Student:Roman Tudor

**Group:30234**

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

Use Swing/C# API to design and implement an application for the order managers of a furniture manufacturer. The application should have two types of users (a regular user represented by the  order manager and an administrator user) which have to provide a username and a password in order to use the application.

# Functional Requirements

The regular user can perform the following operations:

* Add/update/view order information (customer, shipping address, identification number, delivery date, status.).
* Create/update/delete/view product information (title, description, color, size, price, stock etc).
* Add products to order and update order value and stock accordingly.

The administrator user can perform the following operations:

* CRUD on employees’ information.
* Generate reports for a particular period containing the activities performed by an employee.

# Non-functional Requirements

To store data into a database.

To use the Model View Presenter pattern and Factory Method for generating the reports.

Reports written in either xml or simple text format file

Login performed in a secured manner (the password cannot be stored in it’s original form, it needs to be crypted).

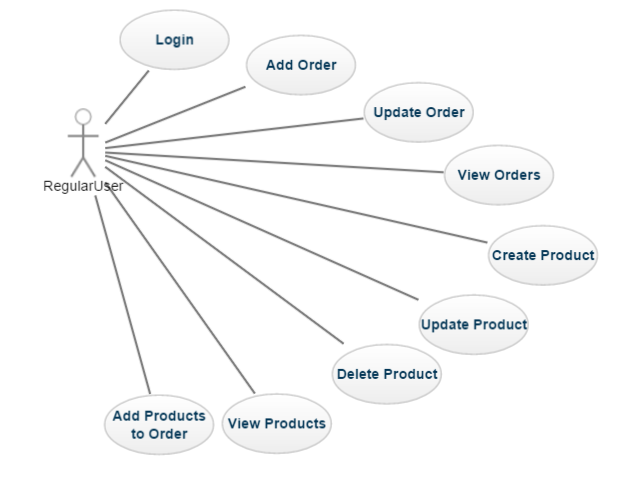
2. Use-Case Model

Use case: adding a product

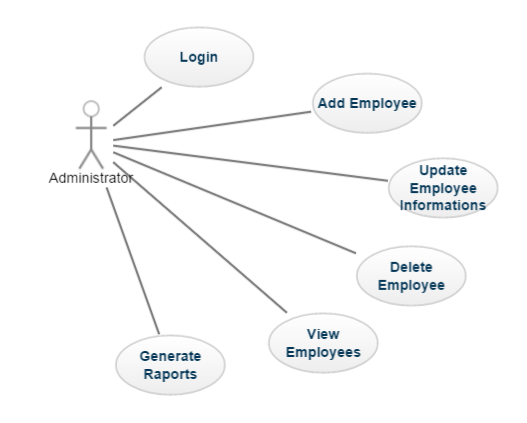
Level: user-goal level

Primary actor: regular user

Main success scenario: login (username and password required)-go to products tab – fill in the information about the product – click “Add Product” button

Extensions: failure : the product already exists in the database

[Use case diagram for a regular user]

[Use case diagram for an administrator user]

3. System Architectural Design

**3.1 Architectural Pattern Description**

MVP is a user interface architectural pattern engineered that follows the separation of concerns in the presentation logic. It is derived from the MVC design pattern. In MVC, all the presentation logic is moved to the presenter, hence make it suitable for automated unit testing.

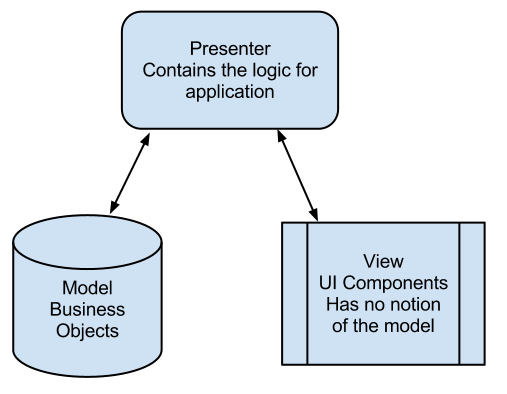
**The view** reflects the data that will be stored in the data-layer.

**The model** is usually very simple , it stores and model the data that the application is going to use. Basically the model contains classes with attributes specifying real life objects.

