

**Tema**

SOCKETS EN APACHE NETBEANS

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

30/11/2022

SOCKETS EN APACHE NETBEANS

[1. MARCO TEÓRICO 4](#_Toc120637635)

[1.1. OBJETIVOS 4](#_Toc120637636)

[1.2.1. OBJETIVO GENERAL 4](#_Toc120637637)

[1.2.2. OBJETIVOS ESPECÍFICOS 4](#_Toc120637638)

[1.2. ARQUITECTURA CLIENTE – SERVIDOR 4](#_Toc120637639)

[Cliente 4](#_Toc120637640)

[Servidor 5](#_Toc120637641)

[1.3. SOCKETS 5](#_Toc120637642)

[PROPIEDADES 5](#_Toc120637643)

[ATRIBUTOS 6](#_Toc120637644)

[TIPOS DE SOCKETS 6](#_Toc120637645)

[2. PARTE PRÁCTICA 6](#_Toc120637646)

[2.1. CREACIÓN DE CARPETAS PROYECTO 6](#_Toc120637647)

[2.2. CREACION DEL PROYECTO 7](#_Toc120637648)

[2.3. CREACIÓN DE PAQUETES PARA APLICAR UN MODELO MVC EN EL CLIENTE Y SERVIDOR. 8](#_Toc120637649)

[2.4. CODIFICACIÓN PROYECTO CLIENTE 10](#_Toc120637650)

[2.5. CODIFICACIÓN DEL PROYECTO SERVIDOR 23](#_Toc120637651)

[2.6. EJECUCIÓN DE LA APLICACIÓN 30](#_Toc120637652)

[3. CONCLUSIONES 36](#_Toc120637653)

[4. RECOMENDACIONES 36](#_Toc120637654)

[5. REFERENCIAS 36](#_Toc120637655)

**INDICE DE IMÁGENES**

[Figura 1. Modelo Cliente – Servidor. 5](#_Toc120637530)

[Figura 2. Sockets 6](#_Toc120637531)

[Figura 3. Estructura del proyecto. 7](#_Toc120637532)

[Figura 4. Creación Nuevo Proyecto 7](#_Toc120637533)

[Figura 5. Selección del tipo de aplicación Java. 7](#_Toc120637534)

[Figura 6. Nombres y rutas del nuevo proyecto. 8](#_Toc120637535)

[Figura 7. Estructura de los proyectos creados anteriormente. 8](#_Toc120637536)

[Figura 8. Creación de los paquetes en el proyecto Socket\_Client\_Grupo5 8](#_Toc120637537)

[Figura 9. Asignación de nombres para los paquetes del Cliente 9](#_Toc120637538)

[Figura 10. Asignación de nombres para los paquetes adicionales del Cliente. 9](#_Toc120637539)

[Figura 11. Creación de los paquetes en el proyecto Socket\_Server\_Grupo5 9](#_Toc120637540)

[Figura 12. Asignación de nombres para los paquetes del Servidor. 10](#_Toc120637541)

[Figura 13. Estructura de los proyectos Cliente y Servidor 10](#_Toc120637542)

[Figura 14. Creación de los archivos del paquete modelo en el proyecto Cliente. 11](#_Toc120637543)

[Figura 15. Creacipon y asignación de los nombre de los archivos del paquete modelo del proyecto Cliente. 11](#_Toc120637544)

[Figura 16. Creación de los archivos ReceivingFileThread y SendingFileThread. 17](#_Toc120637545)

[Figura 17. Creación de los archivos para la vista del Cliente. 23](#_Toc120637546)

[Figura 18. Creación de los archivos para el paquete modelo del servidor. 24](#_Toc120637547)

[Figura 19. Asignación del nombre para el archivo del modelo. 24](#_Toc120637548)

[Figura 20. Creación y asignación de los archivos ServerThread y SocketThread. 27](#_Toc120637549)

[Figura 21. Creación y asignación del nombre al archivo del paquete vista del proyecto Servidor. 30](#_Toc120637550)

[Figura 22. Ejecución del proyecto Servidor. 30](#_Toc120637551)

[Figura 23. Se hace clic sobre iniciar servidor 31](#_Toc120637552)

[Figura 24. Servidor iniciado. 31](#_Toc120637553)

[Figura 25. Ejecución del proyecto cliente. 31](#_Toc120637554)

[Figura 26. Ventana principal del proyecto Cliente 32](#_Toc120637555)

[Figura 27. Creación de los usuarios Michael y Carlos. 32](#_Toc120637556)

[Figura 28. Conversación entre Carlos y Michael. 33](#_Toc120637557)

[Figura 29. Envío de archivos. 33](#_Toc120637558)

[Figura 30. Selección de archivos a enviar. 33](#_Toc120637559)

[Figura 31. Insertar el Username. 34](#_Toc120637560)

[Figura 32. Aceptar archivo recibido. 34](#_Toc120637561)

[Figura 33. Se indica la ruta donde se desea guardar el archivo recibido. 34](#_Toc120637562)

