Notebox

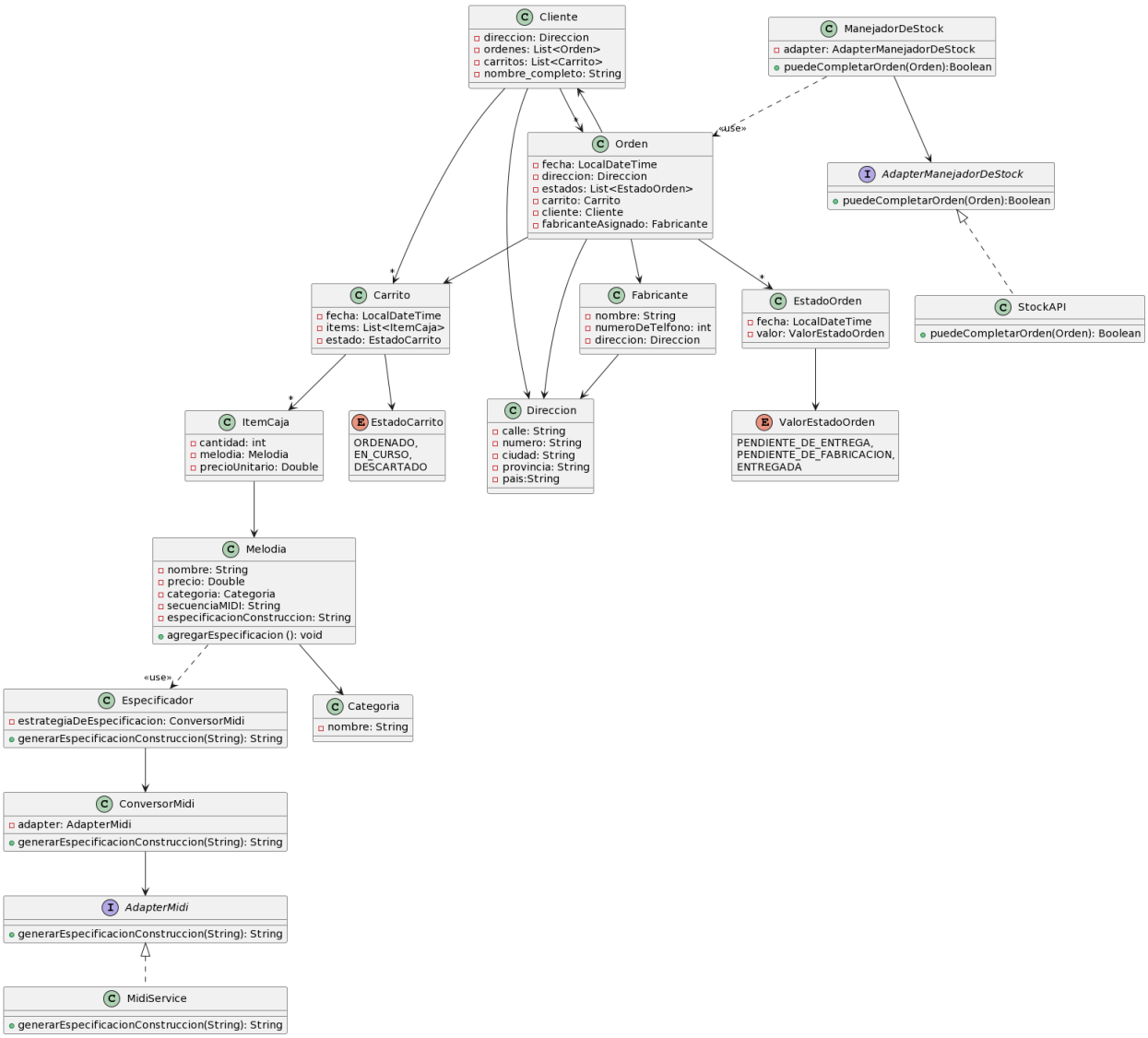
Final DDS 20220219.pdf

Clase de consulta:
https://drive.google.com/drive/folders/1HTUzoHNT7xeFdyG8eVIEL_fTrcmxoJyg

Hecho por: Facundo Piaggio, Chabela Lamas

Modelado de objetos

Diagrama de clases



Decisiones de diseño

Dado el presente contexto, se considero una clase Dirección dado que es una entidad muy relevante en nuestro dominio. De la misma manera, tal como se menciona en el enunciado, un cliente tiene una dirección pero esta puede cambiar en el tiempo, es por ello que sumamos una dirección en la clase Orden.

