Universidad de Buenos Aires
Facultad de Ingeniería

75.61 - Taller de Programación III

Trabajo Práctico N° 2
Buzzer

1º Cuat. - 2017

Titular ..……..	Andres Veiga
Jefe de Trabajos Prácticos.....…..	Pablo Roca
Alumno …….	Pablo Méndez

Fecha de entrega ……..	10/04/2017

Índice

1. Introducción **2**

2. Solución propuesta **3**

2.1. Vista de escenarios seleccionados 3

2.2. Vista de procesos 5

2.3. Vista lógica 8

2.4. Vista de despliegue 11

3. Código fuente **14**

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 mismos mediante un broker de mensajería; en este caso RabbitMQ, el cual conforma un sistema de mensajería empresarial completo y altamente confiable basado en el estándar AMQP 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 física, Vista de despliegue, Vista de procesos y Vista de escenarios seleccionados.

Para este caso se han seleccionado 4 de ellas; mediante las 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 despliegue: 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.

- Vista de escenarios seleccionados: Esta vista es modelada 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. Vista de escenarios seleccionados

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

- **Seguir Usuario**: El cliente envía una petición al sistema de que desea seguir los mensajes que postea un usuario determinado. El sistema registra el pedido en su base de registro de usuarios de manera que de ahora en más, el solicitante comience a recibir los mensajes que el usuario publica.

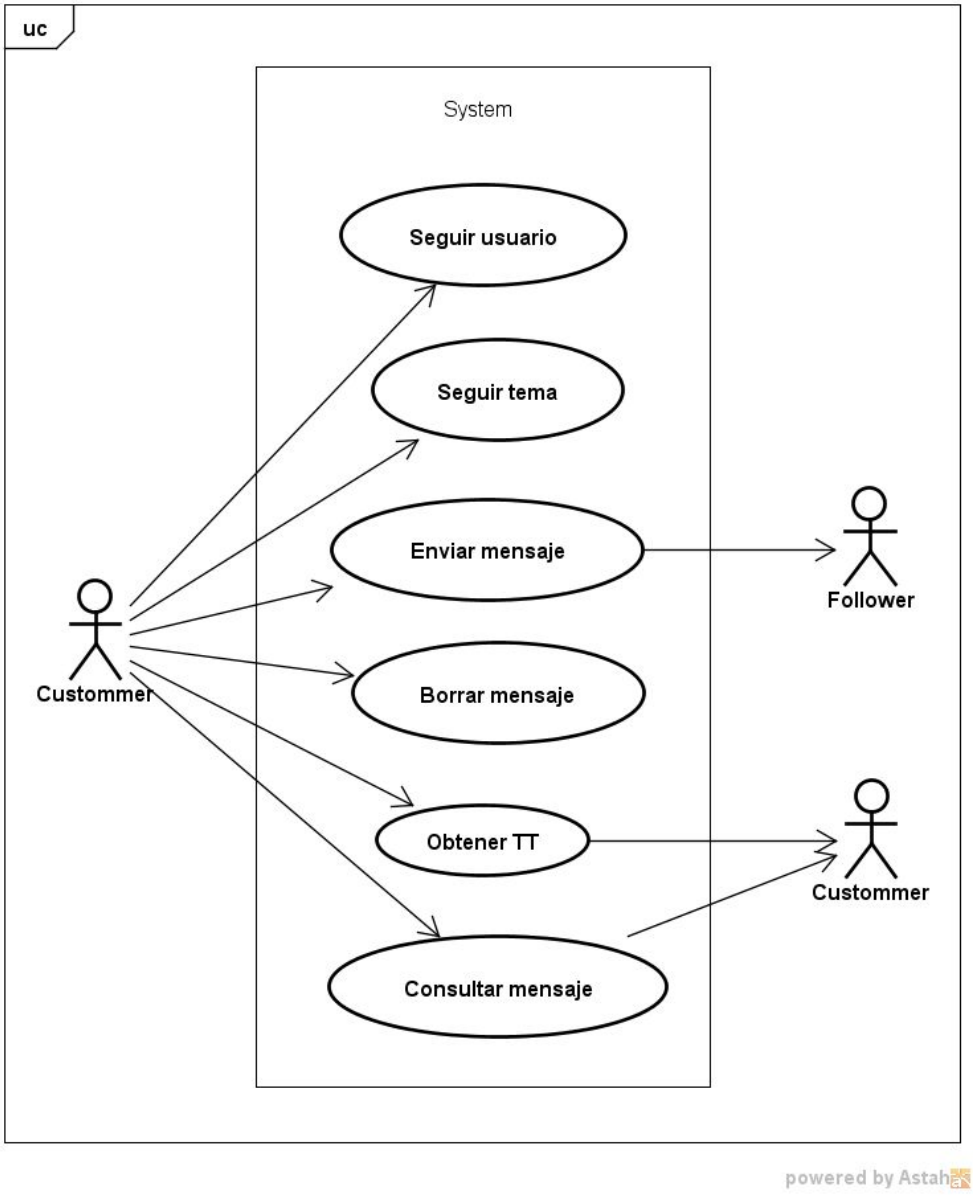
- **Seguir Tema**: El cliente envía una petición al sistema de que desea seguir los mensajes que se postean con un tema determinado. El sistema registra el pedido en su base de registro de usuarios de manera que de ahora en más, el solicitante comience a recibir los mensajes asociados a dicho tema.