[Figura 34. Mensaje de éxito tras enviar y recibir el archivo. 34](#_Toc120637563)

[Figura 35. Visualización de acciones desde el servidor. 35](#_Toc120637564)

[Figura 36 Detener el servidor. 35](#_Toc120637565)

**ÍNDICE DE TABLAS**

[Tabla 1. Codificación del archivo ClientThread 12](#_Toc120708561)

[Tabla 2. Codificación del archivo MessageStyle. 16](#_Toc120708562)

[Tabla 3. Codificación del archivo ReceivingFileThread 18](#_Toc120708563)

[Tabla 4. Codificación del archivo SendingFileThread 20](#_Toc120708564)

[Tabla 5. Codificación del archivo OnlineListThread 25](#_Toc120708565)

[Tabla 6. Codificación del archivo ServerThread 28](#_Toc120708566)

# MARCO TEÓRICO

## OBJETIVOS

### **OBJETIVO GENERAL**

Desarrollar e implementar un Chat que permita el envío de mensajes y archivos haciendo uso de Sockets en el lenguaje de programación java.

### **OBJETIVOS ESPECÍFICOS**

* Desarrollar e implementar un proyecto Servidor haciendo uso de la arquitectura MVC.
* Desarrollar e implementar un proyecto Cliente los cuales se conecten a una misma conversación haciendo uso de la arquitectura MVC.
* Permitir el envío de archivos (texto, imágenes, audios) entre los usuarios que se encuentren en línea.

## ARQUITECTURA CLIENTE – SERVIDOR

El modelo o arquitectura Cliente – Servidor comenzó a ser aceptado a finales de los 80’s. El funcionamiento o su implementación es sencillo ya que se tiene una máquina cliente la cual requiere de un servicio que se encuentra en una máquina servidor.

Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma.

Cuando se habla de la arquitectura cliente – servidor, el cliente envía un mensaje realizando una solicitud de cualquiera de los servicios del servidor lo cual se conoce como una petición, y el servidor responde enviando uno o varios mensajes al cliente.

Al ser considerada una arquitectura distribuida se puede decir que las máquinas clientes puede ser también servidores dependiendo de la necesidad, es decir cumplen el rol de cliente y servidor.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son más adecuadas a sus características.

## Cliente

El cliente es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor, se le conoce con el término front-end.

El objetivo principal del cliente es la manipulación y el despliegue de los datos, y tiene como característica el desarrollo sobre interfaces gráficas de usuario (GUI).

Procesos de un cliente

* Administrar la interfaz de usuario.
* Interactuar con el usuario.
* Procesar la lógica de la aplicación y hacer validaciones locales.
* Generar requerimientos de bases de datos.
* Recibir resultados del servidor.
* Formatear resultados.

## Servidor

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se le conoce con el término back-end.

El objetivo principal del servidor es el manejar y gestionar las reglas del negocio, además, de los recursos de los datos.

Procesos de un servidor

* Aceptar los requerimientos de bases de datos que hacen los clientes.
* Procesar requerimientos de bases de datos.
* Formatear datos para transmitirlos a los clientes.
* Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

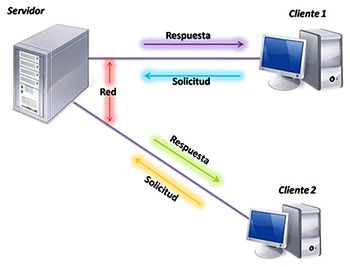


Figura 1. Modelo Cliente – Servidor.

## SOCKETS

Un socket es un mecanismo de comunicación entre procesos los cuales permiten que un proceso emita o reciba información con otro proceso incluso estando en distintas máquinas.

También se puede mencionar que un socket no es más que un canal de comunicación entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador.

## PROPIEDADES

* Fiabilidad de la transmisión, los datos que transmiten no se pierden.
* Conservación del orden de los datos, los datos mantienen el orden de llegada tal como fueron emitidos.
* No existen datos duplicados, los datos llegan una sola vez a su destino.
* Comunicación en modo conectado, la conexión se encuentra establecida antes de comenzar con la comunicación.
* Conservación de los límites de los lenguajes, los límites de mensajes emitidos pueden encontrarse o conocerse en el destino.
* Envío de mensajes urgentes, permite el envío de mensajes fuera del flujo cuando se envía un mensaje fuera del flujo

## ATRIBUTOS

* Dominio: especifica el medio de comunicación de la red que el socket utilizará.
* Protocolo: especifica que protocolo se va a utilizar.
* Tipo: los protocolos de internet proveen dos niveles distintos de servicio, flujo y datagramas.

## TIPOS DE SOCKETS

1. Orientados a conexión: contiene comunicaciones fiables y circuitos virtuales.
2. No orientados a conexión: el programa de aplicación da la fiabilidad.