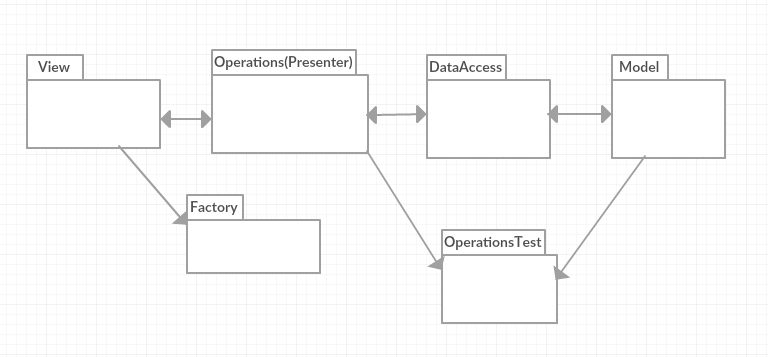
**The presenter** does all the work: listening for events fired by the view and synchronizing the data displayed in the view with the data stored in the data-layer (via the model).

**3.2 Diagrams**

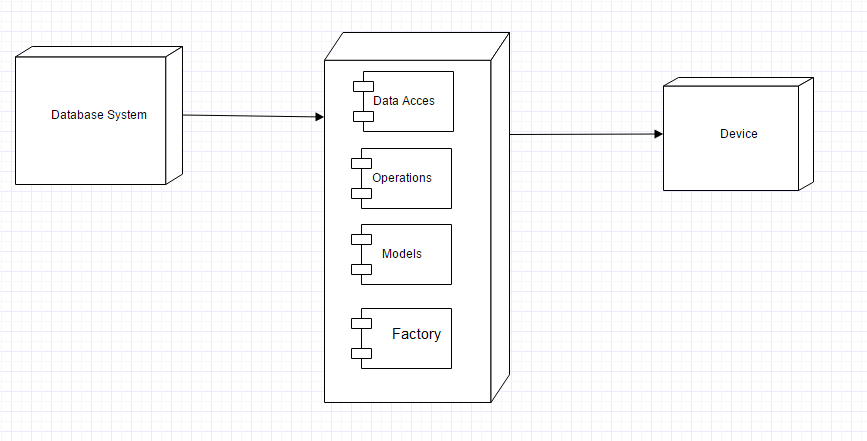
**3.2.1. Conceptual Diagram**

****

[Model View Presenter architecture conceptual diagram]

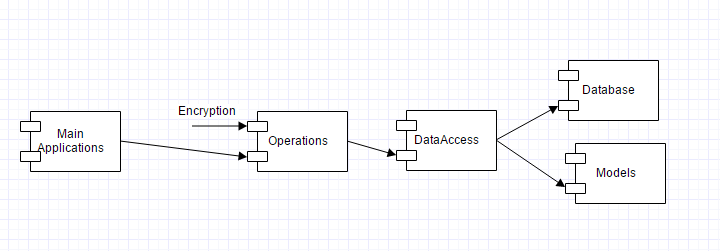
**3.2.2 Package Diagram**

3.2.3 Deployment Diagram



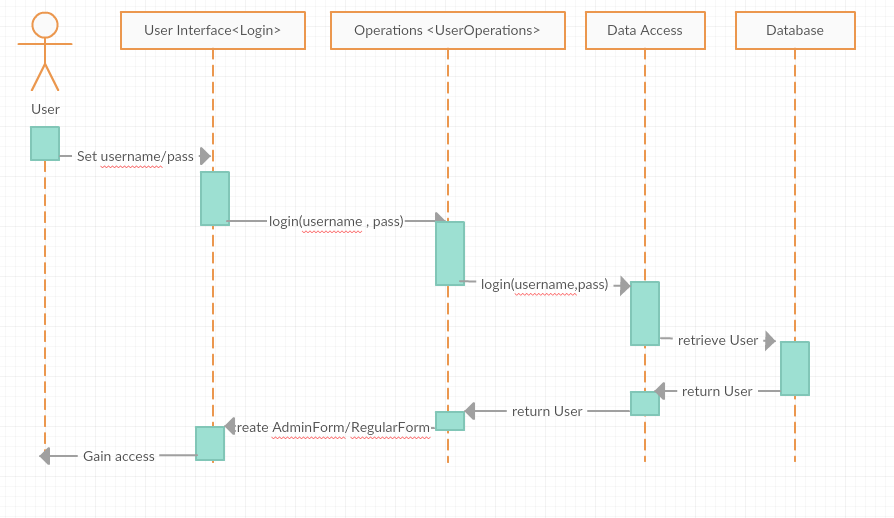
[Deployment Diagram]

