Best Furniture Deals

Analysis and Design Document

Student: Tuturuga Nicolae

**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

[Application description]

Application title is Best Furniture Deals which should be a deal search engine for furniture products. Users should be able to create accounts, login and search for deals, they also should be able to filter deals by price, name and type and add associated product to their cart. They should be able to pay only with cash and post a feedback when order is completed. Their orders should be validated and updated by stuff.

# Functional Requirements

*[Present the functional requirements]*

# Normal users should be able to:

* search for furniture
* create account
* login
* add product to cart and checkout
* provide feedback

Staff should be able to:

* validate orders
* update order state
* manage deals
* manage furniture

# Non-functional Requirements

*[Discuss the non-functional requirements for the system]*

First requirement is to implement the application and test it as we build it, another requirement is to use an OOP language (I choose Java). Next requirements are about design, first of them is to use a client server architecture, another one consist of implementing observer design pattern to notify user when an order state is updated. And another one is to validate all the inputs of the application.

2. Use-Case Model

*[Create the use-case diagrams and provide one use-case description (according to the format below).*

*Use-Case description format:*

*Use case: <use case goal>*

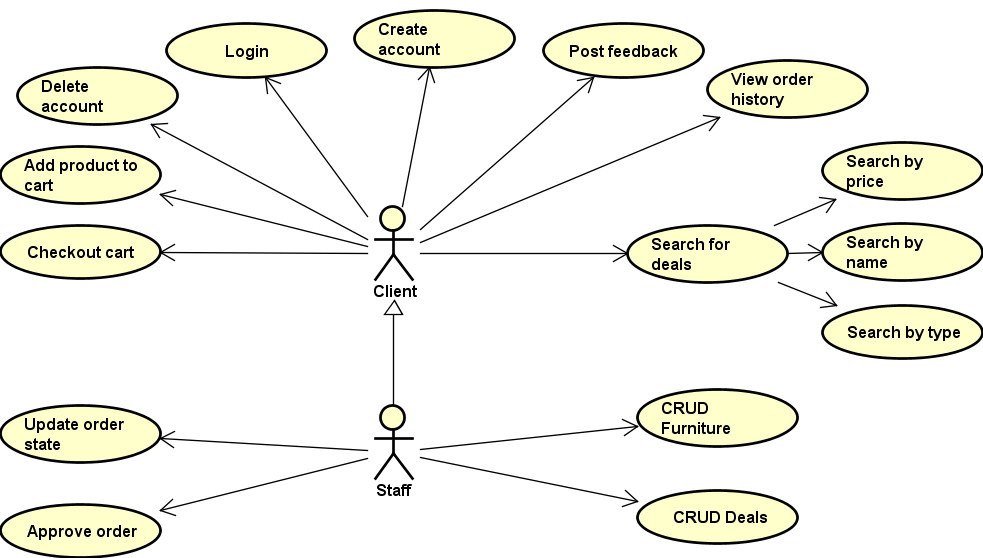
*Level: <one of: summary level, user-goal level, sub-function>*

*Primary actor: <a role name for the actor who initiates the use case>*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*

*]*



**Use case description:**

**Create account:**

**Use case goal:**

The purpose of create account is to register a new account into the the application, by saving the data into the database.

**Success:**

Primary actor is the client who wants to register. This use case begins when client wants to signup into the app.

1. Client requests for a register form.
2. Server redirects him to one.
3. Client inserts his data.
4. Server verifies inserted data to be valid.
5. If inserted data is correct user is inserted.

**Error:**

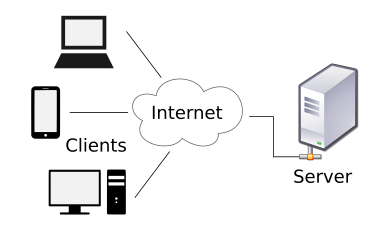
If inserted data of an user is invalid when he tries to login, he will be redirected to the same register form, but he will see an error message.

