UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2015 - 2do Cuatrimestre

TALLER DE PROGRAMACIÓN III (75.61)

TRABAJO PRÁCTICO N.º 2
TEMA: Colas en RabbitMQ
FECHA: 15/10/2015

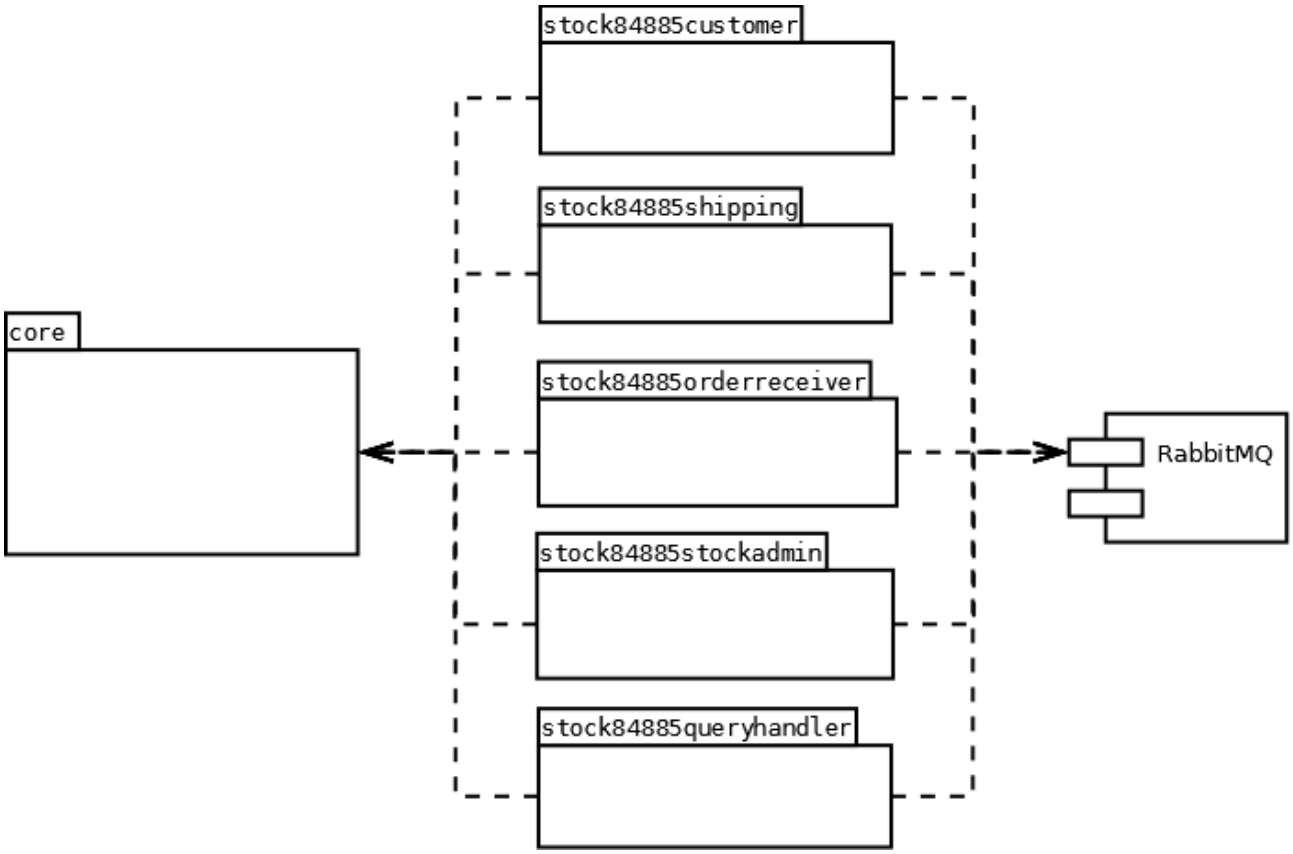
Autor	Padrón
Darío Eduardo Ramos	84885

Objetivos del sistema

- Tomar pedidos de una cantidad masiva de usuarios simultáneos (escalabilidad).
- Mantener un control de stock consistente (siempre mayor a cero) en todo momento.
- Diseñar los componentes de manera que puedan ser fácilmente distribuidos.
- Mantener un control detallado de todos los pedidos, con fines de auditoría.
- Utilizar un patrón de comunicaciones persistentes para garantizar el envío de los mensajes.

