Online Pizza Ordering System
Analysis and Design Document

Student: Margin Razvan Cristian
Group: 30432

Revision History

Date	Version	Description	Author
04/04/18	<1.0>	Added Architectural Design + Domain Model	Margin Razvan Cristian
25/04/18	<1.1>	Iteration 1.2	Margin Razvan Cristian
21/05/18	<1.2>	Adjustments	Margin Razvan Cristian

Table of Contents

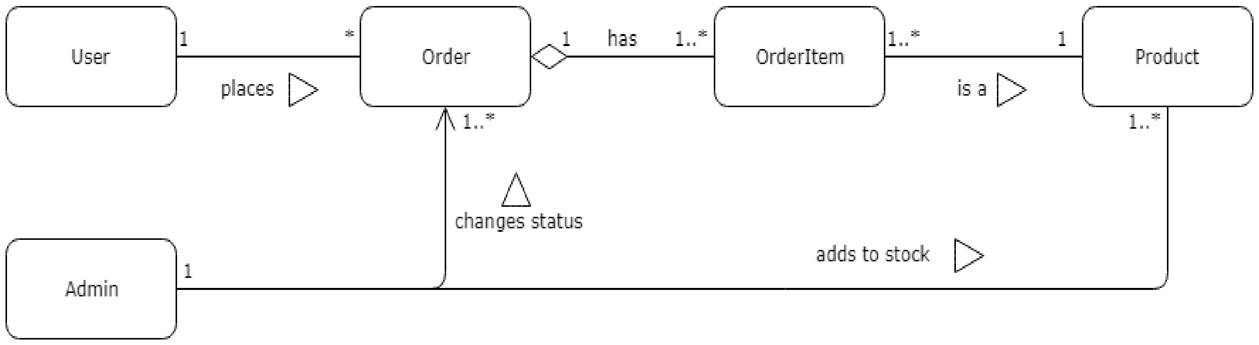
I. Project Specification

The project consists in developing a web application from which a client of a pizza restaurant, does not need to place an order in a “face-to-face” approach. Instead, he can simply order something online and pick up his order when it is ready.

II. Elaboration – Iteration 1.1

1. Domain Model

The domain model consists in the 2 main actors of our system: the User (Client), and the Employee (Admin). As mentioned above, the User places an Order, which contains all the items that he wants to purchase at a transaction: OrderItems. Finally, each OrderItem is associated to a Product. The Admin has the responsibilities of changing the status of an order (from not finished to finished), and update the Menu (the Products) if there is need for this.



2. Architectural Design

2.1 Conceptual Architecture

I will use in this project the **layered architectural pattern**, because it helps us organize the structure of our application much easier, grouping components into layers, according to their responsibility.