3. System Architectural Design

**3.1 Architectural Pattern Description**

*[Describe briefly the used architectural patterns.]*

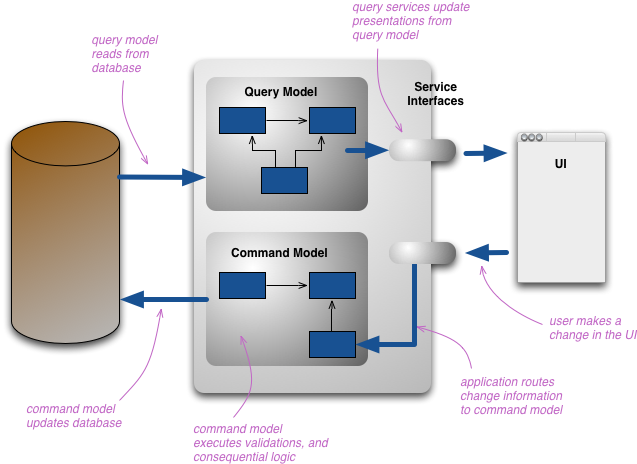
Client–server model is a [distributed application](https://en.wikipedia.org/wiki/Distributed_application) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](https://en.wikipedia.org/wiki/Server_(computing)), and service requesters, called [clients](https://en.wikipedia.org/wiki/Client_(computing)).



CQRS stands for Command Query Responsibility Segregation. It's a pattern that I first heard described by [Greg Young](https://twitter.com/gregyoung). At its heart is the notion that you can use a different model to update information than the model you use to read information. For some situations, this separation can be valuable, but beware that for most systems CQRS adds risky complexity.

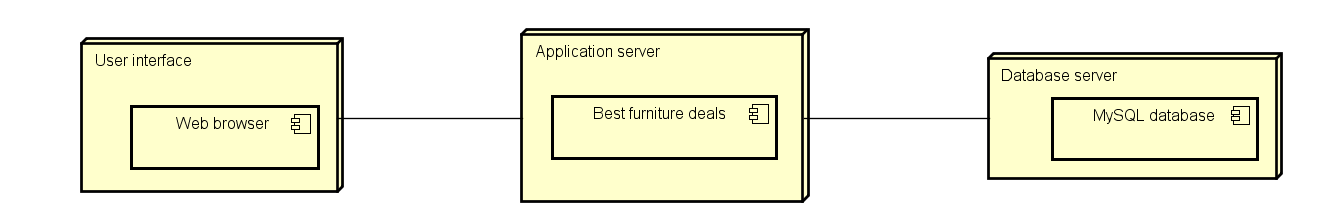
The mainstream approach people use for interacting with an information system is to treat it as a CRUD datastore. By this I mean that we have mental model of some record structure where we cancreate new records, read records, update existing records, and delete records when we're done with them. In the simplest case, our interactions are all about storing and retrieving these records.

The change that CQRS introduces is to split that conceptual model into separate models for update and display, which it refers to as Command and Query respectively following the vocabulary of [CommandQuerySeparation](https://martinfowler.com/bliki/CommandQuerySeparation.html). The rationale is that for many problems, particularly in more complicated domains, having the same conceptual model for commands and queries leads to a more complex model that does neither well.

