Pharmacy

Analysis and Design Document

Student: Prodan Andreea

**Group: 30235**

Table of Contents

[1. Requirements Analysis 2](#_Toc511503291)

[1.1 Assignment Specification 2](#_Toc511503292)

[1.2 Functional Requirements 2](#_Toc511503293)

[1.3 Non-functional Requirements 3](#_Toc511503294)

[2. Use-Case Model 3](#_Toc511503295)

[3. System Architectural Design 4](#_Toc511503296)

[3.1 Architectural Pattern Description 4](#_Toc511503297)

[3.2 Diagrams 5](#_Toc511503298)

[4. UML Sequence Diagrams 7](#_Toc511503299)

[5. Class Design 8](#_Toc511503300)

[5.1 Design Patterns Description 8](#_Toc511503301)

[5.2 UML Class Diagram 9](#_Toc511503302)

[6. Data Model 9](#_Toc511503303)

[7. System Testing 10](#_Toc511503304)

[8. Bibliography 11](#_Toc511503305)

# 1. Requirements Analysis

## Assignment Specification

The application is designed for the employees of a pharmacy, having two types of users: the chemist and the administrator. The chemists can search for a medication by name or ingredients and can sell a medication. The administrator can perform CRUD operations on chemists informations and on medications and generate report files in pdf or csv format, with the medication out of stock.

## Functional Requirements

**Interface requirements**

The id, quantity and price must have a number format.

**Regulatory/Compliance Requirements**

Clicking the Sign In button sends the user to its specific page depending on the type it is (agent or chemist).

When the agent selects a row from the clients table, the data from the row is put in the specific fields.

If the user enters a wrong data in a field, he/she will receive a warning message.

**Security Requirements**

The administrator can’t access the data about itself.

The chemist can’t access any data about users.

## Non-functional Requirements

**Access Security**

The application can be accesed just by authorized users, using their username and password.

**Accessibility**

Any person should be able to use the application after reading the user manual.

**Response time**

Each action must last less than 1.5 seconds in 90 precent of cases and less than 5 seconds in rest.

The data from tables should load in 2 seconds or less.

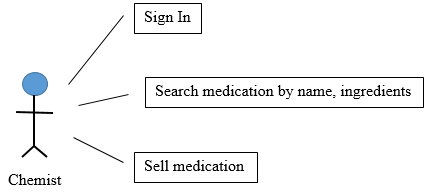
**Availability**

The application should be available allways between 6am and 6pm, the updates being applied just outside this hours.

**Reliability**

Probability of failure on demand (POFOD) shall be 0.0001 (1 out of 10000 plays) when an agent requests to see a specific data.

# 2. Use-Case Model



Use case: Sign In

Level: user goal

Primary actor: chemist

Main success scenario: The chemist enters its username and password correctly and presses the Sign In button. The chemist is signed in and send to the chemist’s page successfully

Extensions: If one of the introduced input is incorrect or the username is not found in the database, a message is displayed

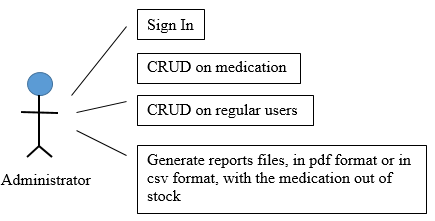
Use case: Search for medication by its name

Level: user goal

Primary actor: chemist

Main success scenario: The chemist introduces the name of an existing medication and clicks the Name button. The medication’s information is displayed in the table.

Extensions: The introduced name doesn’t exist. A warning message is displayed.



Use case: Add a new chemist

Level: user goal

Primary actor: administrator

Main success scenario: The administrator introduces all the user’s data correctly in the fields and presses the Add button. The user is added, the modification being visible in the user’s table.

Extensions: One of the introduced data is incorrect. The adding operation isn’t performed and a warning message is displayed.

# 3. System Architectural Design

## 3.1 Architectural Pattern Description

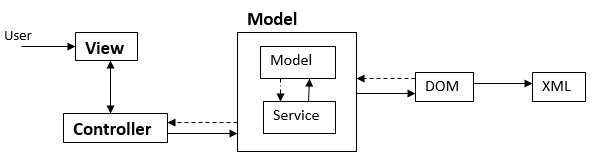
The MVC architectural pattern is used to organize the application. The architecture devides the application into three interconnected parts: view, model, controller. This is done to separate internal representations of information from the ways information is presented to and accepted from the user, allowing for future code reuse.

