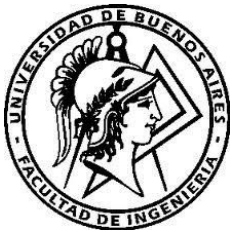


Universidad de Buenos Aires

Facultad de Ingeniería

75.61 - Taller de Programación III

Trabajo Práctico N° 2

Buzzer - Kafka

2º Cuat. - 2017

Titular ..……..	Andres Veiga
Jefe de Trabajos Prácticos.....…..	Pablo Roca
Ayudante……...…..	Ezequiel Torres Feyuk
Alumno …….	Pablo Méndez

Fecha de re-entrega ……..……..	02/10/2017

Índice

1. Introducción

El presente trabajo consta en desarrollar un sistema distribuido que simule una red social; mediante la cual, los usuarios podrán realizar el envío y recepción de mensajes entre ellos de acuerdo a su interés particular, así como también su posterior consulta ya sea por el usuario que lo originó o por hashtag, borrar mensajes posteados con anterioridad y consultar el *trending topic* del día.

Para este trabajo es necesario desarrollar varios componentes que conforman el sistema y realizar la comunicación entre ellos mediante un broker de mensajería; en este caso Apache Kafka, el cual conforma un sistema de mensajería empresarial completo y altamente confiable brindando una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos y mediante el cual se busca cumplimentar los objetivos que este trabajo implica y que son listados a continuación:

- Crear un módulo de despacho de mensajes para que puedan ser recibidos por otras personas ya sean, seguidores del mismo o estén interesados en alguno de los temas de los que ellos hacen referencia.

- Crear un módulo de eliminación de mensajes que le permita a los usuarios borrar de forma permanente aquellos que fueron creados con anterioridad.

- Crear un módulo de consultas para poder obtener una lista de los diez últimos mensajes posteados con anterioridad ya sea por usuario o por hashtag.

- Crear un módulo de consultas para obtener el trending topic del día, el cual presenta una lista de los diez temas más comentados hasta el momento junto con la cantidad de temas comentados en total durante el día.

- Crear un módulo de almacenamiento permanente de mensajes para que los mismos puedan ser recuperados en consultas posteriores o borrados definitivamente del sistema.

- Crear un módulo de registración y seguimiento que le permita a los clientes del sistema seguir a otros usuarios o temas de interés (*hashtags*).

- Crear un módulo de auditoría que permita realizar la trazabilidad de los mensajes que circulan diariamente por el sistema.

2. Solución propuesta

A continuación se presenta la solución propuesta basada en las vistas del modelo de arquitectura 4+1 de Kruchten; esto es, Vista lógica, Vista de desarrollo, Vista física, Vista de procesos y Escenarios seleccionados.

Para este caso se presentarán y analizarán todas ellas a partir de sus diagramas correspondientes mediante los cuales se intentará describir la arquitectura del sistema desarrollado:

- Vista lógica: Esta vista presenta la funcionalidad que el sistema proporciona a los usuarios finales. Para su modelado se utilizarán los diagramas de clases.

- Vista de desarrollo: Esta vista que muestra al sistema desde la perspectiva de un programador haciendo foco en cómo está dividida la aplicación en componentes de software y las dependencias que hay entre estos últimos.

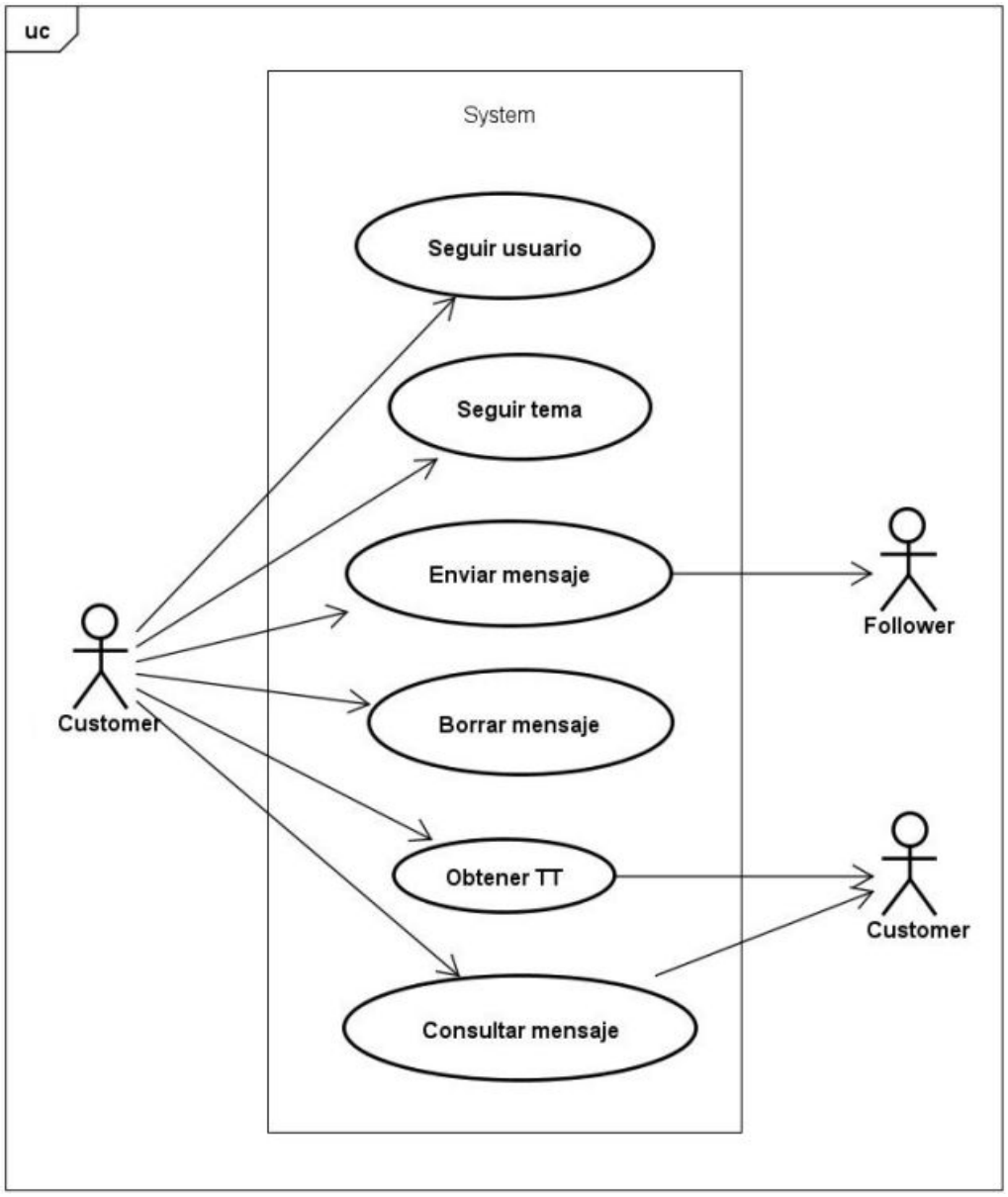
- Vista física: Modela el sistema presentando un esquema de los componentes físicos, lógicos y como es la comunicación entre ellos. En este caso se utilizará un diagrama de despliegue y otro de robustez.

- Vista de procesos: Muestra los procesos que conforman el negocio y como es la comunicación entre ellos. Para modelizarla se utilizará un diagrama de actividades.

- Escenarios seleccionados: Estos escenarios están modelados por el diagrama de casos de uso y cumple la función de unir las otras cuatro vistas mediante la presentación de la interacción de los usuarios con el sistema.

