Trabajo Práctico Final

DISEÑO DE BASES DE DATOS - MG IS 2021

Se presenta en este documento el trabajo práctico final del curso, el cuál consiste en persistir un modelo de objetos en dos tipos diferentes de bases de datos: relacional (utilizando MySQL) y basada en documentos (utilizando MongoDB). El proyecto representa un sistema de deliverys de productos ofrecidos por distintos proveedores.

Se les proporcionará un proyecto en Java ya inicializado, el cual se trata de una API REST, utiliza el framework Spring y Maven como gestor de dependencias. El proyecto cuenta ya con una arquitectura establecida, similar a las vistas en otros proyectos del curso, así como también el modelo de clases y relaciones ya creado.

El proyecto se encuentra en el siguiente repositorio de GitHub:
https://github.com/fedediclaudio/tpfinal_bd

Tecnologías necesarias

Para el desarrollo del proyecto dado, serán necesarias de las siguientes tecnologías:

- Java JDK8
- Maven
  Como gestor de dependencias
- MySQL v5.7 en adelante
  Como base de datos relacional
- MongoDB
  Como base de datos orientada a documentos
- GitHub
  Como software de control de versiones

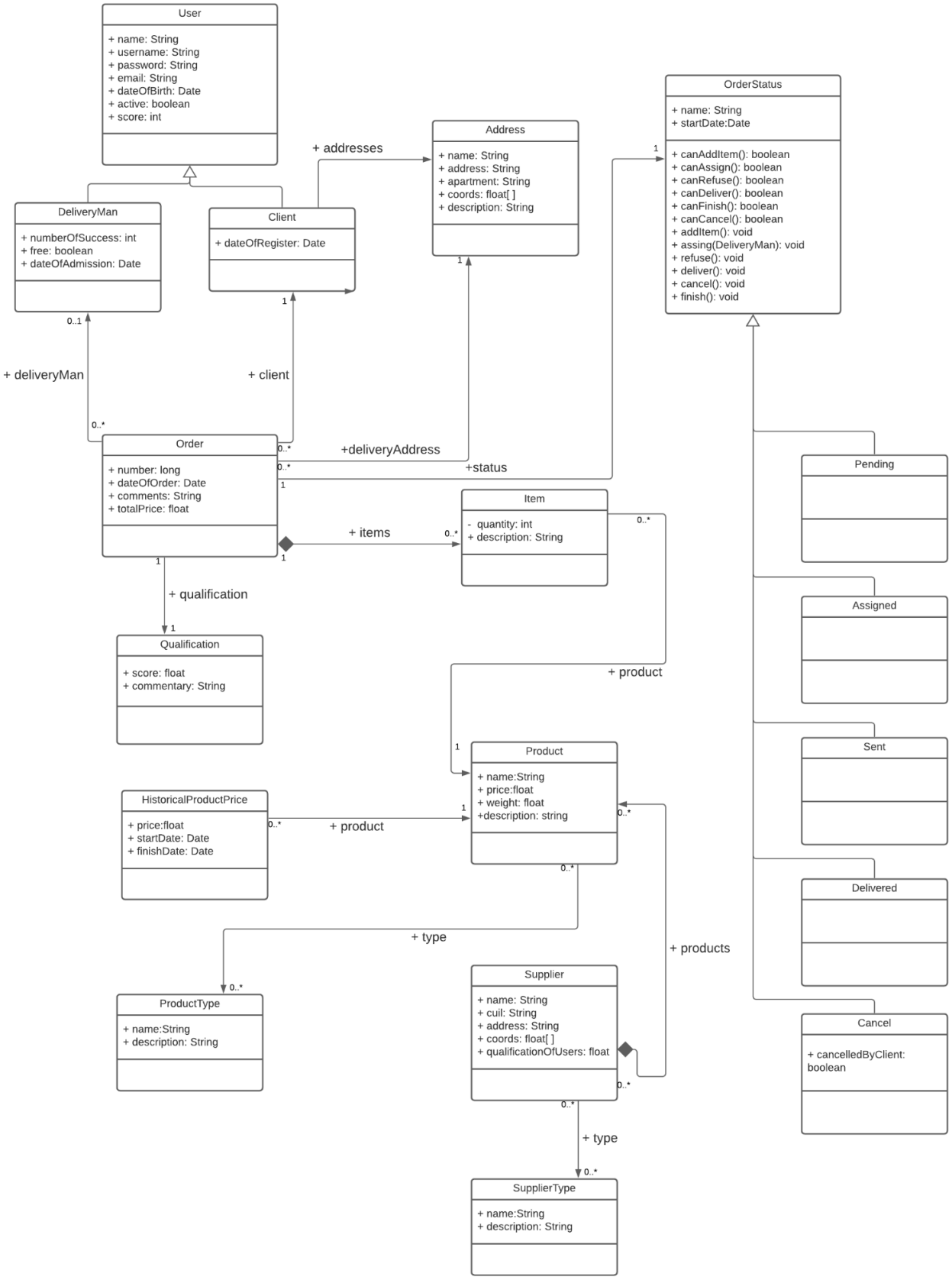
Se recomienda utilizar IntelliJ como IDE, aunque puede usarse el de su preferencia.

