Best Furniture Deals

Analysis and Design Document

Student: Ruști Maria

**Group:30238**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

This is an online application that allows users to search and buy furniture products. In order to do that, they must first create accounts and then log in. The products are being managed by the administrator, who is in charge of keeping everything up-to-date.

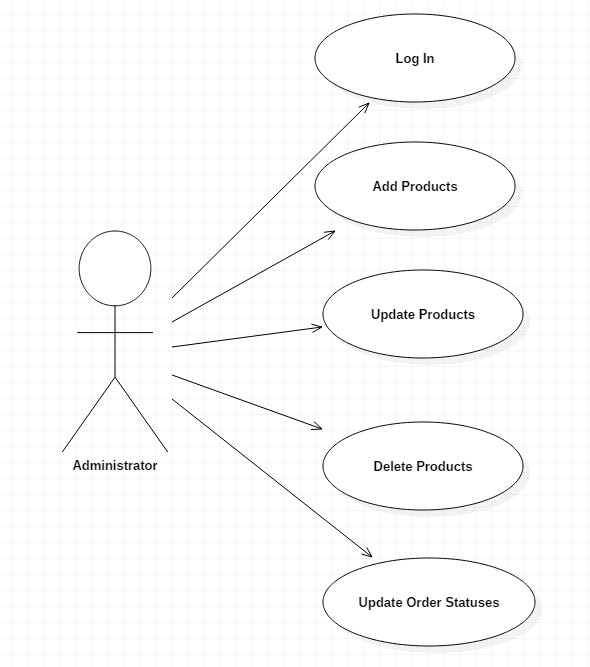
# Functional Requirements

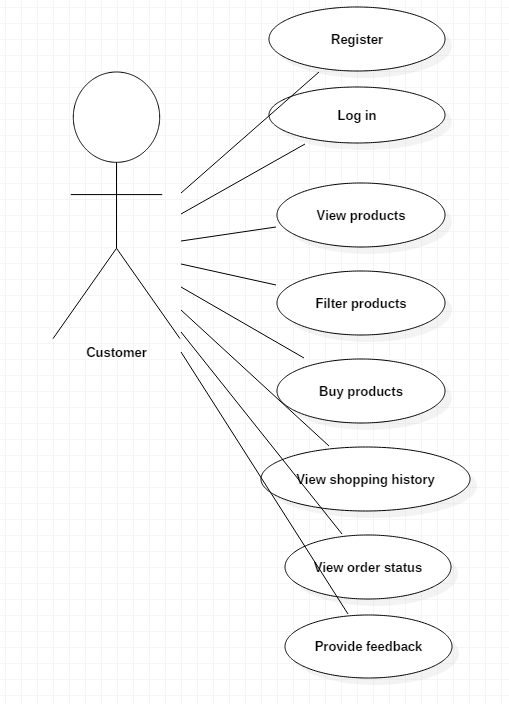
* Allow customers to register
* Allow customers to see the products available at the time
* Provide filtering options so customers can easily find what they are looking for
* Allow customers to choose products in order to buy them
* Implement an adequate checkout mechanism
* Provide feedback form for customers
* Allow customers to access their shopping history
* Allow customers to receive messages
* Allow the administrator to manage the products

# Non-functional Requirements

* Access of confidential data
* Availability
* Security
* Maintainability
* Portability

2. Use-Case Model

**

**

*Use case:* buy products

*Level:* user-goal level

*Primary actor:* customer

*Main success scenario:* The customers access the application page. If they want to buy some products, they will have to register if they don’t have an account or to log in if they already registered. After that, they have the opportunity to choose the products that satisfy their needs and proceed with the order.

*Extensions:* The customer might be unable to register or to log in due to some invalid data.