2.1. Escenarios seleccionados

A continuación se presenta el diagrama de casos de uso del sistema desarrollado, junto con el detalle la especificación de cada uno de los casos de uso que lo conforman.



Caso de Uso: CU001 - Seguir Usuario
Descripción: El cliente envía una petición al sistema para seguir los mensajes que postea un usuario determinado
Actores participantes: Customer
Pre-condiciones: -

Flujos
Flujo Principal
1 | El customer inicia sesión en el sistema
2 | El sistema solicita ingresar un comando
3 | El customer ingresa el comando FOLLOW: <NOMBRE DE USUARIO A SEGUIR> {**A1**} {**E2**}
4 | El sistema registra la solicitud {**E1**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla que ha ocurrido un error el el procesamiento de la solicitud
E1.2 | El caso de uso continúa el el punto 2 del Flujo principal
E2.1 | El sistema informa por pantalla que el comando ingresado es inválido
E2.2 | El caso de uso continúa el el punto 2 del Flujo principal
Post-condiciones: El customer queda registrado como seguidor del usuario solicitado y a partir de este momento comienza a recibir los mensajes que emite este último.

Caso de Uso: CU002 - Seguir Tema
Descripción: El cliente envía una petición al sistema para seguir los mensajes que contengan un hashtag determinado
Actores participantes: Customer
Pre-condiciones: -

Flujos
Flujo Principal
1 | El customer inicia sesión en el sistema
2 | El sistema solicita ingresar un comando
3 | El customer ingresa el comando FOLLOW: #<NOMBRE DEL HASTAG A SEGUIR> {**A1**}
4 | El sistema la solicitud {**E1**} {**E2**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla que ha ocurrido un error el el procesamiento de la solicitud
E1.2 | El caso de uso continúa el el punto 2 del Flujo principal
E2.1 | El sistema informa por pantalla que el comando ingresado es inválido
E2.2 | El caso de uso continúa el el punto 2 del Flujo principal
Post-condiciones: El customer queda registrado como seguidor del hashtag solicitado y a partir de este momento comienza a recibir los mensajes que contengan a este último.

Caso de Uso: CU003 - Enviar Mensaje
Descripción: El cliente envía un mensaje hacia el sistema.
Actores participantes: Customer - Follower
Pre-condiciones: -

Flujos
Flujo Principal
1 | El customer inicia sesión en el sistema
2 | El sistema solicita ingresar un comando
3 | El customer ingresa el comando PUBLISH: <MENSAJE A ENVIAR> {**A1**}
4 | El sistema registra el mensaje, actualiza estadísticas, y re-envía el mensaje a los seguidores o aquellos que sigan hashtags incluidos en el mismo e informa por pantalla que la creación del mismo ha sido exitosa {**E1**} {**E2**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla que ha ocurrido un error el el procesamiento de la solicitud
E1.2 | El caso de uso continúa el el punto 2 del Flujo principal
E2.1 | El sistema informa por pantalla que el comando ingresado es inválido
E2.2 | El caso de uso continúa el el punto 2 del Flujo principal
Post-condiciones: Los Followers del usuario ó de algun hastag que contenga el mensaje enviado por este reciben una copia del mismo.

Caso de Uso: CU004 - Consultar Trending Topic
Descripción: El cliente solicita consultar la lista de los 10 temas mas *posteados*.
Actores participantes: Customer
Pre-condiciones: -

Flujos
Flujo Principal
1 | El customer inicia sesión en el sistema
2 | El sistema solicita ingresar un comando
3 | El customer ingresa el comando QUERY: TT {**A1**}
4 | El sistema procesa el pedido, busca en el storage los diez temas más comentados y los informa en la pantalla del customer {**E1**} {**E2**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla que ha ocurrido un error el el procesamiento de la solicitud
E1.2 | El caso de uso continúa el el punto 2 del Flujo principal
E2.1 | El sistema informa por pantalla que el comando ingresado es inválido
E2.2 | El caso de uso continúa el el punto 2 del Flujo principal
Post-condiciones: -

Caso de Uso: CU005 - Consultar Mensaje
Descripción: El cliente realiza una consulta al sistema para obtener una lista de los post realizados por un usuario o tema en particular
Actores participantes: Customer
Pre-condiciones: -

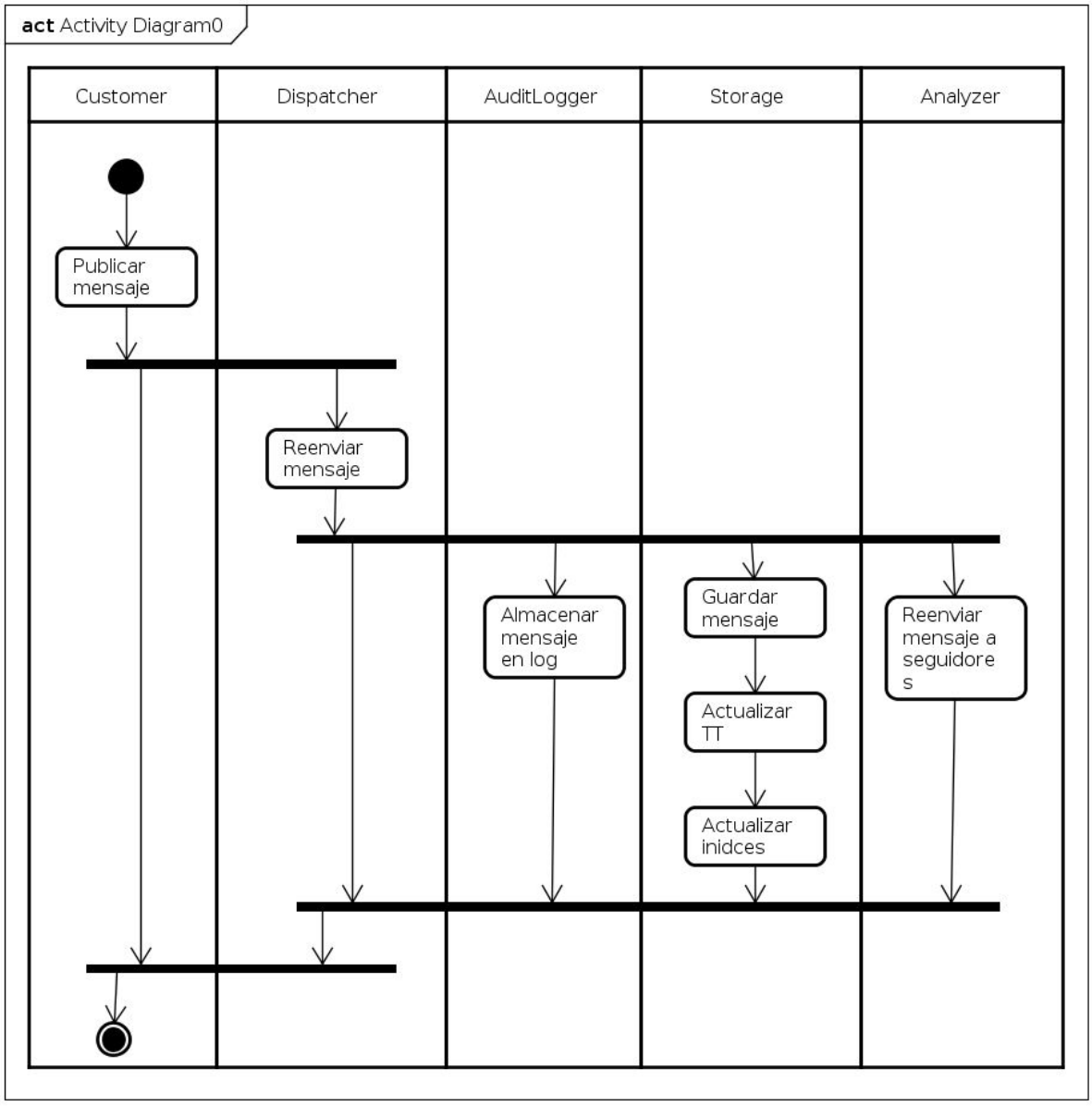
Flujos
Flujo Principal
1 | El customer inicia sesión en el sistema
2 | El sistema solicita ingresar un comando
3 | El customer ingresa el comando QUERY: <NOMBRE DEL USUARIO A CONSULTAR> {**A1**}
4 | El sistema busca en el índice de usuarios los ID's de los diez mensajes más recientes asociados, luego recupera esos mensajes del storage y los informa por pantalla {**E1**} {**E2**}
Flujos Alternativos
-
Flujos de Excepción
E1.1 | El sistema informa por pantalla que ha ocurrido un error el el procesamiento de la solicitud
E1.2 | El caso de uso continúa el el punto 2 del Flujo principal
E2.1 | El sistema informa por pantalla que el comando ingresado es inválido
E2.2 | El caso de uso continúa el el punto 2 del Flujo principal
Post-condiciones: -