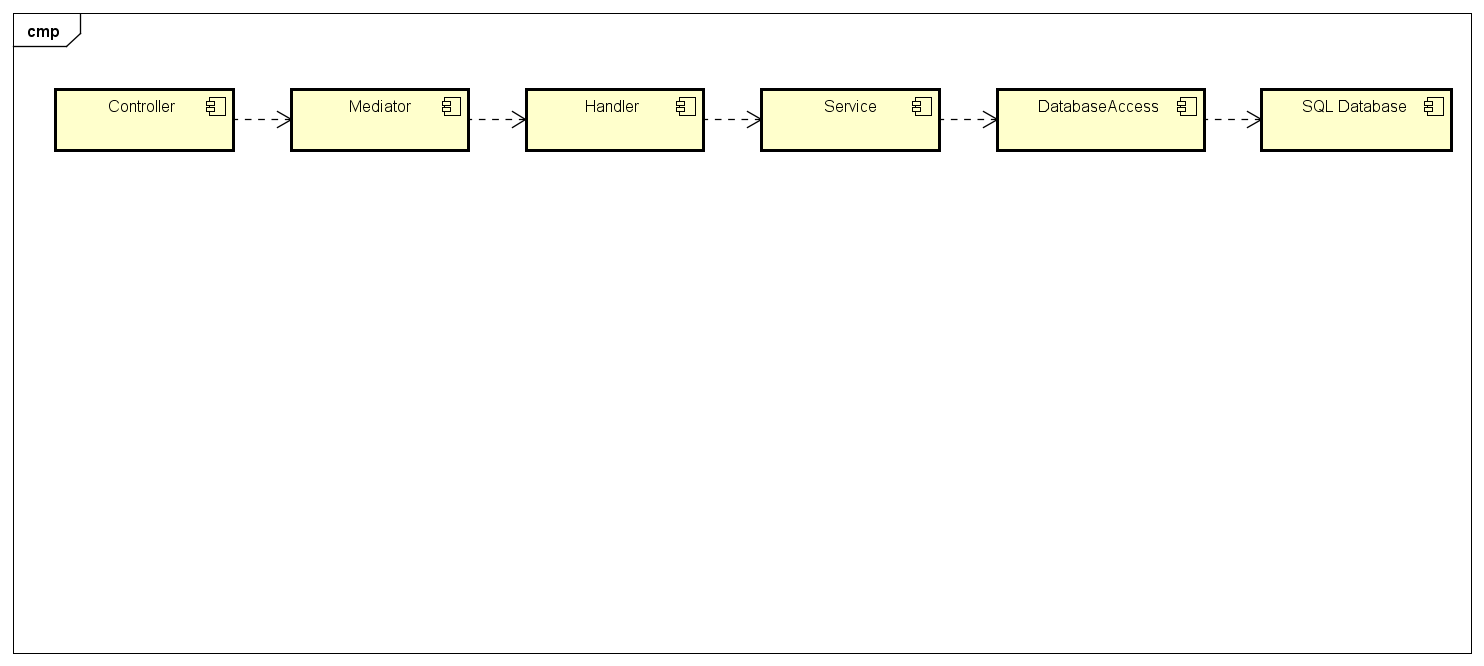


**3.2 Diagrams**

