



Facultad de Ingeniería Universidad de Buenos Aires

75.61 Taller de Programación III

TP N°2 - Ejercicio de Colas (RabbitMQ)

Profesor: Andrés Veiga
JTP: Pablo Roca

Integrantes:

| Padrón | Nombre | Email |
|--------|-----------------------------------|--------------------------------|
| 89579 | Torres Feyuk, Nicolás R. Ezequiel | ezequiel.torresfeyuk@gmail.com |

Índice

| | |
|---|----------|
| 1. Introducción | 3 |
| 2. Arquitectura 4 + 1 | 4 |
| 2.1. Casos de Uso | 4 |
| 2.2. Vista Lógica | 5 |
| 2.3. Vista de Despliegue | 6 |
| 2.3.1. Diagrama de Robustez | 6 |
| 2.3.2. Diagrama de Despliegue | 6 |

1. Introducción

El presente trabajo práctico consiste en diseñar e implementar un sistema de recepción de pedidos masivos. Para realizar esta tarea, se deben implementar procesos que modelen diferentes partes/tareas del sistema. Para comunicar los procesos, se deben utilizar cola de mensajes distribuidas.

RabbitMQ es un middleware orientado a mensajes el cual permite conectar diferentes procesos a través de colas de mensajes distribuidas. El objetivo del presente trabajo consiste en:

- Focalizar el desarrollo de sistema en la recepción fluida de **Orders** y **Queries** realizadas por los clientes
- Comprender el funcionamiento básico de RabbitMQ
- Paralelizar la mayor cantidad de tareas posible, exprimiendo en el máximo nivel posible el middleware de colas a utilizar
- Crear y diseñar una arquitectura escalable horizontalmente de forma de responder de forma eficaz y eficiente ante nuevos requerimientos

2. Arquitectura 4 + 1

2.1. Casos de Uso

En la figura 1 se exhibe el diagrama de casos de uso del sistema. Se detalla a continuación cada uno de las entradas del mismo:

- **Enviar Orden:** El cliente envía una orden de compra, la cual contiene un identificador único de la orden más la cantidad que desea obtener un determinado producto. Por el momento el sistema no acepta más de un producto por orden.
- **Consultar Orden:** Un cliente consulta en que estado se encuentra su orden. El sistema puede responderle con alguno de los siguientes estados:
 - *RECEIVED*
 - *ACCEPTED*
 - *REJECTED*
 - *DELIVERED*
- **Procesar Orden:** Como se considera a los Empleados externos al sistema, el procesamiento de una orden es válido como caso de uso. Los empleados obtienen los pedidos a procesar (aquellos que fueron aceptados), y al terminar de trabajar con los mismo proceden a cambiar su estado a *DELIVERED*
- **Incrementar Stock:** Un proveedor se encarga de aumentar el stock de los productos ofrecidos por la aplicación