3. System Architectural Design

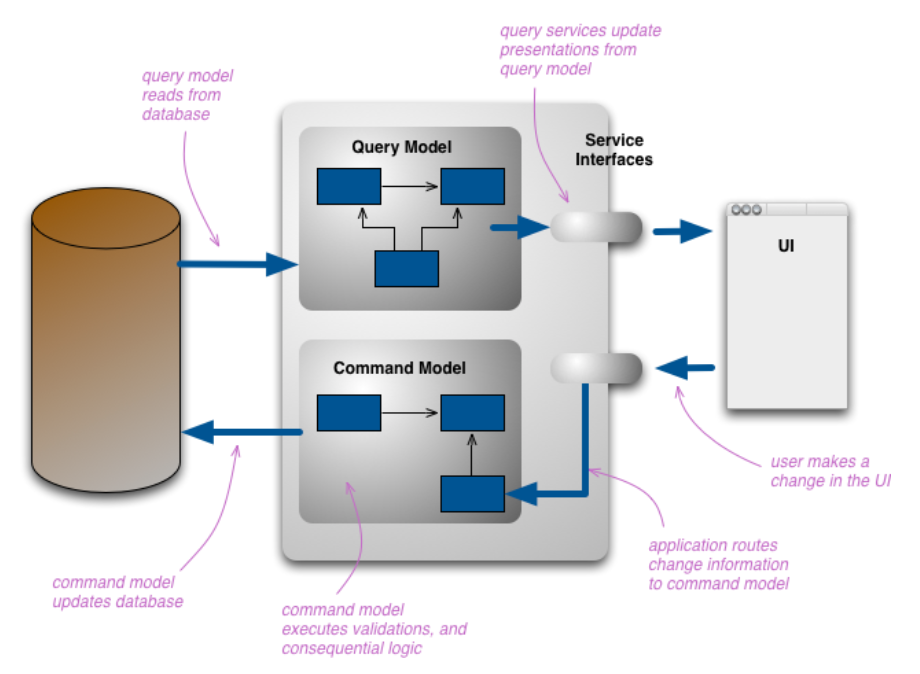
**3.1 System Architecture Description**

**CQRS Architecture**

Segregate operations that read data from operations that update data by using separate interfaces. This can maximize performance, scalability, and security. Supports the evolution of the system over time through higher flexibility, and prevents update commands from causing merge conflicts at the domain level.

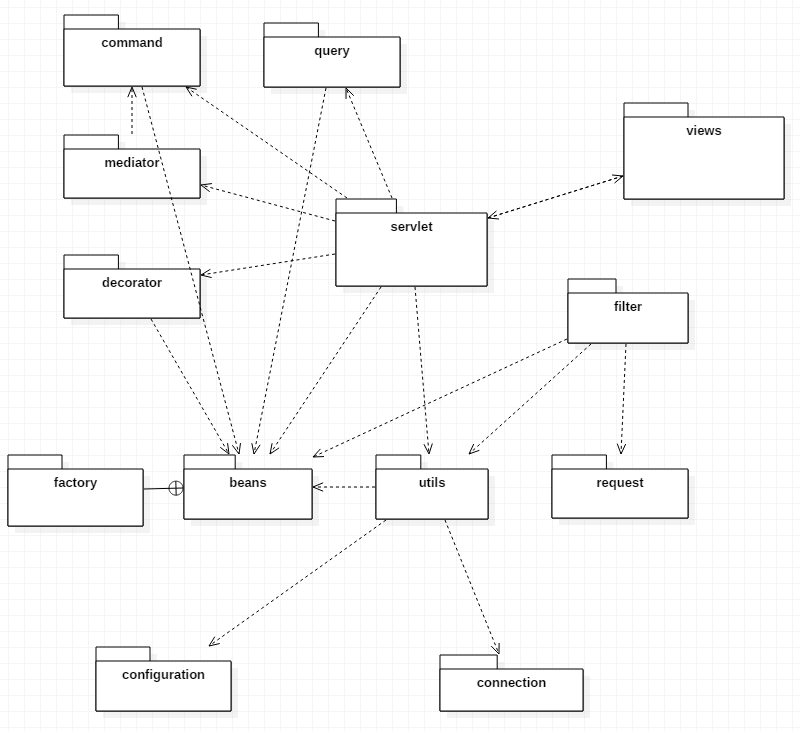
CQRS addresses separates reads and writes into separate models, using **commands** to update data, and **queries** to read data.