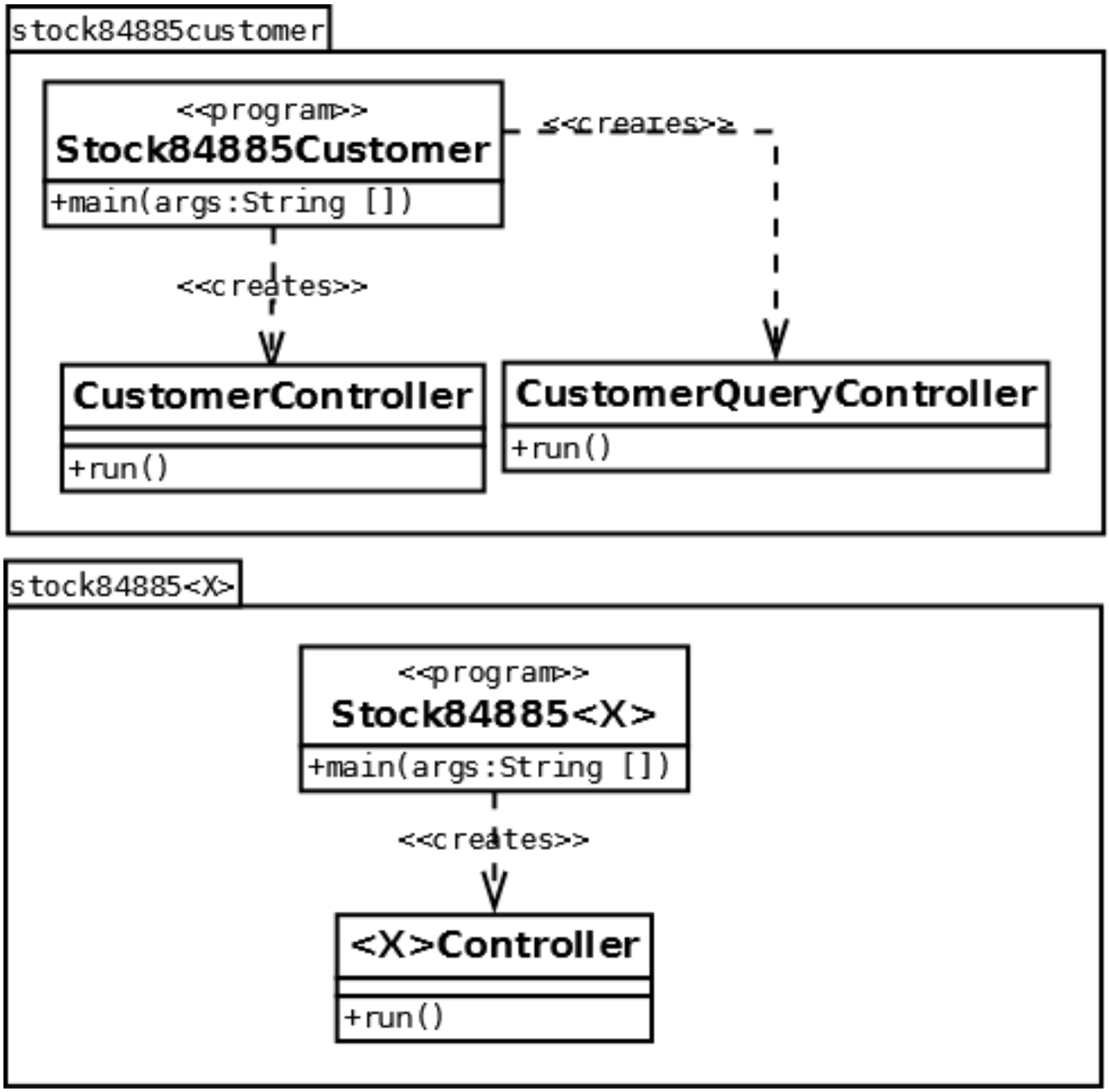
Vista lógica: Diagrama de paquetes

Diagrama de paquetes general



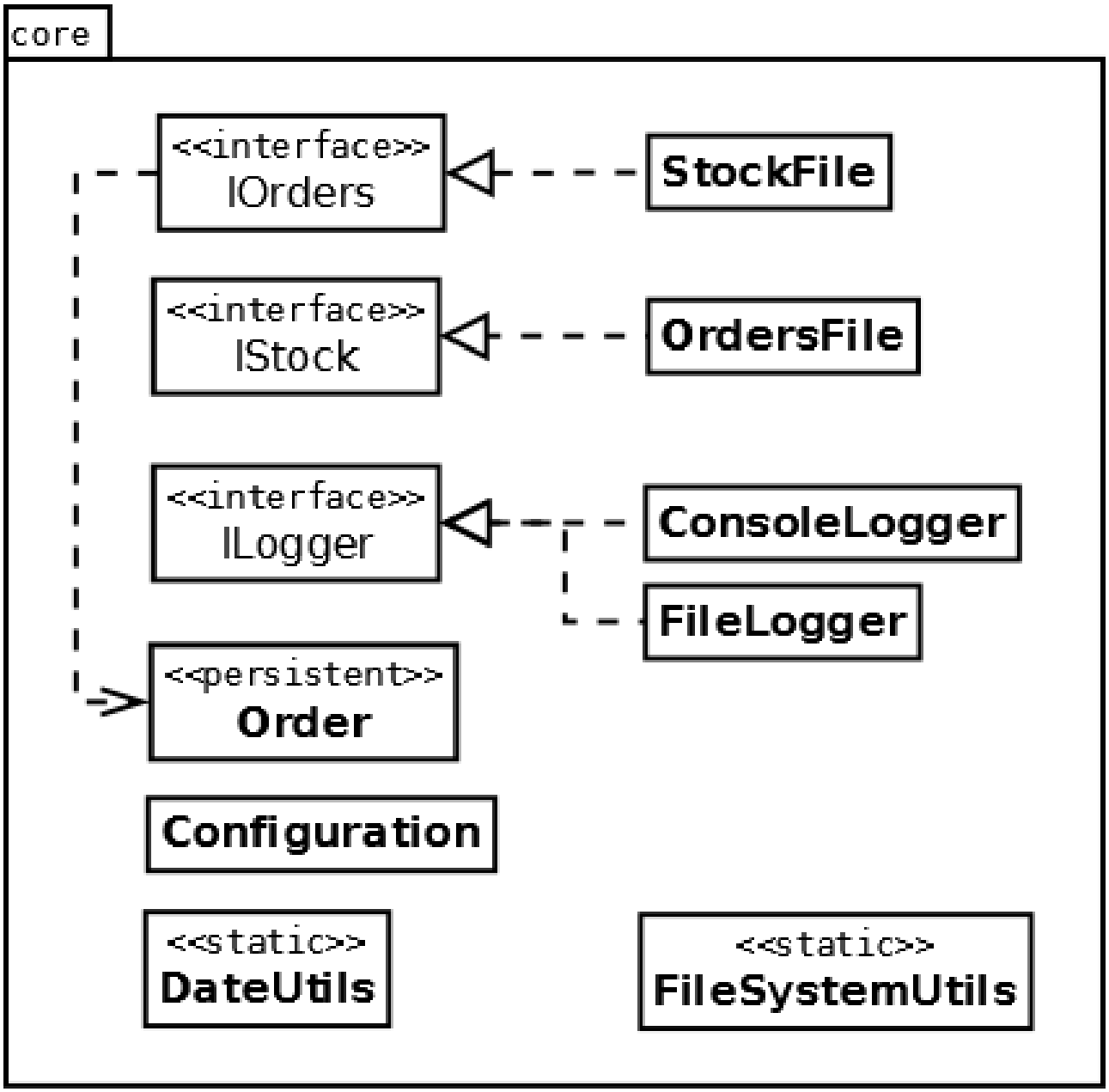
Todos los paquetes del centro están asociados a un ejecutable (.jar), usan RabbitMQ, y clases del Core que no se muestran en este diagrama por simplicidad.

Diagrama de paquetes y clases (ejecutables)



Todos los ejecutables tienen la misma estructura de clases, excepto stock84885Customer, que tiene dos controllers, CustomerController y CustomerQueryController. Todos los otros, simbolizados por una X en el segundo paquete "genérico" (X = stock84885shipping, stock84885orderreceiver, stock84885stockadmin y stock84885queryhandler), constan de un ejecutable y un Controller.

Diagrama de clases del paquete stock84885core



Las interfaces Iorders e Istock abstraen de la implementación de la persistencia de Órdenes (clase Order) y el Stock; las implementaciones provistas, StockFile y OrdersFile, son archivos accedidos con exclusión mutua entre procesos usando un FileLock.