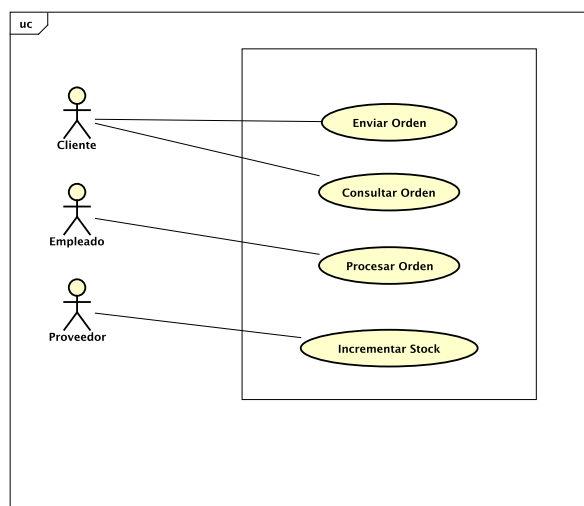


Figura 1: Diagrama de Casos de Uso

2.2. Vista Lógica

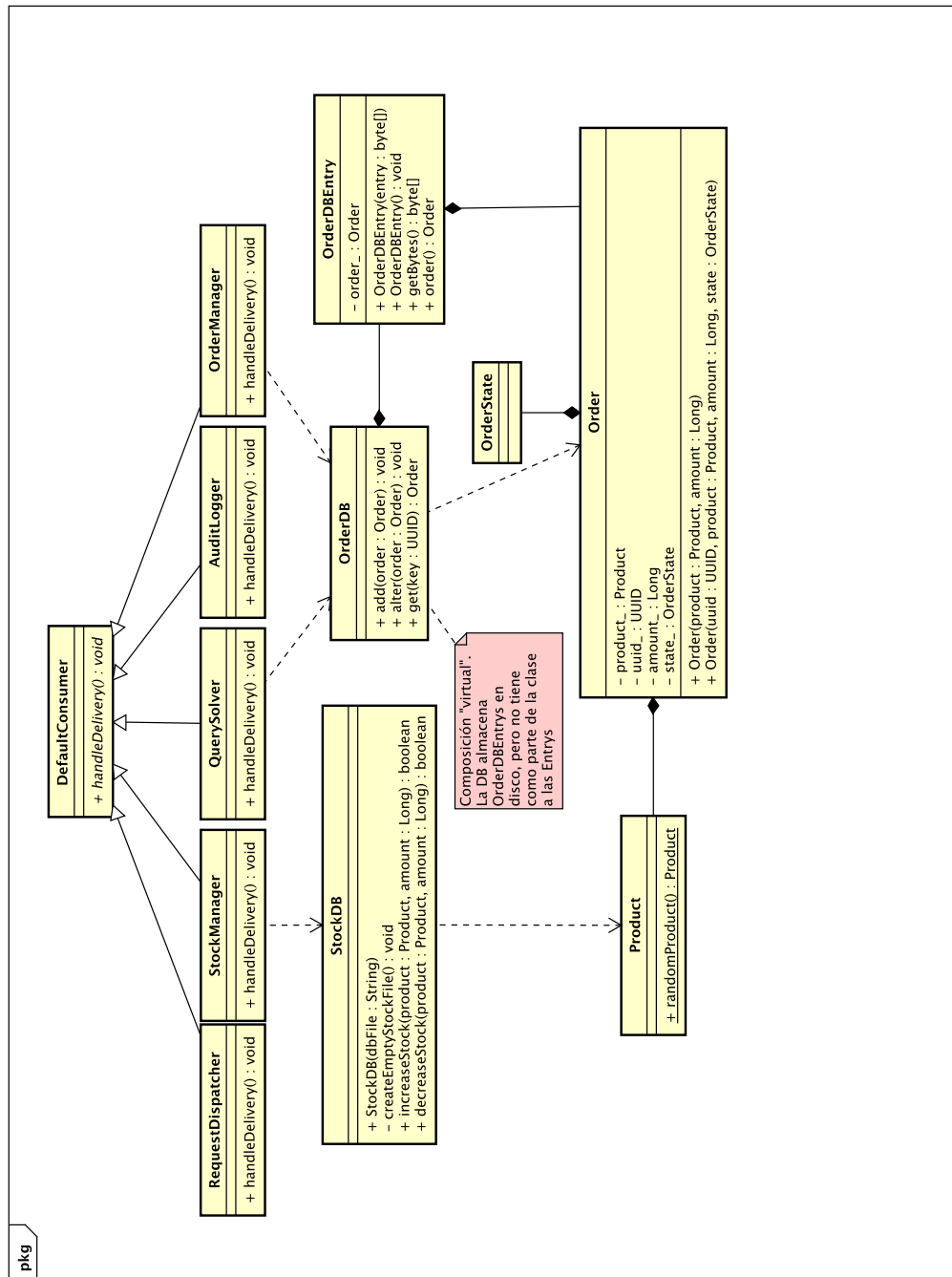


Figura 2: Diagrama de Clases

2.3. Vista de Despliegue

Para la vista de despliegue se decidió realizar un diagrama de robustez y un diagrama de despliegue. El primero explica como los diferentes procesos del sistema se comunican entre sí a través de diferentes colas de mensaje, mientras que el segundo intenta representar la arquitectura física de la aplicación.

2.3.1. Diagrama de Robustez

El mismo se puede visualizar en la figura 4. Debido a que no se encontró una herramienta adecuada para graficar el mismo, se detalla a continuación los elementos utilizados junto con el significado de cada uno:

- **Círculos Rojos:** Procesos que envían estímulos al sistema
- **Círculos Verdes:** Controladores. Puede haber N de ellos
- **Círculos Amarillos:** Controladores. Por cuestiones de modelo de negocio, solo puede haber uno de estos controladores de cada tipo al mismo tiempo
- **Colas:** RabbitMQ Queues. Las flechas entrantes indican que un controlador ingresa mensajes en la cola. Flechas salientes indican que un controlador saca elementos de la cola.
- **Discos:** Archivos físicos los cuales, en el presente trabajo, simulan el funcionamiento de bases de datos relacionales.