Para la instalación de las herramientas mencionadas puede utilizar la guía publicada al inicio del curso.

La utilización de contenedores Docker es opcional.

Descripción de la aplicación

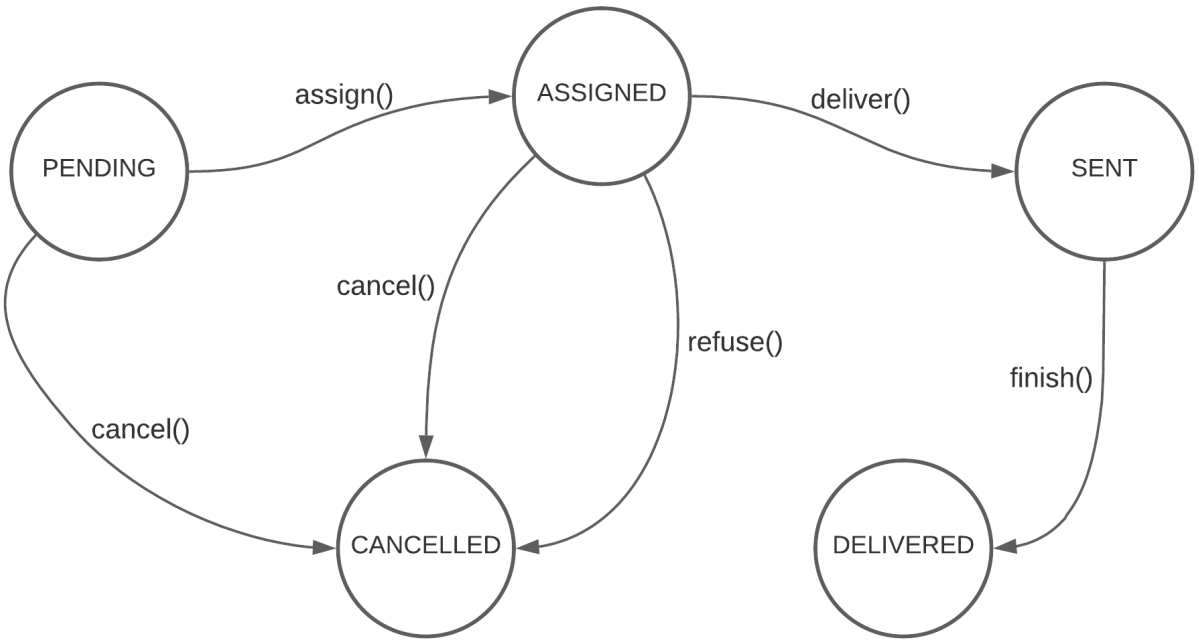
El modelo a persistir se trata de un sistema de delivery similar a soluciones existentes en el mercado (como PedidosYa, Rappi, etc.). Dicho modelo posee el siguiente diagrama de clases:


User
+ name: String
+ username: String
+ password: String
+ email: String
+ dateOfBirth: Date
+ active: boolean
+ score: int
DeliveryMan
+ numberOfSuccess: int
+ free: boolean
+ dateOfAdmission: Date
Client
+ dateOfRegister: Date
Address
+ name: String
+ address: String
+ apartment: String
+ coords: float[ ]
+ description: String
OrderStatus
+ name: String
+ startDate:Date
+ canAddItem(): boolean
+ canAssign(): boolean
+ canRefuse(): boolean
+ canDeliver(): boolean
+ canFinish(): boolean
+ canCancel(): boolean
+ addItem(): void
+ assing(DeliveryMan): void
+ refuse(): void
+ deliver(): void
+ cancel(): void
+ finish(): void
+ addresses
+ deliveryMan
+ client
+deliveryAddress
+status
Order
+ number: long
+ dateOfOrder: Date
+ comments: String
+ totalPrice: float
Item
- quantity: int
+ description: String
+ items
+ qualification
Qualification
+ score: float
+ commentary: String
+ product
Pending
Assigned
Sent
Delivered
Cancel
Product
+ name:String
+ price:float
+ weight: float
+description: string
HistoricalProductPrice
+ price:float
+ startDate: Date
+ finishDate: Date
+ product
+ type
+ products
ProductType
+ name:String
+ description: String
Supplier
+ name: String
+ cuil: String
+ address: String
+ coords: float[ ]
+ qualificationOfUsers: float
+ type
SupplierType
+ name:String
+ description: String

La aplicación dispone de un conjunto de usuarios (*User*) que pueden ser de dos tipos: repartidores (*DeliveryMan*), los cuales se encargan de llevar los pedidos generados por, el segundo tipo de usuarios, los clientes (*Client*). Cada usuario posee, además de ciertos datos propios del usuario, un atributo que determina si está activo (*active*) en el sistema, así como también un puntaje (*score*). Este puntaje se calcula de diferente manera dependiendo del tipo de usuario: un cliente suma un punto por cada pedido finalizado y resta un punto cuando cancela uno ya confirmado y asignado; un repartidor suma un punto cuando completa una entrega mientras que resta dos puntos cuando rechaza un pedido que le fue asignado.

Cada cliente posee un conjunto de direcciones (*Address*) guardadas en el sistema. Cuando este genera un pedido nuevo debe elegir una de sus direcciones como lugar de entrega.

Las órdenes (*Order*) poseen, además de sus datos básicos, un estado (*OrderState*), que irá cambiando a medida que la entrega avanza. Este estado, define el comportamiento dinámico ante las diferentes acciones que los usuarios pueden realizar sobre un pedido. Inicialmente, el estado del pedido será pendiente (*Pending*). Una vez confirmado por el cliente, el pedido se asigna a un repartidor libre y pasa a un estado de asignado (*Assigned*). Dicho repartidor puede rechazar el pedido (descontando su puntaje), en cuyo caso el estado pasa a un estado de cancelado (*Cancelled*), o puede aceptarlo y comenzar con el reparto del mismo, así este último pasa a un estado de en proceso (*Sent*). Una vez entregado, el pedido pasa a un estado de finalizado (*Finished*) y se actualizan los puntajes. Antes de que el pedido fuera aceptado por el repartidor, este puede ser cancelado por el cliente en cualquier momento, llevándolo a un estado de cancelado (*Cancelled*). El siguiente diagrama muestra de una manera gráfica la transición de estados de un pedido:



Cada orden se compone de una serie de items (*Item*), cada uno se trata de un producto solicitado con una cantidad del mismo. Opcionalmente el cliente puede agregar una descripción o aclaración sobre el producto ordenado.

Una vez finalizado el pedido el cliente puede generar una calificación (*Qualification*) en la cual podrá establecer un puntaje (de una a cinco estrellas) y una opinión en forma de texto.

El sistema también modela a los proveedores de los productos (*Supplier*), mediante los datos básicos de dichos proveedores, junto con una dirección y un calificación, la cual será el promedio entre las calificaciones recibidas por los clientes. Además, cada proveedor posee un tipo (*SupplierType*) que indica si se trata, por ejemplo, de un restaurant, heladería, quiosco, etc.

