**Informe de Diseño de Software: Bot de Telegram**

# Introducción

Este informe detalla la implementación y diseño de un bot de Telegram que se comunica con la API de Telegram utilizando longpollingBot. Este diseño se eligió por su simplicidad y facilidad de implementación en comparación con otras alternativas como WebhookBot. A continuación, se presenta una explicación detallada de las decisiones de diseño, las ventajas y desventajas de las tecnologías utilizadas, el diagrama de componontes y la justificación de cada elección.

# Diagrama de Componentes



## Bot

### Comunicación con la API de Telegram mediante long polling

Long polling es la técnica de conexión elegida. Consiste en que el bot mantiene una solicitud abierta a la API de Telegram y espera hasta que haya un mensaje disponible antes de responder. Este método se eligió por las siguientes razones:

#### Ventajas

* + **Simplicidad:** Requiere menos configuración inicial en comparación con WebhookBot, lo cual permitió un desarrollo más rápido del proyecto.
  + **Escalabilidad:** Aunque WebhookBot puede ser más escalable a largo plazo, longpollingBot también ofrece opciones para escalar el proyecto según la demanda actual.
  + **Menos infraestructura necesaria:** No requiere configurar un servidor web adicional para recibir las actualizaciones.

#### Desventajas

* + **Latencia:** Puede haber un ligero aumento en la latencia en comparación con WebhookBot, que recibe las actualizaciones de manera más inmediata.
  + **Uso de recursos:** Mantener una conexión abierta puede consumir más recursos a largo plazo, especialmente en proyectos de gran escala.

### Redis como caché del bot

Redis se seleccionó como la caché del bot para este proyecto por varias razones:

#### Ventajas

* + **Estructura conveniente:** Redis es una base de datos clave valor. Esto nos proporciona la estructura que necesitamos, ya que todo valor que necesitamos guardar va a ser accedido mediante el ID del chat que nos proporciona telegram.
  + **Rendimiento:** la rapidez que tiene Redis nos permite realizar estas consultas de manera casi instantáneas, sin perder una cantidad significativa de rendimiento en cada interacción con el usuario.
  + **Escalabilidad horizontal:** al no tener la información de las interacciones con el usuario en memoria, sino que centralizada en un nodo aparte, podemos lanzar varias instancias del bot, aumentando el rendimiento del bot en las interacciones.

#### Desventajas

* + **Complejidad:** contar con un componente de arquitectura más siempre agrega más responsabilidades, testeos y controles.
  + **Consistencia:** En comparación con las bases de datos SQL, Redis puede presentar problemas de consistencia debido a su modelo de consistencia eventual no relacional.
  + **Single point of failure:** el bot depende enteramente del Redis para su funcionamiento. Si el servidor Redis se cae, el bot no funcionará.

### Pruebas unitarias

Para las pruebas unitarias del bot, principalmente utilizamos 3 paquetes de Java:

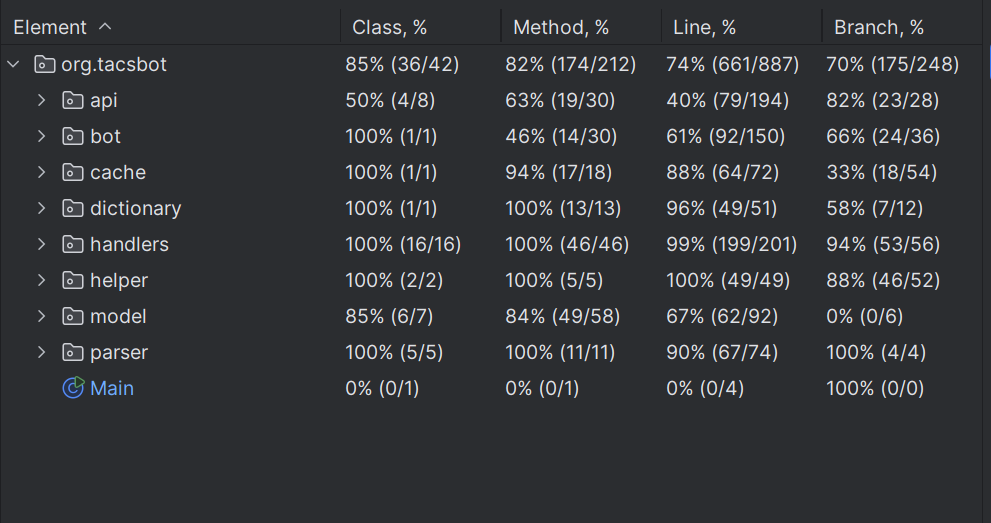
* **Mockito:** para mockear objetos y poder realizar pruebas que de otra forma serían imposibles, como cuantas veces se llamó a determinado método, lo cual fue indispensable para comprobar que se manden mensajes, o probar respuestas simulando interacciones con la API.  
  URL: <https://mvnrepository.com/artifact/org.mockito/mockito-core>
* **Redis embebido de ozimov:** para poder levantar un Redis en memoria y poder hacer tests completos del bot, con la interacción con el bot incluida.  
  URL: <https://mvnrepository.com/artifact/com.github.codemonstur/embedded-redis>
* **JUnit:** para correr las pruebas unitarias en Java.  
  URL: <https://mvnrepository.com/artifact/junit/junit>