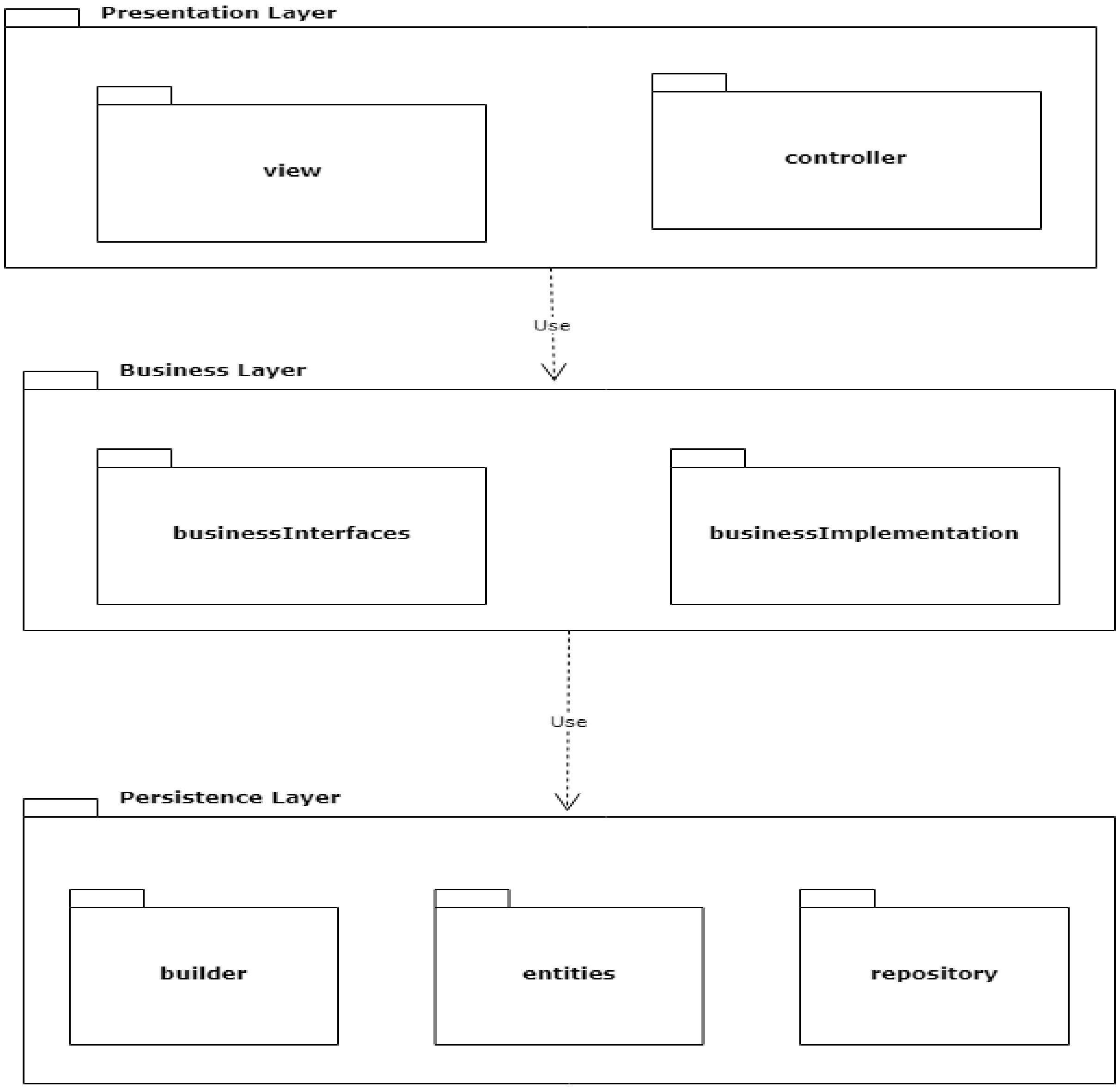
Presentation Layer
The Presentation Layer will manage the interactions of our application with the users. Therefore, this layer will be responsible with retrieving data from the user and displaying it back, according with the events occurred.

Business Layer
The Business Layer encapsulates all the business logic of our system. Its job is to process the information passed on by the presentation layer.

