Assignment 3

Analysis and Design Document

Student: Vadean Andrada-Anastasia

**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]

I was tasked to build a deal search engine for furniture products. The user should be able to create an account and login to search for various provided deals. The deals must be managed by staff and can be filtered by price, name and type. If a deal is available an user can add the associated product to his cart and proceed to checkout.

Payments can be done via a cash only policy and need to be validated by staff. This creates an order in the system that can be tracked by the user from the Order History section. The state of an order is updated by staff.

Once an order is delivered the user can provide feedback in a form on the specific Order History entry details.

# Functional Requirements

*[Present the functional requirements]*

I have two kinds of users: normal user and staff.

A user can: login, create an account, search for furniture and filtered them, see their orders history and add products to cart and checkout.

A staff should be able to: login, create account, see all the products, change the price and the quantity of a product, delete a product and add a product. Also, he can validate orders.

# Non-functional Requirements

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

I have some non-functional requirements like using a OOP language (Java), implement and test the application or commit my work in my Git repository. We have to use a client-server architecture and store all the data in a database. All the inputs of the application have to be validated. Another requirement is to use a observer pattern for getting notification when the state of an order changes.

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>*

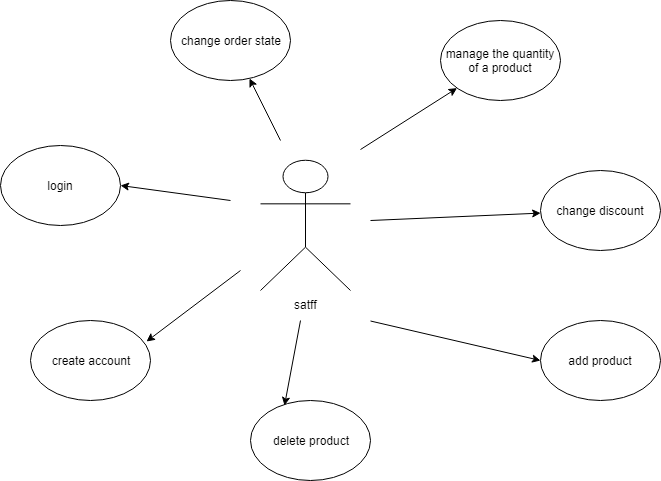
*Level: <one of: summary level, user-goal level, sub-function>*

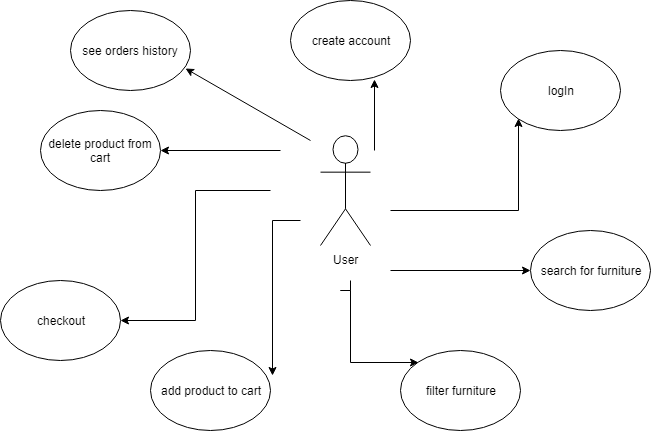
*Primary actor: <a role name for the actor who initiates the use case>*

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

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

*]*

****



Use case: Login user

Level: user-goal level

Primary actor: user

Main success scenario: the user enters his data and he can log in because he has an account

Extensions: the user doesn’t have an account and he can’t login

3. System Architectural Design

**3.1 Architectural Pattern Description**

*[Describe briefly the used architectural patterns.]*

An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

I organized the system using a client server architecture. Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. The client-server architecture is also termed as a network-computing structure because every request and their associated services are distributed over a network.

The observer pattern is a [software design pattern](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) in which an [object](https://en.wikipedia.org/wiki/Object_(computer_science)#Objects_in_object-oriented_programming), called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their [methods](https://en.wikipedia.org/wiki/Method_(computer_science)).

The Observer pattern addresses the following problems:

* A one-to-many dependency between objects should be defined without making the objects tightly coupled.
* It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
* It should be possible that one object can notify an open-ended number of other objects

The Mediator Pattern, which is similar to the [Command](https://springframework.guru/command-pattern/), [Chain of Responsibility](https://springframework.guru/chain-of-responsibility-pattern/), and [Iterator](https://springframework.guru/iterator-pattern/) patterns, are part of the Behavioral pattern family of the [Gang of Four](http://springframework.guru/gang-of-four-design-patterns/) design patterns. Behavioral patterns address responsibilities of objects in an application and how they communicate between them. The Mediator pattern allows the loose coupling between a set of objects in an application by handling the interactions between the objects.