2.2. Vista de procesos

A continuación se presentan los diagramas de secuencia modelados para representar el negocio sobre el cual está basado el sistema.

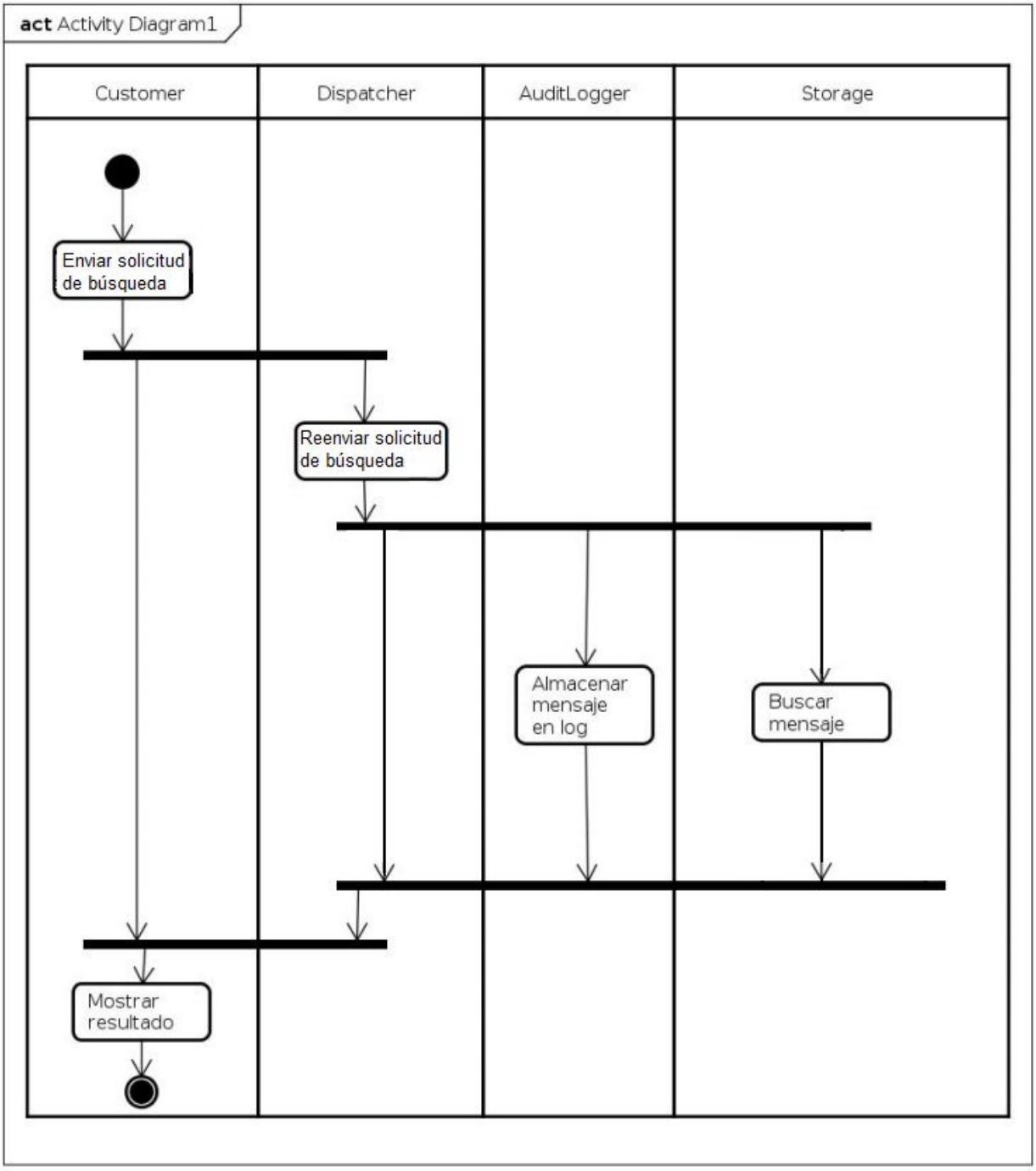
Escenario: Publicación de un mensaje

Descripción: Flujo de actividades desde que un cliente realiza la publicación de un mensaje hasta que el mismo almacenado en la base de datos, se registra en el log de auditoría y entregado a los seguidores en caso de ser necesario.


act Activity Diagram0
Customer
Dispatcher
AuditLogger
Storage
Analyzer
Publicar mensaje
Reenviar mensaje
Almacenar mensaje en log
Guardar mensaje
Actualizar TT
Actualizar inidces
Reenviar mensaje a seguidore s
powered by Astah