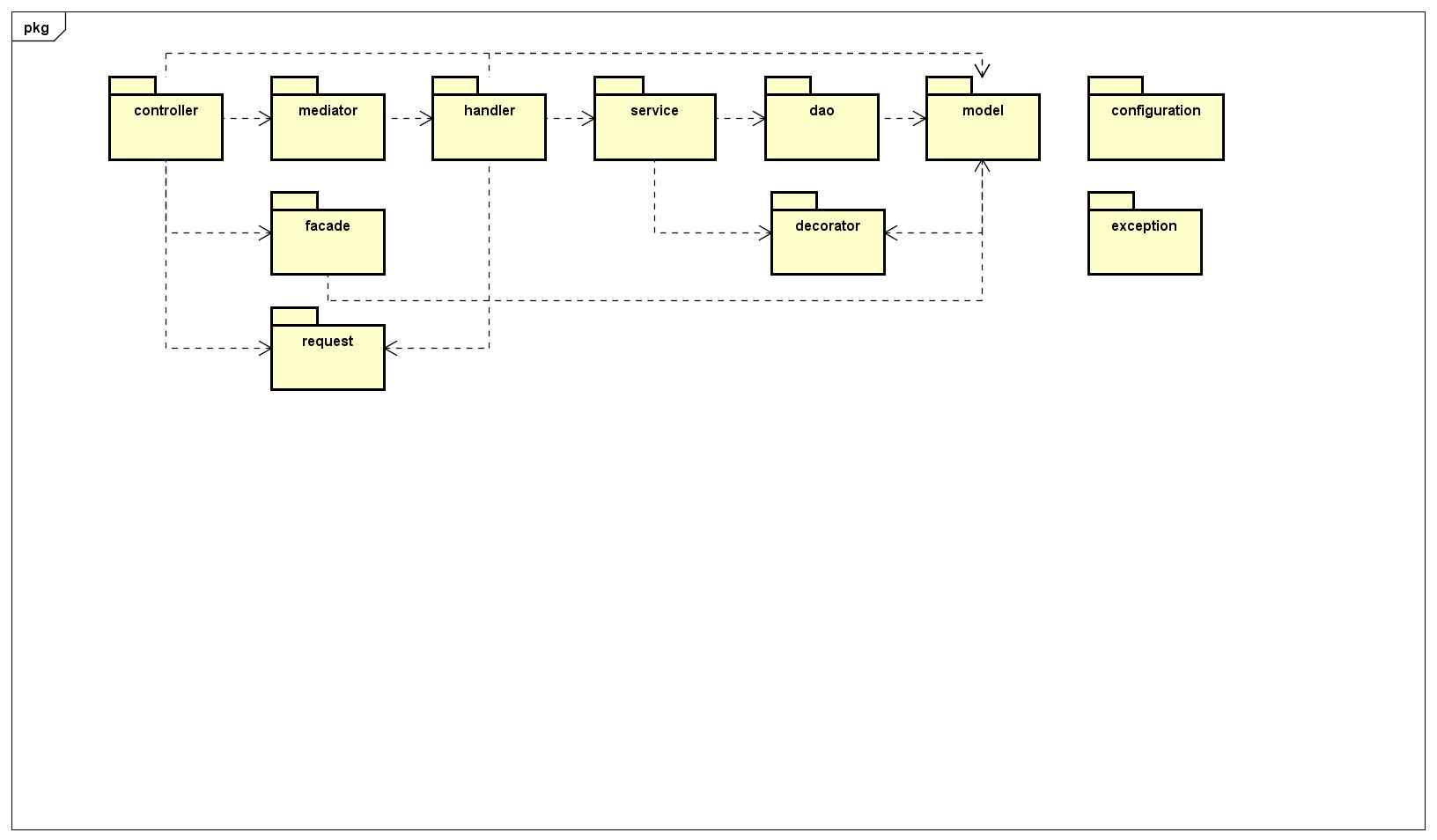
*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