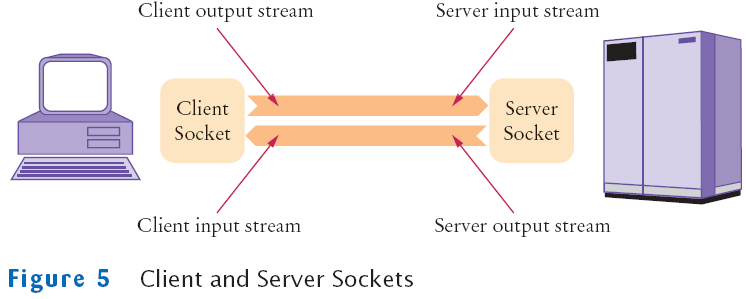


Figura 2. Sockets

Para comenzar con el desarrollo del proyecto es importante crear las carpetas que nos permitan tener organizado de mejor manera el cliente y el servidor.

# PARTE PRÁCTICA

## CREACIÓN DE CARPETAS PROYECTO

Para comenzar con el desarrollo del proyecto es importante crear las carpetas que nos permitan tener el proyecto organizado de tal forma que se separe la aplicación cliente, de la aplicación servidor y la documentación del mismo, para ello se tendrá la carpeta principal que es 07\_SOCKETS\_GRUPO5 y dentro de esta se tendrá dos subcarpetas que son APLICATIVO y DOCUMENTACIÓN, y dentro de APLICATIVO se tendrá dos subcarpetas que serán CLIENTE y SERVIDOR.

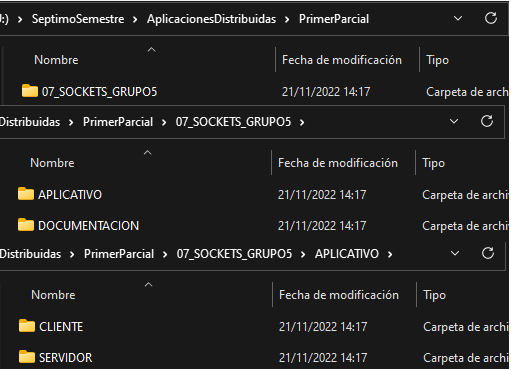


Figura 3. Estructura del proyecto.

## CREACION DEL PROYECTO

Una vez creadas las carpetas donde se va a guardar la aplicación cliente y servidor creamos un nuevo proyecto en Apache Netbeans para ello se hace clic ebn File, New Project, Java with Ant y finalmente Java Application.



Figura 4. Creación Nuevo Proyecto

En la siguiente ventana, seleccionar la categoria Java with Ant, Java Application y finalmente se hace clic en Next.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 5. Selección del tipo de aplicación Java.

Se definen los nombres para dos proyectos que se van a crear los cuales son Socket\_Client\_Grupo5 y Socket\_Server\_Grupo5, se selecciona las rutas de las carpetas creadas anteriormente las cuales serán APLICATIVO/CLIENTE y APLICATIVO/SERVIDOR respectivamente, se hace clic en Finish.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 6. Nombres y rutas del nuevo proyecto.

A continuación se puede visualizar los dos proyectos creados uno para el Cliente y otro para el servidor con la siguiente estructura.

Texto

Descripción generada automáticamente

Figura 7. Estructura de los proyectos creados anteriormente.

## CREACIÓN DE PAQUETES PARA APLICAR UN MODELO MVC EN EL CLIENTE Y SERVIDOR.

Para poder continuar con el desarrollo de la práctica se deberá crear los siguientes paquetes MODELO, VISTA, CONTROLADOR que son los encargados de reestructurar nuestro proyecto.

Para ello se deberá hacer clic derecho sobre Source Packages, seguido de eso en New y Java Package.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 8. Creación de los paquetes en el proyecto Socket\_Client\_Grupo5

Seguido de eso se mostrará la siguiente ventana donde se configuran los nombres del paquete los cuales serán, **ec.edu.monster.modelo**, **ec.edu.monster.controlador** y **ec.edu.monster.vista.**

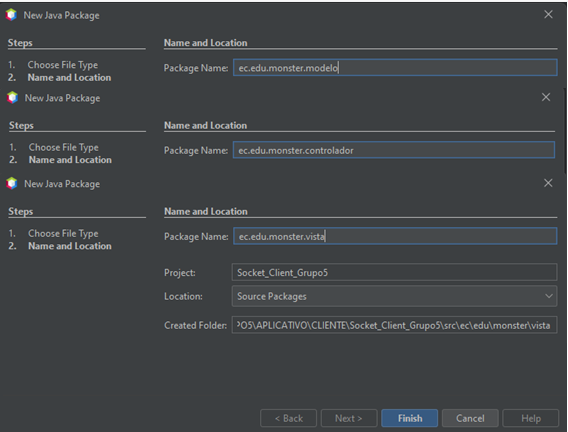


Figura 9. Asignación de nombres para los paquetes del Cliente

Además, se crearan dos paquetes adicionales que tienen como objetivo el almacenamiento de las imágenes y los audios que se utilizarán en el desarrollo de la aplicación.