#### Componentes de la aplicación sin pruebas

No se logró testear los componentes de interacción con la API, ya que para eso debíamos hacer una prueba de integración más compleja que se nos iba del alcance del trabajo.

#### Coverage del código

A continuación podemos ver una captura del análisis de Coverage reportado por Intellij.



## API

La API que se comunica con el bot fue desarrollada utilizando un servidor Tomcat. Esta API sigue el estilo de arquitectura REST y se conecta a una base de datos MongoDB.

### Elección de MongoDB como Base de Datos

MongoDB se seleccionó como la base de datos para este proyecto por varias razones:

#### Ventajas

* + **Flexibilidad en el manejo de datos:** MongoDB, siendo una base de datos NoSQL, permite almacenar datos en un formato flexible, ideal para manejar estructuras de datos no uniformes y cambiantes.
  + **Escalabilidad horizontal:** Es fácil de escalar horizontalmente añadiendo más servidores a la base de datos.
  + **Rendimiento:** Ofrece un alto rendimiento para operaciones de lectura y escritura, lo que es crucial para un sistema que maneja múltiples solicitudes concurrentes.

#### Desventajas

* + **Consistencia:** En comparación con las bases de datos SQL, MongoDB puede presentar problemas de consistencia debido a su modelo de consistencia eventual.
  + **Transacciones:** Aunque MongoDB ha mejorado en este aspecto, las transacciones complejas pueden ser más difíciles de implementar en comparación con bases de datos relacionales como PostgreSQL.

### CRON de deadline

Para el caso de uso de la deadline de cada artículo, se recurrió a un CRON que corre todos los días a las 00hs. Este CRON está configurado para correr en un container Ubuntu mediante crontab, y lo que hace es un POST al path /CRON/deadline de la API, haciendo que la API procese el cerrado de los artículos ya vencidos.

### Pruebas unitarias

Para las pruebas unitarias de la API, principalmente utilizamos 3 paquetes de Java:

* **Mockito:** para mockear objetos.  
  URL: <https://mvnrepository.com/artifact/org.mockito/mockito-core>
* **JUnit:** para correr las pruebas unitarias en Java.  
  URL: <https://mvnrepository.com/artifact/junit/junit>
* **MongoDB embebido:** para generar una base mongo en memoria.

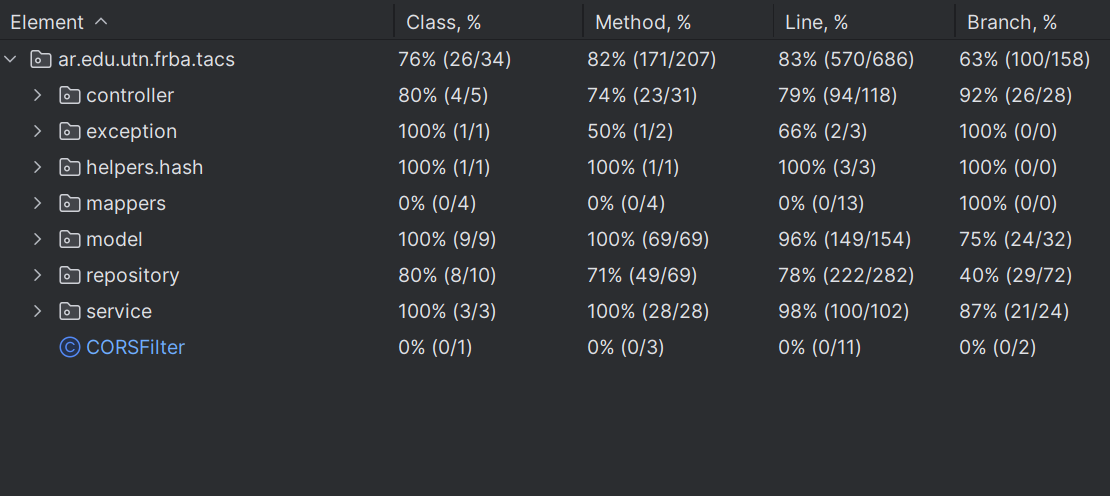
URL: <https://mvnrepository.com/artifact/de.flapdoodle.embed/de.flapdoodle.embed.mongo>

#### Componentes de la aplicación sin pruebas

Se pudo probar todo en la API.