**

**Component :**

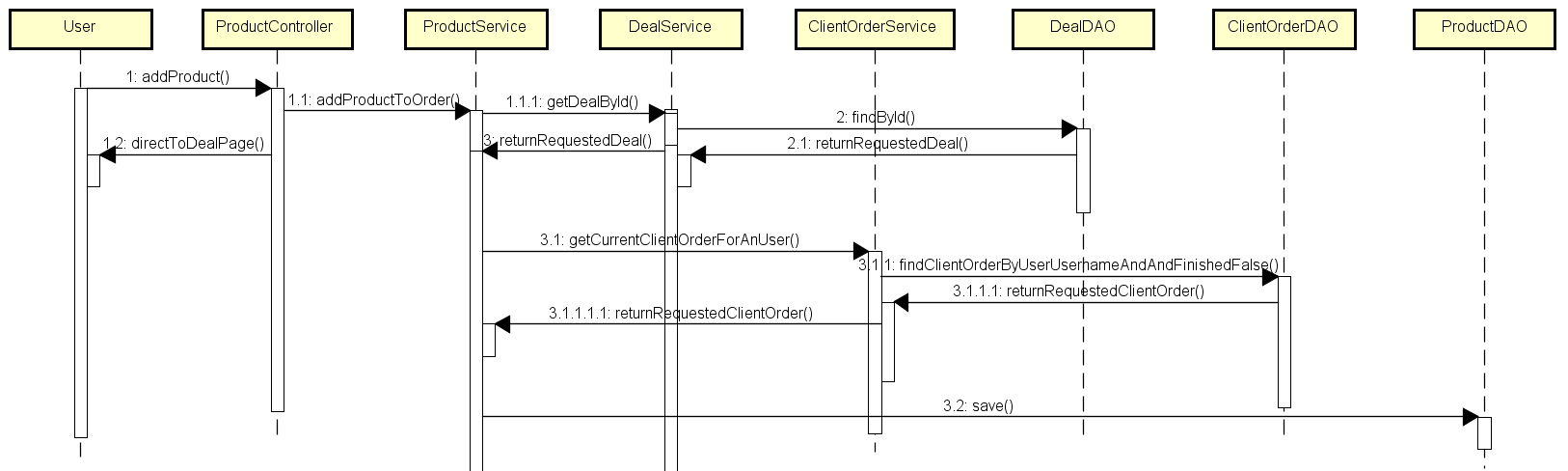
**

**Package diagram :**

****

4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*



5. Class Design

**5.1 Design Patterns Description**

*[Describe briefly the used design patterns.]*

Client server architecture is represented by cqrs architecture. And cqrs is exemplified by the separations of dto models, services and controller for query and command dtos, services and controllers for commands.

Table data gateway, usage of this pattern is exemplified by DAO classes, like UserDAO, DealDAO etc.

Observer design pattern is exemplified by the model abstract classes Observer and Observable and by ObserverService.

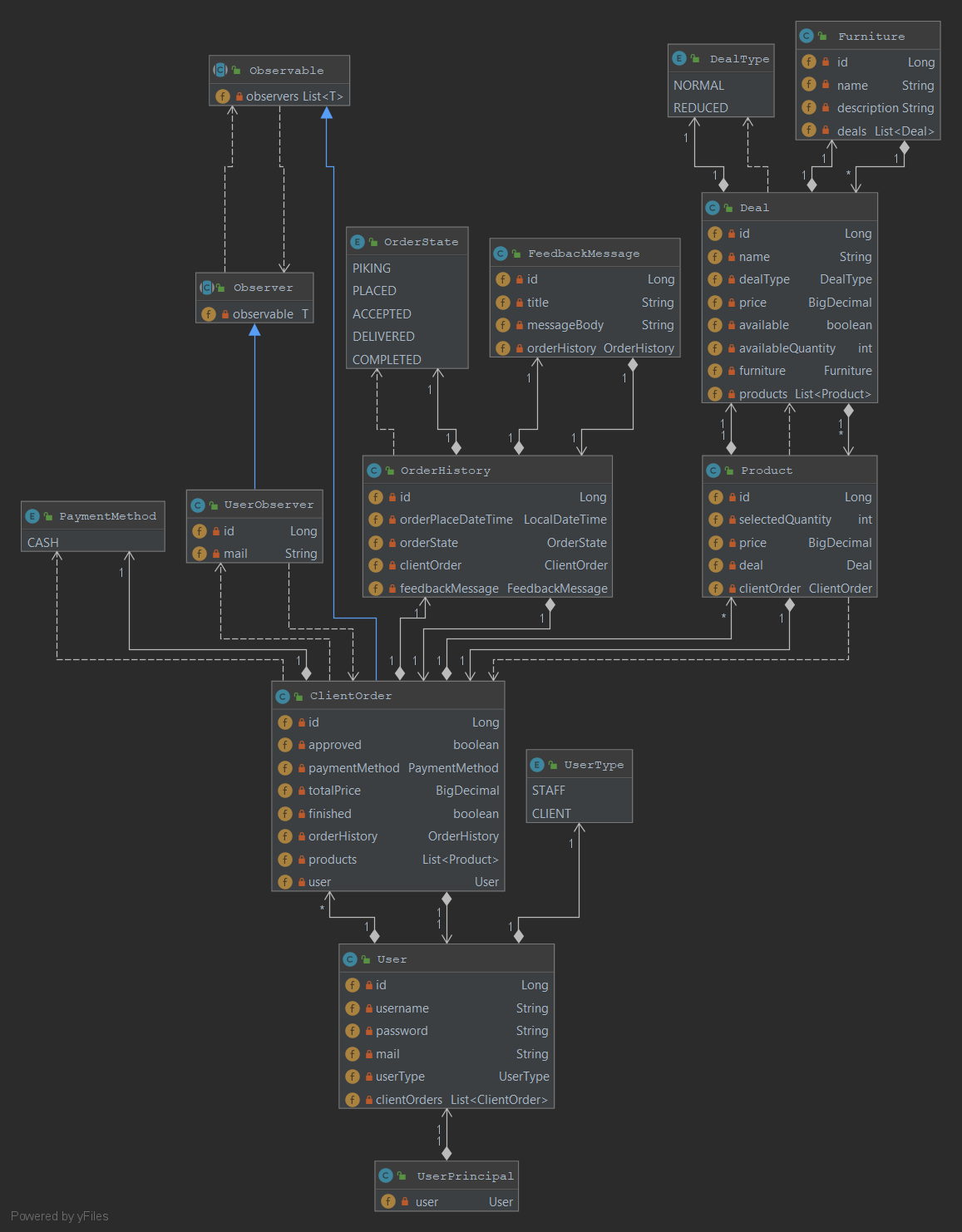
Facade design pattern is exemplified in facade package which is used to encapsulate the logic for converting the dto to a model entity class call wanted service, and convert obtained model if exists in a dto that controller can send to view.

