**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**



**Sistemas Distribuidos**

**Proyecto Final**

*Sistema de mensajería utilizando Redis PubSub*

**Integrantes**

*Sixto Castro Redrobán*

*Jordy Vásquez Cepeda*

*Marlon Espinoza Pacheco*

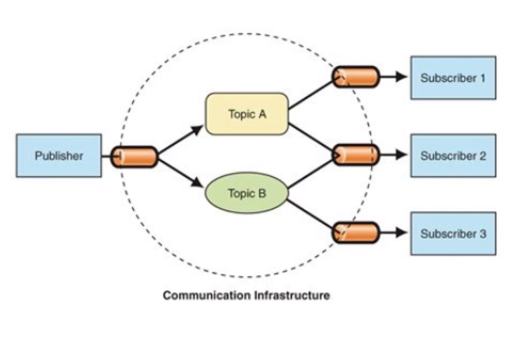
**Fecha de entrega**

16/08/2016

**OBJETIVOS:**

* Implementación de la comunicación cliente-servidor utilizando Redis como Middleware.
* Comunicación en tiempo real, es un tópico indispensable en un sistema de mensajería instantánea.
* Envío y recepción de mensajes entre varios clientes suscriptos a un canal específico.
* Implementación de la suscripción multicanal de tal manera que el Cliente pueda suscribirse y chatear en distintos canales.

**INTRODUCCIÓN**



Las características de un modelo publicador-suscriptor tal como se muestra en la imagen son:

* Extender la infraestructura de la comunicación mediante tópicos (temas) para **cada mensaje.**
* Permitir que las aplicaciones que escuchan (suscriptores) puedan recibir mensajes sólo de tópicos específicos.
* Crear un mecanismo que envíe mensajes a todos los suscriptores interesados.

**DESCRIPCIÓN DEL PROYECTO - PROPUESTA**

Este proyecto consiste en la implementación de un sistema de mensajería multiusuario y multicanal mediante la utilización de Redis, un middleware que abstrae la gestión del puerto de escucha y envío de mensajes bajo un esquema publish-subscribe.

El programa está hecho en Java, en el cual el usuario puede crear grupos de chat o suscribirse en grupos ya creados para luego poder acceder al chat respectivo con los usuarios que forman parte del canal. También se le da al usuario la opción de desafiliarse de un grupo (unsubscribe) si ya no quiere recibir mensajes o pertenecer a un grupo en específico.

Y como **extra** al proyecto**,** se ha implementado esta aplicación con sus respectivas interfaces gráficas (GUI) con el fin de tener una mejor interacción por parte del usuario ya que en un principio se lo había hecho sin interfaz gráfica (solamente en consola).

**METODOLOGÍA**

**Middleware**

Se escogió Redis como Middleware en nuestro proyecto dada las siguientes razones:

● Redis pub/sub es uno de los patrones de cola de mensajes más comunes donde hay múltiples publishers(publicadores), que envían mensajes a múltiples suscriptores que están escuchando mensajes en canales específicos.

● Redis es un almacén de estructura de datos de alto rendimiento, tiene agregado soporte para pub/sub por encima de la funcionalidad existente.

● Redis está construido en 3 comandos: publish, subscribe, y unsubscribe.

● Todos los comandos en una transacción son secuencialmente ejecutados como una sola operación aislada (operación atómica). Cada tipo de datos tiene su conjunto de características para manipular los datos en una manera atómica.

● Peso ligero (Lightweight): Usando Redis desde cualquier lenguaje o plataforma es muy sencillo, aprovisionarlo, y mantenerlo en un servidor de producción es fácil.

● Redis pub/sub provee escalabilidad: Cuando se levanta un servicio, se desea más de una instancia del servidor ejecutándose. El middleware provee la facilidad de poder escalar el servicio de chat.

**Lenguaje de programación - Librerías**

Java debido a que este lenguaje es multiplataforma (se lo puede ejecutar en varios sistemas operativos) y fácil de implementar en este lenguaje.

Redis ofrece un API para la parte del Cliente que en este caso vendría a ser la librería Cliente de Java “Jedis”, la cual es recomendada por la página oficial de Redis.

Librerías utilizadas:

* jedis 2.4.2
* commons-pool2-2.4.2

**Estructuras de datos - clases utilizadas**

Entre las clases principales, se implementaron las clases Publisher y Subscriber. La clase Publisher tiene su propia método constructor, y método para poder publicar mensajes; mientras que la clase Subscriber hereda de la clase JedisPubSub y se le hace una sobreescritura al método principal onMessage, el cual es llamado cada vez que el suscriptor recibe un mensaje.

Se utilizó un hashmap en el cual se registra como clave el canal y como valor el nombre de usuario del Cliente (puede tener un nombre de usuario igual o distinto para cada canal). Esta estructura se utilizó con el fin de saber a qué canales el Cliente se ha suscrito, y luego mostrarle los grupos suscritos en la sección “Ver grupos” de la aplicación.