2.3.2. Diagrama de Despliegue

El diagrama de despliegue se exhibe en la figura ???. Cada nodo representa a un servidor físico independiente, el cual corre aplicaciones y puede o no tener Base de Datos. Se procede a detallar cada uno de los nodos:

- **DispatchServer:** En este nodo corren los procesos RequestDispatcher que se encargan de recibir las **Orders** de los clientes para derivarlas a otros nodos. En el diagrama se puede ver que existen dos nodos de este tipo. Con esto se quiere representar que se puede escalar este nodo agregando N de los mismos de forma transparente al sistema.
- **AuditLogServer:** Nodo donde se encuentra el log de auditoría. Como los logs deben ser almacenados en orden, solo puede haber un proceso corriendo de este tipo. El proceso debe correr en un servidor aparte debido a que este es un punto sensible del sistema y debe ser optimizado en la mayor medida posible.
- **StockServer:** En el presente nodo se encuentra la base de datos en donde se almacena el Stock de los productos. En el nodo se pueden ver más de una aplicación, lo cual indica que se pueden correr N instancias de este proceso en el dispositivo físico, pero que no se puede agregar más nodos de este tipo. Esto último se debe a que los procesos a través de I/O syscalls a la DB las cuales deben correrse en el mismo servidor físico donde se encuentra la DB en cuestión.

- **OrderServer:** En el presente nodo se encuentran los procesos OrderManager y QuerySolver. Se cumplen los mismos requisitos que para el proceso StockManager, dado que estos procesos también realizando un acceso físico a un DB (OrderDB).