The Mediator pattern says that instead of allowing a set of objects to directly interact with them, define an object (mediator) that will handle the interactions.

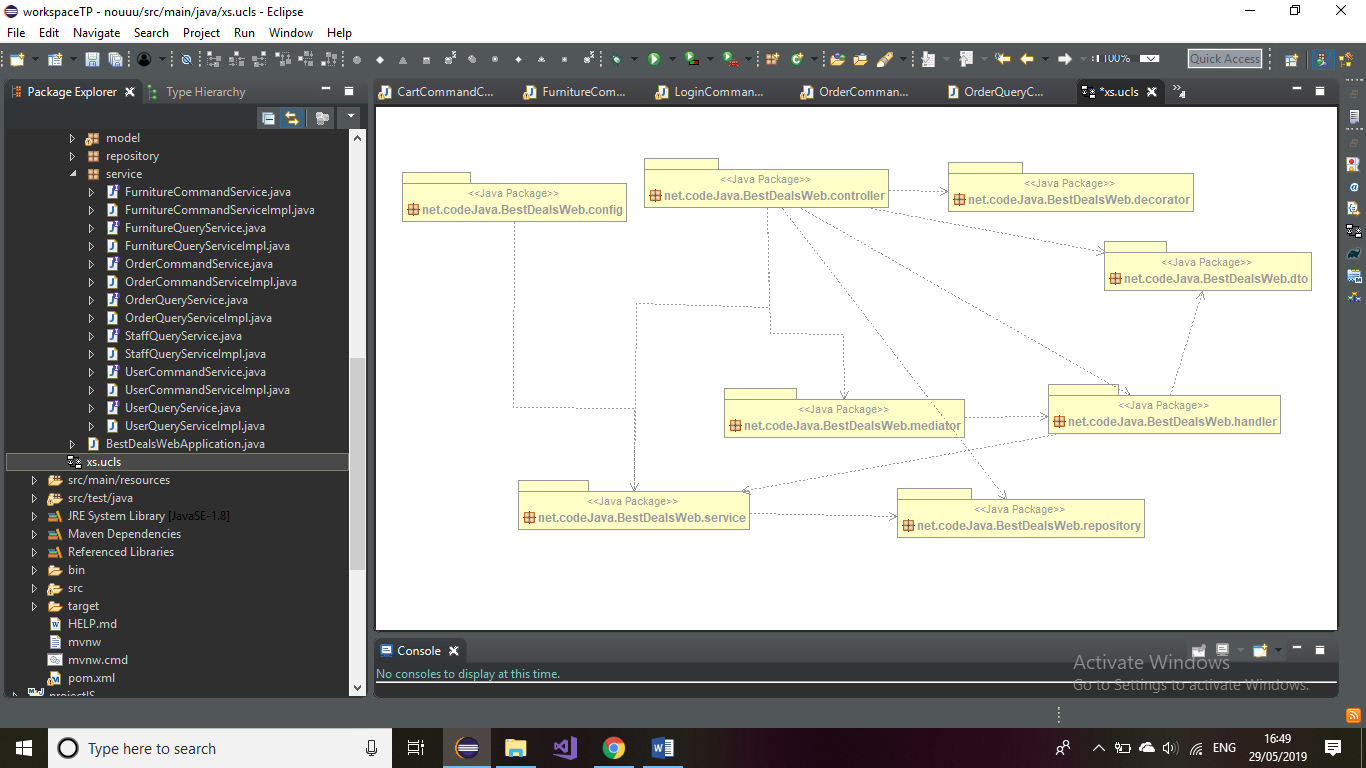
Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

**3.2 Diagrams**

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

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.]*

Like I said before, I used a client server architecture that is represented by MVC architecture.

Observer pattern is used to notify the user that the state of his order has change. For implement this pattern I used two interfaces Observer and Observable. The User implements the Observer Interface and the Order class implements the Observable Interface. So, when an update is done for the order state the user will be notify by sending an email to his email address.