Consideramos la existencia de la clase Carrito, con una fecha, los items pedidos y un estado. Tener esta clase nos proporciona la ventaja de mantener el histórico de como fueron evolucionando los carritos solicitados (o no) por un cliente.

La clase Orden representa las posibles ordenes de un cliente, la misma tiene una fecha, la dirección de entrega y los estados por los que pudo ir pasando esta orden. El

estadoDeOrden es una clase compuesta por una fecha y un valorEstadoOrden el cual es un enum que representara el estado al que esta cambiando en si. Ademas, la orden, tiene una lista de ItemCaja solicitados por el cliente para esa orden en particular. Luego, la clase ItemCaja contempla una melodía particular, con una cantidad y su precioUnitario, valor que desnormalizamos para mantener el histórico del mismo y no perder la consistencia de datos. Esto nos da la posibilidad de que el cliente solicite varios items, y que incluyan X cantidad de cajas de una Y melodía.

Respecto a la melodía consideramos una clase, que contiene un atributo de precio, categoría, secuenciaMIDI, nombre. Cuando se crea una melodía, se llama al Especificador que se encarga de generar el archivo para la especificación de construcción a raíz del archivo MIDI que se encuentra en la clase. Asimismo, en Melodía se hace referencia a una instanciación de la clase categoría que luego permitirá realizar filtrados, entre otras funciones.

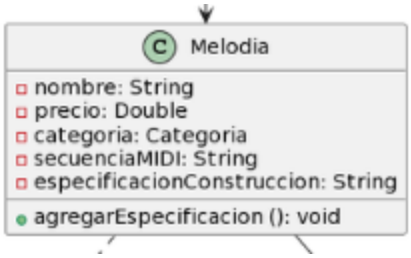
En el caso particular de los fabricantes, solo sumamos un atributo nombre, dirección y número de teléfono dado que se aclara que la disponibilidad y selección se realiza de manera telefónica, es decir que queda por fuera del alcance de nuestro sistema. Cabe aclarar que, se le asigna un fabricante a la orden y que esta puede ser llenada por un administrador. Esto no se presenta a nivel objetos ya que puede ser administrado por roles y permisos del aplicativo.

Por lo que respecta a el servicio que nos integramos para el manejo de stock y para la conversión del archivo de especificación de construcción, utilizamos un patrón Adapter ya que ganamos cohesión en las clases adaptadas, y delegamos la responsabilidad en objetos particulares, aumentando de esta manera la mantenibilidad al no tener acoplamiento entre la clase adaptada y el cliente.

Por último, la clase Fabricante no es un actor del sistema, solo se genera para guardar data y referencia de la creación de cajitas musicales. Permite cumplir con la trazabilidad mencionada de saber quién realiza la cajita a demanda. Cabe aclarar que, en el controller, una vez que se realice una orden se llama a la API REST para verificar si hay stock o no. En caso de que no haya, se debe realizar la llamada telefónica a los fabricantes (esto no se plasma en nuestro sistema, queda por fuera de nuestro alcance).