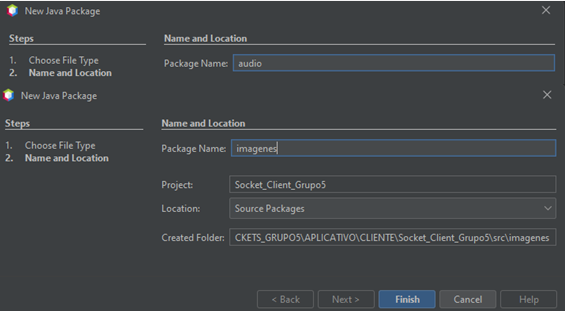


Figura 10. Asignación de nombres para los paquetes adicionales del Cliente.

De igual forma se realizará la creación de los paquetes para el proyecto del Servidor, para ello se deberá hacer clic derecho sobre Source Package del proyecto servidor, New y finalmente Java Package.

Interfaz de usuario gráfica, PowerPoint

Descripción generada automáticamente

Figura 11. Creación de los paquetes en el proyecto Socket\_Server\_Grupo5

Seguido de eso se crearán los paquetes de **ec.edu.monster.modelo**, **ec.edu.monster.controlador** y **ec.edu.monster.vista** como se muestra en la siguiente pantalla.

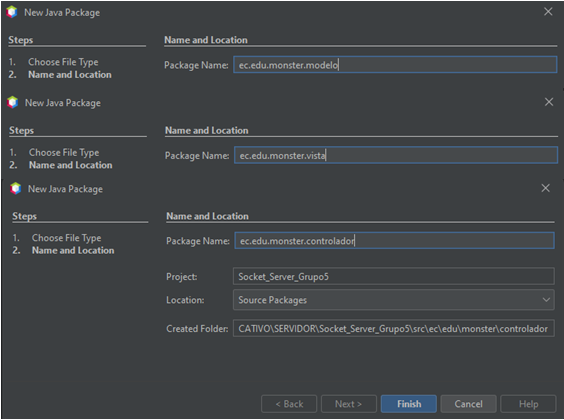


Figura 12. Asignación de nombres para los paquetes del Servidor.

Finalmente una vez creados los paquetes se tiene la siguiente estructura en los proyectos Cliente y Servidor.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 13. Estructura de los proyectos Cliente y Servidor

Una vez creados los paquetes de cada uno de los proyecto se procede a la creación y codificación de los archivos.

## CODIFICACIÓN PROYECTO CLIENTE

Para ello comenzamos con la creación de los archivos que se van a utilizar en cada uno de los paquetes.

**2.4.1. PAQUETE MODELO PROYECTO CLIENTE**

Comenzamos en el paquete del modelo, y para ello se va a dar clic derecho sobre el paquete ec.edu.monster.modelo, new y finalmente Java Class.

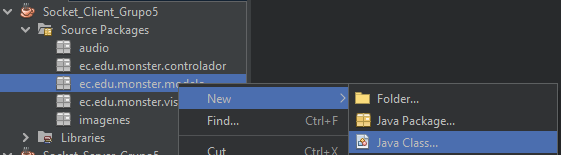


Figura 14. Creación de los archivos del paquete modelo en el proyecto Cliente.

Finalmente se crean los dos archivos que son ClientThread y MessageStyle como se muestra en la siguiente pantalla, donde se indica el nombre de los archivos.

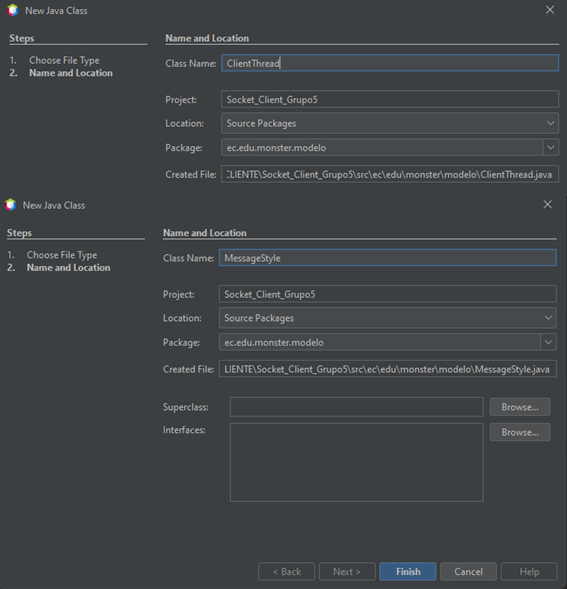


Figura 15. Creacipon y asignación de los nombre de los archivos del paquete modelo del proyecto Cliente.

Tabla 1. Codificación del archivo ClientThread.

