Best Furniture Deals

Analysis and Design Document

Student: Păcurar Aralda

**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 4

4. UML Sequence Diagrams 5

5. Class Design 5

6. Data Model 6

7. System Testing 7

8. Bibliography 7

1. Requirements Analysis

# Assignment Specification

The application is a search engine for furniture products. A user is able to create an account and login to search for various provided deals. Deals are managed by staff and can be filtered by price, name and type. 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

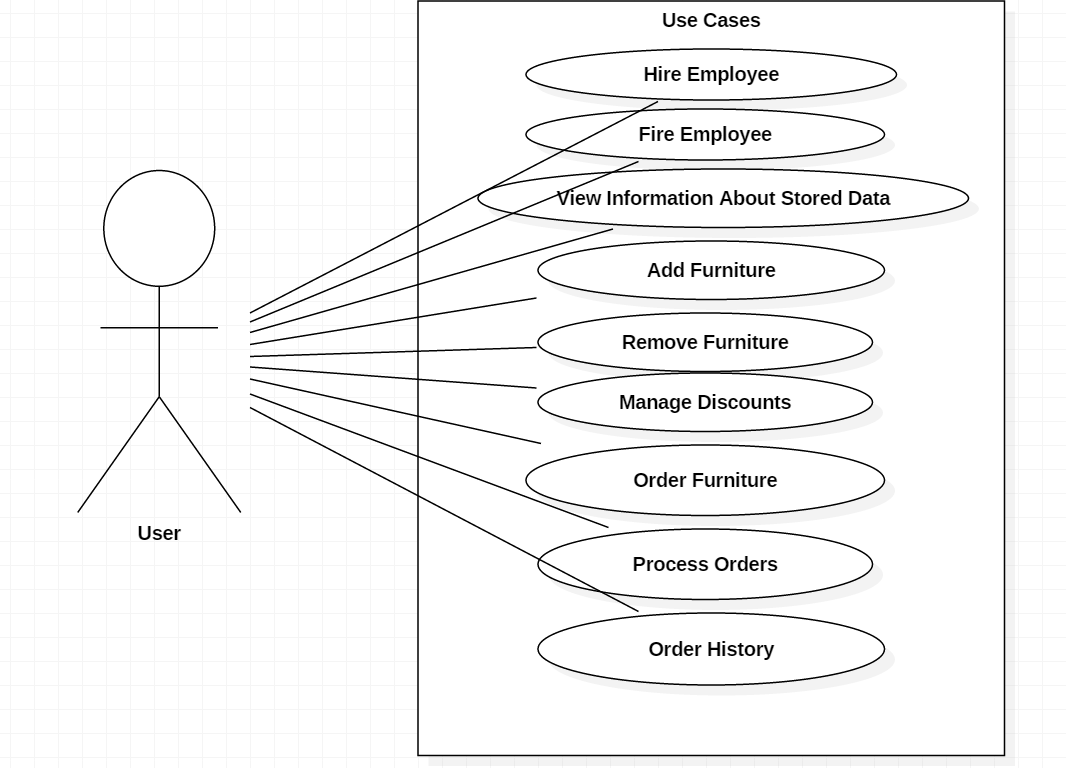
In order to use the application, every user is supposed to create an account and login. The application has three types of users, each one of them having a specific set of rights. The administrator has the rights to perform any operation, employees can manage products, discounts and orders as well as view and order items while customers can view and order products, check their order history and give feedback on the products they bought.

The application allows hireing and fireing employees which implies inserting and removing them from the database. Users, roles and the furniture can also be listed. Furniture can be added and removed from the system. Discounts can also be managed by adding or removing a 10% or 20% price change. This also modifies the product’s price in the database. Every type of user can place orders and view their order history as well as leave feedback to the products they bought and received. Staff members can view and manage the orders, updating their tracking information.

# Non-functional Requirements

The “Best Furniture Deals” application has a good response time, it’s a maintainable application that can be updated and upgraded any time. The information provided by the users is safely stored in a database, passwords are hashed using the SHA-256 algorithm.

2. Use-Case Model

**

Use case: Hire Employee

Level: sub-function

Primary actor: admin

Main success scenario:

* Admin clicks Hire Employee from the menu
* Admin enters the new employee’s information
* Admin clicks Hire Employee
* The entered data is valid and the new employee is registered and saved in the database

Extensions:

* The data entered by the admin is not valid and the new employee isn’t registered or saved in the database

3. System Architectural Design

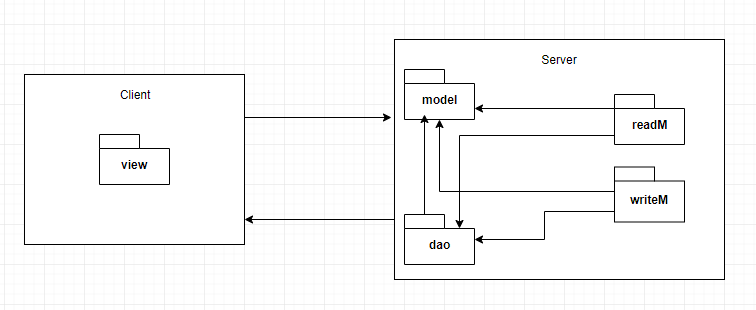
**3.1 Architectural Pattern Description**

The 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. This type of architecture has one or more client computers connected to a central server over a network or internet connection.