En un principio la aplicación sólo permitía que el Cliente pueda suscribirse a un canal. Luego se lo extendió a suscripción “multicanal”; y para poder ver los mensajes de cualquier canal en específico, se implementó:

Un hashmap el cual contenga como clave el canal al cual el Cliente se acaba de suscribir, y como valor una lista, la cual contiene todos los mensajes de ese canal a partir del momento que el Cliente se suscriba.

Cabe recalcar que al recibir un mensaje en cualquier canal suscrito, mediante una variable string de control puedo saber en qué canal me encuentro para luego validar el canal, buscarlo en el hashmap, y actualizar la respectiva lista de mensajes asociada al canal.

En lo que respecta a Eliminación de grupo (Unsubscribe), dado el hecho de que se tiene en un hashmap el canal como clave y el suscriptor (subscriber) como valor; mediante el canal se puede obtener el suscriptor para luego desafialiarlo del grupo indicado al aplicar el método unsubscribe que es uno de los métodos principales ya implementados en la librería Jedis.

**Manejo de errores - Validaciones**

Mediante las respectivas depuraciones (debuggings) en cada uno de los escenarios que presenta la aplicación, se pudo encontrar y solucionar falencias que presentaba al comienzo.

A continuación se detalla las principales soluciones:

* Se usa los respectivos try-catch con el fin de que la aplicación no deje de funcionar, por ejemplo: cuando existe una suscripción fallida, o una conexión fallida con el servidor.
* Se utiliza mensajes de advertencia y validaciones:
* Cuando el usuario trata de suscribirse a un canal ya suscrito anteriormente. Como feedback al usuario se le muestra el respectivo mensaje de que ya se encuentra suscrito a dicho canal.
* Cuando el usuario no ingresa todos los campos al crear un canal. Se le muestra un mensaje indicando que debe llenar los campos faltantes para que pueda crear el canal.
* Al usuario se le asocia un hashmap para tener clasificado por canal, la lista de mensajes recibidos; debido a que al principio fue un inconveniente saber de qué canal recibía mensajes. En cambio con una estructura de datos apropiada, se pudo facilitar para poder ver los mensajes por canal.

**Mecanismos de sincronización**

No fue necesario implementar mecanismos de sincronización debido a que el Middleware garantiza atomicidad en cada una de sus transacciones (operaciones).

Todas las operaciones son atómicas, lo que asegura que si dos o más clientes acceden simultáneamente al servidor Redis obtendrán el valor actualizado.

**LINK DEL REPOSITORIO**

[**https://github.com/Sixto-Castro93/Proyecto-Sistemas-Distribuidos**](https://github.com/Sixto-Castro93/Proyecto-Sistemas-Distribuidos)

**CONCLUSIONES**

* Redis es un middleware que abstrae la parte de la conexión socket. Con tan solo mandarle la ip del servidor, internamente ya hace la conexión y asume el puerto 6379 por default.
* Fue útil utilizar las funciones principales que la librería Cliente Jedis ofrece tanto para suscripción de canales y para publicar mensajes en diferentes canales.
* Los comandos en Redis al momento de realizar una transacción son secuencialmente ejecutados como una sola operación aislada, lo cual garantiza atomicidad en las operaciones.

**BIBLIOGRAFÍA**

Redis Pub/Sub, Fecha de acceso: 27-07-2016

**Link:** [**http://redis.io/topics/pubsub**](http://redis.io/topics/pubsub)

Redis - Publish Subscribe, Fecha de acceso: 28-07-2016

**Link:** [**http://www.tutorialspoint.com/redis/redis\_pub\_sub.htm**](http://www.tutorialspoint.com/redis/redis_pub_sub.htm)

David Marquis, Reliable Delivery Pub/Sub Message Queues with Redis, Enero 3, 2013, Fecha de acceso: 05-08-2016

**Link:**

[**https://davidmarquis.wordpress.com/2013/01/03/reliable-delivery-message-queues-with-redis/https://davidmarquis.wordpress.com/2013/01/03/reliable-delivery-message-queues-with-redis/**](https://davidmarquis.wordpress.com/2013/01/03/reliable-delivery-message-queues-with-redis/https://davidmarquis.wordpress.com/2013/01/03/reliable-delivery-message-queues-with-redis/)

[Sankha Narayan](https://thelimbeck.wordpress.com/author/sankha93/), Simple chatrooms with Redis, Mayo 25, 2013, Fecha de acceso: 25-07-2016

**Link:** [**https://thelimbeck.wordpress.com/2013/05/25/simple-chatrooms-with-redis/**](https://thelimbeck.wordpress.com/2013/05/25/simple-chatrooms-with-redis/)

Robert Nimmo, Publish/Subscribe A Message Pattern, Abril 12, 2015

**Link:** [**https://www.youtube.com/watch?v=frGy-nGoGUY**](https://www.youtube.com/watch?v=frGy-nGoGUY)