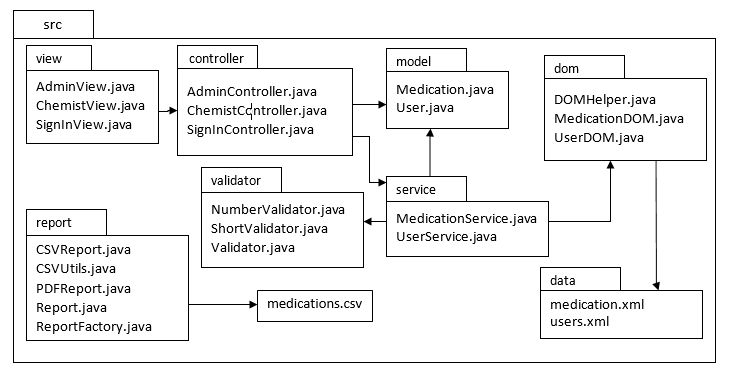
## 3.2 Diagrams

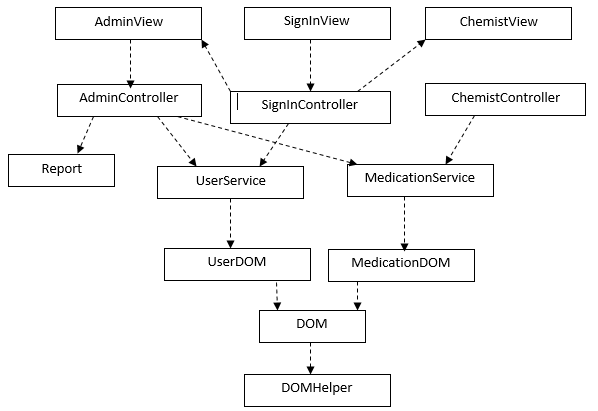
The model is responsible for managing the data of the application. It receives user input from the controller and sends back information about updates or errors where needed. Additionally, here is are added services classes, which validate the data received from the controller, make any other necessary business operations and realise the connection between the controllers and dom.

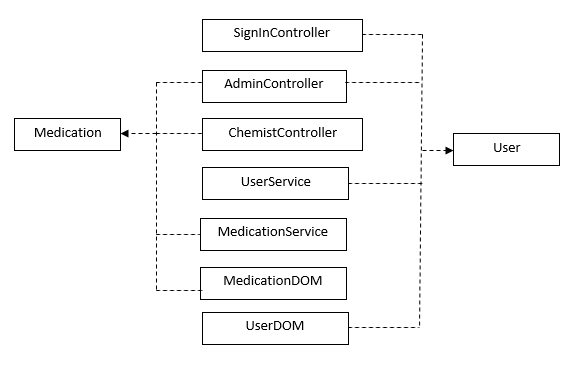
The view represents the presentation of the model and is what the user can see and access directly. When the user performes an action, the view signals the controller about it and doesn’t takes any other action, unless the controller tells so. The view, therefore, has no knowledge about how the data is processed or stored.

The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the actions performed by the user and input, optionally validates the input and then passes the input to the model. Also, it receives back information from the model, prepares the information in the specific format for the view (String or Object), so the view will have no knowledge about the model, and then passes it to the view. Therefore, the controller doesn’t care how the information is processed or the logical operations performed for achievieng a specific result, it does care about the steps which have to be followed and the results which should be obtained for each action performed by a user.