#### Coverage del código

A continuación podemos ver una captura del análisis de Coverage reportado por Intellij.



## Frontend

El frontend es el componente de la aplicación que se corre en los navegadores y permite que los usuarios no dependan de una aplicación en concreto para poder interactuar con ella.

## React como framework

…

#### Ventajas

* + El código se ejecuta siempre del lado del cliente por lo que el servidor tiene menor carga y responsabilidades.
  + React permite cambiar de estado a los componentes sin tener que recargar la página o tener que llamar al servidor por lo que permite tener una web dinámica más fluida.
  + Ahorra muchas idas y vueltas al servidor ya que el HTML lo genera en el cliente.

#### Desventajas

* + Al ejecutarse el código del lado del cliente hay una menor seguridad porque los requests a la API se hacen desde el cliente.

## Despliegue

### Contenedores Docker

El bot, la API y la base de datos están embebidos en contenedores Docker, lo que proporciona varias ventajas:

#### Ventajas

* + Portabilidad: Los contenedores Docker aseguran que el software funcione de manera idéntica en cualquier entorno, eliminando problemas de compatibilidad.
  + Aislamiento: Permite aislar las aplicaciones y sus dependencias, mejorando la seguridad y la estabilidad del sistema.
  + Facilidad de despliegue: Simplifica el proceso de despliegue y escalado de las aplicaciones.

### Despliegue en AWS y Orquestación con Kubernetes

Todo el sistema está desplegado en AWS, utilizando Kubernetes para la orquestación y escalado automático en función de la demanda.

#### Ventajas

* + Escalabilidad automática: Kubernetes permite escalar los servicios automáticamente según la demanda, asegurando un rendimiento óptimo.
  + Alta disponibilidad: Kubernetes distribuye las solicitudes de manera eficiente, garantizando alta disponibilidad del servicio.
  + Gestión simplificada: Facilita la gestión de los contenedores y los recursos del sistema.

### Instancia del Bot en Modo Cron

Además de las funcionalidades interactivas, se implementó un hilo en el bot que corre los procesos relacionados con el envío de notificaciones. En la versión ideal, se haría una instancia diferente del bot en modo Cron para el envío de notificaciones pendientes. Esta instancia no recibe mensajes, sino que se encarga de enviar notificaciones programadas y actualizar el estado de las mismas en la base de datos para evitar inconsistencias y duplicados.

Se implementaron notificaciones para los siguientes casos de uso:

* **Un usuario se suscribe a un artículo**:   
  -Se notifica al dueño del articulo que se ha suscripto un nuevo miembro.  
  -Se notifica al resto de suscriptos que un usuario nuevo se ha suscrito al artículo.
* **El artículo se cierra sin llegar a la cantidad mínima de usuarios suscritos**:  
  -Se notifica al dueño del articulo que se ha suscripto un nuevo miembro.  
  -Se notifica al resto de suscriptos que un usuario nuevo se ha suscrito al artículo.  
  Nota: En ambos casos se informa a su vez que se cancela la operación por no llegar a la cantidad mínima de suscriptores. Cabe aclarar que un articulo se cierra por 3 motivos: se alcanza el deadline, se alcanza el maximo de usuarios suscritos o lo cierre el dueño.
* **El artículo se cierra alcanzando o superando la cantidad mínima de usuarios suscritos**:  
  -Se notifica al dueño del articulo que se ha suscripto un nuevo miembro.  
  -Se notifica al resto de suscriptos que un usuario nuevo se ha suscrito al artículo.  
  Nota: En ambos casos se informa a su vez que ya se puede realizar la operación.

**Ejemplos de Notificaciones enviadas**

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# 

# Conclusión

La combinación de longpollingBot, una API REST en Tomcat, MongoDB, contenedores Docker, AWS y Kubernetes ha permitido desarrollar un bot de Telegram eficiente y escalable. Aunque hay algunas desventajas asociadas con cada elección tecnológica, las ventajas y la justificación de diseño hacen que estas elecciones sean adecuadas para las necesidades actuales y futuras del proyecto.

*Nota:* Este informe indica el proyecto ideal a futuro pero no representa el estado actual del proyecto.