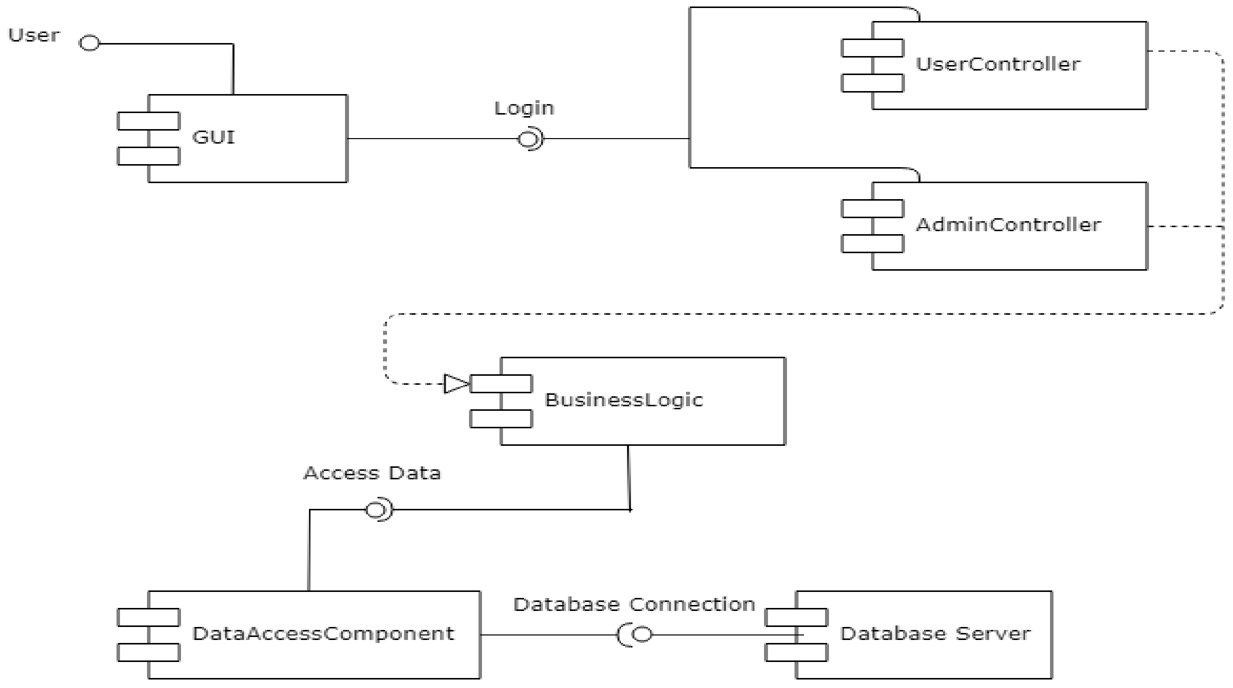
Data Layer
The Data Layer consists of holding the data of our application, having the job of giving access to data that is stored outside our system and passing it to the Business Layer.

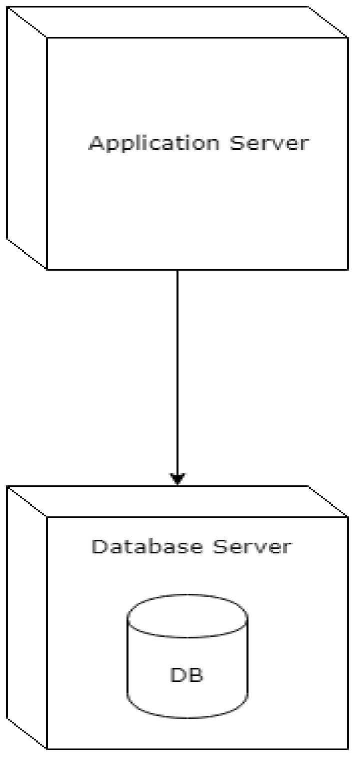
Furthermore, because we are using Spring, we will take advantage and will use the **MVC architectural pattern**, which will make easier the exchange of information between the View, Controller and Model.