/\*

 \* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

 \* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

 \*/

package ec.edu.monster.modelo;

import ec.edu.monster.controlador.ReceivingFileThread;

import ec.edu.monster.vista.MainForm;

import ec.edu.monster.vista.SoundEffect;

import java.awt.Color;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import java.text.DecimalFormat;

import java.util.StringTokenizer;

import java.util.Vector;

import javax.swing.JOptionPane;

/\*\*

 \*

 \* @author Michael

 \*/

public class ClientThread implements Runnable {

    Socket socket;

    DataInputStream dis;

    DataOutputStream dos;

    MainForm main;

    StringTokenizer st;

    protected DecimalFormat df = new DecimalFormat("##,#00");

    public ClientThread(Socket socket, MainForm main) {

        this.main = main;

        this.socket = socket;

        try {

            dis = new DataInputStream(socket.getInputStream());

        } catch (IOException e) {

            main.appendMessage("[IOException]: " + e.getMessage(), "Error", Color.RED, Color.RED);

        }

    }

    @Override

    public void run() {

        try {

            while (!Thread.currentThread().isInterrupted()) {

                String data = dis.readUTF();

                st = new StringTokenizer(data);

                /\*\*

                 \* Get Message CMD \*

                 \*/

                String CMD = st.nextToken();

}

switch (CMD) {

                    case "CMD\_MESSAGE":

                        SoundEffect.MessageReceive.play();

                        String msg = "";

                        String frm = st.nextToken();

                        while (st.hasMoreTokens()) {

                            msg = msg + " " + st.nextToken();

                        }

                        main.appendMessage(msg, frm, Color.BLACK, Color.BLACK);

                        break;

                    case "CMD\_ONLINE":

                        Vector online = new Vector();

                        while (st.hasMoreTokens()) {

                            String list = st.nextToken();

                            if (!list.equalsIgnoreCase(main.username)) {

                                online.add(list);

                            }

                        }

                        main.appendOnlineList(online);

                        break;

                    case "CMD\_FILE\_XD":

                        String sender = st.nextToken();

                        String receiver = st.nextToken();

                        String fname = st.nextToken();

                        int confirm = JOptionPane.showConfirmDialog(main,

                                "De " + sender + "\nNombre archivo: " + fname + "\nAceptar archivo?");

                        if (confirm == 0) {

try {

                                dos = new DataOutputStream(socket.getOutputStream());

                                String format = "CMD\_SEND\_FILE\_ACCEPT " + sender + " Aceptado";

                                dos.writeUTF(format);

                                Socket fSoc = new Socket(main.getMyHost(), main.getMyPort());

                                DataOutputStream fdos = new DataOutputStream(fSoc.getOutputStream());

                                fdos.writeUTF("CMD\_SHARINGSOCKET " + main.getMyUsername());

                                new Thread(new ReceivingFileThread(fSoc, main)).start();

                            } catch (IOException e) {

                                System.out.println("[CMD\_FILE\_XD]: " + e.getMessage());

                            }

                        } else

                            try {

                                dos = new DataOutputStream(socket.getOutputStream());

                                String format = "CMD\_SEND\_FILE\_ERROR " + sender + " Usuario rechazo o se desconecto!";

                                dos.writeUTF(format);

                            } catch (IOException e) {

                                System.out.println("[CMD\_FILE\_XD]: " + e.getMessage());

                            }

                        }

                        break;

                    default:

                        main.appendMessage("[CMDException]: Comando inválido " + CMD, "CMDException", Color.RED,

                                Color.RED);

                        break;

                }

            }

        } catch (IOException e) {

            main.appendMessage(" Se perdio la conexion con clon .!", "Error", Color.RED, Color.RED);

        }

    }

}

Tabla 2. Codificación del archivo MessageStyle.

package ec.edu.monster.modelo;

import java.awt.Color;

import javax.swing.text.AttributeSet;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import javax.swing.text.StyleContext;

/\*\*

 \*

 \* @author Michael

 \*/

public class MessageStyle {

    public static AttributeSet styleMessageContent(Color color, String fontFamily, int size) {

        StyleContext sc = StyleContext.getDefaultStyleContext();

        AttributeSet aset = sc.addAttribute(SimpleAttributeSet.EMPTY, StyleConstants.Foreground, color);

        aset = sc.addAttribute(aset, StyleConstants.FontFamily, fontFamily); // FontFamily

        aset = sc.addAttribute(aset, StyleConstants.Alignment, StyleConstants.ALIGN\_JUSTIFIED);

        aset = sc.addAttribute(aset, StyleConstants.FontSize, size);

        return aset;

    }

}

**2.4.2.** PAQUETE CONTROLADOR PROYECTO CLIENTE

Una vez codificado el paquete modelo continuamos con la codificación de los archivos del paquete controlador para ello se van a crear los archivos haciendo clic derecho sobre el paquete ec.edu.monster.controlador, new y finalmente Java Class.