La interfaz Ilogger abstrae al usuario del medio de logging (archivo, consola, socket, etc).

Configuration hace transparente el acceso a los parámetros del sistema.

DateUtils y FileSystemUtils son clases auxiliares, sin estado (por ende el estereotipo static).

Descripción de clases más importantes

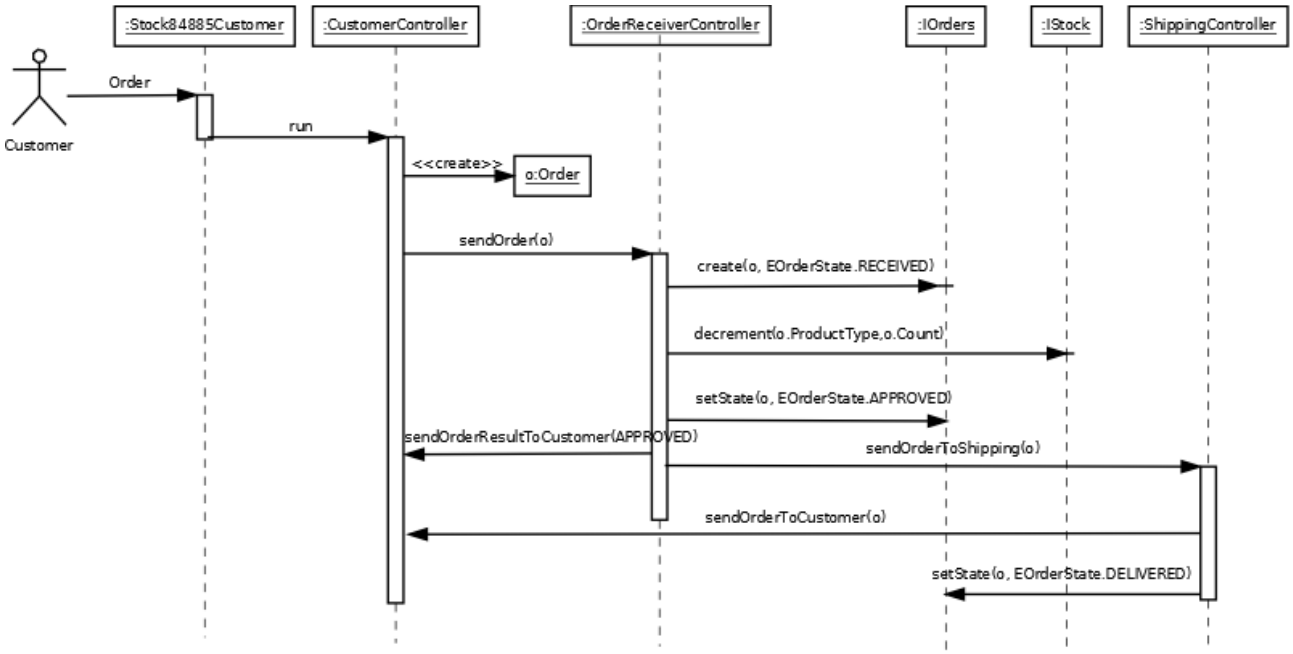
Clase	Descripción
StockFile	**Responsabilidad**: Almacenar el stock actual garantizando consulta y actualización en exclusión mutua.
	Atributos principales: ***_maxStock***: Máximo valor de stock para cualquier tipo de producto. ***_filePath***: Ruta del archivo de stock. ***_lockFilePath***: Ruta del archivo de lock asociado al archivo de stock (podría ser el mismo si se leyera el mismo de forma binaria).
	Métodos principales: ***public boolean decrement(Order.EProductType type, int count)***: Devuelve true sii se pudo reducir el stock del tipo y cantidad especificados. Exclusión mutua garantizada. ***public boolean increment(Order.EProductType type, int count)***: Análogo a increment, pero aumentando stock. Se valida que no supere el máximo valor.
OrdersFile	**Responsabilidad**: Almacenar los pedidos (orders) garantizando consulta y creación en exclusión mutua.
	Atributos principales: ***_filePath***: Ruta del archivo de pedidos. ***_lockFilePath***: Ruta del archivo de lock asociado al archivo de pedidos (podría ser el mismo si se leyera el mismo de forma binaria).
	Métodos principales: ***public List<Order> getOrdersByUserName(String userName)***: Devuelve la lista completa de pedidos realizados por el usuario de nombre userName. Si nunca hizo un pedido, se devuelve una lista vacía. La consulta se hace en exclusión mutua. ***public void create(Order order, Order.EOrderState initialState)***: Crea un pedido con los datos contenidos en order, y el estado de pedido inicial especificado por initialState. ***public void setState(Order order, Order.EOrderState state)***: Altera el estado de order, garantizando exclusión mutua. ***public void setState(String orderID, Order.EOrderState state)***:

	Overload para mayor comodidad. Busca el pedido con orderID dado, y actualiza su estado garantizando exclusión mutua. Si dicho pedido no existe, lanza una excepción.
Order	**Responsabilidad**: Representar un pedido, con sus valores y estado dentro del sistema (recibido, aprobado, rechazado o enviado).
	Atributos principales: ***CustomerName***: Nombre del cliente que realizó el pedido. ***ProductType***: Tipo de producto encargado en el pedido. ***_id***: Identificador unívoco del pedido dentro del sistema. ***Count***: Cantidad de unidades de producto de tipo ProductType encargados. ***State***: Estado actual del pedido: recibido, aprobado, rechazado o enviado. Como los pedidos son creados al ser recibido, el estado inicial siempre es recibido.
	Métodos principales: ***public byte[] serialize()***: Devuelve una representación binaria del pedido. ***public static Order deserialize(byte[] bytes)***: A partir de una representación binaria creada por serialize, se obtiene un objeto equivalente al original.
CustomerController	**Responsabilidad**: Simular a un cliente, enviando un pedido generado aleatoriamente a un OrderReceiver, y recibiendo la respuesta al mismo.
	Atributos principales: ***_deliveryChannel***: Canal para recibir la entrega del pedido. ***_resultChannel***: Canal para recibir el resultado del pedido (aprobado o rechazado). Se recibe usando _name como routing key. ***_name***: Nombre del cliente en el sistema. Sirve para identificarlo al enviarle el resultado o entrega.
	Métodos principales: ***public void run()***: Inicializa los canales, genera un pedido aleatorio, lo envía y recibe la respuesta.
CustomerQueryController	**Responsabilidad**: Simular a un cliente que consulta el estado de su pedido.

	Atributos principales: ***_queryChannel***: Canal para enviar la consulta. ***_queryResultChannel***: Canal para recibir el resultado de la consulta. Se usa _name como routing key. ***_name***: Nombre del cliente en el sistema. Sirve para identificarlo al enviarle el resultado.
	Métodos principales: ***public void run()***: Inicializa los canales, genera una consulta aleatoria, la envía y espera la respuesta.
OrderReceiverController	**Responsabilidad**: Procesar pedidos, actualizando el stock pertinentemente y despachando pedidos a Shipping.
	Atributos principales: ***_ordersChannel***: Canal para recibir los pedidos. ***_resultsChannel***: Canal para enviar los resultados a cada cliente. Se usa el nombre del cliente como routing key. ***_shippingChannel***: Canal para enviar pedidos a Shipping. ***_auditLogger***: Para registrar hora de creación de cada pedido. ***_orders***: Repositorio de pedidos. Se accede para crearlos y actualizar su estado (aprobado o rechazado). ***_stock***: Archivo de stock. Se accede para actualizarlo luego de aprobar un pedido.
	Métodos principales: ***public void run()***: Inicializa los canales y entra en un loop donde recibe y procesa pedidos. Si el pedido no es rechazado, se actualiza el stock, se registra el pedido en el sistema y es enviado a shipping. El procesamiento es secuencial (un pedido a la vez).
QueryHandlerController	**Responsabilidad**: Recibir y responder consultas de estado de pedidos.
	Atributos principales: ***_queriesChannel***: Canal para recibir las consultas. ***_queriesResultChannel***: Canal para enviar los resultados de las consultas. Se usa el nombre del cliente como routing key.
	Métodos principales:

	public void run(): Inicializa los canales y entra en un loop donde recibe y procesa consultas. Si el cliente que consulta nunca hizo un pedido, se le devuelve una lista de pedidos vacía.
ShippingController	**Responsabilidad**: Recibir pedidos y enviarlos al cliente que los realizó.
	Atributos principales: ***_shippingChannel***: Canal para recibir los pedidos a reenviar. ***_deliveryChannel***: Canal para enviar los pedidos. Se usa el nombre del cliente como routing key; el mismo está contenido en el pedido mismo. ***_orders***: Repositorio de pedidos. Se accede para marcar el pedido como enviado una vez hecha tal cosa.
	Métodos principales: ***public void run()***: Inicializa los canales y entra en un loop donde recibe y procesa pedidos.
StockAdminController	**Responsabilidad**: Actualizar stock.
	Atributos principales: ***_stock***: Para actualizar el stock garantizando exclusión mutua. ***_maxProductCount***: Máximo delta de stock (positivo o negativo). Es configurable.
	Métodos principales: ***public void run()***: Genera un tipo de producto y un delta aleatorios e intenta actualizar el stock con los mismos.