2.2 Package Design



2.3 Component and Deployment Diagrams



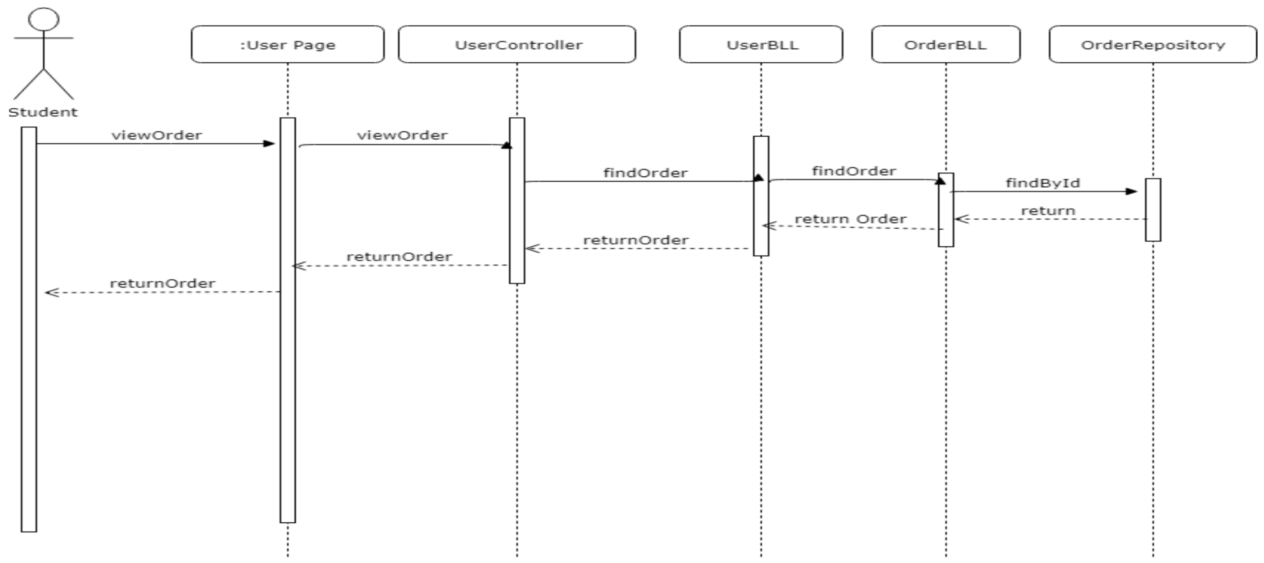


III. Elaboration – Iteration 1.2

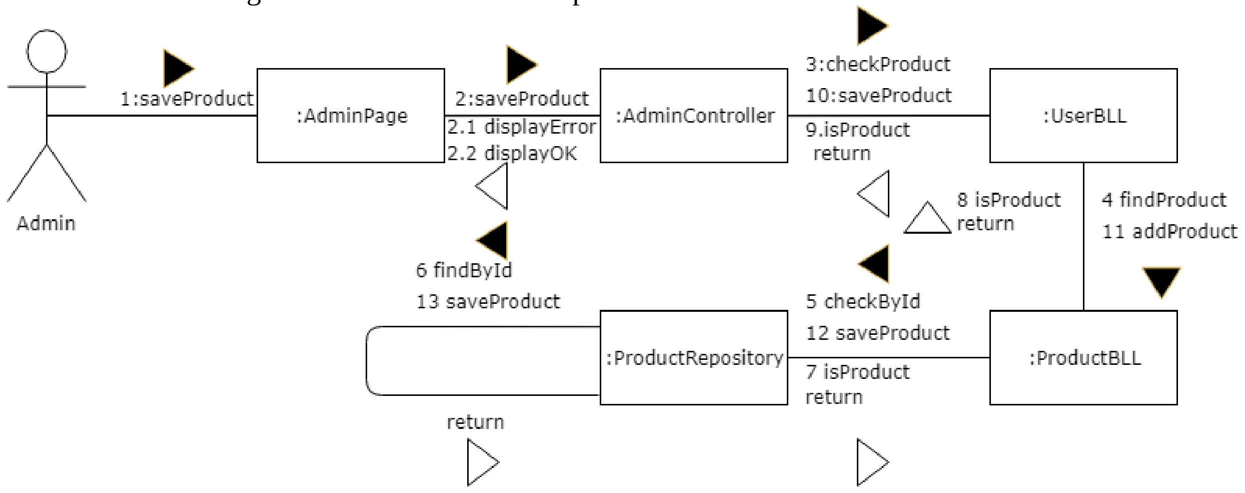