* Commands should be task based, rather than data centric. Commands may be placed on a queue for asynchronous processing, rather than being processed synchronously.
* Queries never modify the database. A query returns a DTO that does not encapsulate any domain knowledge.

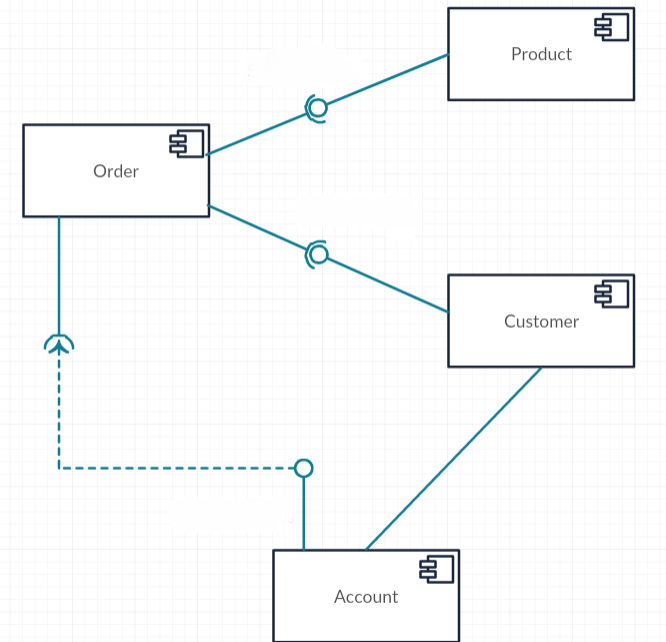


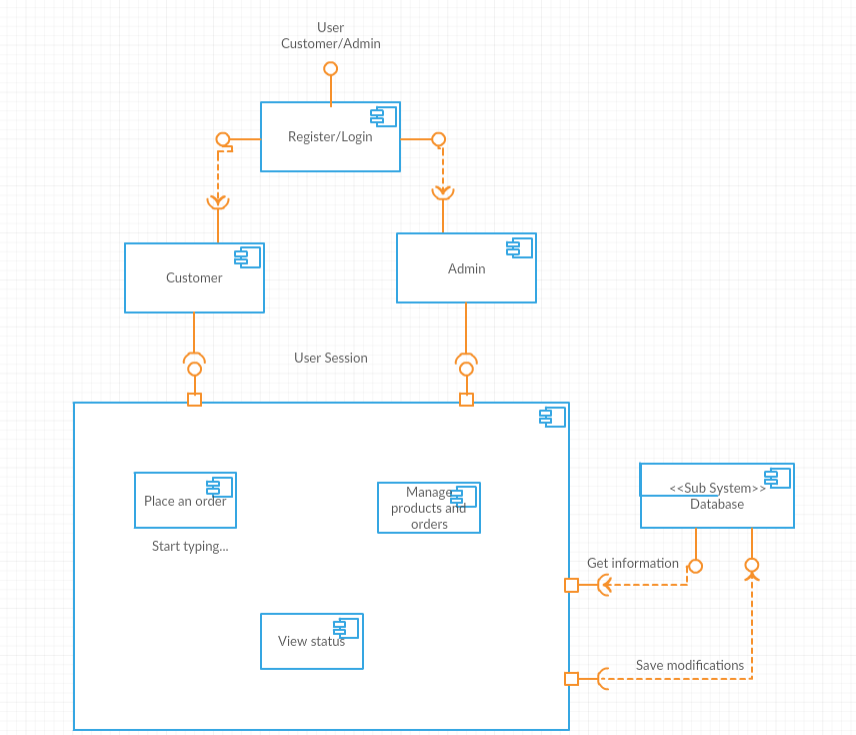
**3.2 Diagrams**

Package Diagram