- **Enviar mensaje**: El cliente envía un mensaje hacia el sistema. Este último lo almacena en su base de datos, deja registro en el log de auditoría y lo reenvía a los interesados que lo sigan o sigan alguno de los temas de los que el mismo trata en dicho mensaje.

- **Borrar Mensaje**: El cliente envía una petición de eliminación con un id de un mensaje posteado con anterioridad. El sistema busca el mensaje y elimina el registro correspondiente de su base de datos en caso de encontrarlo. En caso contrario informa un error. Este mensaje eliminado ya no puede ser consultado nuevamente.

- **Obtener TT**: El cliente envía una petición al sistema de que él mismo le retorne la lista con los diez temas más comentados durante el día. El sistema analiza la consulta y retorna la información solicitada junto con la cantidad total de temas registrados hasta el momento.

- **Consultar Mensaje**: El cliente realiza una consulta al sistema para obtener una lista de los post realizados por un usuario o tema en particular. El sistema analiza la consulta y retorna una lista de diez entradas más reciententemente generadas en base a la consulta realizada.

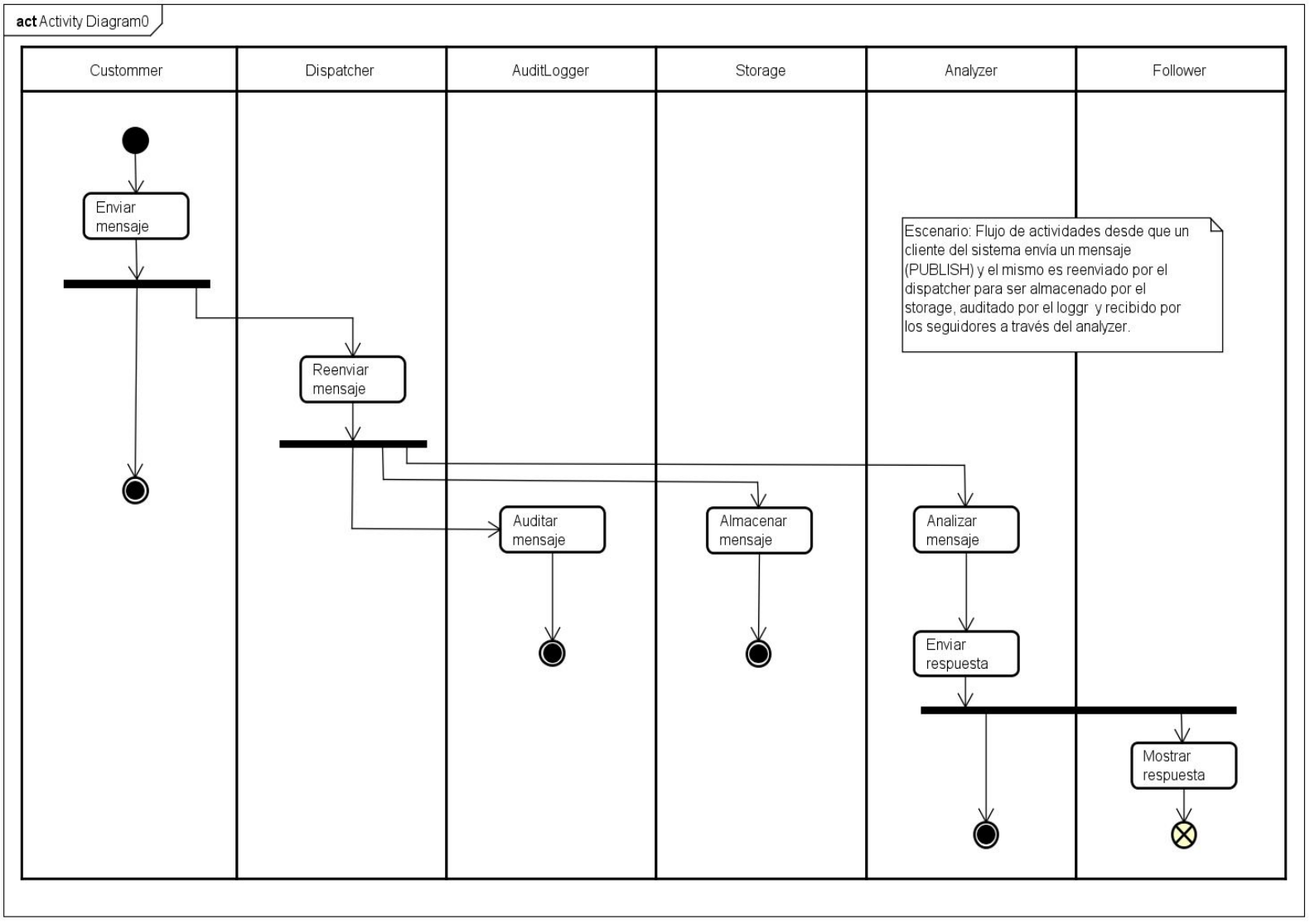


2.2. Vista de procesos

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

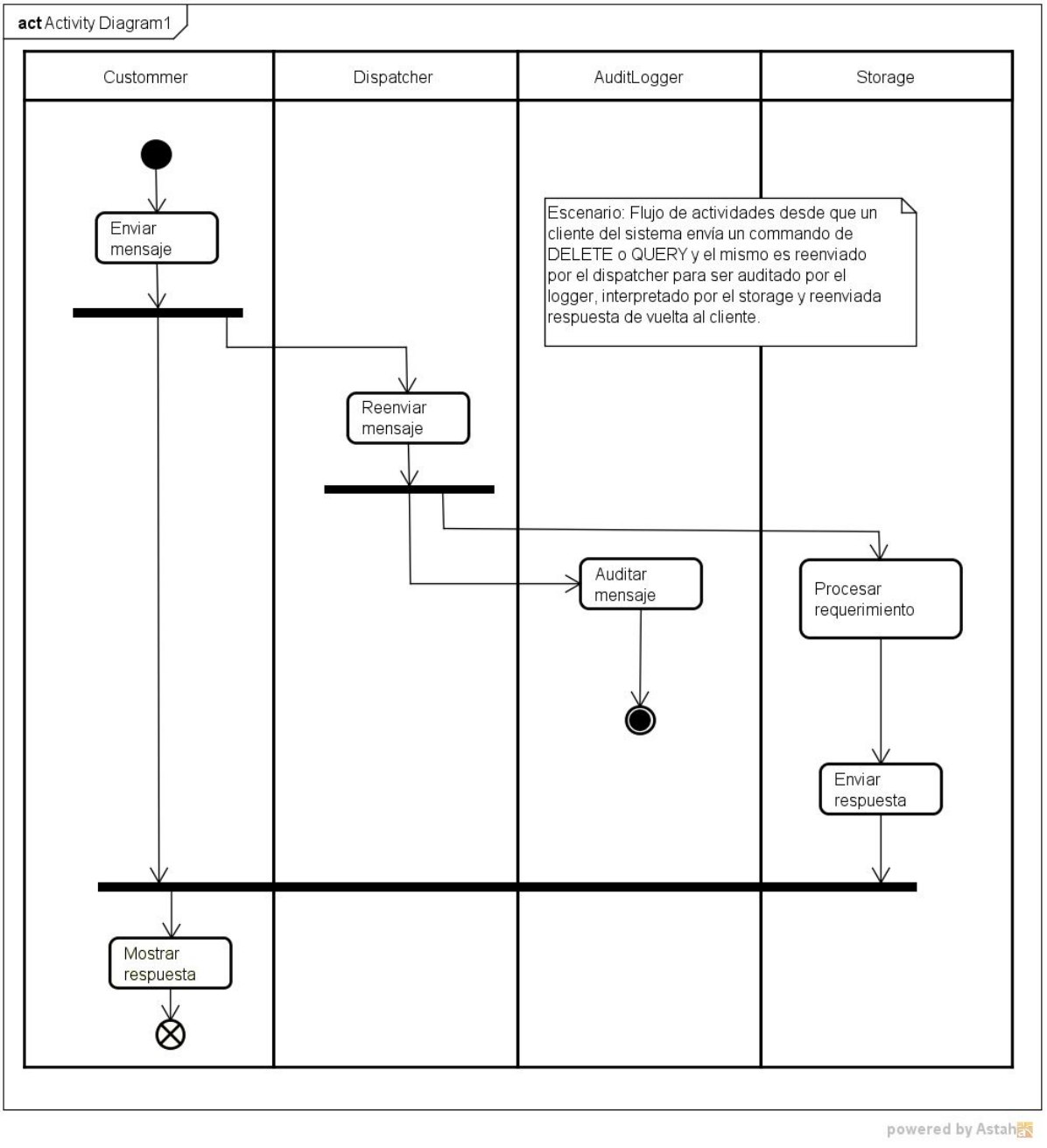
Escenario: Publicación de un mensaje

Descripción: Flujo de actividades desde que un cliente realiza la publicación de un mensaje hasta que el mismo almacenado en la base de datos y entregado a los seguidores.



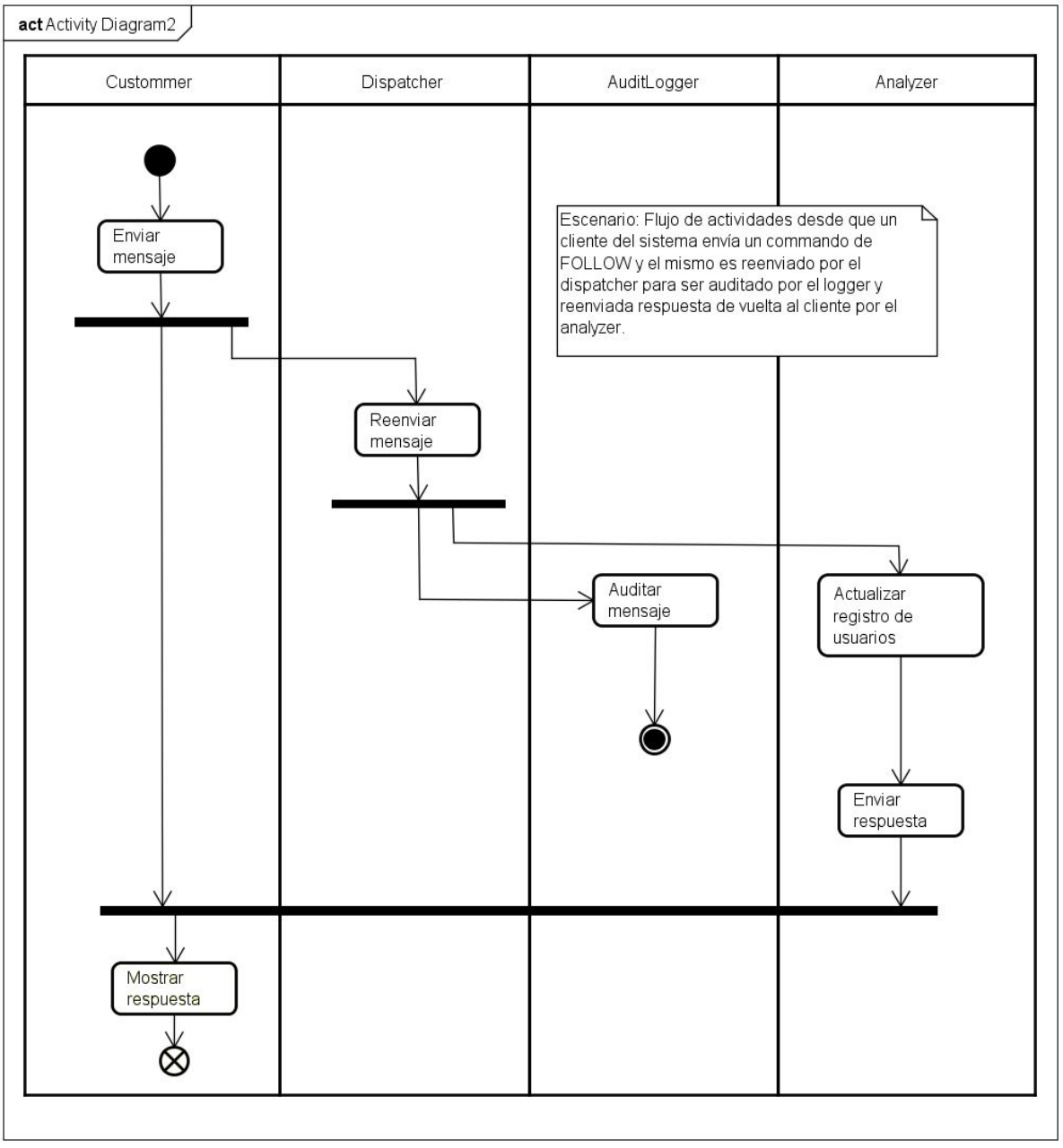
Escenario: Borrado y consulta de un mensaje

