**UNIVERSIDAD NACIONAL DE LA PLATA**

**FACULTAD DE INFORMÁTICA**

**Diseño de Bases de Datos**

Año: 2021

Profesores: Javier Bazzocco y Federico Diclaudio

Alumnos: Alejandra Lucia Pérez Lucero y Gustavo Lazarte

**Introducción**

En este trabajo se ha implementado un sistema para gestión de pedidos. Dicho sistema permite a clientes finales realizar pedidos a distintos proveedores para ser entregados mediante repartidores registrados en el sistema.

**Desarrollo**

Se ha implementado el sistema utilizando principalmente Python 3.9 y Django 3.2.9. Estas herramientas permiten facilitar el manejo de las requests y abstraer el manejo de la base de datos mediante modelos.

Para poder proveer de funcionalidades correspondientes a una REST API, se utilizó la librería django-rest-framework. Mediante ella es que se implementó el procesamiento de las requests entrantes al sistema.

**Desarrollo: decisiones de diseño de las tablas**

Para el diseño de la aplicación se ha seguido mayormente el diseño propuesto en el enunciado del proyecto, con algunas modificaciones:

* Para la implementación del estado de la orden, se utilizó una única tabla donde el estado de la orden es representado por un número entero. Esta decisión se tomó para evitar la creación de múltiples tablas para el manejo del estado; en su lugar se utiliza una sola donde se realizan todas las operaciones. Esto permite simplificar las operaciones de modificación del estado al permitir realizar una única operación por transacción.
* La relación entre la tabla Supplier y la tabla Product fue cambiada por una relación one-to-many en lugar de many-to-many. Esto se hizo para simplificar las queries y porque representa una relación más “realista” de un sistema de pedidos.
* Todas las relaciones pueden ser consideradas bidireccionales. Esto es debido a Django en realidad, que provee el campo related\_field en cada columna de clave foránea para poder ser usada de forma inversa.
* Todas las relaciones many-to-one fueron cambiadas por relaciones one-to-many. Esto es debido a que, entre estas dos, Django sólo soporta las one-to-many, proveyendo del campo mencionado anteriormente para emular el comportamiento de una relación many-to-one.
* Los campos de fecha de creación o modificación que las tablas puedan poseer son manejados de manera automática por Django.
* Todas las relaciones one-to-many se han implementado agregando una constraint para impedir que si un elemento es borrado, sus dependencias también lo sean. Es decir, no se han agregado constraints de cascada en las tablas, sino que se han agregado constraints de protección.
* Por otro lado, se optó por utilizar transacciones únicamente al crear o actualizar el precio de un producto. Esta operación también agrega una entrada a la tabla HistoricalProductPrice, por el cual existe la necesidad de preservar el orden de las operaciones. Realizar ambas operaciones en una única transacción permite conservar la integridad de los datos en la base de datos.
* Finalmente, algunos aspectos mencionados en el enunciado tales como repositorios o versionado no aplican en Django dado que éste no brinda soporte para ese tipo de operaciones.

**Desarrollo con MySQL**

En primer lugar, se realizó una implementación utilizando MySQL 8 como base de datos. Al ser una base de datos relacional, se pudo implementar el control de la base de datos usando exclusivamente las librerías propias de Django.

Diagram

Description automatically generated

**Desarrollo con Mongo**

De manera similar, se realizó un desarrollo utilizando una base de datos no relacional: MongoDB. Para este proyecto, se utilizó MongoDB 4. Se optó por no utilizar MongoDB 5 (la versión más reciente al momento de realizar este proyecto) por problemas de compatibilidad entre la imagen de Docker y algunas computadoras que utilizan MacOS.

A continuación, se muestra el diagrama de la relación entre las distintas colecciones implementado en Mongo:

Diagram

Description automatically generated

Se puede observar que hay muchas similitudes entre esta implementación y la realizada en MySQL. El mayor cambio está en los campos usados como clave primaria: Mongo exige utilizar un tipo propio denominado ObjectId, mientras que en MySQL comúnmente se utilizan números enteros para representarlas. Si bien en ambos casos los valores son autogenerados, esto presenta algunas diferencias al utilizar los IDs en las llamadas a la REST API.

A diferencia de con MySQL, se requirió utilizar una librería adicional (djongo). Ésta permite utilizar las distintas funcionalidades de Django utilizando MongoDB como base de datos, sin tener que realizar cambios mayores en el código. Esto permite incrementar la productividad al no realizar migraciones excesivamente largas desde la versión con MySQL durante el desarrollo de la implementación.

**Conclusión**

Para el proyecto que representa un sistema de deliverys de productos ofrecidos por distintos proveedores se desarrolló un modelo de objetos en dos tipos diferentes de bases de datos: relacional, utilizando MySQL, y basada en documentos, utilizando MongoDB.

**Anexo: Instrucciones de ejecución**

Para facilitar la ejecución del sistema, se ha encapsulado la base de datos y la REST API en imágenes de Docker, las cuales pueden ser controladas mediante docker-compose.

Las imágenes de Docker pueden ser lanzadas utilizando el comando docker-compose up.

Los endpoints pueden probarse utilizando alguna aplicación de test (curl, Postman, etc.) o accediendo al link <http://localhost:8080/>. Este link corresponde a una imagen de Docker de Swagger/OpenAPI que es activada junto con la imagen de la base de datos y la imagen de la REST API. En ella están listados todos los endpoints implementados en la aplicación, los parámetros que toma cada una y las respuestas devueltas por cada una.