📌 Se podría sumar trazabilidad a las direcciones y que un cliente pueda poner varias direcciones que tengan un boolean de predeterminado. No lo dice la consigna pero es ver más allá del enunciado

> Justificar en forma detallada cómo resuelve el siguiente requerimiento: “Como persona administradora, poder agregar cajas musicales al catálogo, disparando el proceso de conversión de la secuencia MIDI a la especificación de construcción”.




```
>> AgregarMelodiaController
crearMelodia(datos) {

Melodia melodia = FactoryMelodia.crearMelodia(datos);
String especificacion = this.especificador.generarEspecificacio
                                                (melodia.getSecu
melodia.setEspecificacionConstruccion(especificacion);
this.repoMelodias.merge(melodia);

}
```

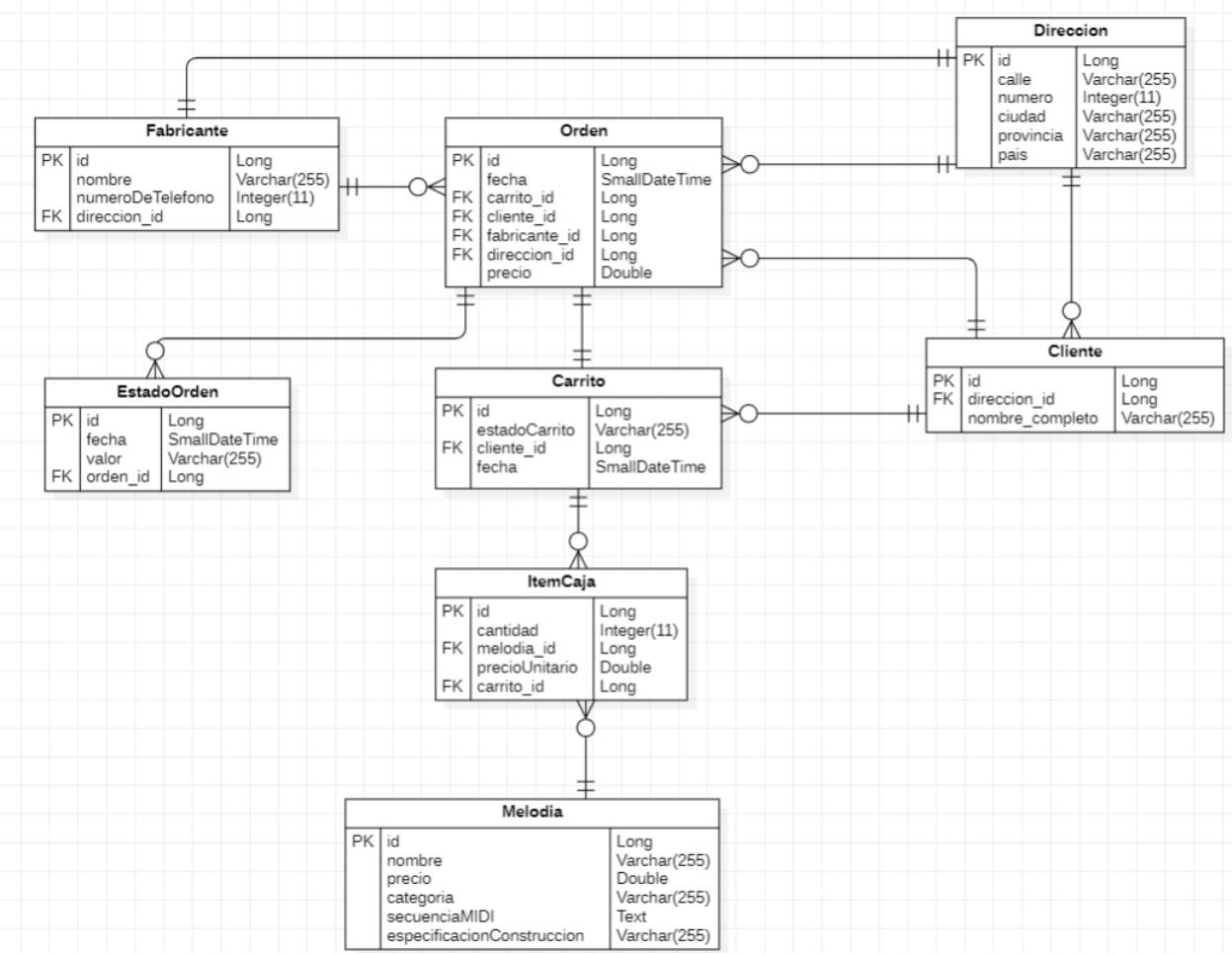
Una persona administradora, actor de nuestro sistema, presionara el botón de “Agregar melodía”, luego el sistema le mostrará el formulario de creación de una melodía el cual

el administrador llenara y enviará. Luego, un método de nuestro controller recibirá esos datos, y se creará una melodía con los mismos. Finalmente se llama al especificador, que se encargará de realizar especificación de construcción.