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

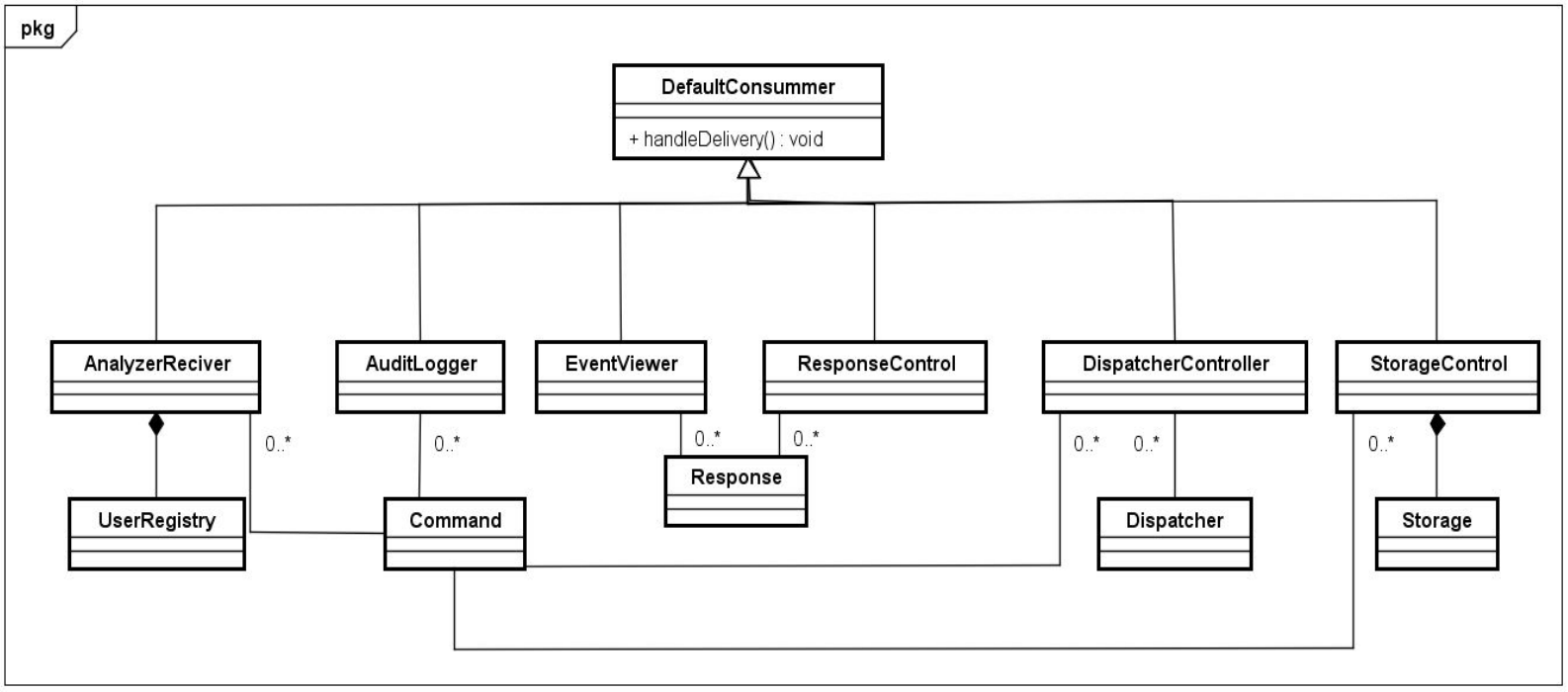
Descripción: Flujo de actividades desde que un cliente realiza una petición de seguir un usuario o tema en particular hasta que la respuesta de la misma es devuelta al cliente.