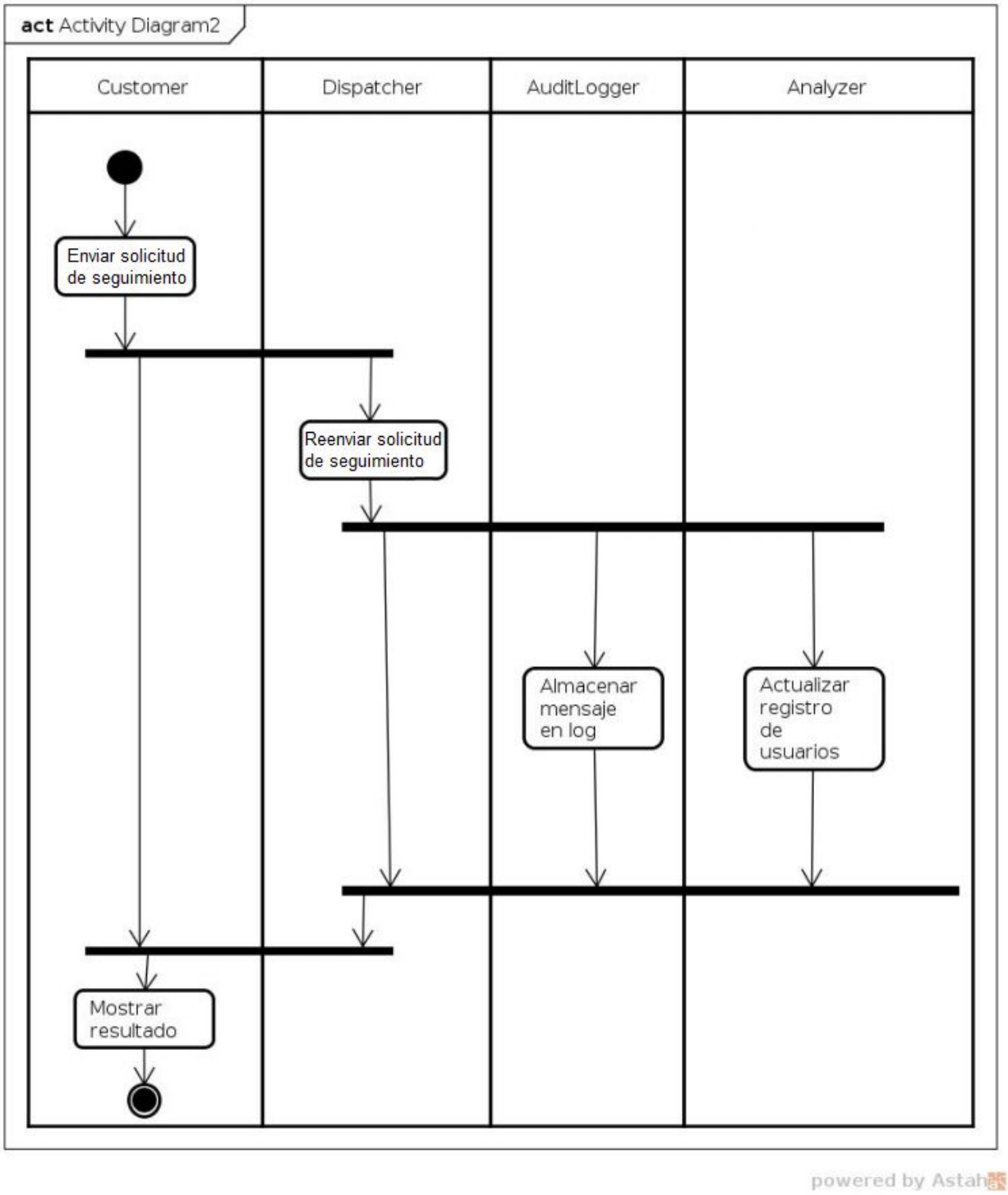
Escenario: Borrado y consulta de un mensaje

Descripción: Flujo de actividades desde que un cliente realiza una consulta o una petición de borrado de un mensaje hasta que la respuesta de la misma es devuelta al cliente.



Escenario: Seguimiento de usuario o tema

Descripción: Flujo de actividades desde que un cliente realiza una petición de seguir un usuario o tema en particular hasta que la respuesta de la misma es devuelta al cliente.



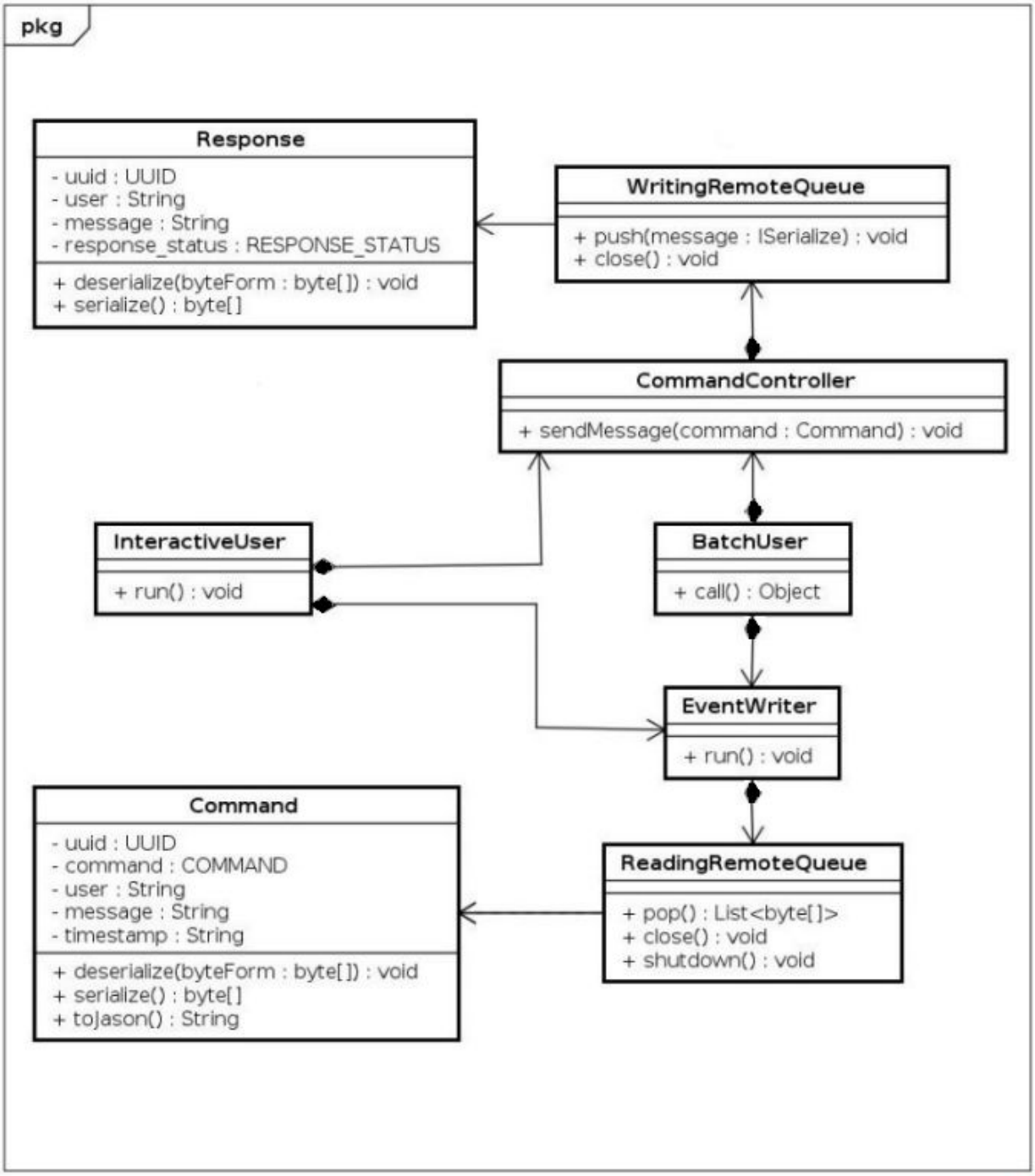
2.3. Vista lógica

A continuación se presentan los diagramas de clases que modelan los distintos componentes del sistema.