El especificador lo tomamos como un atributo del controller, como también al repositorio de melodías. A su vez, se podría crear un Factory de melodías para delegar la creación del objeto y que baje el acoplamiento, aumentando la cohesión.

Persistencia

Modelo de datos



Decisiones de diseño

Consideramos que la Direccion es una entidad relevante en el contexto, por lo que decidimos persistirla en una tabla propia, y no como datos embebidos dentro de otras tablas.

Para el estado del carrito usamos un converter, y lo que a nivel objetos tenemos como un enum, lo mapeamos como un varchar(255).

Para el estado de las ordenes, consideramos persistir el histórico de los estados que atravesó la orden en una tabla aparte, en donde persisitimos la fecha del cambio y ademas el valor del estado. Para el valor del estado, que a nivel objetos es un objeto, lo persistimos como un varchar(255) usando un converter.

En la Orden desnormalizamos el precio por consistencia de datos y performance al igual que en el ItemCaja. De esta manera, se fija el valor al cual tuvo que pagar el cliente la compra, manteniendo el histórico del precio. Cabe aclarar que el precio del ItemCaja se setea cuando se suma el carrito a una orden y éste cambia de estado a Ordenado.

DUDA: categoría debería ser tabla

Entidades necesarias a persistir

- Cliente
- Dirección
- Orden
- Carrito
- ItemCaja
- Melodía
- Fabricante
- EstadoOrden

Las interfaces, clases stateless de los adapters, clases como Especificador o ManejadorDeStock no deben ser persistidas, ya que representan comportamiento del sistema y no estado.

Impedance mismatch

Tipo de dato objetos	Tipo de dato relacional
LocalDateTime	SMALLDATETIME
String	VARCHAR(255)
Int	INTEGER(11)
String	TEXT

Respecto al impedance mismatch de Identidad, tuvimos que agregar un campo “id” (clave subrogada) para identificar unívocamente a cada una de nuestras entidades.

Modelo desnormalizado

Para el presente ejercicio se desnormalizaron los precios, tal como mencionamos anteriormente, por consistencia de datos y performance. A su vez, desnormalizamos la dirección en la orden ya que el cliente puede cambiarla luego de realizar una compra, tal como menciona el enunciado.