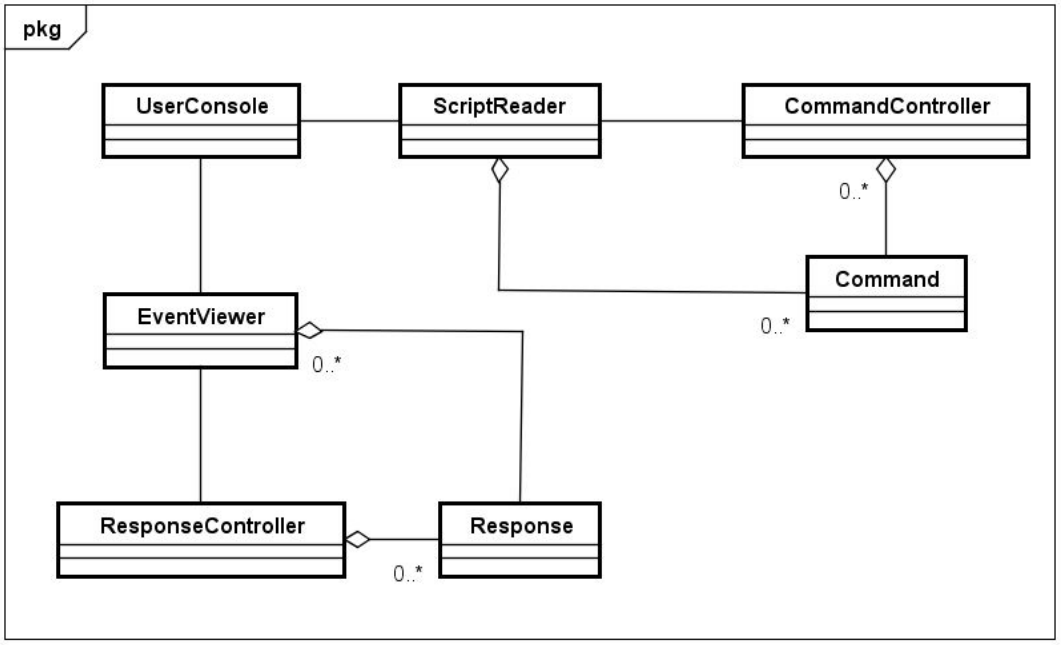
2.3. Vista lógica

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

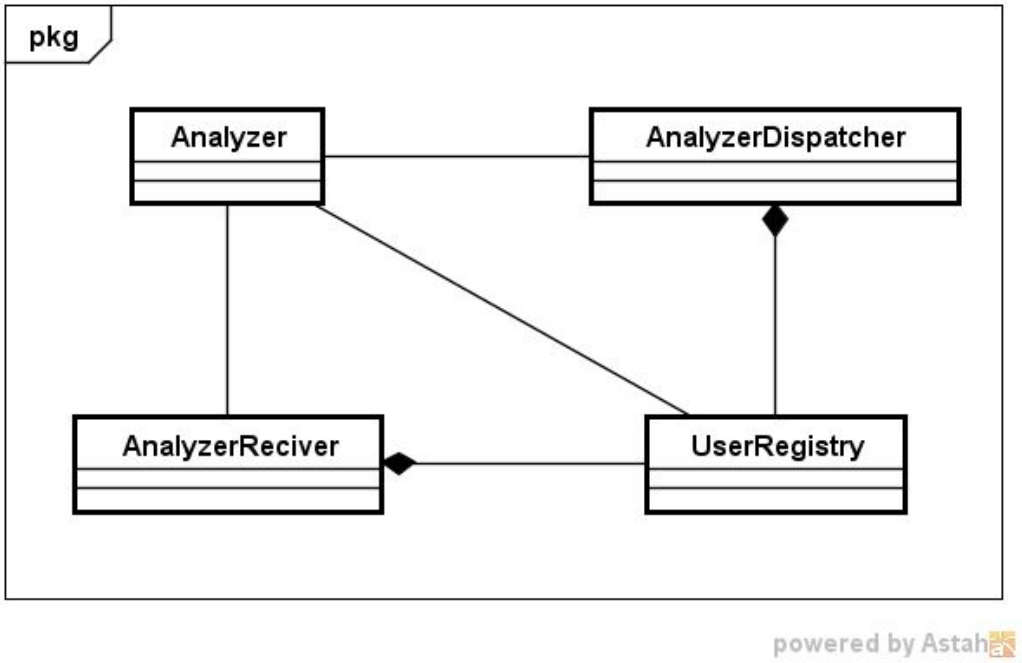
- Diagrama de clases que muestra el modelado de los consumidores del broker así como también qué tipos de mensajes maneja cada uno y en donde persisten la información recibida en caso de ser necesario.



- Diagrama de clases que 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 a través del ResponseController y los muestra al cliente a través del EventViewer. 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*.
  - **DELETE**: Permite borrar un mensaje determinado posteado con anterioridad.
  - **FOLLOW**: Permite seguir los posts de un usuario o referidos a un tema en particular.



- Diagrama que muestra la relación entre las clases que conforman al analyzer. El mismo está compuesto por una clase AnalyzerReciver que recibe mensajes del broker, un AnalyzerDispatcher que envía mensajes a los seguidores y un UserRegistry que almacena información de las suscripciones. Este componente se encarga de analizar los comandos del tipo **PUBLISH** y **FOLLOW**.