Decorator pattern is used to create a Product object based on his deal, reduced or normal.

Mediator pattern is used to catch request sent by controllers and handles them based on their type.

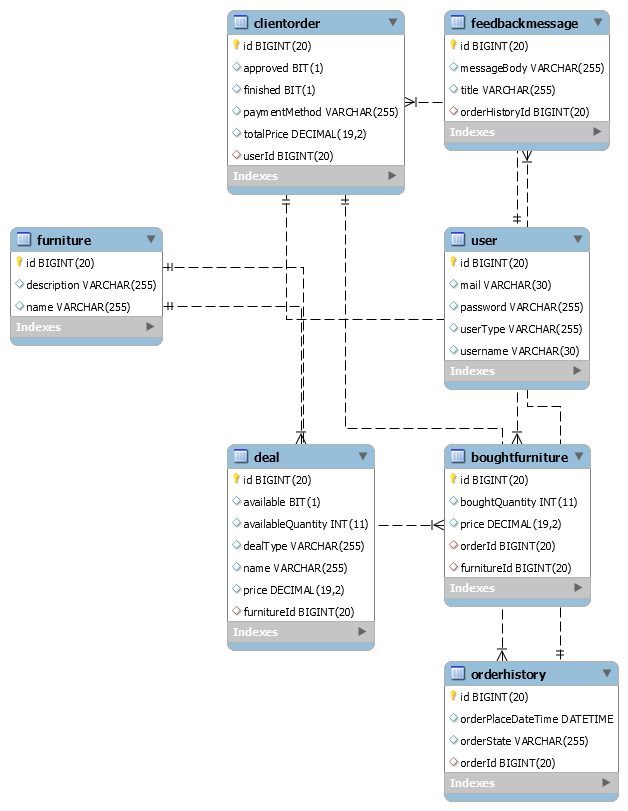
**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*



6. Data Model

*[Present the data models used in the system’s implementation.*



Is represented by model package, it consists of classes Product, ClientOrder, Deal, FeedbackMessage, Furniture, OrderHistory, User, where each attribute of class corresponds with a column in the table.

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

To test my application, I wrote java tests using an in memory database with preinserted data by a SQL script I wrote and spring boot testing functionalities to load that data and context, I used integration test to test service and dao layers and integration tests for others, I also wrote tests that check corner cases. Another way I used to test my application was with the help of GUI I implemented.

8. Bibliography

<https://martinfowler.com/eaaCatalog/tableDataGateway.html>

<https://creately.com/blog/diagrams/deployment-diagram-tutorial/>

<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

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

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

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