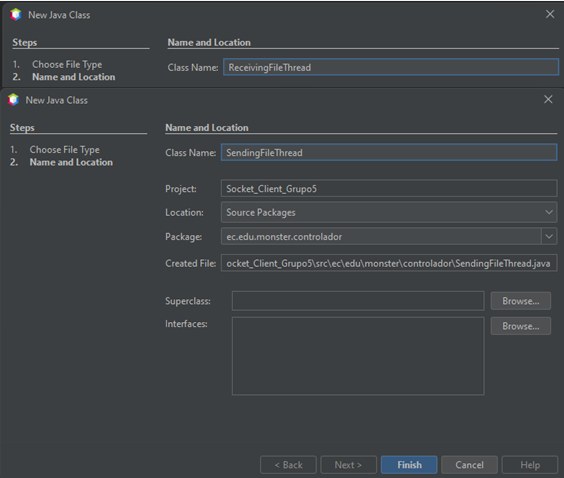


Figura 16. Creación de los archivos ReceivingFileThread y SendingFileThread.

Tabla 3. Codificación del archivo ReceivingFileThread

package ec.edu.monster.controlador;

import ec.edu.monster.vista.MainForm;

import java.io.BufferedInputStream;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.Socket;

import java.text.DecimalFormat;

import java.util.StringTokenizer;

import javax.swing.JOptionPane;

import javax.swing.ProgressMonitorInputStream;

public class ReceivingFileThread implements Runnable {

    protected Socket socket;

    protected DataInputStream dis;

    protected DataOutputStream dos;

protected MainForm main;

    protected StringTokenizer st;

protected DecimalFormat df = new DecimalFormat("##,#00");

    private final int BUFFER\_SIZE = 100;

    public ReceivingFileThread(Socket soc, MainForm m) {

        this.socket = soc;

        this.main = m;

        try {

            dos = new DataOutputStream(socket.getOutputStream());

            dis = new DataInputStream(socket.getInputStream());

        } catch (IOException e) {

            System.out.println("[ReceivingFileThread]: " + e.getMessage());

        }

    }

    @Override

    public void run() {

        try {

            while (!Thread.currentThread().isInterrupted()) {

                String data = dis.readUTF();

                st = new StringTokenizer(data);

                String CMD = st.nextToken();

                switch (CMD) {

                    case "CMD\_SENDFILE":

                        String consignee = null;

                        try {

                            String filename = st.nextToken();

                            int filesize = Integer.parseInt(st.nextToken());

                            consignee = st.nextToken();

                            main.setMyTitle("Cargando archivo....");

                            System.out.println("Cargando archivo....");

                            System.out.println("Desde: " + consignee);

} catch (IOException e) {

            System.out.println("[ReceivingFileThread]: " + e.getMessage());

        }

    }

}

Tabla 4. Codificación del archivo SendingFileThread

package ec.edu.monster.controlador;

import ec.edu.monster.vista.SendFile;

import java.io.BufferedInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

import java.text.DecimalFormat;

import javax.swing.JOptionPane;

/\*\*

 \*

 \* @author Michael

 \*/

public class SendingFileThread implements Runnable {

    protected Socket socket;

    private DataOutputStream dos;

    protected SendFile form;

    protected String file;

    protected String receiver;

    protected String sender;

 protected DecimalFormat df = new DecimalFormat("##,#00");

    private final int BUFFER\_SIZE = 100;

    public SendingFileThread(Socket soc, String file, String receiver, String sender, SendFile frm) {

        this.socket = soc;

        this.file = file;

        this.receiver = receiver;

        this.sender = sender;

        this.form = frm;

    }

    @Override

    public void run() {

        try {

            form.disableGUI(true);

            System.out.println("Enviar archivo..!");

            dos = new DataOutputStream(socket.getOutputStream());

            File filename = new File(file);

            int len = (int) filename.length();

            int filesize = (int) Math.ceil(len / BUFFER\_SIZE); //

            String clean\_filename = filename.getName();

            dos.writeUTF("CMD\_SENDFILE " + clean\_filename.replace(" ", "\_") + " " + filesize + " " + receiver + " "

                    + sender);

            System.out.println("De: " + sender);

            System.out.println("Para: " + receiver);

  InputStream input = new FileInputStream(filename);

            OutputStream output = socket.getOutputStream();

            BufferedInputStream bis = new BufferedInputStream(input);

            byte[] buffer = new byte[BUFFER\_SIZE];

            int count, percent = 0;

 while ((count = bis.read(buffer)) > 0) {

                percent = percent + count;

                int p = (percent / filesize);

                form.updateProgress(p);

                output.write(buffer, 0, count);

            }

            form.setMyTitle("Archivo enviado.!");

            form.updateAttachment(false); //

            JOptionPane.showMessageDialog(form, "Archivo enviado con éxito.!", "Éxito",

                    JOptionPane.INFORMATION\_MESSAGE);

            form.closeThis();

            output.flush();

            output.close();

            System.out.println("Archivo enviado..!");

        } catch (IOException e) {

            form.updateAttachment(false); //

            System.out.println("[SendFile]: " + e.getMessage());

        }

    }

}

**2.4.3.** PAQUETE VISTA PROYECTO CLIENTE

Finalmente se codifica los archivos de la vista para el cliente, por la cantidad de líneas de código existentes en estos archivos unicamente se mostrará su creación de los archivos.