3.2.4 Component Diagram



[Component Diagram]

4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

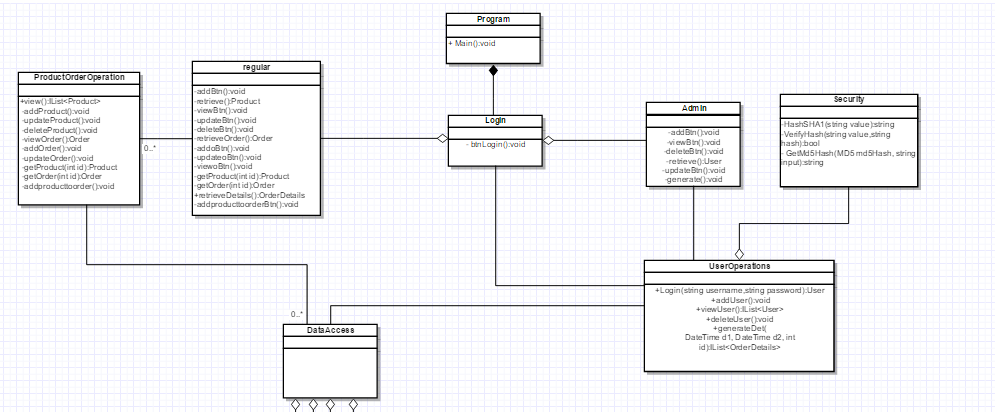
**MVP** is a user interface architectural pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic:

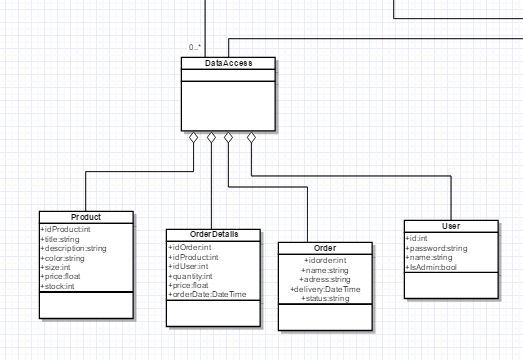
The model is an interface defining the data to be displayed or otherwise acted upon in the user interface.

The presenter acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

The view is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.

**The factory method** pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor**.**

**5.2 UML Class Diagram**

****

[UML Class Diagram]

6. Data Model

As data models I used in my implementation the following:

* Product
* Order
* User
* Order Details

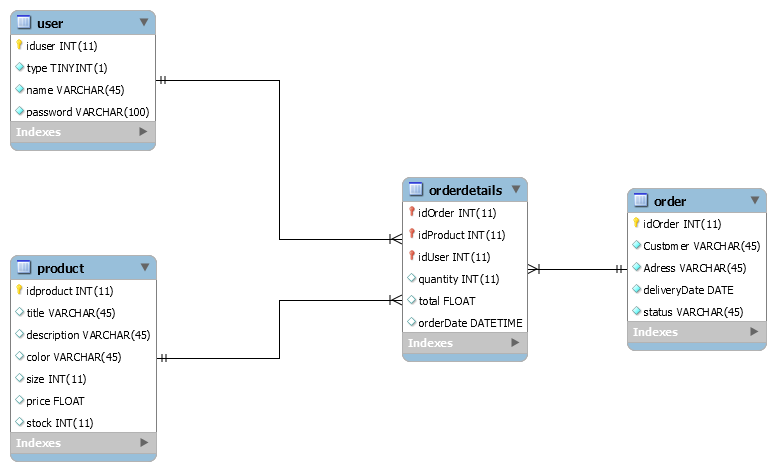
These represents the four classes that I used to represent real world objects.

The products have attributes like: title ,description, identification number, color, size, price and number in stock.

The orders have attributes like: identification number, customer information, address for delivery, delivery date and status.

The users have basic attributes like username, password, identification number and type.

For even more information about orders there are Order Details objects, which have attributes like order identification number, product identification number, total, user identification number and order date.

**

7. System Testing

Unit testing was the main method used for testing the system. Using assertion database operations and method based on data that were retrieved from the database were tested.

Unit testing is based on black box testing, basically the tester chooses an input for a specific method, knowing the expecting output, and with an assert operation tests if the output is satisfying.

8. Bibliography

<https://www.techopedia.com/definition/24649/three-tier-architecture>

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

<http://exponential.io/blog/2015/03/05/3-layer-architecture-in-detail/>