Procesamiento de comandos del cliente

En el siguiente diagrama de clases se modela el componente que se encuentra del lado del cliente y se encarga de leer el script de comandos (ScriptReader) y enviarlos al sistema (CommandController). A su vez, recibe las respuestas del sistema y los muestra al cliente a través del EventWriter. Mediante este componente se pueden ejecutar los siguientes comandos:

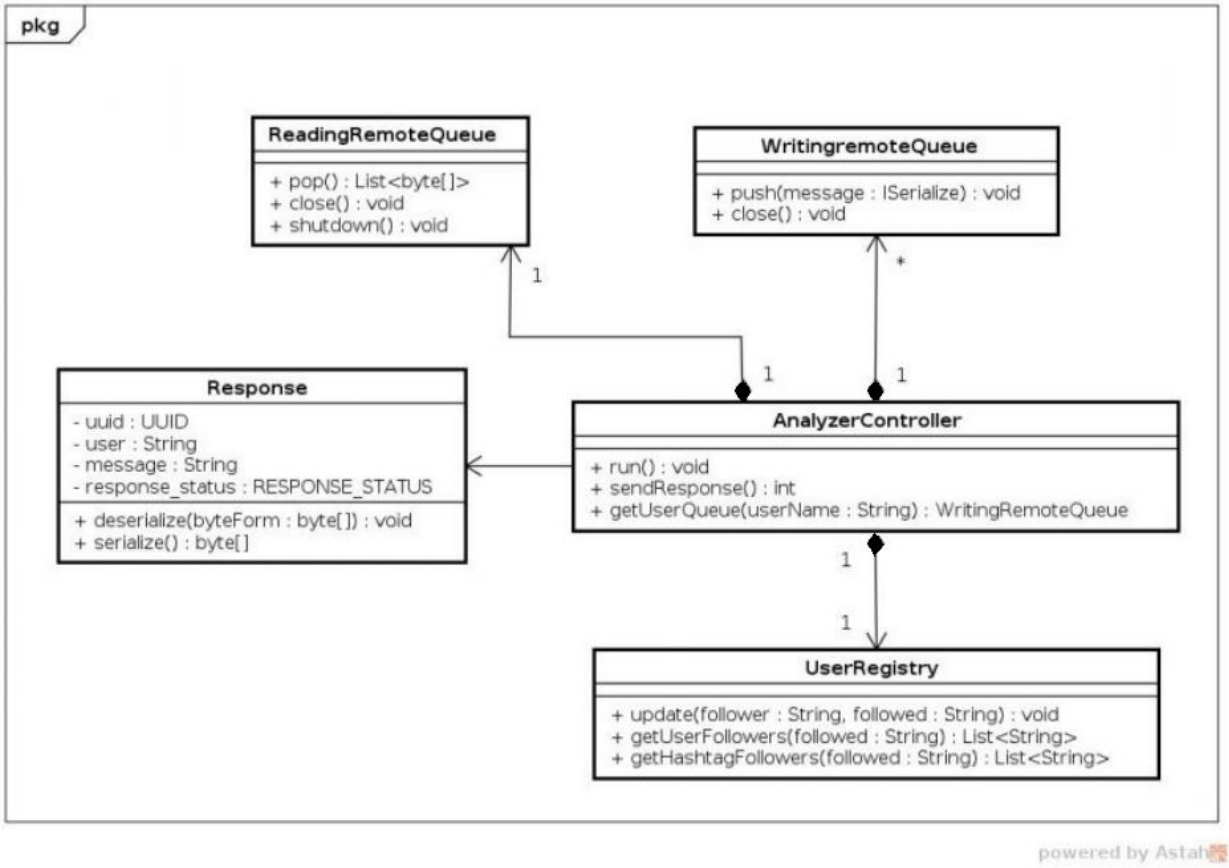
- **PUBLISH**: Permite publicar un mensaje determinado.
- **QUERY**: Permite realizar una de las siguientes consultas al sistema: Últimos *posts* de un usuario determinado, últimos *posts* sobre un tema determinado y *trending topic*.
- **FOLLOW**: Permite seguir los posts de un usuario o referidos a un tema en particular.



Reenvío de mensajes y registro de seguidores

En el siguiente diagrama se muestra la relación entre las clases que conforman al analyzer. El mismo está compuesto por un AnalyzerController que recibe mensajes del broker y los reenvía a los seguidores. Tambien posee un UserRegistry que almacena información de las

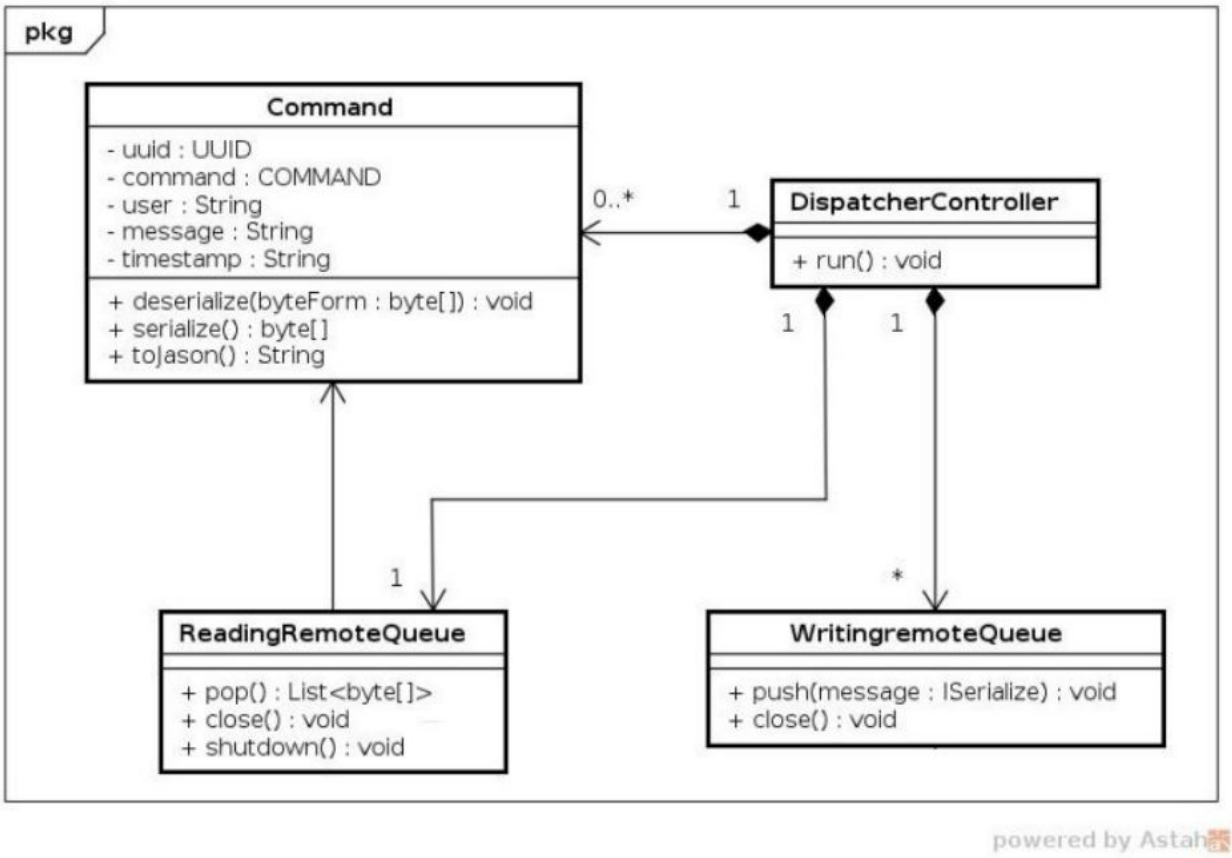
suscripciones. Este componente se encarga de analizar los comandos del tipo **PUBLISH** y **FOLLOW**.