Vista de procesos: Diagrama de secuencia



Este diagrama corresponde al escenario más común: un cliente hace un pedido, hay stock suficiente y se le entrega el pedido sin problemas. Se omitieron los mecanismos de comunicación por simplicidad; los mismos pueden verse en la vista de despliegue (diagrama de robustez).

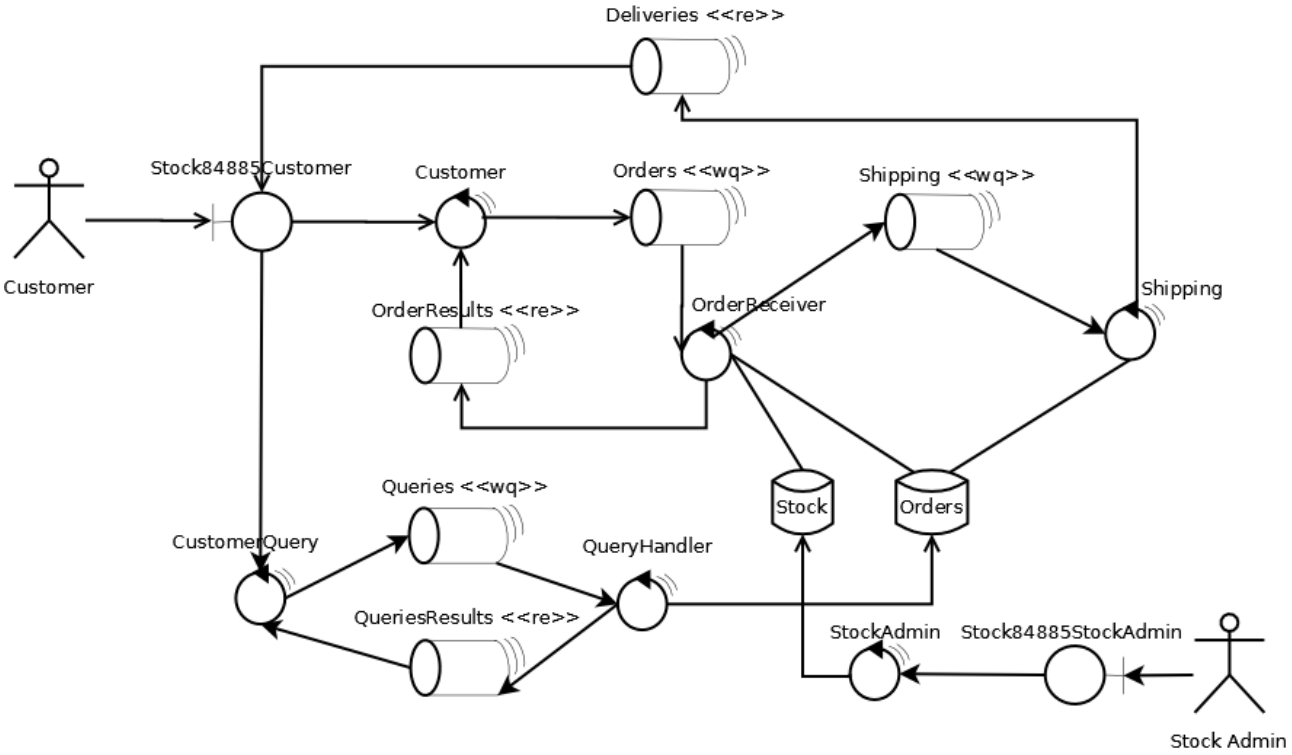
El cliente (Customer) manda su pedido a través de un objeto de borde de tipo **Stock84885Customer**, quien invoca a **CustomerController**. Éste instancia una **Order**, o, con los datos del pedido, y se la envía a **OrderReceiverController**. El OrderReceiver registra la orden con estado inicial RECEIVED. Luego, intenta decrementar el stock según el pedido. En este escenario, lo logra, y por ende cambia el estado del pedido a APPROVED, notifica al cliente que su pedido fue aprobado y envía la orden a un **ShippingController**.

