObject() , cerramos la conexión y esperamos a que se termine el ciclo del for.

```
for (int i = 0; i < usuario.getListado_usuarios().length; i++) {  
    if (usuario.obtener_usuario(i) == null) {  
        break;  
    }  
    Socket enviar_destino = new Socket(usuario.obtener_usuario(i).getIp(),  
                                       usuario.obtener_usuario(i).getPuerto());  
    out_paquete = new ObjectOutputStream(enviar_destino.getOutputStream());  
  
    out_paquete.writeObject(datos_recibidos);  
  
    System.out.println("mensajes enviados a " + usuario.obtener_usuario(i).getNombre());  
    enviar_destino.close();  
  
}  
  
System.out.println("-----");  
}  
in_paquete.close();  
socket_cliente.close();
```

Conclusión

Me gusto mucho realizar este tipo de proyecto ya que me pone a prueba mis capacidades de aprendizaje y enseñanzas en el transcurso del semestre , aunque creo que se me facilito a la hora de la creación y funcionamiento de thread por que un semestre antes lleve una materia relacionada a eso , pero en ocasiones se me dificultaba por que tuve que aprender a mandar información por paquetes y no enviarla una por uno . Pero en si , el proyecto quedo muy bien .