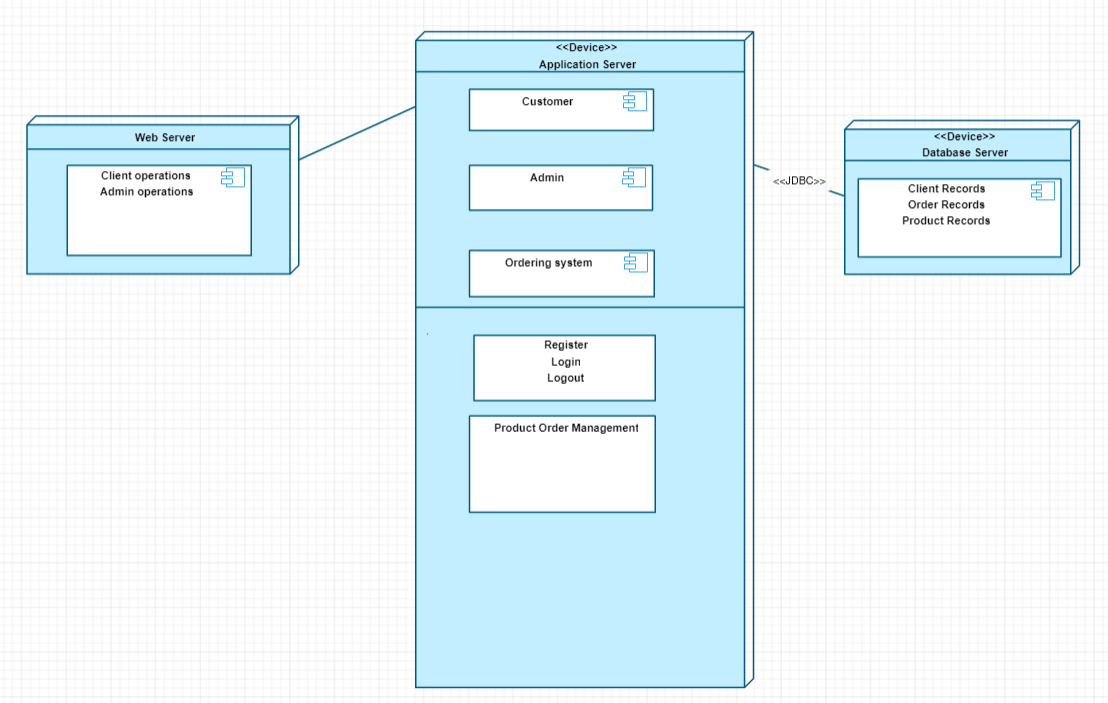


Component Diagram



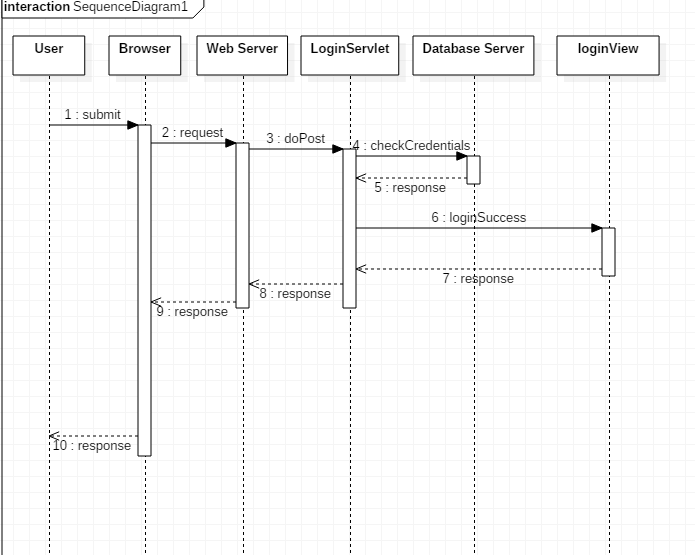


Deployment Diagram



4. UML Sequence Diagrams

Scenario: user login



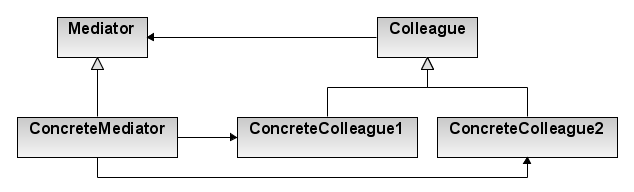
5. Class Design

**5.1 Design Patterns Description**

**Mediator Design Pattern**

Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling. Mediator pattern falls under behavioral pattern category.