Despacho de mensajes

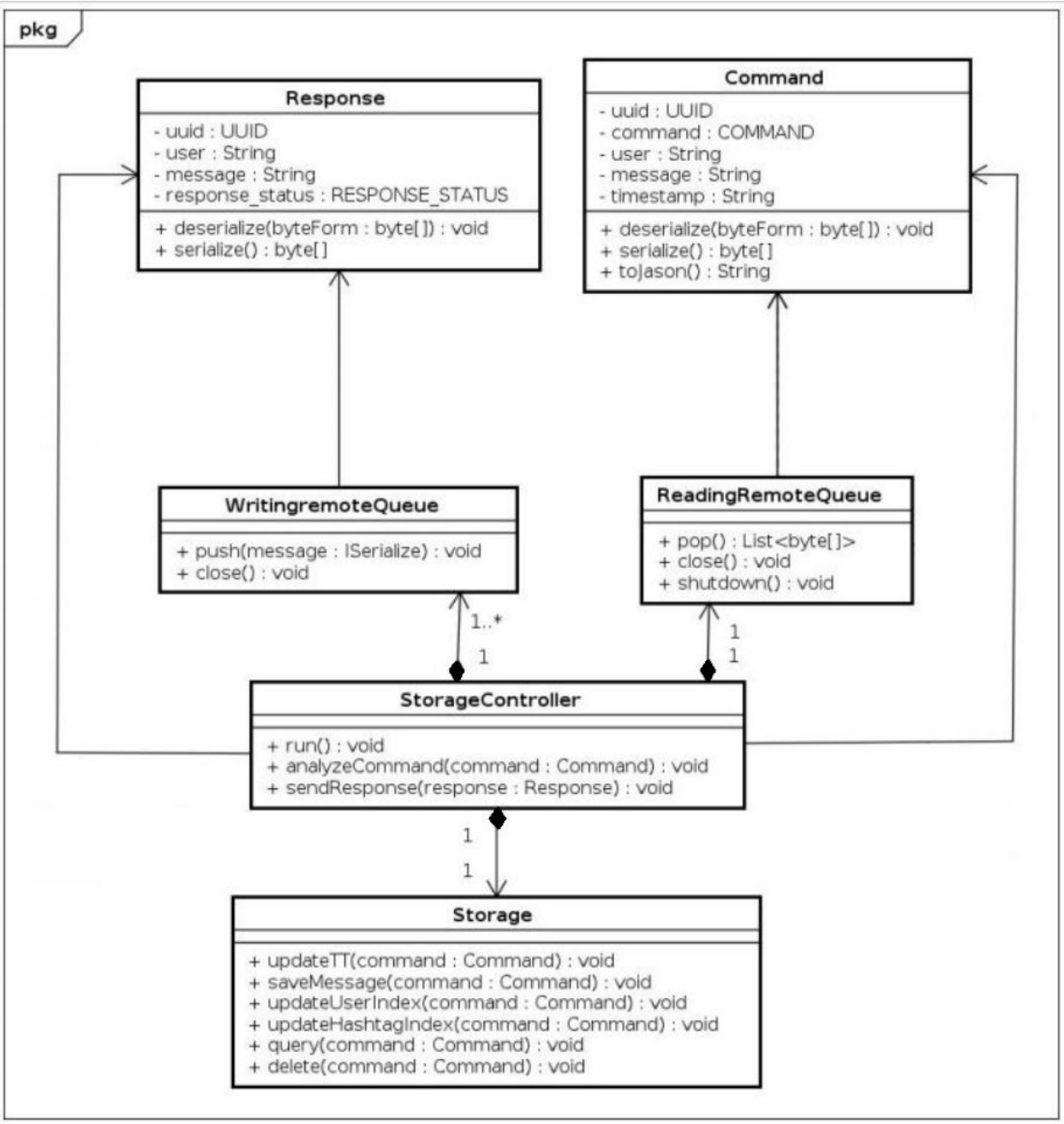
Diagrama de clases que muestra la arquitectura del Dispatcher el cual se encarga de establecer comunicación con el resto de los componentes del sistema para *forwardear* los comandos que los usuarios ingresan. La distribución realizada es la siguiente:

- **PUBLISH**: Son reenviados al storage, analyzer y logger.
- **QUERY**: Son reenviados hacia el storage y el logger.
- **FOLLOW**: Son reenviados hacia el analyzer y el logger.



Almacenamiento de mensajes y actualización de índices en el Storage

Diagrama de clases que modela el storage, en donde los mensajes ingresan al mismo a través del StorageController y de acuerdo al tipo se acción solicitada se procede a crear el mensaje en la base, eliminarlo o realizar una consulta. A su vez la las consultas son manejadas por la clase Storage y mediante el mismo *controller* se envían las respuestas hacia los clientes indicados. Para soportar el *sharding* de la información se ha protegido el acceso a los archivos de la base mediante *locks* de manera de poder levantar varias instancias del storage al mismo tiempo. Las operaciones protegidas son creación y consulta. La protección del borrado de mensajes queda fuera del alcance debido a la complejidad que la misma conlleva.

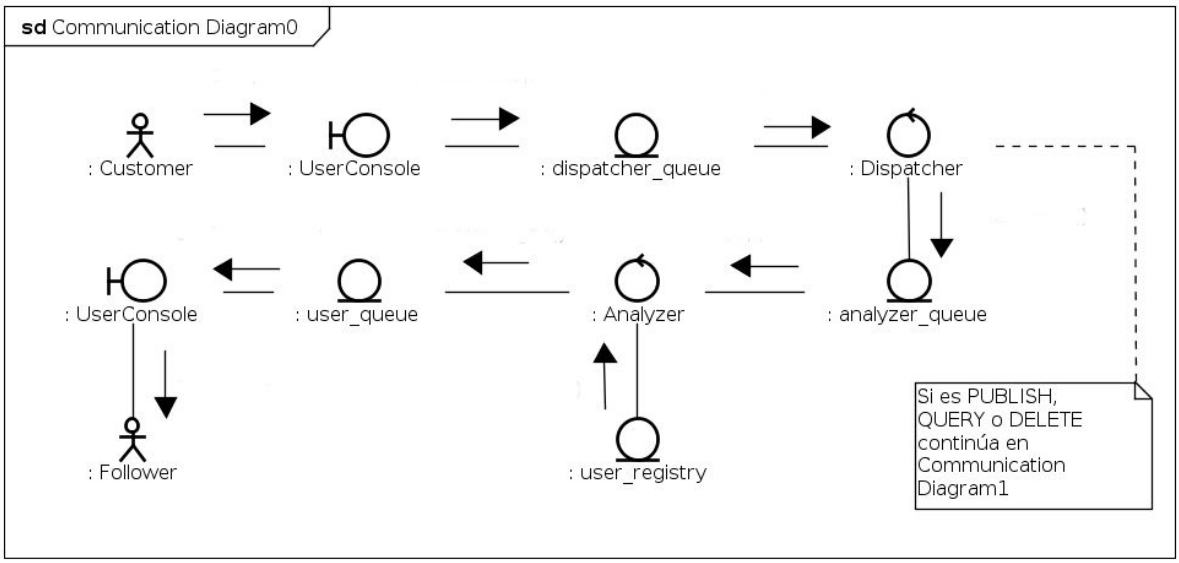


2.4. Vista de física

A continuación se presenta el diagrama de robustez el cual muestra cómo se comunican los diferentes procesos del sistema y el diagrama de despliegue con la distribución de los componentes en los diferentes nodos.

Diagramas de robustez

En el siguiente diagrama se muestra la publicación de un mensaje y el reenvío del mismo realizado por el dispatcher y luego por el analyzer. Las colas mostradas corresponden a colas de Kafka.



En el siguiente diagrama se muestra la distribución de los comandos efectuados por el cliente hacia el log de auditoría y el storage (almacenado, borrado y respuesta de consultas) y el retorno de la respuesta a los mismos. Al igual que en el diagrama anterior las colas mostradas pertenecen a colas de Kafka.