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:

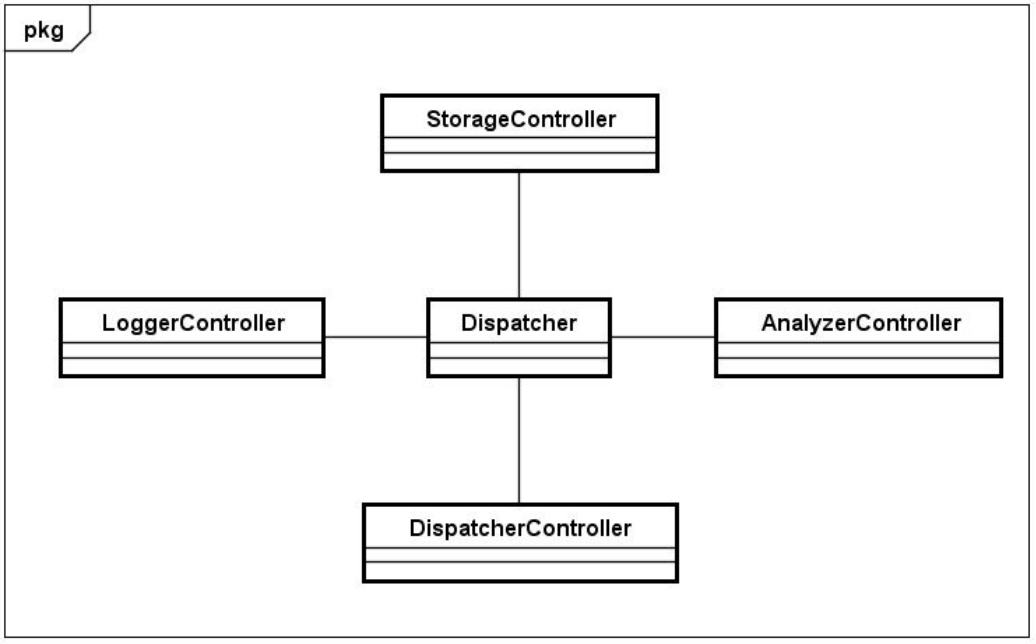
índice de usuarios (user.json)

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

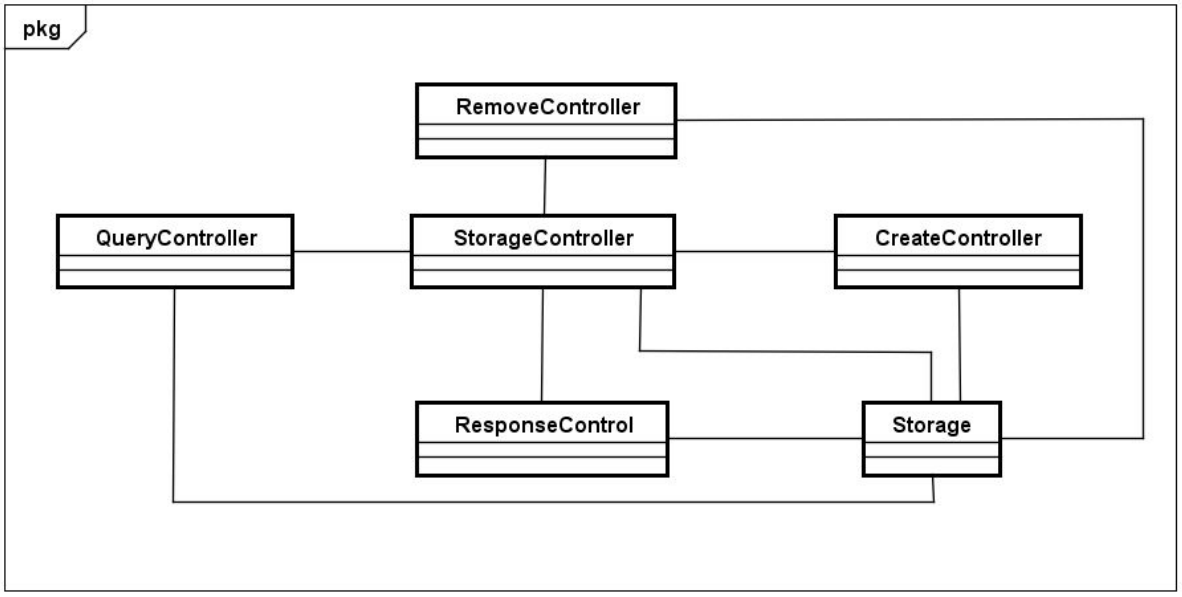
Índice de hashtags (hashtag.json)

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

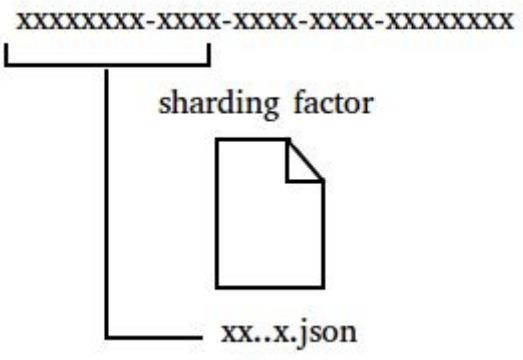
- Diagrama de clases que muestra la arquitectura del Dispatcher el cual controla los *threads* que se encargan de comunicarse con el resto de los componentes del sistema para *forwardear* los comandos que los usuarios ingresan. La distribución realizada es la siguiente:
  - **PUBLISH**: Son reenviados al storage, analyzer y logger.
  - **QUERY**: Son reenviados hacia el storage y el logger.
  - **DELETE**: Son reenviados hacia el storage y el logger.
  - **FOLLOW**: Son reenviados hacia el analyzer y el logger.



