**DESIGN document for**

**backend of Sales Order**

# Index

[Index](#h.hd6n4g7jr62v)

[Introduction](#h.xb527l9s42k5)

[UML Class Diagram](#h.pvgeqn9exvp3)

[Database Diagram](#h.sxjk2ku81x2g)

[Technologies Used](#h.bvikjnqegj02)

[Logging of the system](#h.t9imce1dkdii)

[JUnit tests](#h.6uaoir8zkhat)

# 

# Introduction

The backend system was implemented using a layered approach where there are 3 different layers:

Controller(uy.com.orders.controller)

This layer is in charge of receiving the requests and call the appropiate service to process it.

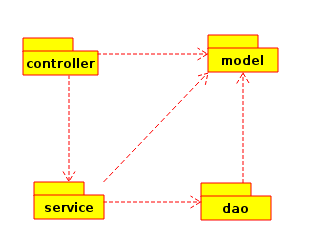
Service(uy.com.orders.service)

This layer is the mediator between controllers and the model (data). It dispatches to the proper dao to process the request.

Model (uy.com.orders.model, uy.com.orders.dao)

This layer is the one responsible to communicate with database. CRUD operations are done here. It exposes its methods through DAOs interfaces and then communicates with the database through model classes defined with JPA annotations and processed by hibernate.