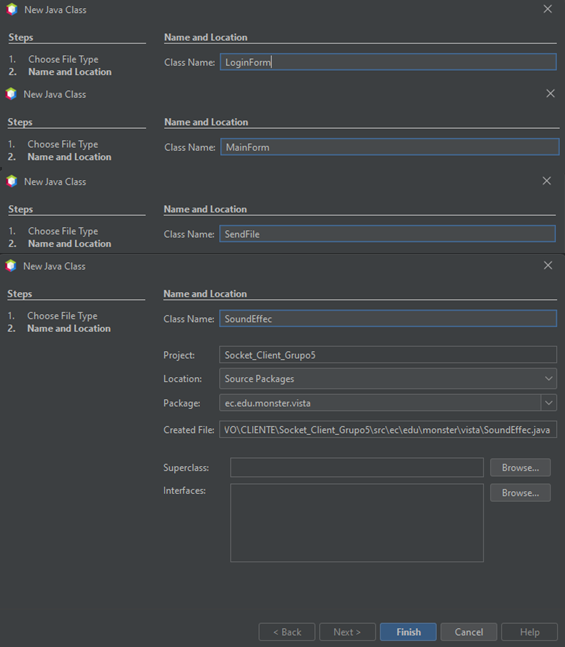


Figura 17. Creación de los archivos para la vista del Cliente.

## CODIFICACIÓN DEL PROYECTO SERVIDOR

Para ello comenzamos con la creación de los archivos que se van a utilizar en cada uno de los paquetes.

**2.5.1.** PAQUETE MODELO PROYECTO SERVIDOR

Para el desarrollo del proyecto servidor se deberán crear los archivos en el paquete **ec.edu.monster.modelo** para ello se deberá hacer clic derecho sobre el paquete, new y finalmente Java Class.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 18. Creación de los archivos para el paquete modelo del servidor.

Se configura el nombre del archivo y seguido de eso se hace clic en Finish.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 19. Asignación del nombre para el archivo del modelo.

Tabla 5. Codificación del archivo OnlineListThread

package ec.edu.monster.modelo;

import ec.edu.monster.vista.MainForm;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class OnlineListThread implements Runnable {

    MainForm main;

    public OnlineListThread(MainForm main) {

        this.main = main;

    }

    @Override

    public void run() {

        try {

            while (!Thread.interrupted()) {

                String msg = "";

                for (int x = 0; x < main.clientList.size(); x++) {

                    msg = msg + " " + main.clientList.elementAt(x);

                }

                for (int x = 0; x < main.socketList.size(); x++) {

                    Socket tsoc = (Socket) main.socketList.elementAt(x);

                    DataOutputStream dos = new DataOutputStream(tsoc.getOutputStream());

                    if (msg.length() > 0) {

                        dos.writeUTF("CMD\_ONLINE " + msg);

                    }

                }

                Thread.sleep(1900);

            }

        } catch (InterruptedException e) {

            main.appendMessage("[InterruptedException]: " + e.getMessage());

        } catch (IOException e) {

            main.appendMessage("[IOException]: " + e.getMessage());

        }

    }

}

**2.5.2.** PAQUETE CONTROLADOR PROYECTO SERVIDOR

Creamos los archivos que se van a utilizar en el controlador del proyecto servidor, por la cantidad de líneas de código existentes en el archivo SocketThread únicamente se agregará el código de ServerThread.

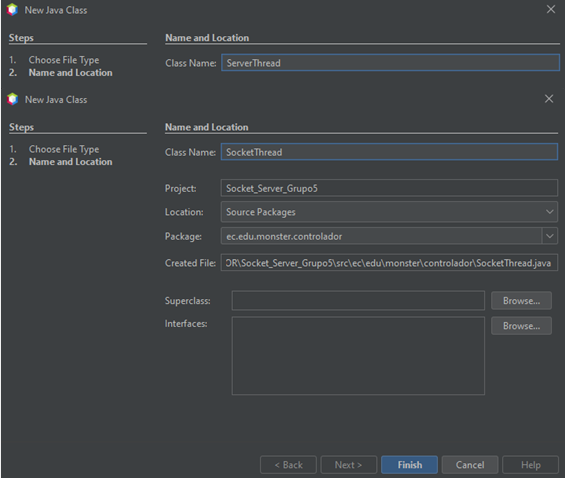


Figura 20. Creación y asignación de los archivos ServerThread y SocketThread.