- Diagrama de clases que modela la arquitectura del storage en donde los mensajes ingresan al mismo a través del StorageController y de acuerdo al tipo son redireccionadas hacia el CreateController, RemoveController o QueryController. A su vez la entrada y salida de disco es manejada por la clase Storage y mediante el ResponseController se envía la respuesta de las consultas hacia los clientes.



➢ 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 (.json)

{"UUID":{"message":"MESSAGE","user":"USER","command":"COMMAND","timesta
m
p":"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}

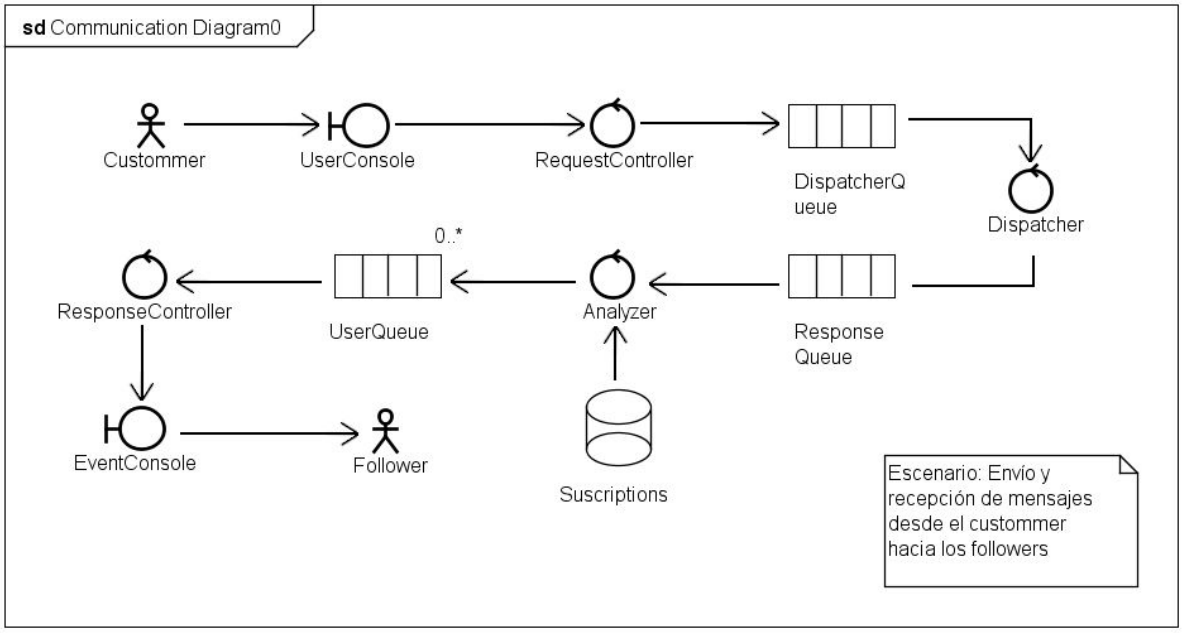
2.4. Vista de despliegue

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

Para los diagramas de robustez se decidió representar las entidades con una simbología distinta a la tradicional de manera de hacer más sencilla su legibilidad; esto es, las colas por rectángulos horizontales y los puntos de almacenamiento de datos por cilindros verticales.

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 RabbitMQ.


sd Communication Diagram0
Custommer
UserConsole
RequestController
DispatcherQueue
Dispatcher
0..*
ResponseController
UserQueue
Analyzer
Response Queue
EventConsole
Follower
Suscriptions
Escenario: Envío y recepción de mensajes desde el custommer hacia los followers
powered by Astah

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 RabbitMQ.