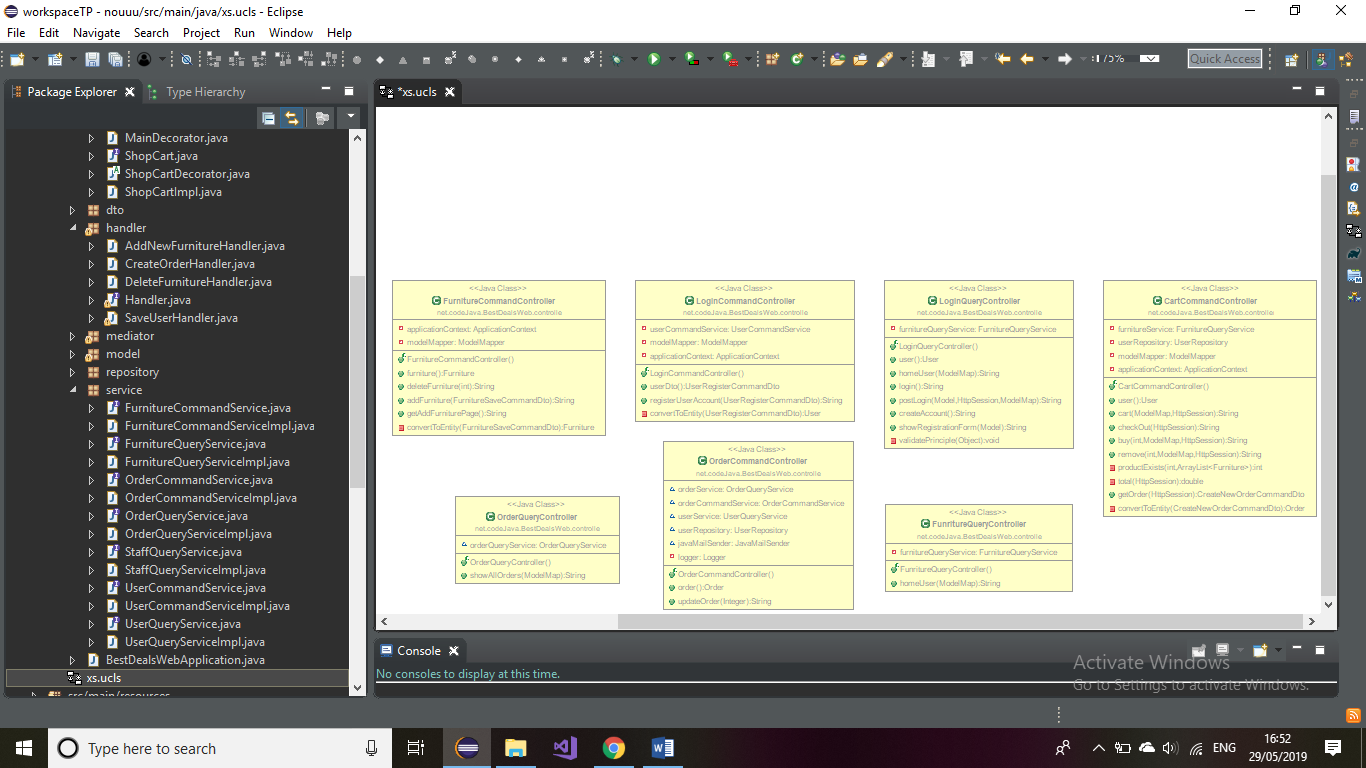
The architecture of this project is split I two parts. We have two types of services and controllers: a type used for writing operation (such as save or update) and the other type for reading operation (such as find or findAll).

I used the decorator pattern for applying deals to the shopping cart.