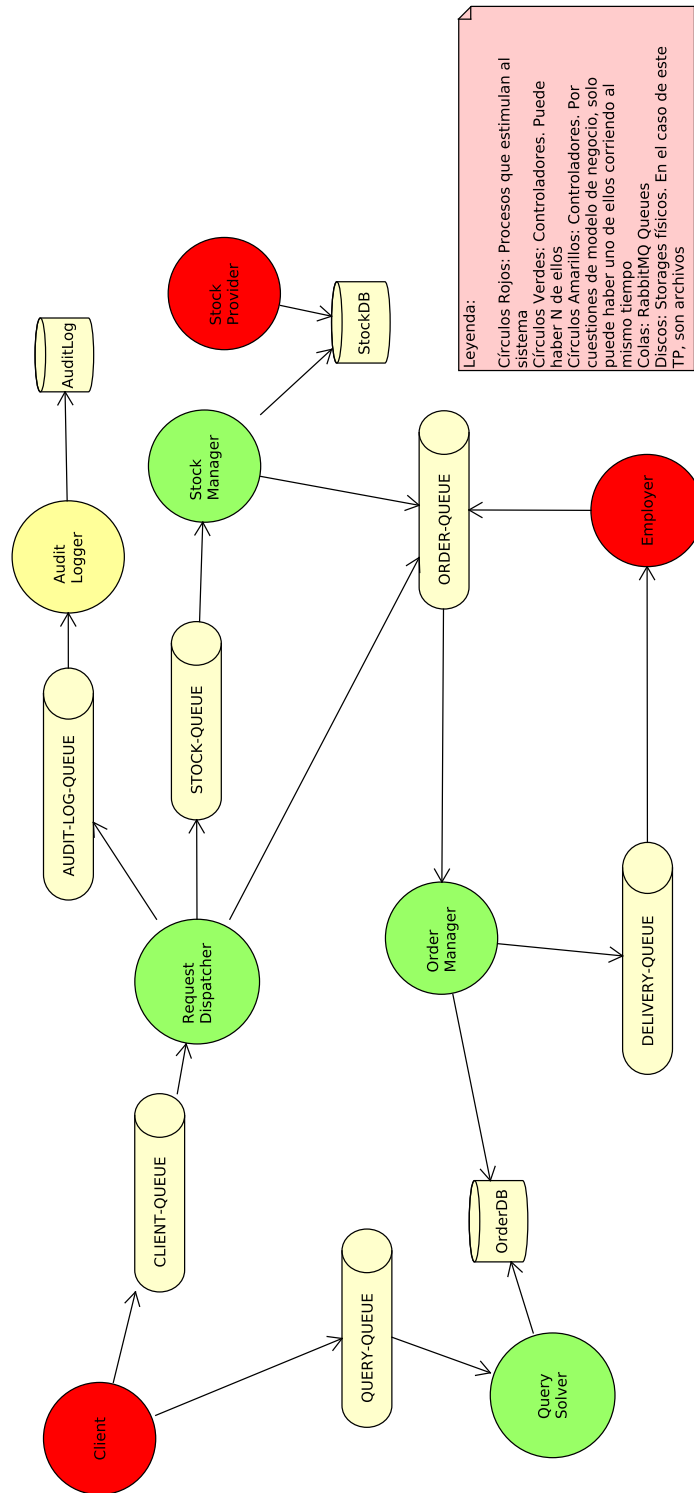


Figura 3: Diagrama de Robustez

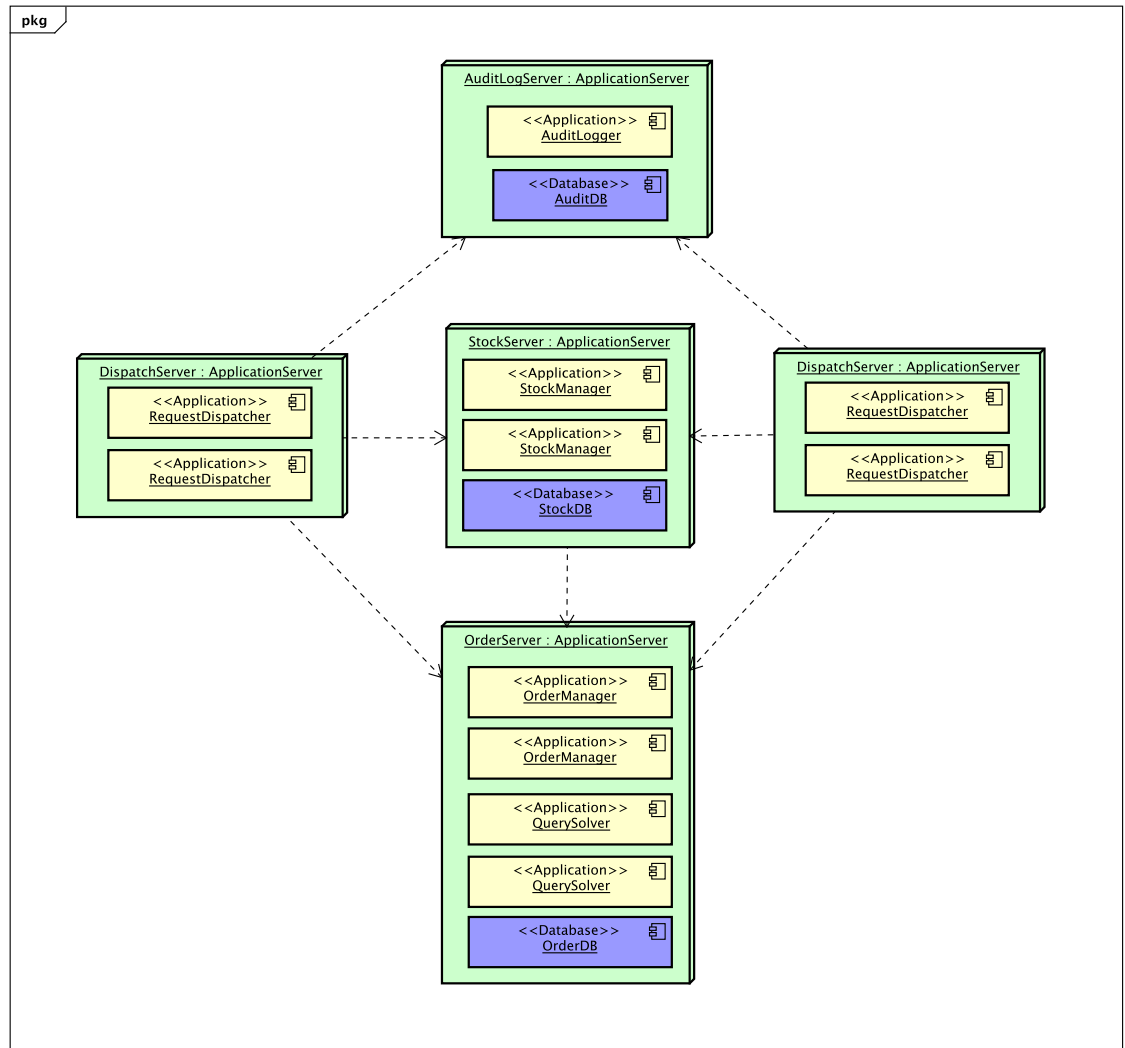


Figura 4: Diagrama de Despliegue