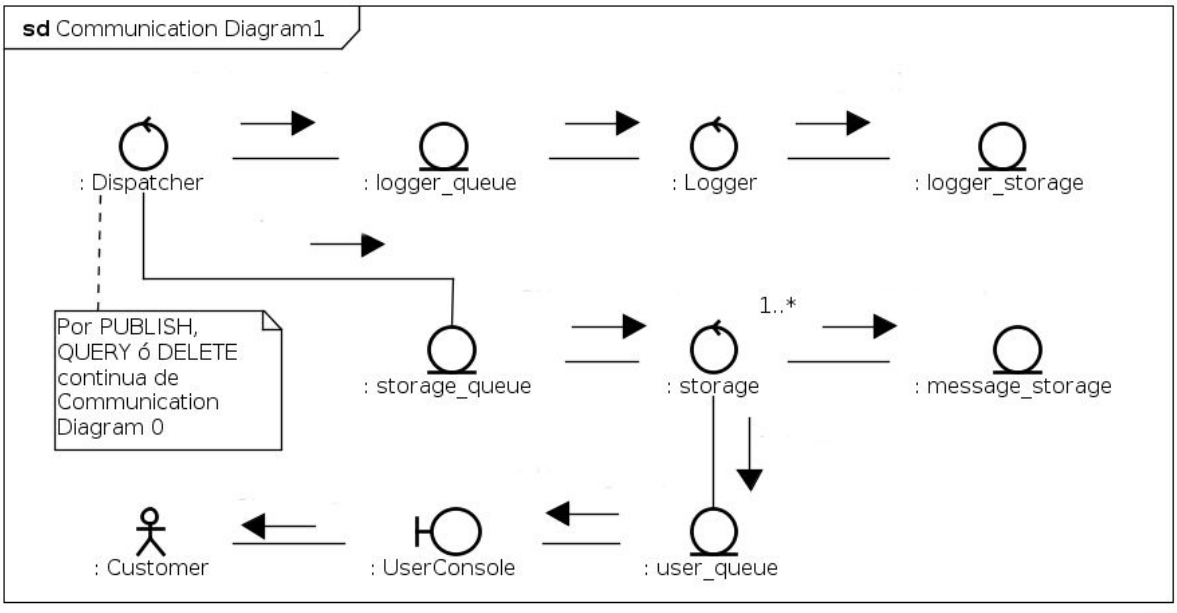


Diagrama de despliegue

En el siguiente diagrama de despliegue se muestra la distribución de los componentes, los cuales pueden ser nodos (servidores) o artefactos (aplicaciones o bases de datos), los cuales se describen a continuación:

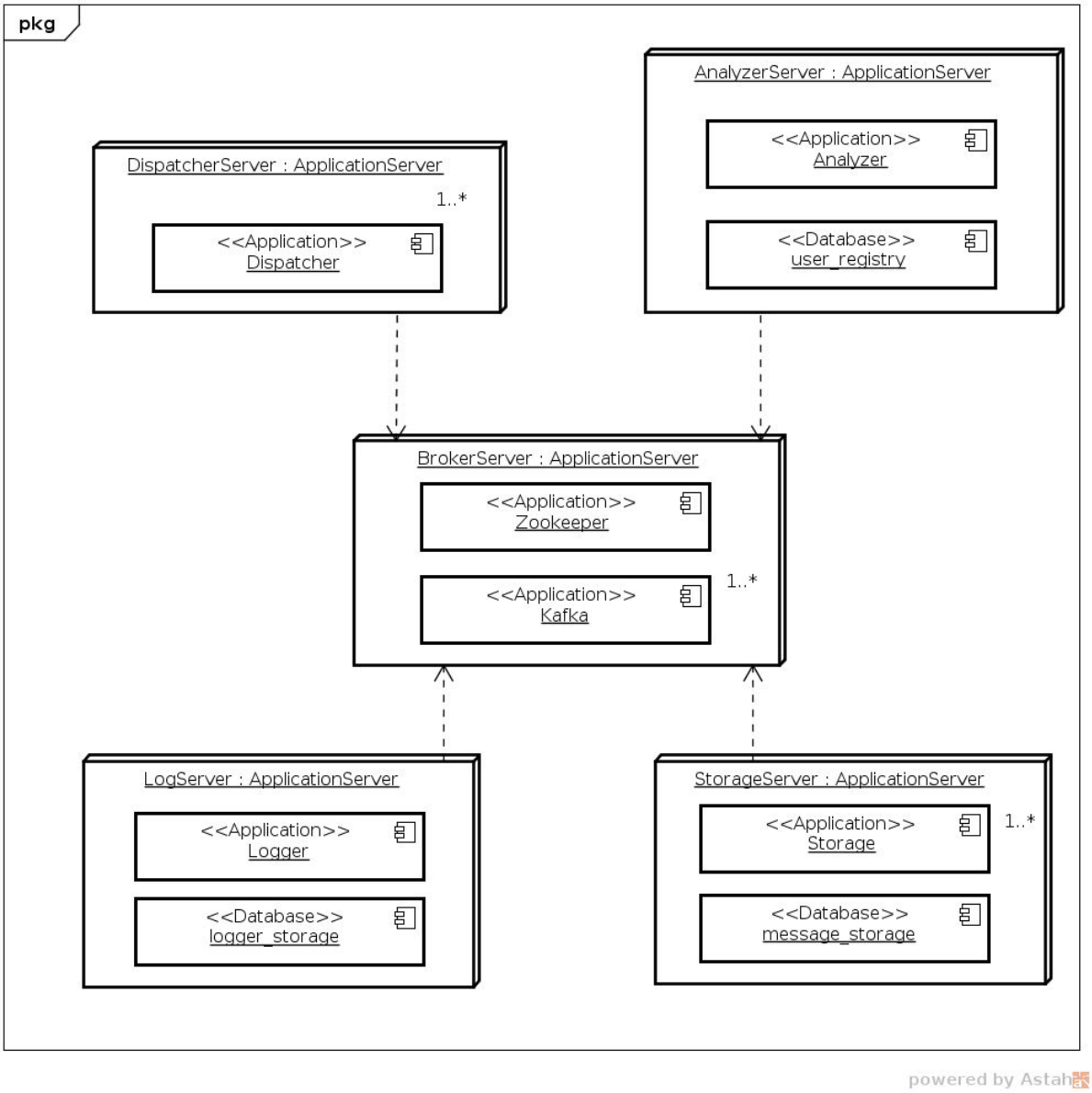
- **DispatcherServer**: Nodo que puede tener varias instancias corriendo del artefacto Dispatcher de manera de poder procesar varios comandos al mismo tiempo. Este nodo se conecta al resto a través de colas RabbitMQ.

- **AnalyzerServer**: Nodo en donde puede haber corriendo varias instancias del Analyzer y además una instancia de la base de datos de suscripción de usuarios.

- **LoggerServer**: Nodo en el que se encuentra corriendo una instancia del artefacto que controla el log de auditoría y la correspondiente base de datos que almacena los logs.

- **StorageServer**: Nodo en el que se encuentra corriendo el controlador de la base de datos de mensajes el cual se encarga de atender las consultas así como también las peticiones de creación y borrado.

- **BrokerServer:** Nodo en el que se encuentran corriendo instancias de Apache Kafka y Zookeeper.