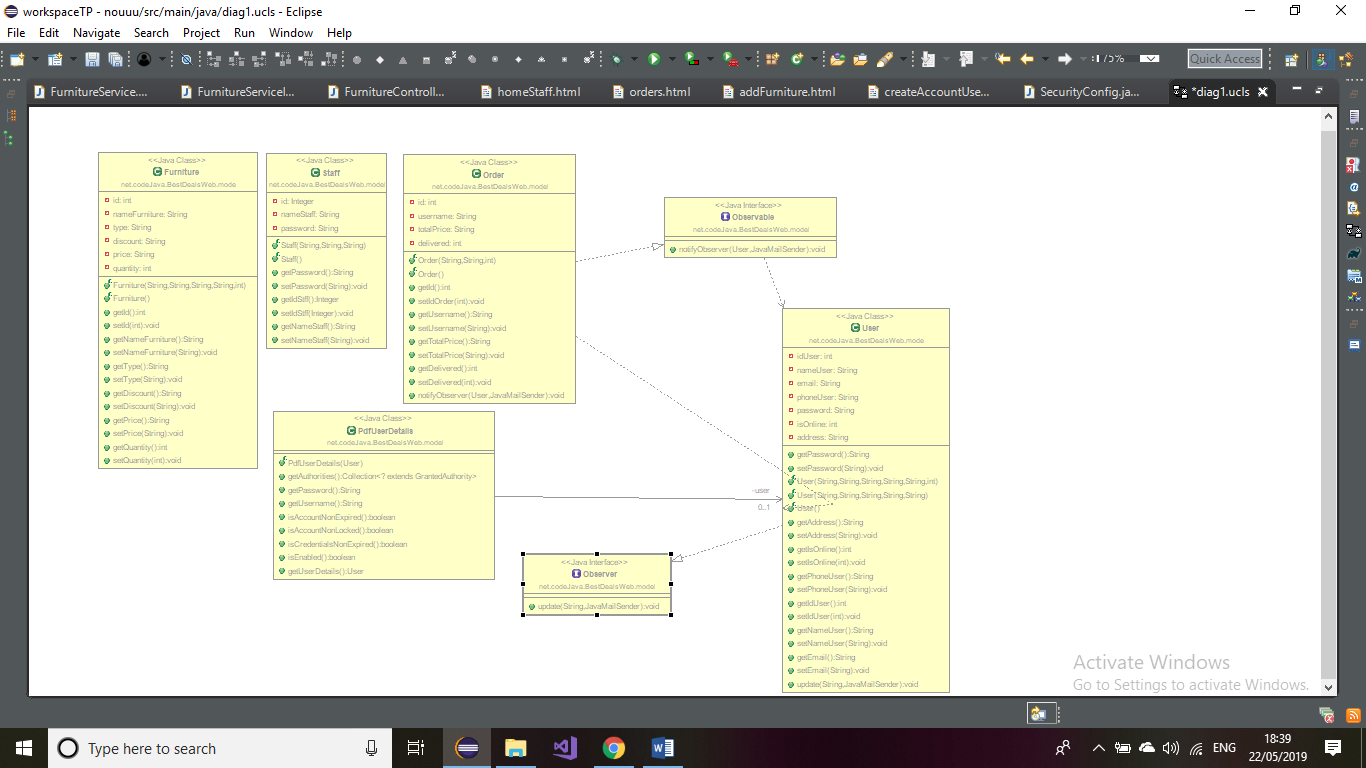
**5.2 UML Class Diagram**

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

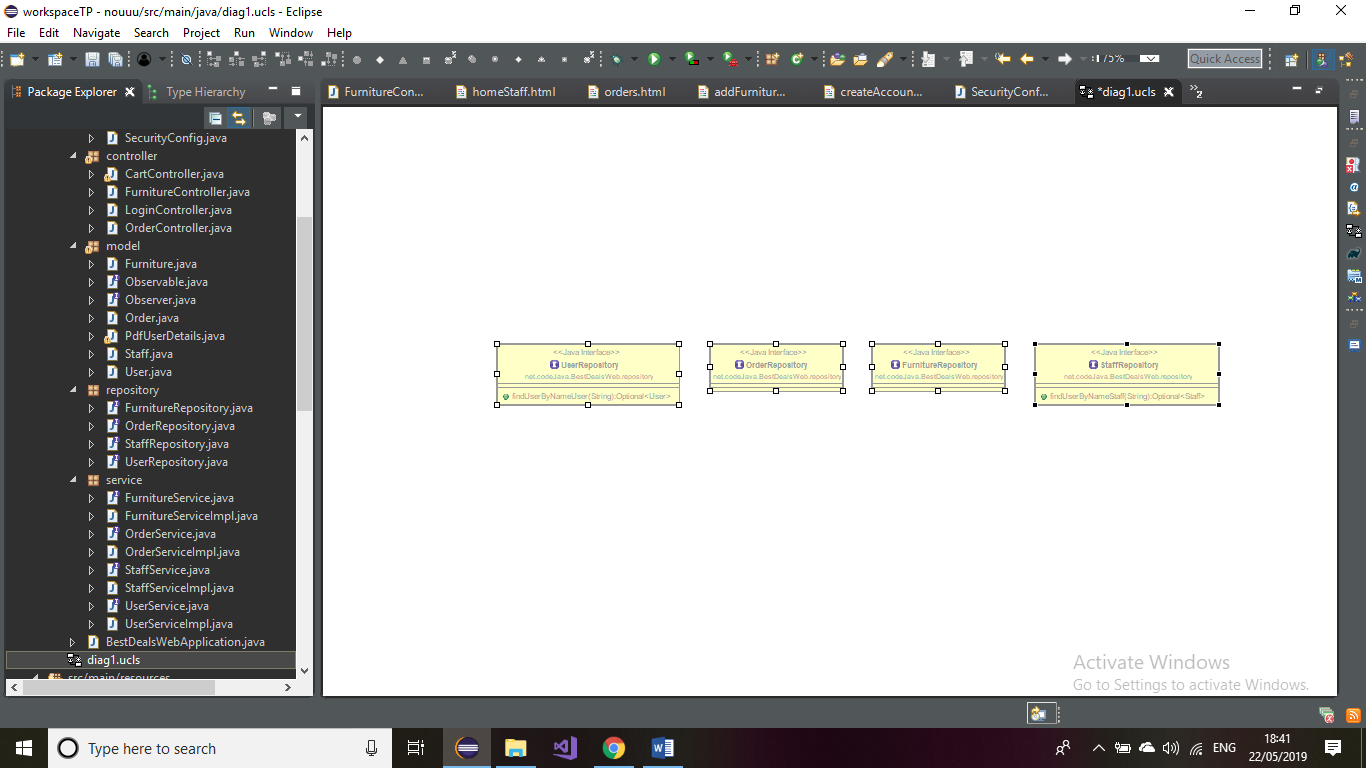
controller package:



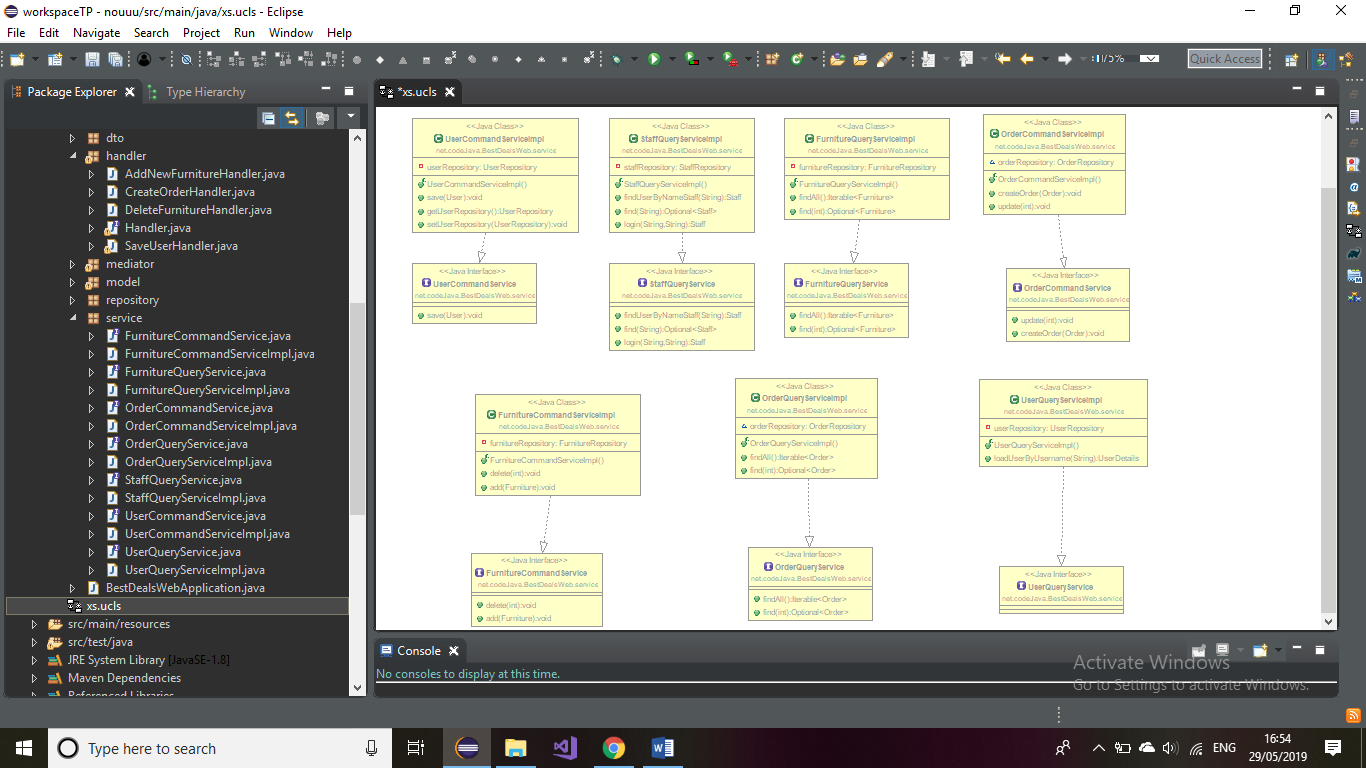
model package:



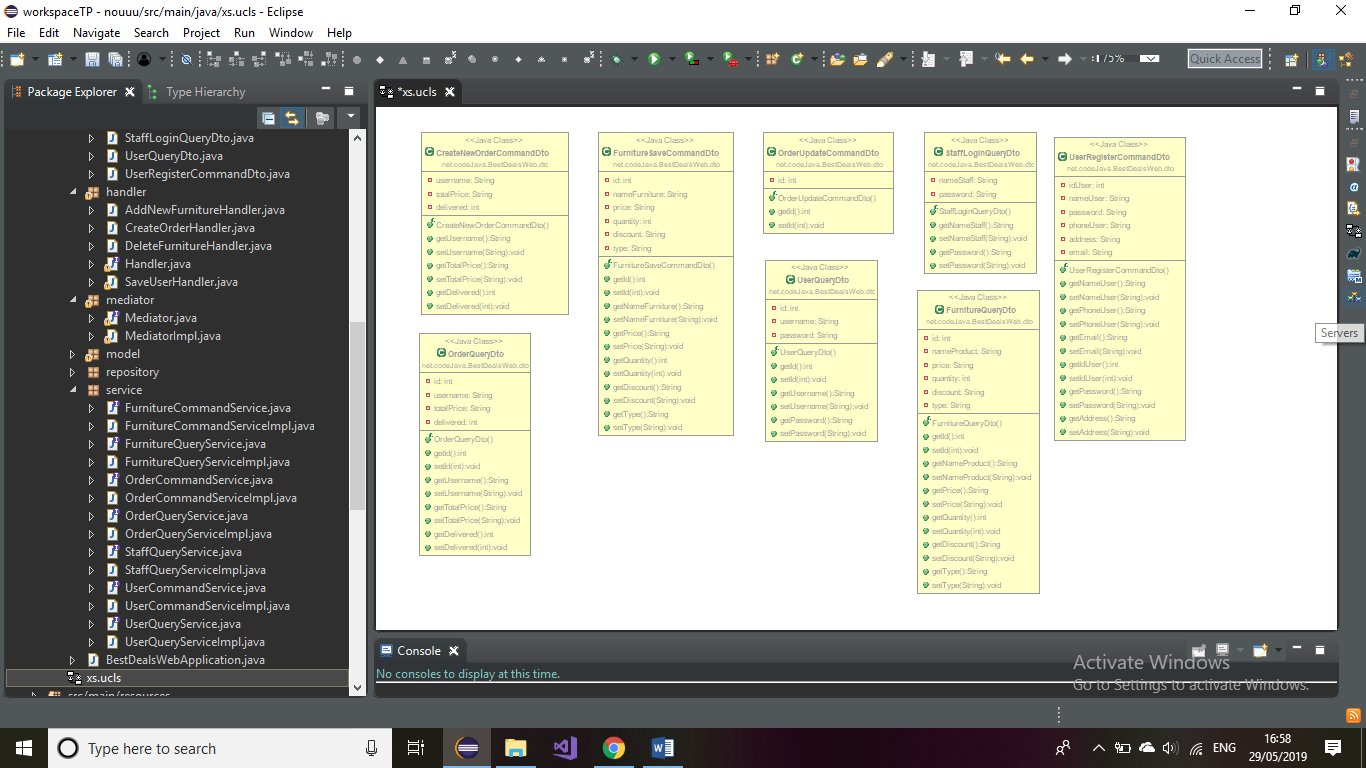
repository package:



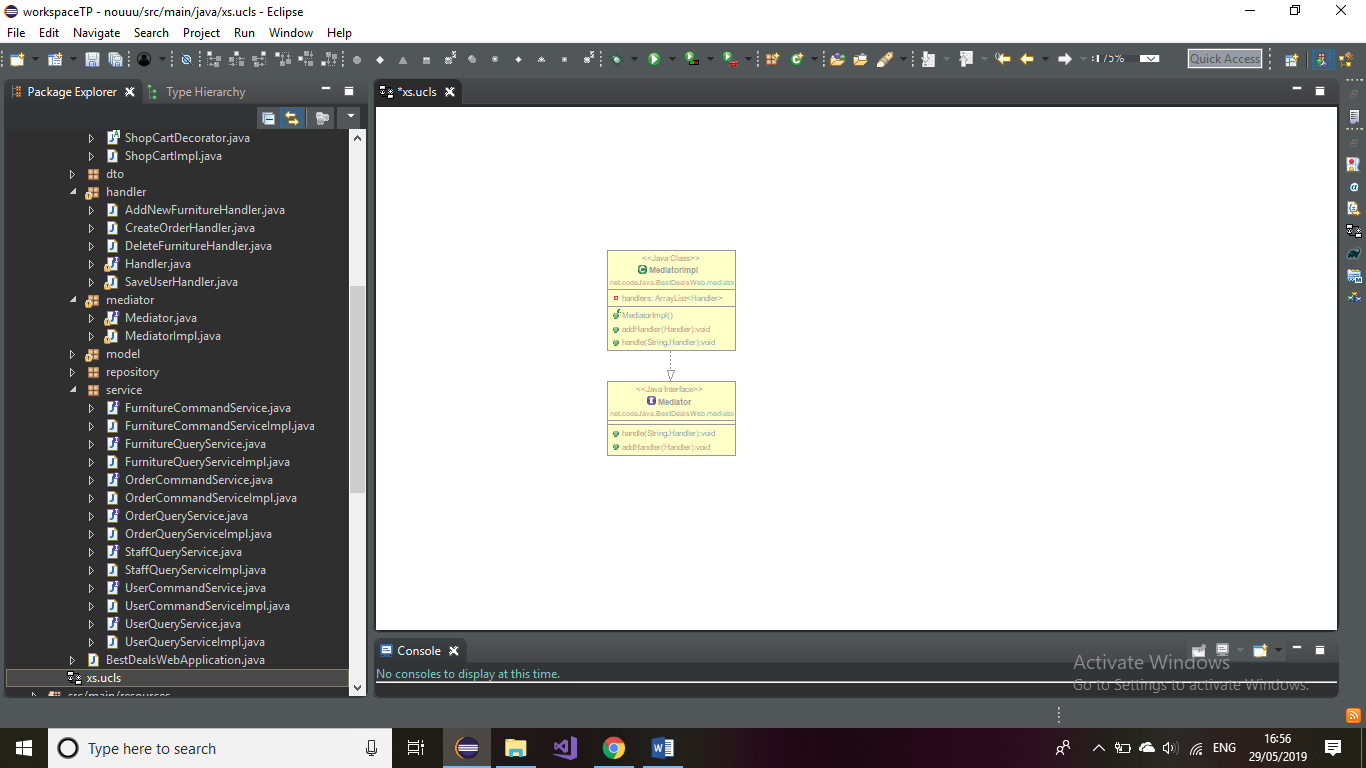
service package:



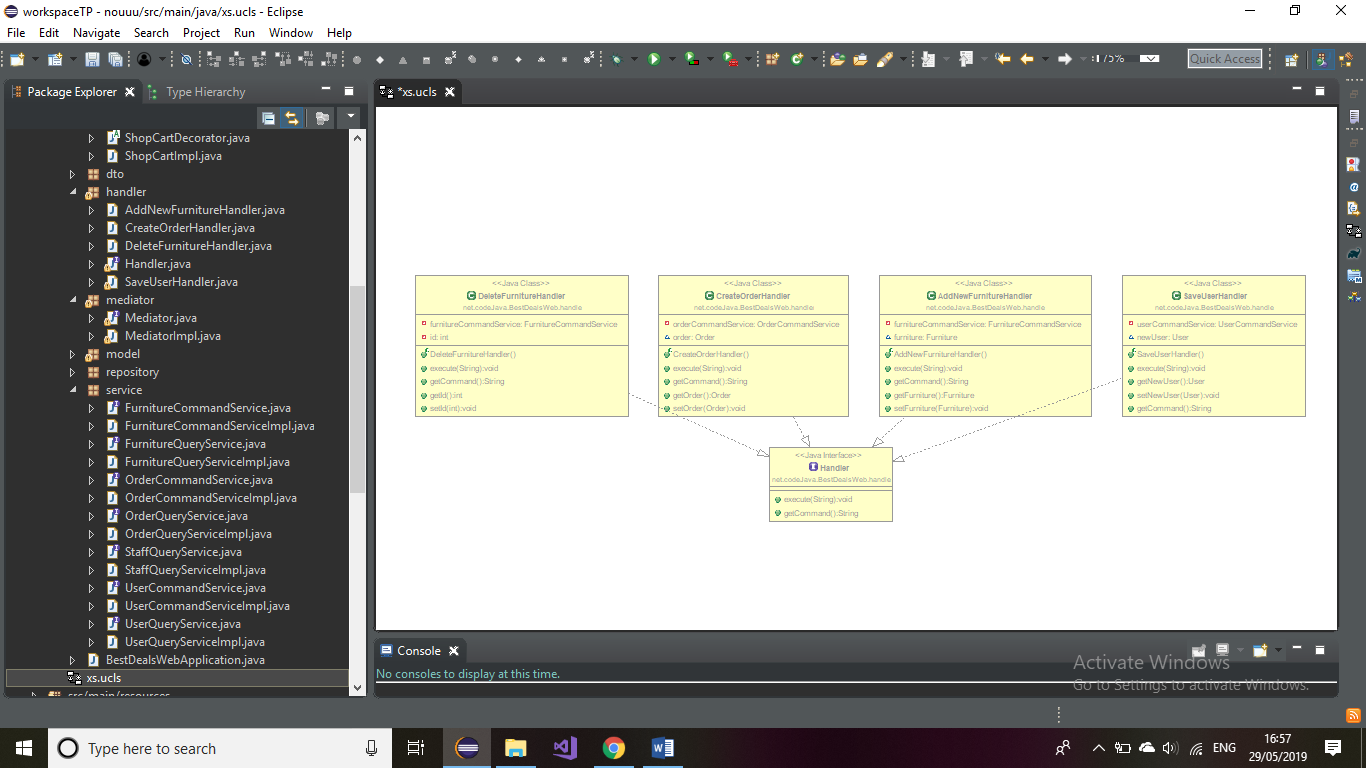
dto package:



mediator package



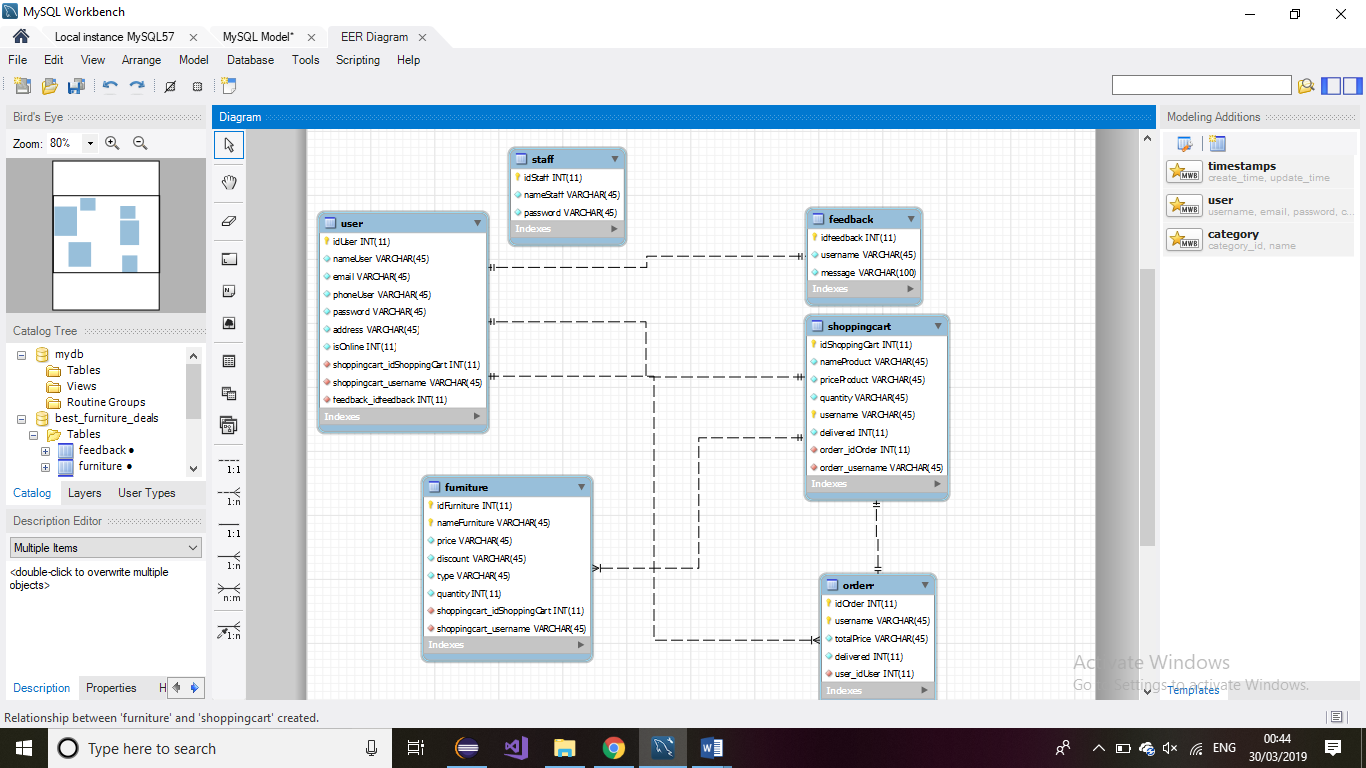
handler pakage:



6. Data Model

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

Data Model is representing in the package model, by the classes User, Staff, Furniture and Order which will be the objects used for implementing the system. Each class contain the principal attributes, which correspond with the column of the tables in database. For example, the class User has idUser, nameUser, address, password, email, isOnline and phoneUser.



7. System Testing

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

I tested my application using java test class. I use TestEntityMager to have access at records already setup in my database. The TestEntityManager provided by Spring Boot is an alternative to the standard JPA EntityManager that provides methods commonly used when writing tests. Another way to test my application was with my html pages.

8. Bibliography

<https://stackoverflow.com/>

<https://www.baeldung.com/spring-boot-testing>

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