The essence of the Mediator Pattern is to "define an object that encapsulates how a set of objects interact". It promotes loose coupling by keeping objects from referring to each other explicitly, and it allows their interaction to be varied independently. Client classes can use the mediator to send messages to other clients, and can receive messages from other clients via an event on the mediator class.

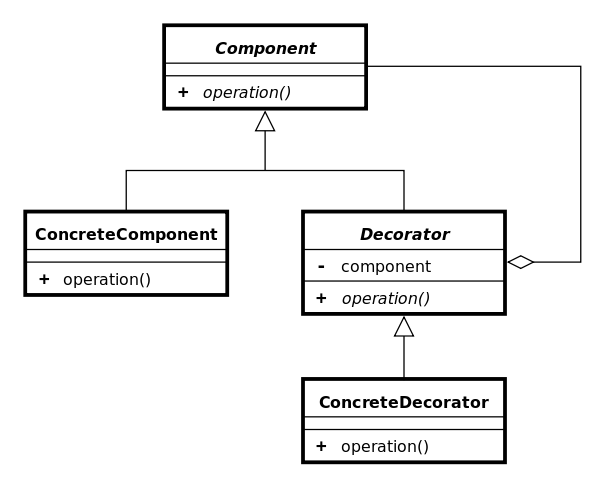


<https://commons.wikimedia.org/w/index.php?curid=1784483>

**Decorator Design Pattern**

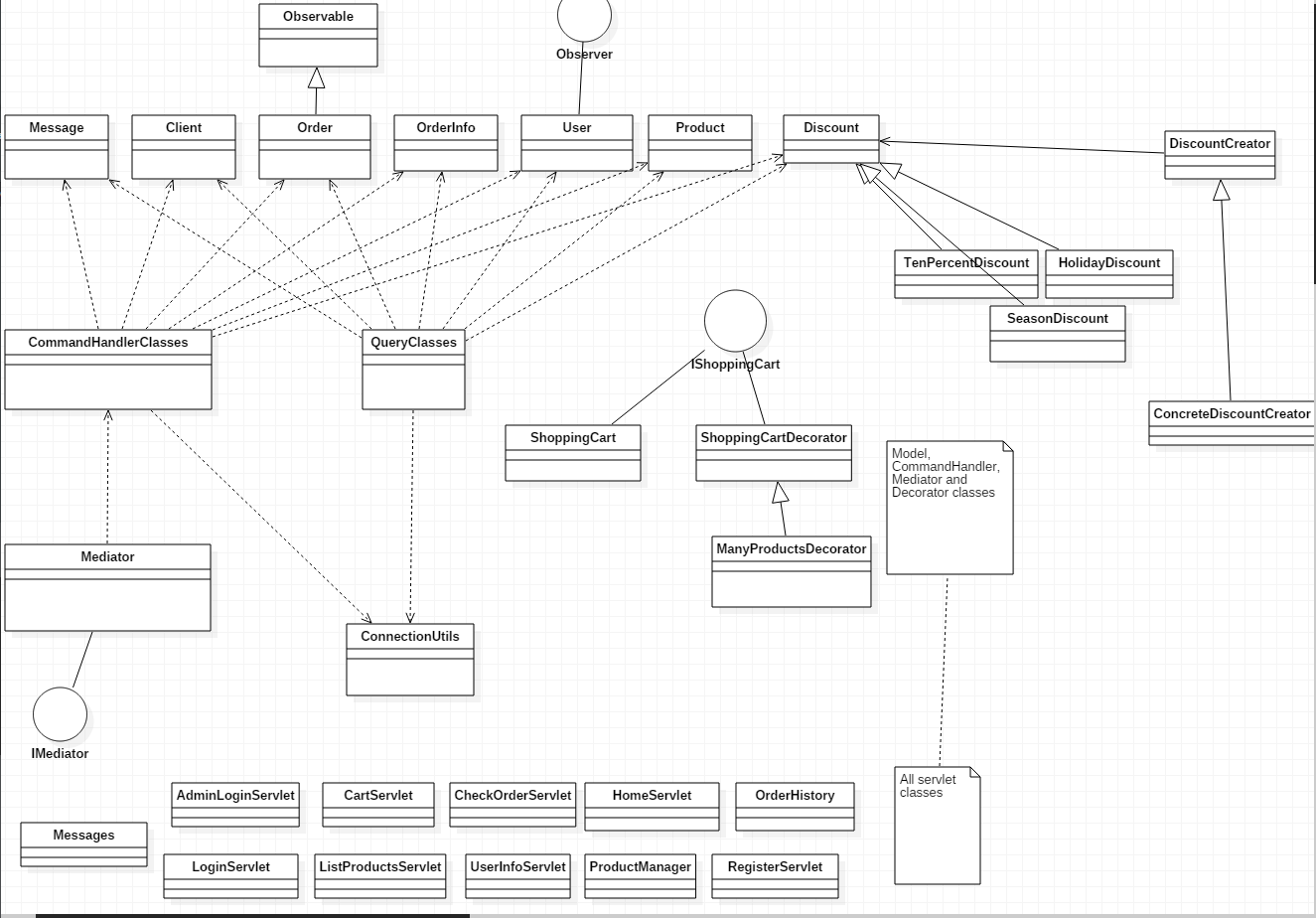
The **decorator pattern** is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class. The decorator pattern is often useful for adhering to the Single Responsibility Principle, as it allows functionality to be divided between classes with unique areas of concern. The decorator pattern is structurally nearly identical to the chain of responsibility pattern, the difference being that in a chain of responsibility, exactly one of the classes handles the request, while for the decorator, all classes handle the request.

