

# -Trabajo Práctico N. 2

## Reddit Memes Analyzer

**Fecha:**

9 de Junio  
2022

**Docentes:**

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Ana Czarnitzki
- Cristian Raña

**Alumno:**

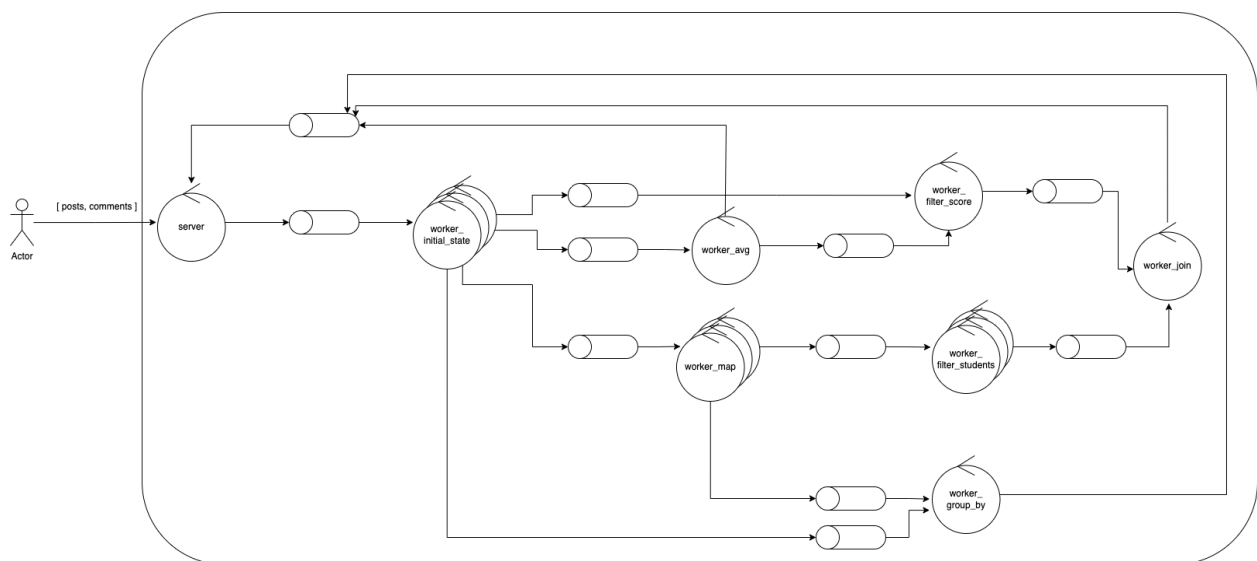
- Sebastian Ripari (96453)

# Introduccion

El trabajo práctico fue desarrollado en **Rust**. Se llega a la solución mediante procesamiento coordinado de varios procesos. Para la comunicación de los mismos se utilizó queues, provistos por **RabbitMQ**.

## Procesos

Se cuenta con procesos que poseen diferentes responsabilidades. Los que se encargan de tareas más costosas y tienen la ventaja de que eran sin estados, fueron replicados.



*Diagrama de Robustez*

Se enuncia a continuación la principal actividad de cada proceso:

**server**: Recibe y envía mensajes al cliente por socket.

**worker\_initial\_state**: consume posts y comments, para dárselos al proceso correspondiente.

**worker\_avg**: consume todos los score, calcula el average.

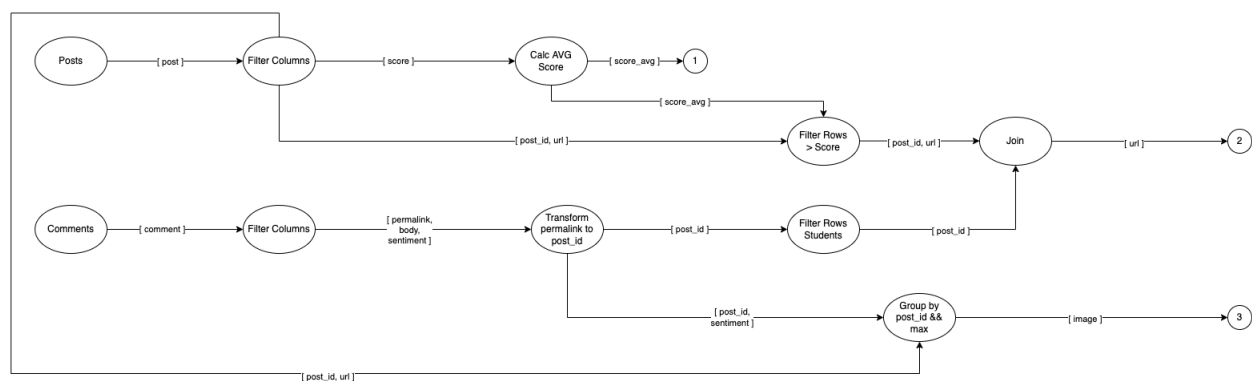
**worker\_map**: consume todos los permalink de los comments, los parsea para encontrar el post\_id.

**worker\_filter\_students**: consume todos los body de los comments, indicando cuales matchean con un estudiante.

**worker\_filter\_score:** consume todos los post y score average filtrando aquellos post que tienen un score mayor al average.

**worker\_join:** consume los post que tengan un score superior al average y los comments de estudiantes. Hace un join de estos por post\_id, encontrando cuáles post cumplen ambas condiciones.

**worker\_group\_by:** consume los post y los comments, hace un group\_by post\_id de los comments y realiza la función de agregación average sobre el sentiment. Luego se queda con el que posee el máximo.



DAG

## Queues

Se cuenta con varias queue, estas son:

AVG\_TO\_FILTER\_SCORE  
QUEUE\_INITIAL\_STATE  
QUEUE\_COMMENTS\_TO\_FILTER\_STUDENTS  
QUEUE\_COMMENTS\_TO\_GROUP\_BY  
QUEUE\_COMMENTS\_TO\_JOIN  
QUEUE\_COMMENTS\_TO\_MAP  
QUEUE\_POSTS\_TO\_AVG  
QUEUE\_POSTS\_TO\_FILTER\_SCORE  
QUEUE\_POSTS\_TO\_GROUP\_BY  
QUEUE\_POSTS\_TO\_JOIN

Cada una de ellas fue pensada como punto de entrada de cada uno de los diferentes workers, por ejemplo `QUEUE_COMMENTS_TO_FILTER_STUDENTS` es aquella que es utilizada por el `worker_filter_students`.

# Middleware

Toda la comunicación de un proceso con otro, es encapsulada por medio de un middleware. Este realiza toda la funcionalidad, desde la creación y destrucción del canal y de los recursos, hasta el envío de mensajes. Estos mensajes pueden ser de dos tipos, el mensaje normal que contiene información, o el mensaje end, que indica que ya no habrá más mensajes para consumir.

Funcionalidad:

```
middleware_connect  
middleware_create_channel  
middleware_declare_queue  
middleware_create_consumer  
middleware_create_exchange  
middleware_send_msg_end  
middleware_send_msg  
middleware_end_reached  
middleware_consumer_end
```

Hubiese sido ideal juntar connect, create\_channel y create\_exchange en una sola, pero por dificultades del lenguaje no fue posible. Otro deseable hubiese sido poder crear una clase que tenga como atributos, el exchange, el channel y la conexión, pero por lo mismo no fue posible. Así que la interfaz es al estilo C, donde por parámetro se pasa todo lo que se necesita.

## Detección de parar de consumir

Se utilizó la técnica de mandar mensajes indicando el end. Debido a que algunos procesos están replicados, es necesario que los productores manden tanto mensajes de end como consumidores allá. Toda lógica se encuentra dentro del middleware, más precisamente en `middleware_consumer_end`. Comienza usando la función `middleware_end_reached` detecta cuando le llegó la cantidad de ends necesaria para parar de consumir. Y en este punto `middleware_send_msg_end` sabe como enviar los ends a los consumidores, cuántos de ellos enviar.