IMPLEMENTACIÓN DE TECNOLOGÍAS SOCKETS, PERSISTENCIA DE DATOS Y TRANSACCIONES

ENTREGA 1 SEMANA 3

PRESENTADO A:

NELSON PEREZ

PRESENTADO POR:

MARIAN YISEKT ROMERO ORTIZ

SANTIAGO BETANCOURT

JUAN ESTEBAN VELASQUEZ

JHON SEBASTIAN RODRIGUEZ RINCON

CARLOS YECID ASCENCIO CELY

UNIVERSIDAD POLITÉCNICO GRAN COLOMBIANO

PERSISTENCIA DE DATOS Y TRANSACCIONALIDAD

NOVIEMBRE DE 2023

INTRODUCCION

Un socket es un punto de conexión utilizado en comunicaciones de red que permite la interacción entre dispositivos. En programación, los sockets se utilizan para establecer comunicaciones entre procesos remotos y locales. Estos pueden residir en el mismo sistema o en sistemas de redes diferentes.

**¿Qué es un socket?**

*Martin Santana & A. David, 2019*; afirma que: Los sockets son mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas. Una forma de conseguir que dos programas se transmitan datos.

Un socket no es más que un "canal de comunicación" entre dos programas que pueden o no estar en la misma máquina.

**TIPOS DE SOCKETS**

Existen dos tipos de socket:

socket stream (TCP) y socket datagram (UDP), según *(Pereira, 2019)*, existe una tercera categoría sockets raw, para los interesados en desarrollar nuevos protocolos de comunicación, en *(Murillo Morera & Juan de Dios, 2010)* define que socket stream está orientado a la conexión en flujo continuo y el datagrama está orientado al transporte sin conexión , básicamente el socket stream está orientado al protocolo TCP que garantiza el envío de toda la data y en el mismo orden de envío, y para el socket datagram está orientado al protocolo UDP que no garantiza que los paquetes de data lleguen pero es más rápido comparado con TCP.

Es decir, para analizar la creación de los sockets debemos tener en cuenta el protocolo y de qué lado se encuentra, es decir si es servidor o es cliente.

En *(https://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf, 2019),* indica que: Java nos proporciona el paquete java.net, que permite utilizar las clases para proporcionar comunicación de red independiente del sistema. Las clases URL, URLConnection, Socket, y SocketServer utilizan TCP para comunicarse a través de la Red. Las clases DatagramPacket y DatagramServer utilizan UDP.

El paquete *Java.net* nos proporciona entre otras las siguientes clases según *(https://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf, 2019):*

*Socket*: Implementa un extremo de la conexión TCP.

*ServerSocket*: Se encarga de implementar el extremo Servidor de la conexión en la que se esperarán las conexiones de los clientes.

*DatagramSocket*: Implementa tanto el servidor como el cliente cuando se utiliza UDP.

*DatagramPacket*: Implementa un datagram packet, que se utiliza para la creación de servicios de reparto de paquetes sin conexión.

*InetAddress*: Se encarga de implementar la dirección IP.

**Socket TCP. Servidor**

Según *(Belmonte Fernandez & Óscar)*: Para crear un Socket que acepte conexiones en un determinado puerto, Java nos proporciona la clase *java.net.ServerSocket*, en el momento de crear una instancia de esta clase indicamos el puerto de escucha. Para aceptar conexiones utilizamos el método *public Socket accept() throws IOException*, que es bloqueante, es decir, la ejecución se detiene en este método hasta que un cliente se conecta. Cada vez que un cliente se conecta al Socket servidor, el método *accept()* nos devolverá una referencia al Socket cliente (instancia de la clase Socket esta vez), a partir de la cual podremos obtener Flujos de lectura o escritura.

En*(https://www.infor.uva.es/-fidiaz/sd/doc/java.net.pdf, 2019)*, se detalla tanto los métodos como los constructores que se pueden utilizar según la clase *java.net* por la tanto se tiene que:

***Métodos:***

*public Socket accept () throws IOException*:

Sobre un ServerSocket se puede realizar una espera de conexión por parte del cliente mediante el método *accept()*. Hay que decir, que este método es de bloqueo, el proceso espera a que se realice una conexión por parte del cliente para seguir su ejecución. Una vez que se ha establecido una conexión por el cliente, este método devolverá un objeto tipo Socket, a través del cual se establecerá la comunicación con el cliente.

*public void close () throws IOException:*

Se encarga de cerrar el socket.

*public InetAddress getInetAddress ():*

Retorna la dirección IP remota a la cual está conectado el socket. Si no lo está retornará null.

*public int getLocalPort ():*

Retorna el puerto en el que está escuchando el socket.

*public static void setSocketImplFactory (SocketImplFactory fac) throws IOException:*

Este método establece la compañía de implementación del socket para la aplicación. Debido a que cuando una aplicación crea un nuevo socket, se realiza una llamada al método *createSocketImpl()* de la compañía que implementa al socket. Es por tanto en el parámetro fac, donde se especificará la citada compañía.

*public String toString()*:

Retorna un string representando el socket.

***Constructores:***

*public ServerSocket (int port) throws IOException:*

Se crea un socket local al que se enlaza el puerto especificado en el parámetro *port*, si se especifica jun 0 en dicho parámetro creará el socket en cualquier puerto disponible. Puede aceptar hasta 50 peticiones en cola pendientes de conexión por parte de los clientes.

*public ServerSocket (int port, int count) throws IOException:*

Aquí, el parámetro *count* sirve para que pueda especificarse el número máximo de peticiones de conexión que se pueden mantener en cola. Hay que recordar, que es fundamental que el puerto escogido sea conocido por el cliente, en caso contrario, no se podría establecer la conexión.