# Clases

* **Entity**: Formato DB
* **DTO**: Transferir datos
* **Services**: lógica de negocio
* **Resources**: API REST HTTP

# Componentes

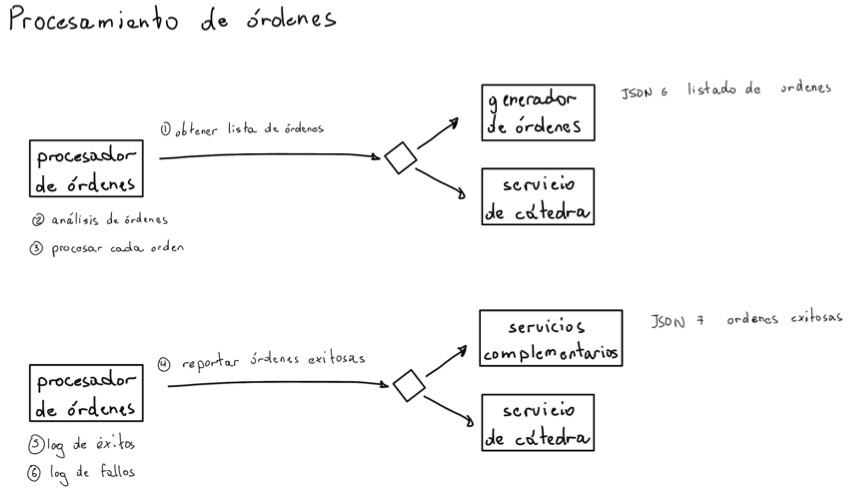
* Servicio catedra
* Procesador de ordenes (Este tengo que hacer)
* Generador de ordenes
* Servicios complementarios

# Proyecto

Servicio de compra/venta de acciones en la bolsa.

Existen servicios que serán los encargados de hacer funcionar el sistema, para la compra y venta de acciones se van a crear órdenes de compra/venta que quedarán almacenadas hasta que sean procesadas.

El procesamiento de las órdenes se realiza de forma asíncrona donde el servicio de **procesamiento de órdenes** se conectará con el almacén de operaciones pendientes para ser procesadas.



# Procesador de órdenes

## Obtener listado de órdenes

### Ordenes fija

(<http://192.168.194.254:8000/api/ordenes/ordenes/>)

Este endpoint de **servicio cátedra** permite recuperar un conjunto de órdenes necesarias para el servicio de **procesamiento de órdenes**. Este endpoint devuelve siempre la misma lista de órdenes, se utiliza durante el desarrollo para poder tener la posibilidad de consultar un servicio y que devuelva una lista de órdenes con las cuales podemos trabajar.

### Orden espejo

(<http://192.168.194.254:8000/api/ordenes/espejo>)

Este endpoint hará́ la retransmisión de un JSON que nosotros le inyectamos como valores de entrada. La idea es crear nuestra propia lista de operaciones manualmente, la inyectamos al endpoint y el endpoint nos devolverá́ el objeto creado. Este endpoint recibirá́ un objeto JSON representando una lista de órdenes que serán retransmitidas.

## Procesamiento de órdenes

El servicio analiza una por una las órdenes (*analizarOrdenes()*) y realiza las siguientes acciones:

1. Verifica si es posible de realizar la operación (*puedeRealizarOperacion()*) de acuerdo a las condiciones de procesamiento:

* Una orden instantánea no puede ejecutarse fuera del horario de transacciones, antes de las 09:00 y después de las 18:00.
* Una orden debe tener asociado un cliente y una acción de una compañía. Se debe verificar que el Id de cliente y el Id de la acción sean válidos. Para esto se debe consultar el **servicio cátedra** buscando por Id de ambos.
* Una orden no puede tener un número de acciones <=0. Para verificar este punto se deberá́ hacer una consulta a **servicios de la cátedra.**

1. Si **NO** es posible realizar la operación (*noEsPosibleOperar()*), se almacenará la operación en una lista de operaciones fallidas y continuamos con la siguiente.
2. Si es posible de realizar la operación (*EsPosibleOperar()*), almacenará la operación en una lista de operaciones finalizadas y, además, hay que definir un método de compra que devuelve un valor verdadero simulando el proceso de compra (*comprarOrden()*), y otro método para la venta (*venderOrden()*) que también devolverá́ un valor verdadero.

* Procesar órdenes inmediatas: Las órdenes de compra o venta cuyo modo es inmediato se ejecutan en el momento del análisis.
* Programar órdenes programadas (*programarOrden()*): Las órdenes de compra o venta cuyo modo sea al final del día (18:00), o a principio del día (09:00) se deberán programar para ser ejecutadas cuando corresponda.

El **resultado** del análisis es una lista de órdenes que pueden procesarse y una lista de órdenes que no pueden procesarse.

## Registrar resultado de órdenes

¿¿¿Tengo que guardarlo en una DB local o en el endpoint de la catedra???

(<http://192.168.194.254:8000/api/reporte-operaciones/reportar/>)

Todas las órdenes que se reciben deben almacenarse en la base de datos local del servicio, cada operación debe tener información del estado de la misma para saber si se pudo ejecutar con éxito (o no), si está programada o si la orden tiene errores:

* FALLIDO: Ordenes con errores.
* PROGRAMADO: Ordenes no ejecutadas ya que están programadas.
* VENDIDO: Ordenes ejecutadas con éxito.
* COMPRADO: Ordenes ejecutadas con éxito.

El endpoint de **servicio catedra** permite registrar el resultado de la operación sobre las órdenes. Puede recibir múltiples órdenes en un solo llamado.

## Logs de actividad relevante

Órdenes recuperadas del servidor, órdenes procesadas, ordenes programadas y ordenes que no se pudo realizar la operación.

## Pruebas (testing)

Usar mockito.

## Seguridad

No sé a qué se refiere.

## Reportes

Lista de órdenes procesadas, filtrar por (el filtrado puede ser por múltiples campos):

* Cliente
* Acción
* Fecha

Lista de órdenes no procesadas y mostrar motivo (ID cliente inexistente inválido, ID acción inválida, etc.)