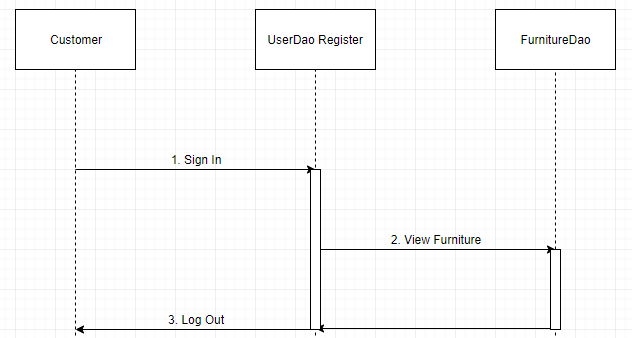
Client/server architecture is a producer/consumer computing architecture where the server acts as the producer and the client as a consumer. The server houses and provides high-end, computing-intensive services to the client on demand.

**3.2 Diagrams**

* Package diagram



4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

* **Factory Method**

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. The Factory Method pattern suggests that you replace direct object construction calls (using new operator) with calls to a special factory method. The objects are still created via the new operator, but it is being called from within the factory method. Objects returned by a factory method are often referred to as “products”.

* **Builder**

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code. The builder pattern suggests that you extract the object construction code out of its own class and move it to separate objects called builders.

* **Observer**

Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing. The observer pattern suggests that you add a subscription mechanism to the subject class so individual objects can subscribe to or unsubscribe from a stream of events coming from that subject.

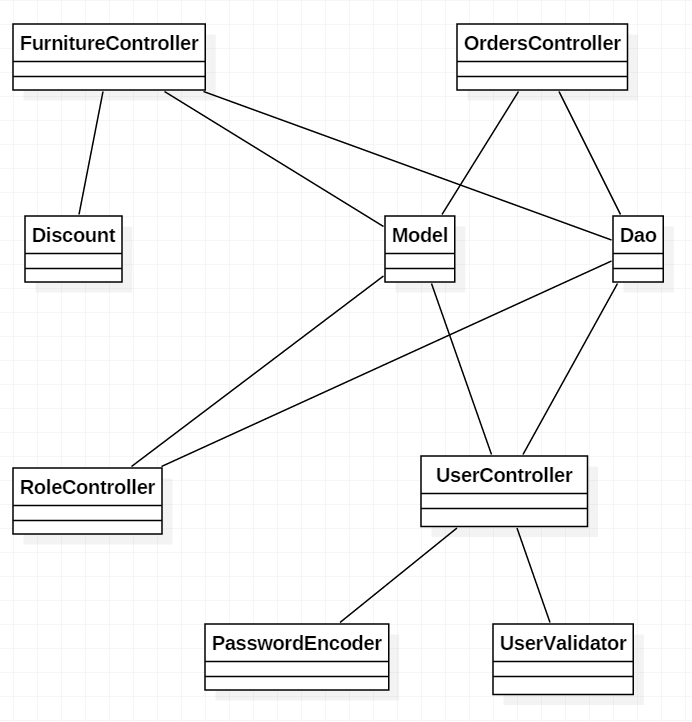
* **Decorator**

Decorator is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors. Composition is the key principle behind many design patterns, including the Decorator. Wrapper is the alternative nickname for the Decorator pattern that clearly expresses the main idea of the pattern. A “wrapper” is an object that can be linked with some target object. The wrapper contains the same set of methods as the target and delegates to it all the requests that it receives. However, the wrapper may alter the result by doing something either before or after it passes the request to the target.

* **Mediator**

Mediator is a behavioral design pattern that lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object. The Mediator pattern suggests that you should cease all direct communication between the components which you want to make independent of each other. Instead, these components must collaborate indirectly, by calling a special mediator object that redirects the calls to appropriate components. As a result, the components depend only on a single mediator class instead of being coupled to dozen of their collegues.

**5.2 UML Class Diagram**



6. Data Model

The data models used in the system’s implementation are:

* Furniture
* Id
* Description
* Name
* Original\_price
* Price
* Furniture\_type\_id
* Furniture\_type
* Id
* Name
* Orders
* Id
* Feedback
* Furniture\_name
* Status
* Username
* Furniture\_id
* User\_id
* Rights
* Id
* My\_right
* Role
* Id
* Role
* Role\_rights
* Roles\_id
* Rights\_id
* User
* Id
* Address
* Email
* Password
* Username
* Role\_id
* Notification
* Id
* Notification
* Order\_id
* User\_id

7. System Testing

The system has been tested by trying out the application’s various features and comparing the obtained results with the desired results. The tests have been successful.

8. Bibliography

<https://www.techopedia.com/definition/438/clientserver-architecture>

<https://refactoring.guru/design-patterns/factory-method>

<https://refactoring.guru/design-patterns/builder>

<https://refactoring.guru/design-patterns/observer>

<https://refactoring.guru/design-patterns/decorator>

<https://refactoring.guru/design-patterns/mediator>

<https://www.draw.io/>