This pattern is designed so that multiple decorators can be stacked on top of each other, each time adding a new functionality to the overridden method(s).



<https://commons.wikimedia.org/w/index.php?curid=1797712>

**5.2 UML Class Diagram**

**

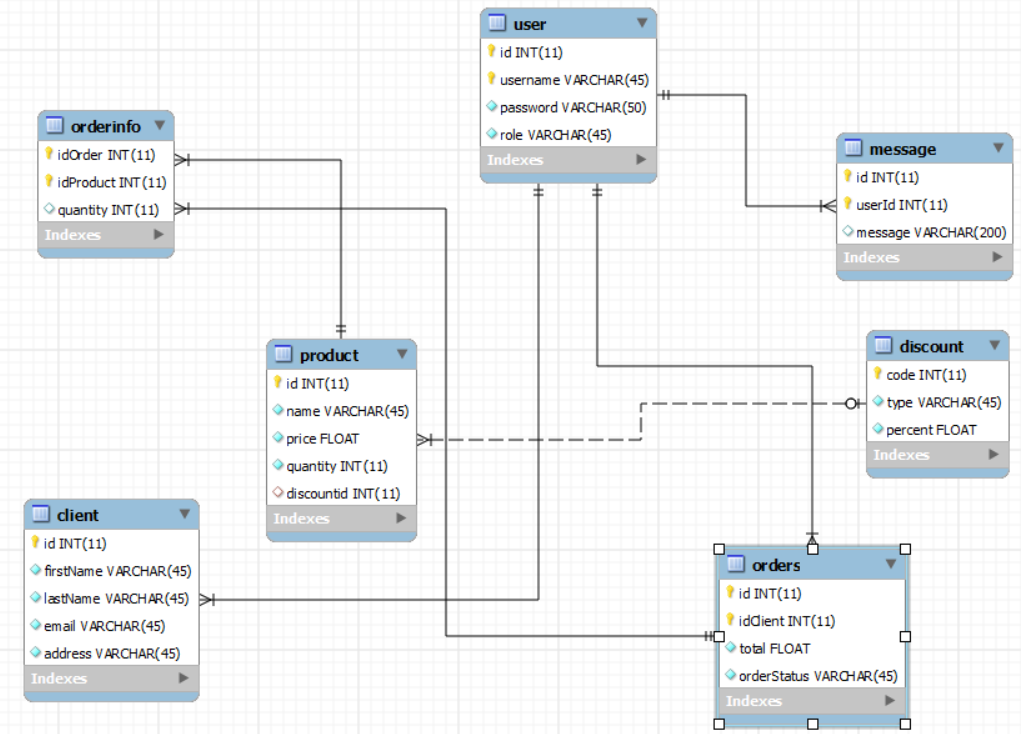
6. Data Model

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities.

The data models used in this application are: client, user, order, orderInfo, product, discount and message.

Each of these models have an associated table in the database used to store the needed information. Each attribute of these classes represents a column in the associated table.

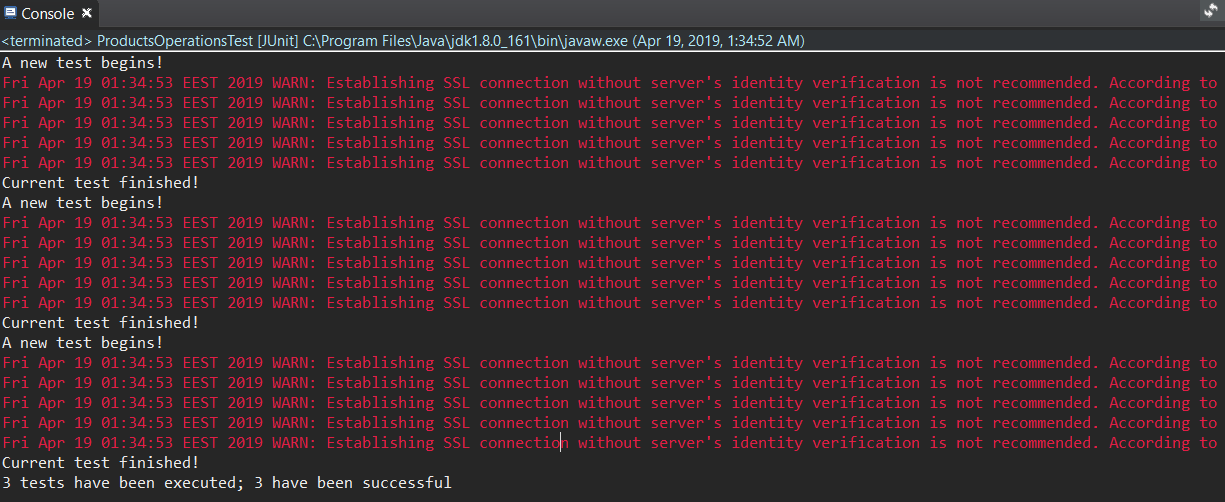
Database structure:



7. System Testing

All the functionalities were tested manually. Junit test methods were also written to test the functionality of products management (add product, update product and delete product). Three tests were executed and all three were successful.





8. Bibliography

<https://o7planning.org/en/11071/create-a-simple-login-application-and-secure-pages-with-java-servlet-filter#a11094140>

<https://www.journaldev.com/1907/java-session-management-servlet-httpsession-url-rewriting>

<https://www.journaldev.com/1854/java-web-application-tutorial-for-beginners>

<https://www.w3schools.com>

<https://en.wikipedia.org/wiki/Mediator_pattern>

<https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm>

<https://en.wikipedia.org/wiki/Decorator_pattern>

<https://martinfowler.com/bliki/CQRS.html>

<https://microservices.io/patterns/data/cqrs.html>

<https://www.codeproject.com/Articles/555855/Introduction-to-CQRS>

<https://www.future-processing.pl/blog/cqrs-simple-architecture/>

<https://creately.com/app/#>