Tabla 6. Codificación del archivo ServerThread

package ec.edu.monster.controlador;

import ec.edu.monster.vista.MainForm;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

 \*

 \* @author Michael

 \*/

public class ServerThread implements Runnable {

    ServerSocket server;

    MainForm main;

    boolean keepGoing = true;

    public ServerThread(int port, MainForm main) {

        main.appendMessage("[Server]: Servidor se esta iniciando en puerto " + port);

        try {

            this.main = main;

            server = new ServerSocket(port);

            main.appendMessage("[Server]: Servidor conectado.!");

        } catch (IOException e) {

            main.appendMessage("[IOException]: " + e.getMessage());

        }

    }

@Override

    public void run() {

        try {

            while (keepGoing) {

                Socket socket = server.accept();

                new Thread(new SocketThread(socket, main)).start();

            }

        } catch (IOException e) {

            main.appendMessage("[ServerThreadIOException]: " + e.getMessage());

        }

    }

    public void stop() {

        try {

            server.close();

            keepGoing = false;

            System.out.println("Servidor desconectado..!");

            System.exit(0);

        } catch (IOException e) {

            System.out.println(e.getMessage());

        }

    }

}

**2.5.3.** PAQUETE VISTA PROYECTO SERVIDOR

Se crea el archivo MainForm en el paquete de la vista, y por la cantidad de líneas de código de este archivo no se agregará su código en este documento.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 21. Creación y asignación del nombre al archivo del paquete vista del proyecto Servidor.

## EJECUCIÓN DE LA APLICACIÓN

Para poder ejecutar la aplicación hay que tener en cuenta que se tienen dos proyectos diferentes el uno es el proyecto Cliente y el otro es el proyecto Servidor, para lo cual primero se deberá ejecutar el proyecto Servidor haciendo clic derecho sobre el proyecto seguido de eso en Run.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 22. Ejecución del proyecto Servidor.

Seguido de eso se deberá levantar el servicio haciendo clic en iniciar servidor.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 23. Se hace clic sobre iniciar servidor

Una vez inicializado el servidor se tiene lo siguiente.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 24. Servidor iniciado.

Una vez iniciado el proyecto servidor se puede inicializar el proyecto cliente, ejecutando dos veces para poder tener dos usuarios diferentes para la conversación.

Captura de pantalla con letras y números

Descripción generada automáticamente

Figura 25. Ejecución del proyecto cliente.

Se mostrará la siguiente pantalla donde se indicará el nombre del usuario.

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Figura 26. Ventana principal del proyecto Cliente

Se indica el nombre del usuario que se va a crear.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 27. Creación de los usuarios Michael y Carlos.

Una vez que se establece la conexión entre los dos usuarios se puede mantener una conversación como se muestra en la siguiente imagen.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 28. Conversación entre Carlos y Michael.

Se puede realizar el envío de archivos entre los usuarios como se muestra en la siguiente imagen, para ello se deberá elegir la opción Escoger archivo, seguido de eso seleccionar.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 29. Envío de archivos.

Se deberá elegir el archivo que se desea enviar.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 30. Selección de archivos a enviar.

Una vez seleccionado el archivo se elige el Username como se muestra en la siguiente imagen.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 31. Insertar el Username.

Y se mostrará un mensaje que si se desea aceptar el archivo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 32. Aceptar archivo recibido.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 33. Se indica la ruta donde se desea guardar el archivo recibido.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 34. Mensaje de éxito tras enviar y recibir el archivo.

Se puede visualizar en el proyecto servidor las acciones que están siendo realizadas por el Cliente.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 35. Visualización de acciones desde el servidor.

Finalmente se detiene el servicio del Servidor.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 36 Detener el servidor.

# CONCLUSIONES

* Los sockets nos permiten implementar una arquitectura cliente – servidor donde la comunicación es iniciada por uno de los procesos el cual es denominado “Cliente” y finalmente se espera al segundo proceso inicie la comunicación por tal razón es conocido como servidor.
* Un socket nos permite el intercambio de información entre procesos en la misma máquina o en una red de tal manera que se puede distribuir el trabajo y tener el acceso a datos centralizados.
* Un socket es un proceso o hilo que existe en la máquina del cliente y servidor, la cual sirve como instancia para que un programa servidor y el cliente tengan acceso a la información la misma que es transmitida por varias capas de red.

# RECOMENDACIONES

* Antes de comenzar con el desarrollo de proyectos haciendo uso de sockets es importante el conocer los hilos.
* Para el desarrollo de la práctica es importante el conocer que es una arquitectura cliente – servidor, además, del Modelo Vista Controlador.

# REFERENCIAS

* <https://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf>
* Rivera, J. (s.f.). Sockets en Java. Obtenido de http://pedrobeltrancanessabiblioteca.weebly.com/uploads/1/2/4/0/12405072/mi\_manual\_sockets.pdf
* ULG. (s.f.). Socket programming. Obtenido de Complement for the programming assignment: <http://www.montefiore.ulg.ac.be/~hiard/Java%20Sockets.pdf>
* <https://cruzado.info/tutojava/V_2.htm>
* <https://www.youtube.com/watch?v=L0Y6hawPB-E&t=4s>
* Vila, J. (s.f.). Sockets en Java Sockets en J. Obtenido de https://poliformat.upv.es/access/content/group/OCW\_6069\_2008/T2.- Comunicaci%C3%B3n%20I%3A%20del%20C\_S%20al%20modelo%20de%20objetos/T ecnolog%C3%ADa%20JAVA/Java-sockets.pdf