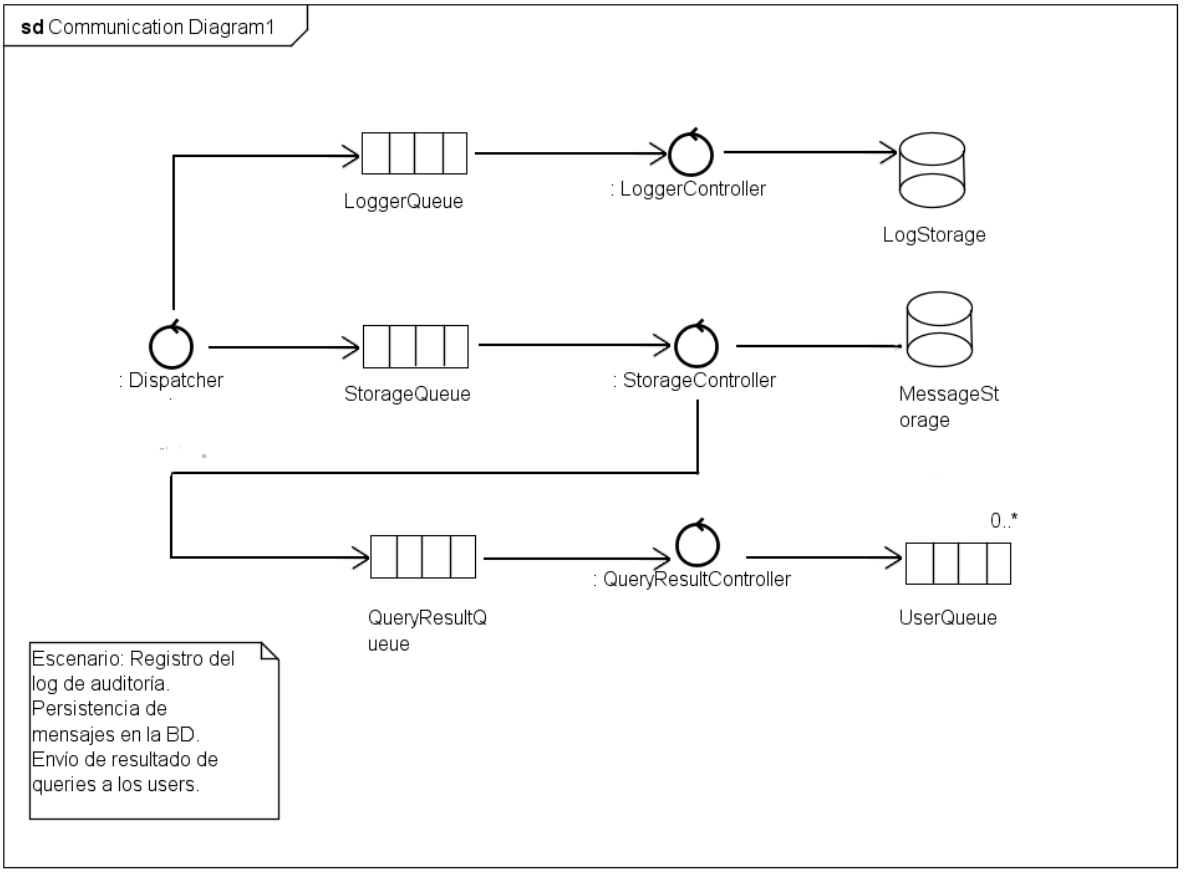


Diagrama de despliegue

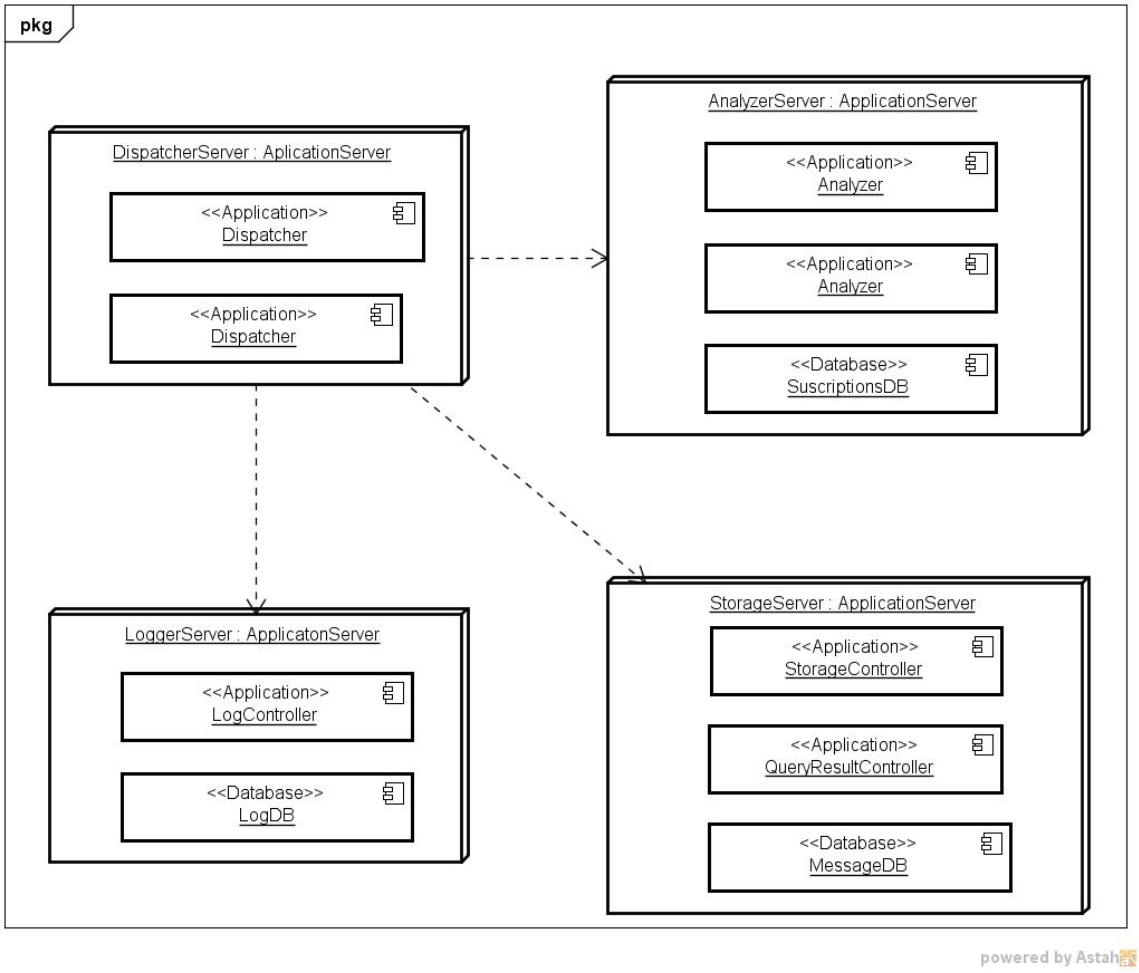
En el siguiente diagrama de despliegue se muestra la distribución de los componentes, los cuales pueden ser nodos (servidores) o artefactos (aplicaciones o bases de datos), los cuales se describen a continuación:

- **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.



3. Código fuente

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