# Patrones EIP

La utilización de patrones EIP facilita la creación de aplicaciones potentes, útiles y bien estructuradas. Facilita la creación de sistemas escalables debido al uso de pautas de diseño proporcionando soluciones sencillas a problemas comunes.

Los dos patrones más importantes en este sistema son el patrón repositorio y la arquitectura dirigida por eventos y varios otros dentro de este último.

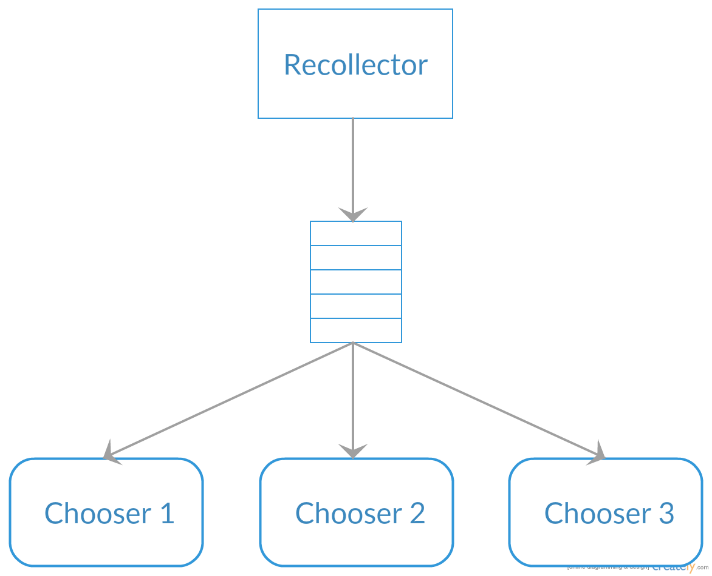
# Patrones de canales

A continuación, se exponen los patrones de canales de mensajería que han sido usados o tenidos en cuenta a la hora de realizar la implementación del sistema.

## Canal punto a punto

Para poder leer y procesar completamente todos los mensajes que provienen de la API de *streaming* de Twitter, se ha tenido que hacer uso de tres *choosers* simultáneos en la aplicación. Se ha habilitado un canal donde los *tweets* se van almacenando a la espera de ser leídos.

El canal permite que haya varios *choosers* consumiendo mensajes concurrentemente, es decir, son consumidores competidores. Se garantiza que cada *tweet* será consumido solo por un *chooser.*

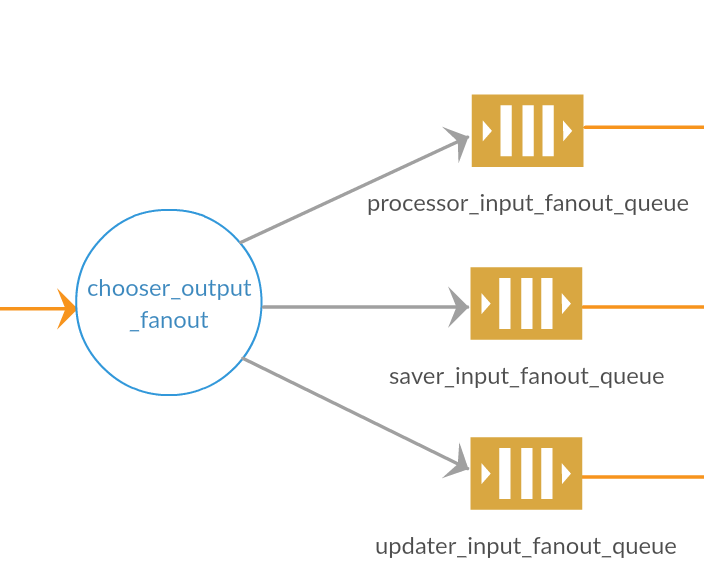


## Publicación-suscripción

El primero se utiliza en los WebSockets entre el servidor y los clientes. En este caso cada cliente se suscribe a un tema y el servidor publica mensajes sobre ese tema. En este caso se realiza con los tweets que se deben mostrar en tiempo real en la interfaz.

Existe otra aplicación de este patrón en la aplicación. Esta vez está implementada con *RabbitMQ*

gracias a unas colas en modo *fanout*. Los *chooser* dejan pasar los tweets que nos interesan. Estos tweets se triplican para el *Saver*, el *Updater* y el *Processor* 1



## Canal con tipo de datos

Este patrón resuelve el problema de que el emisor pueda enviar datos que el suscriptor sepa como procesar. En este caso sabemos que todos los datos que van a viajar por el canal son POJOs de JAVA serializados. Más concretamente se trata de Tweets.

## Adaptador de canal

En este caso queríamos conectar una aplicación al sistema de mensajería. En este caso es la aplicación de *streaming* de Twitter. Se usa un adaptador de canal que accede a la API de *streaming* y publica los tweets como objetos de Java serializados.

# Patrones de mensajes

## Mensaje evento

Los módulos de la aplicación *Saver*, *Updater* y los dos *Processor* son elementos que reaccionan cuando llegan mensajes desde el bróker de AMQP. Por lo tanto, podemos considerar esta arquitectura como un patrón de mensajes de evento.

# Patrones de enrutado de mensajes

## Filtro de mensajes

Este patrón se utiliza para evitar que un componente reciba mensajes que no le interesan. Esto es el ejemplo claro de lo que hacen los *chooser*. Se usa un filtro de mensajes. que elimine los tweets no deseados del canal a partir de ciertos criterios. En este caso, los criterios son modificables en tiempo de ejecución.

# Patrones de transformación de mensajes

## Enriquecedor de contenido

Se hace uso de este patrón para enriquecer el Tweet original que viene de Twitter. Se cambia la estructura del objeto para permitir un acceso más cómodo a los datos que contiene. Esto se hace por ejemplo en el texto original, se modifica la estructura del objeto para permitir modificar este texto.

## Filtro de contenido

Otra manera de ver el trabajo de los *choosers*. Realmente se hace un filtro por el contenido del texto de los tweets. Esto hace que este patrón esté reflejado en el sistema.

# Modelo arquitectural

## Patrón repositorio

El patrón repositorio se utiliza de manera constante en esta aplicación. Esto es debido a que la mayoría de componentes hacen consultas a una base de datos. Estas consultas se realizan a través de los llamados repositorios. Estos repositorios abstraen las consultas a la base de datos. A través de estos repositorios se lee la información relativa a Tweets, Usuarios, Conexiones, Participantes…

Estos repositorios son una abstracción proporcionada por la integración de Spring con MongoDB. Se expone una interfaz de operaciones varias sobre los elementos de cada repositorio.

La base de datos es externa, es decir, está en una máquina ajena al sistema. En este caso concreto, se trata de una base de datos MongoDB alojada en una máquina virtual de Microsoft Azure.