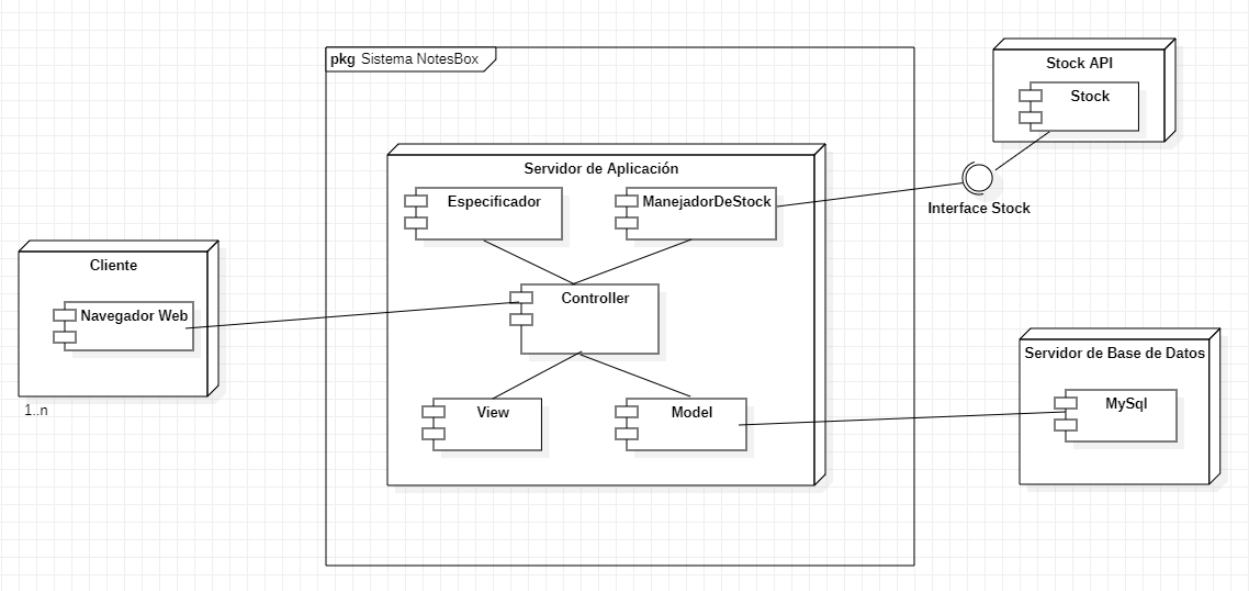
Arquitectura

> 1. Como parte del proceso de diseño, se ha propuesto que el sistema presente una
> arquitectura de microservicios, siendo uno el servicio de clientes y el otro el de administradores.
> a. ¿Qué opinás de esta propuesta? Justificá en base a las cualidades de diseño de
> Escalabilidad y Facilidad de mantenimiento.
> b. Si estás de acuerdo, realizá un diagrama de despliegue de cómo quedaría la solución. En
> caso contrario, proponé un diagrama de despliegue que muestre tu contrapropuesta

No estamos de acuerdo con dicha propuesta ya que nos parece que no es correcta dado el contexto de dicha empresa. Es una empresa familiar que quizás no requiere de tanta estructura y tan compleja para llevar a cabo su servicio. Los microservicios son recomendables cuando se tratan de aplicaciones complejas que necesitan de gran escalabilidad dinámica, sin poder predecir cuándo sucederá el evento que requiera del cambio. Por consiguiente, dado que esta empresa no es muy propensa a querer agregar componentes o cambiar el dominio de manera recurrente y que a su vez, un microservicio requiere de un gran equipo de desarrolladores para realizar dicho mantenimiento, convendría la implementación de un monolítico.

ESCALABILIDAD → Agregar nuevos componentes con poco impacto

FACILIDAD DE MANTENIMIENTO → Capacidad de modificar características existentes con el menor esfuerzo posible



> 2. Notesbox ha sido puesto en producción y nos han informado dos problemas. Para
> cada uno de ellos, explicá cual es una causa probable y qué mejoras de de arquitectura pueden realizarse para evitarlos:
> a. Cuando los administradores del sitio cargan una melodía nueva

> en el catálogo, el sitio les
> arroja un error de timeout y la conversión jamás termina.
> b. La cantidad de usuarios ha crecido notoriamente y estamos empezando a notar que los
> pedidos HTTP empiezan a demorar cada vez más e incluso hemos detectado caídas del
> servidor.

A. Se realiza el proceso de conversión de manera síncrona causando el TimeOut. Para ello recomendamos el uso de una cola de mensajes en pos de realizar dicho evento de manera asíncrona. Cuando se crean las melodías, se las insertan en una cola que luego una cron task, por ejemplo, puede iniciar con el proceso de la conversión. Esto se puede realizar dado que la conversión no tiene la necesidad de ser ejecutada instantáneamente para la parte del cliente y no debería generar un cuello de botella. Cabe aclarar que, la melodía podría ser mostrada una vez que sea convertida ya que sino se le expondría al cliente una melodía sin sonido de prueba.

B. Dicho servicio no cuenta con una cantidad de servidores suficientes para atender la mayor demanda y concurrencia de los clientes, debido a esto se ve afectada la disponibilidad y la performance. Por consiguiente, se podría escalar horizontalmente, contando con mayor cantidad de equipos servidores para atender las solicitudes de los clientes, distribuyendo la carga entre ellos con un Load Balancer. Se podría utilizar un algoritmo como Sticky Session, haciendo que la sesión abierta se mantenga siempre en el mismo servidor.