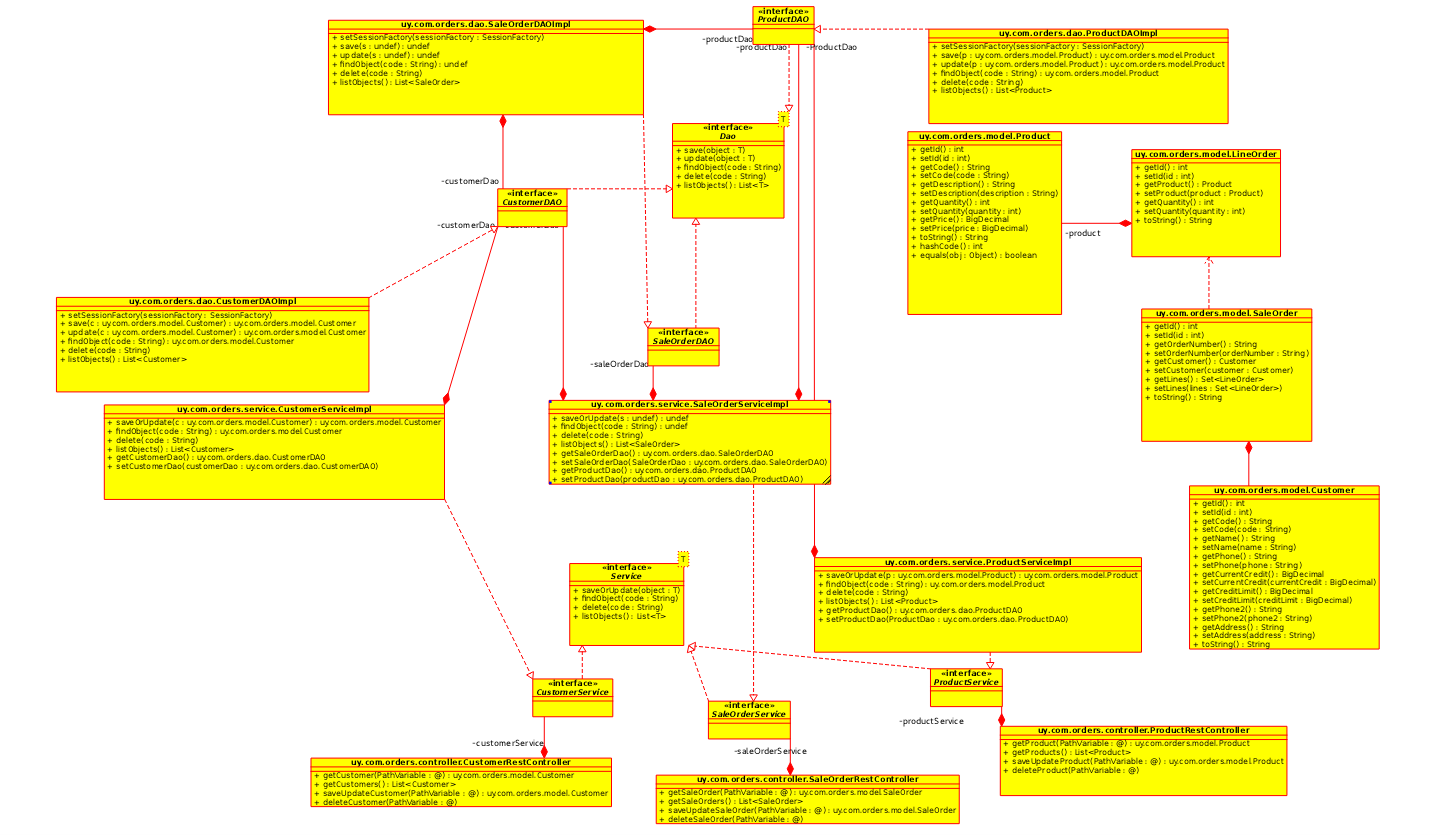
**UML Package Diagram of Backend**



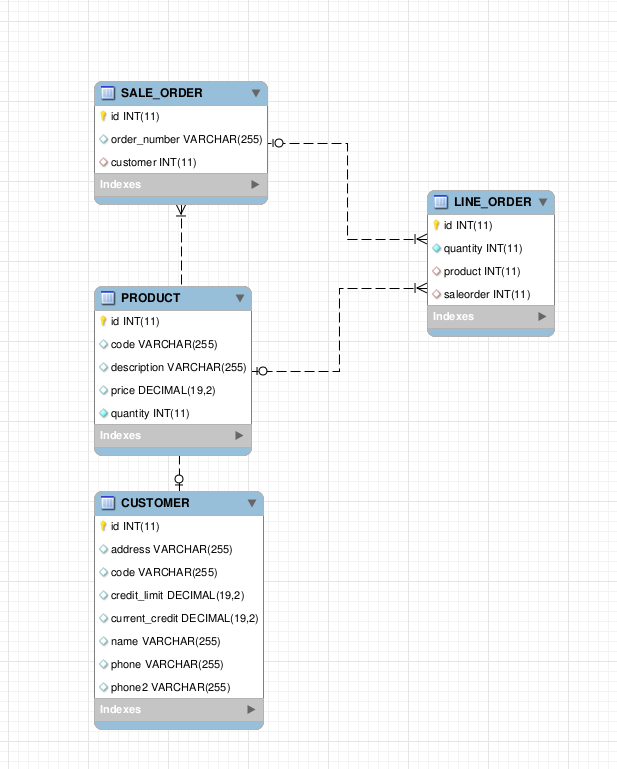
As we can see on this diagram applying the layered pattern, we have unidirectional references of the different packages.

It was a design decision to make the controller and service package reference the model as I did not wanted to create clone classes of the model in other package. It was preferred to gain mantainability but knowing that it affected coupling.

# UML Class Diagram



# Database Diagram



# Technologies Used

Maven 3

Apache Tomcat 8

Spring 4

Hibernate 4

MySQL

RESTFul webservices

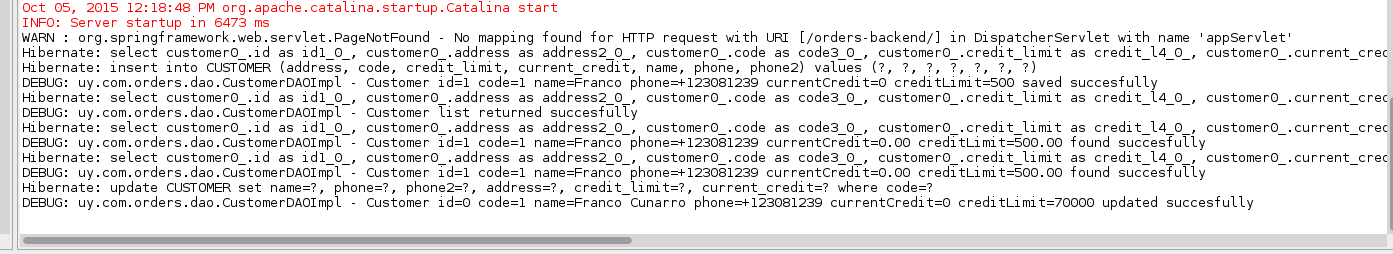
JSON

Jackson libraries for processing JSON

Java 1.8

# Logging of the system

Library used was slf4j.



# JUnit tests

Not implemented