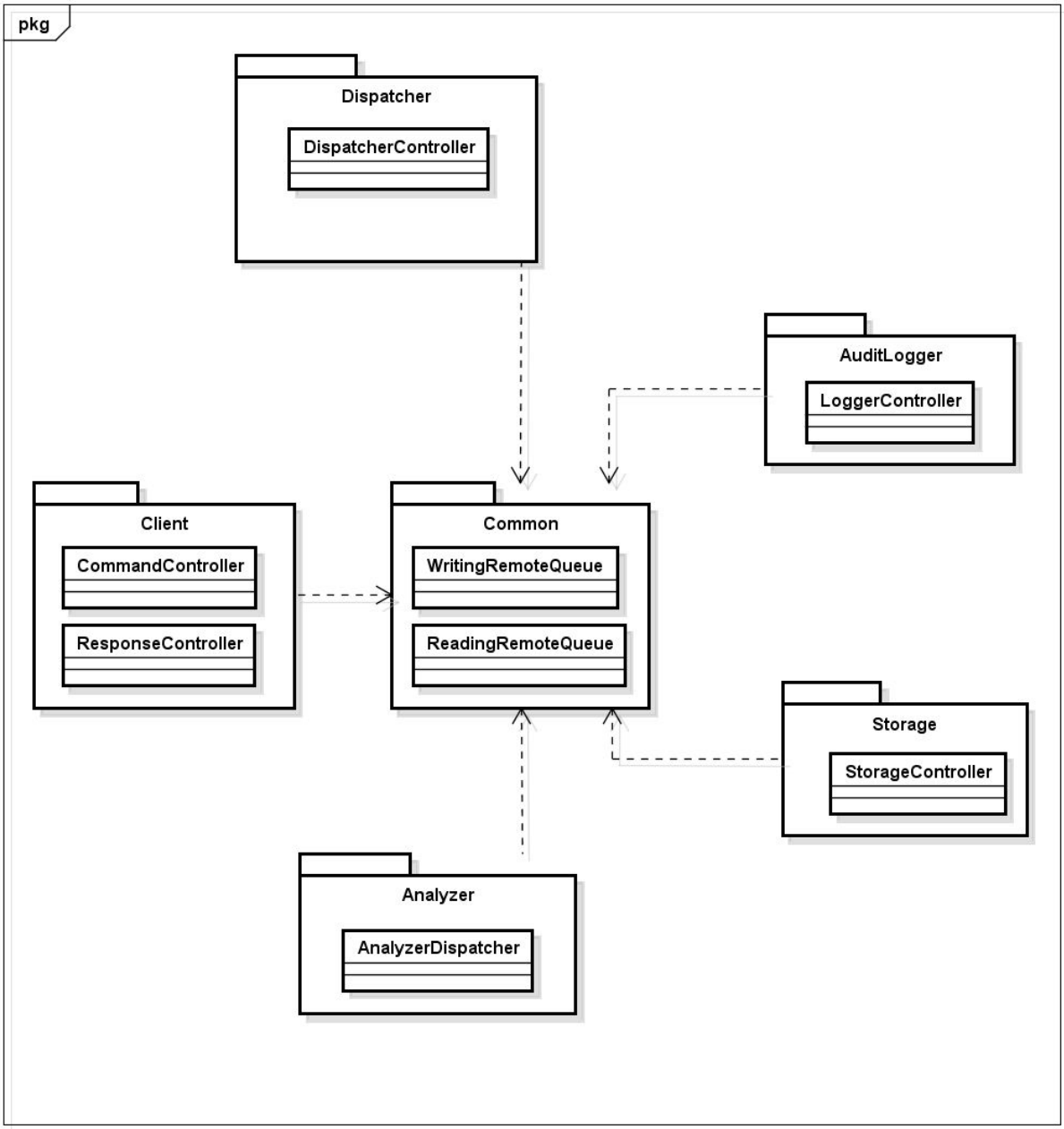


2.5. Vista de desarrollo

A continuación se presentan los diagramas de paquetes correspondientes al sistema desarrollado mediante los cuales se pueden observar las dependencias entre los paquetes del modelo planteado.

En el siguiente diagrama de paquetes se muestra cómo se interrelacionan las clases del paquete **Client** (que contiene las clases para que los clientes se puedan conectar al *Buzzer*), el paquete del **Analyzer**, el del **Dispatcher**, el **Storage** y el **AuditLogger** con el paquete que contiene las clases *wrapper* de las colas de Kafka (**Common**). Como puede observarse la única dependencia entre dichos paquetes se da entre los componentes del *Buzzer* y el Client

con el paquete Common; por lo que a la hora de querer instalar un componentes particular del sistema es necesario solamente distribuir el paquete de la aplicación en particular junto con el paquete common.



Formato de archivos de registro de usuarios (Analyzer)

Para saber a qué usuarios se les debe reenviar los posts, el componente debe consultar dos colecciones de documentos json, los cuales poseen el siguiente formato:

índice de usuarios (user.json)

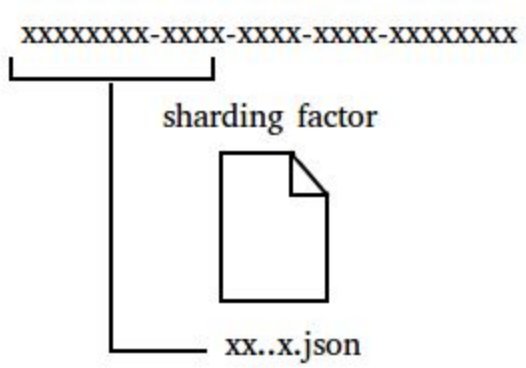
```
{"FOLLOWED 1":["FOLLOWER 1","FOLLOWER 2", ... ,"FOLLOWER N]}
.
.
.
{"FOLLOWED N":["FOLLOWER 1","FOLLOWER 2", ... ,"FOLLOWER N]}
```

Índice de hashtags (hashtag.json)

```
{"HASHTAG 1":["FOLLOWER 1","FOLLOWER 2", ... ,"FOLLOWER N]}
.
.
.
{"HASHTAG N":["FOLLOWER 1","FOLLOWER 2", ... ,"FOLLOWER N]}
```

Formato de archivos del Storage

El almacenamiento de los mensajes en el storage se realiza por medio de un *data sharding* en función de una determinada cantidad de caracteres seleccionados pertenecientes al *UUID* del mensaje recibido (*sharding factor*), el cual determina el nombre del archivo donde se almacenará como se muestra a continuación.



La información del storage está basada en un conjunto de tres colecciones de documentos json para administrar la información de las creaciones, borrados y consultas sobre los datos y cuyos registros poseen el siguiente formato.

Archivo de datos (.txt)

```
{"UUID":{"message":"MESSAGE","user":"USER","command":"COMMAND","timestamp":"yyyy-mm-dd
hh:mm:ss"}}
```

índice de usuarios (user.json)

```
{"USER":["UUID 1","UUID 2", ... ,"UUID N]}
```

Índice de hashtags (hashtag.json)

```
{"HASHTAG":["UUID 1","UUID 2", ... ,"UUID N]}
```

En el caso del *trending topic*, se lleva un registro en un archivo con la cantidad de ocurrencias de cada *topic* encontrado en cada mensaje con el siguiente formato.

Archivo de *trending topic* (tt.json)

```
{"TOPIC 1":#OCCURRENCE OF TOPIC 1, ...  , "TOPIC N":#OCCURRENCE OF TOPIC N}
```

3. Código fuente

A continuación se presenta el código fuente de los componentes desarrollados para implementar la solución descrita anteriormente.