1. Design Model

1.1 Dynamic Behavior

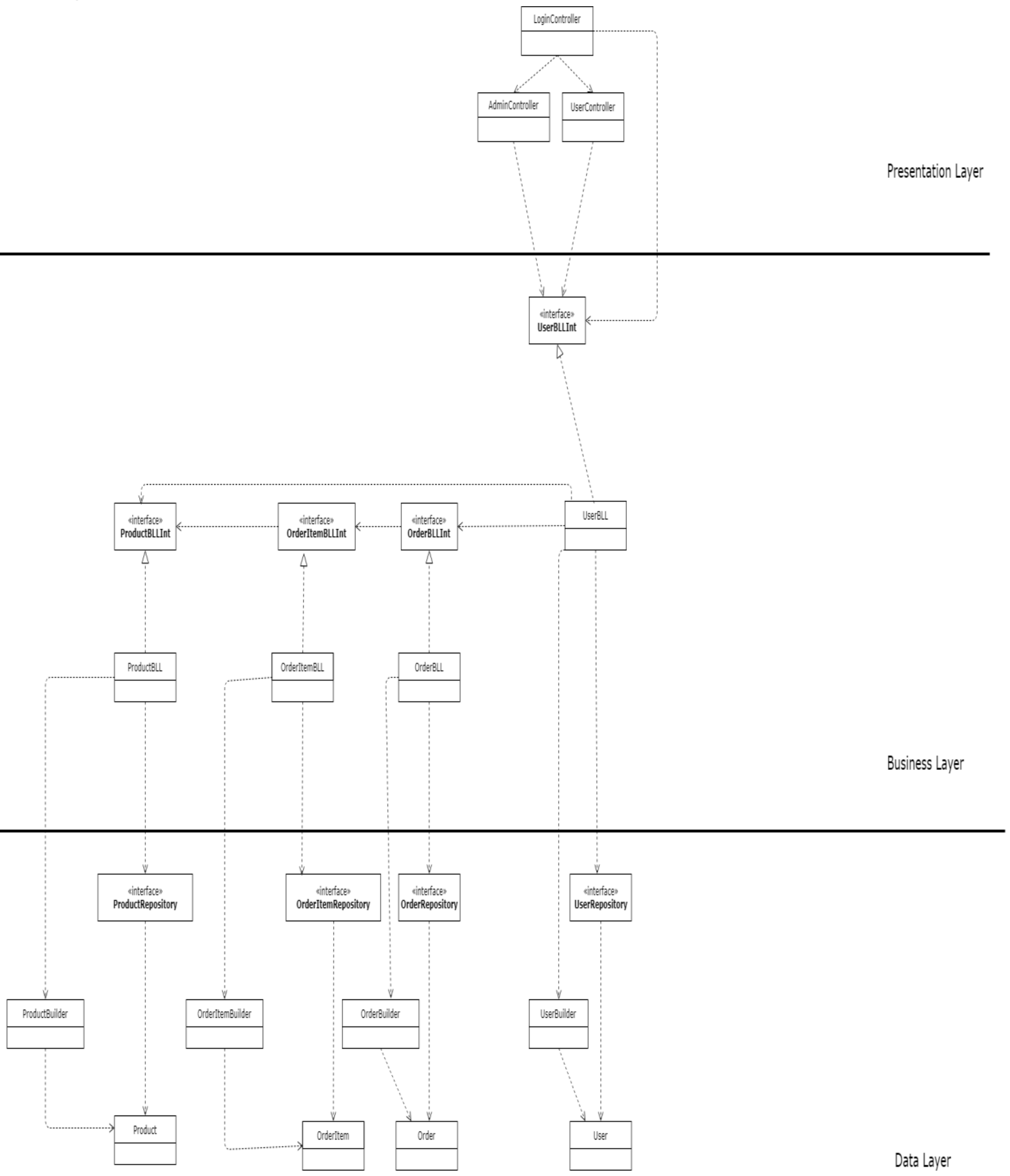
Sequence Diagram for a user to view his order:



Communication Diagram for admin to add new product:



1.2 Class Design

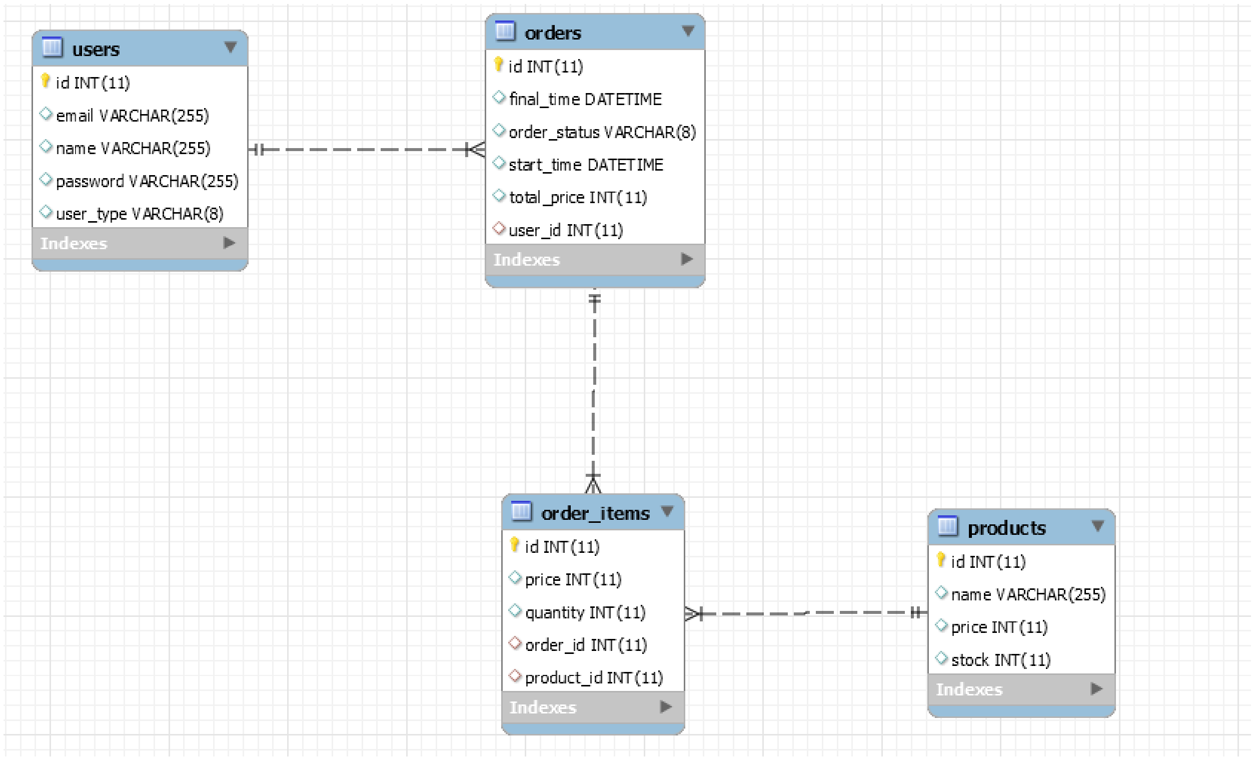


1.3 Design Patterns

- Builder Pattern : used for instantiating objects of our entities (because our entities can hold several attributes, it is much easier to use this pattern in order to create certain objects + useful in the case of integration tests)
- Observer Patter: used for notifying the User when his/her order is being changed (in the case of changing the status of the order)
- Chain of responsibility pattern: used throughout the project and imposed by the architectural layer + also useful when handling orders (the order responsibility is passed on the orderItem responsibility which is passed further on to the product)

2. Data Model

The Data Model represents the entities that we want to store in our Database. As mentioned above, we will have users stored in the users table. It is important to mention that through the user_type column one can check the type of the user, which can be either CLIENT or ADMIN. Each will need an email and a password in order to login. Also, each client will have an order, which will contain a status(delivered/undelivered), a total price for the order, and references to other tables. Furthermore, each order can contain several order items, which contain a single product. The difference between the product and the order items is that a product represents a single item, whereas the orderItems table will hold information about the quantity of the product chosen.



3. Unit Testing

Regarding the testing provided for our product, we will conduct **JUnit** testing in order to verify the behavior of single classes reported to different inputs. Furthermore, after testing each unit from a layer we will go on and start performing integration tests in order to see how our complete system works together.

IV. Elaboration – Iteration 2

1. Architectural Design Refinement

-

2. Design Model Refinement

-

V. Construction and Transition

1. System Testing

In the current state of our product, I have managed to test each component of the persistence layer, performing the CRUD operations, which worked well. Secondly, I started providing testing for the business layer, which until now is performing as expected.

2. Future improvements

As future improvements of the product, if time is given, one could implement a mobile app that performs the same tasks as the web application. Furthermore, the product could invest in a tracking system which could be available on the web app, in order to track your order in real time, on a map.

VI. Bibliography