En Shipping, la orden es despachada al cliente que la realizó, se marca la orden como enviada (DELIVERED), y se cierra el circuito.

El diagrama para el caso en que el pedido es rechazado sería similar, con la diferencia de que se marca la orden como rechazada (REJECTED), se notifica al cliente y se corta el circuito allí.

El acceso a IOrders e IStock es en exclusión mutua para toda objeto que los acceda, sea para leer o escribir.

Vista de despliegue: Diagrama de robustez



Estereotipos:

- re: Routing Exchange. Usa routing para enviar el mensaje a un receptor específico. Por ejemplo, Shipping lo usa para enviarle el pedido al cliente que lo pidió y a ningún otro.
- wq: Workers Queue. El productor pone el mensaje en el exchange, y lo toma cualquier consumidor (se usa el esquema por defecto, o sea Round Robin). Por ejemplo, lo usa el Customer para enviar su pedido y que lo tome cualquier OrderReceiver.

Sobre el diagrama:

Un cliente (**Customer**) envía un pedido (order) por la cola **Orders**, de donde la toma algún **OrderReceiver**. El mismo crea una instancia de Order y la guarda en **Orders** con estado inicial recibido. Luego, revisa el **Stock**, garantizando exclusión mutua, y le responde al **Customer** si el pedido fue rechazado porque no había stock, o aprobado. En ambos casos, se actualiza el estado del pedido.

Si el pedido es aprobado, es enviado a la cola **Shipping**, de donde la toma algún **ShippingController** y la envía por el exchange **Deliveries** al cliente que la pidió. Shipping sabe quién es ese cliente porque dicha información llegó en el pedido que recibió.

Por otro lado, un Customer puede consultar el estado de sus pedidos. Lo hace a través de **CustomerQuery**, quien envía el nombre del cliente por **Queries**, de donde lo toma **QueryHandler**. Éste revisa Orders, y manda la lista de pedidos con su estado al cliente vía **QueriesResults**.

Por último, el actor StockAdmin, a través del **controller homónimo**, puede consultar el stock de cualquier producto en cualquier momento, con exclusión mutua garantizada.

Vista de casos de uso

Caso de uso	Realizar pedido
Actores	Primario: Cliente (Customer)
Pre-condiciones	Stock definido y válido
Descripción	Flujo principal 1. Cliente envía su pedido especificando tipo de producto y cantidad. 2. Sistema registra pedido como recibido. 3. Sistema consulta stock de producto y cantidad solicitados. 4. Sistema marca pedido como aprobado. 5. Sistema responde a Cliente que su pedido fue aprobado. 6. Sistema envía pedido a Shipping 7. Sistema envía el pedido desde Shipping a Cliente.
	Flujo alternativo A1 (si no hay stock para el pedido) A1.1. Sistema marca pedido como rechazado. A1.2. Sistema responde a Cliente que su pedido fue rechazado.
Post-condiciones	Stock actualizado (decrementado) Pedido registrado en el sistema como rechazado o enviado

Caso de uso	Realizar consulta de estado de pedido
Actores	Primario: Cliente
Pre-condiciones	Registro de pedidos activo y válido
Descripción	Flujo principal: 1. Cliente solicita a Sistema el estado de su(s) pedido(s). 2. Sistema devuelve el listado de todos los pedidos realizados por Cliente, incluyendo su estado actual al momento de la consulta. Flujo alternativo A1: (si el cliente nunca hizo un pedido) A1.1. Sistema responde que no hay pedidos registrados
Post-condiciones	El Cliente tiene la información solicitada. No hay cambios en el Sistema.

Caso de uso	Modificar stock
Actores	Primario: Administrador de Stock (StockAdmin)
Pre-condiciones	Registro de stock válido
Descripción	Flujo principal: 1. Admin solicita un cambio en el stock, especificando tipo de producto y variación en stock (positiva o negativa). 2. Sistema realiza modificación. Flujo alternativo A1: (no se puede hacer el cambio porque daría stock negativo o por encima del máximo) A1.1. Sistema informa a Admin que no se puede efectuar el cambio
Post-condiciones	Stock actualizado.