Cada proveedor ofrece una serie de productos (*Product*), de los cuales se registra nombre, precio, peso, una descripción y el tipo de producto que se trata (*ProductType*). Cada producto es exclusivo de un proveedor y un producto puede tener muchos tipos. Tanto el tipo de producto como el tipo de proveedor es necesario en la aplicación para la realización de búsquedas.

La aplicación necesita mantener un historial de los precios de cada producto (*HistoricalProductPrice*), para ello registra los valores que fue presentando entre las distintas fechas.

La aplicación utiliza el patrón State para implementar el estado de las órdenes, si no conoce de dicho patrón le dejamos un link de utilidad: https://refactoring.guru/es/design-patterns/state

Configuración

Para poder persistir los objetos deberá configurar la conexión de su proyecto con la correspondiente base de datos; para ello puede utilizar como guía los proyectos vistos durante el desarrollo del curso.

Las dependencias necesarias, tanto para el mapeo en MySQL como en MongoDB, se encuentran ya cargadas dentro del archivo de dependencias “POM.xml”, pero se encuentran comentadas. Según cuál va a utilizarse, descomente las correspondientes.

Actividad

Se deberá entregar el proyecto en dos resoluciones: una con persistencia de objetos a MySQL (base de datos relacional) y otra a MongoDB (base de datos orientada a documentos). Para ello se deberá implementar el mapeo mediante anotaciones en las clases, y deberá implementar los servicios y los repositorios necesarios.

El proyecto entregado deberá tener en cuenta los siguientes puntos:

- Utilizar relaciones bidireccionales en los casos que están sean necesarias.
- Determinar la carga bajo demanda de objetos relacionados (o lazy) y las operaciones en cascada sobre estas.
- Determinar el mapeo de objetos relacionados a documentos embebidos o referencias en MongoDB.
- Uso de transacciones en el servicio.

- Implementación del control de concurrencia mediante el versionado de objetos en las clases necesarias.

Se deberán implementar en la aplicación endpoints para realizar las siguientes acciones:
- Agregar un ítem a una orden ya creada.
- Confirmar un pedido. Esto implica buscar un repartidor libre y asignarle dicho pedido.
- Añadir una calificación a una orden ya completada. Tenga en cuenta que deberá actualizar la calificación del proveedor.
- Actualizar los datos de un producto. Tenga en cuenta que puede cambiar su precio.
- Eliminar un producto de los ofrecidos por un proveedor.
- Obtener todos los proveedores de un cierto tipo.
- Obtener todos los productos y su tipo, de un proveedor específico.
- Obtener las órdenes con más productos de un proveedor específico.
- Obtener la orden de mayor precio total de un día dado.
- Obtener los diez repartidores con mayor puntaje.
- Obtener los diez proveedores que más órdenes despacharon.
- Obtener los precios de un producto entre dos fechas dadas.
- Obtener el precio promedio de los productos de cada tipo, para todos los tipos.
- Obtener la información de los proveedores que tengan al menos una calificación de una estrella (la más baja). Es necesario también el número de estas calificaciones que el proveedor posee.
- Obtener los proveedores que ofrezcan productos de todos los tipos.

El uso de test para verificar el correcto funcionamiento del servicio creado es opcional.

El proyecto deberá estar acompañado por un informe en el cual se describa las resoluciones empleadas para los anteriores puntos, así como también las decisiones tomadas en las persistencias de objetos.

Forma de entrega

El proyecto será compartido en un repositorio de Github, sobre el cual cada grupo dispondrá de dos ramas: una en la cual deberá estar el proyecto mapeado a MySQL y otra donde deberá estar mapeado a MongoDB.

Será necesario que cada grupo comunique los usuarios de Github de cada integrante para poder asignarle un número de grupo y compartirles dichas ramas, las cuales serán de acceso solo por parte de los integrantes de cada grupo.

Fecha límite de entrega: 31 de Junio de 2022