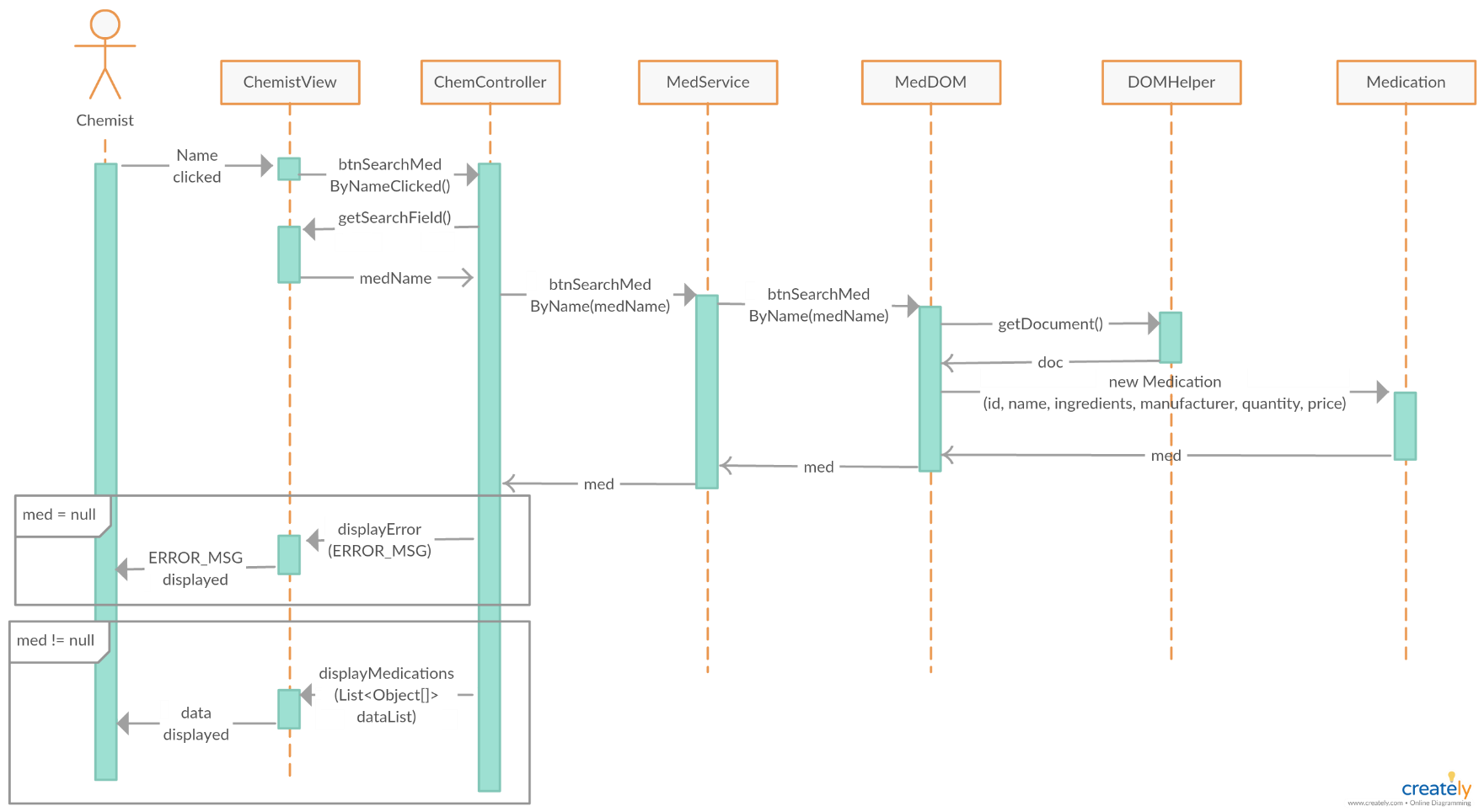






# 4. UML Sequence Diagrams

Search Medication By Name



# 5. Class Design

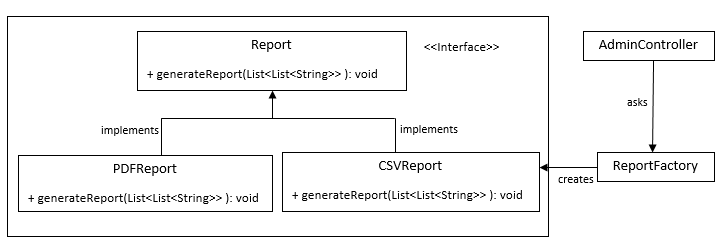
## 5.1 Design Patterns Description

**Domain model**

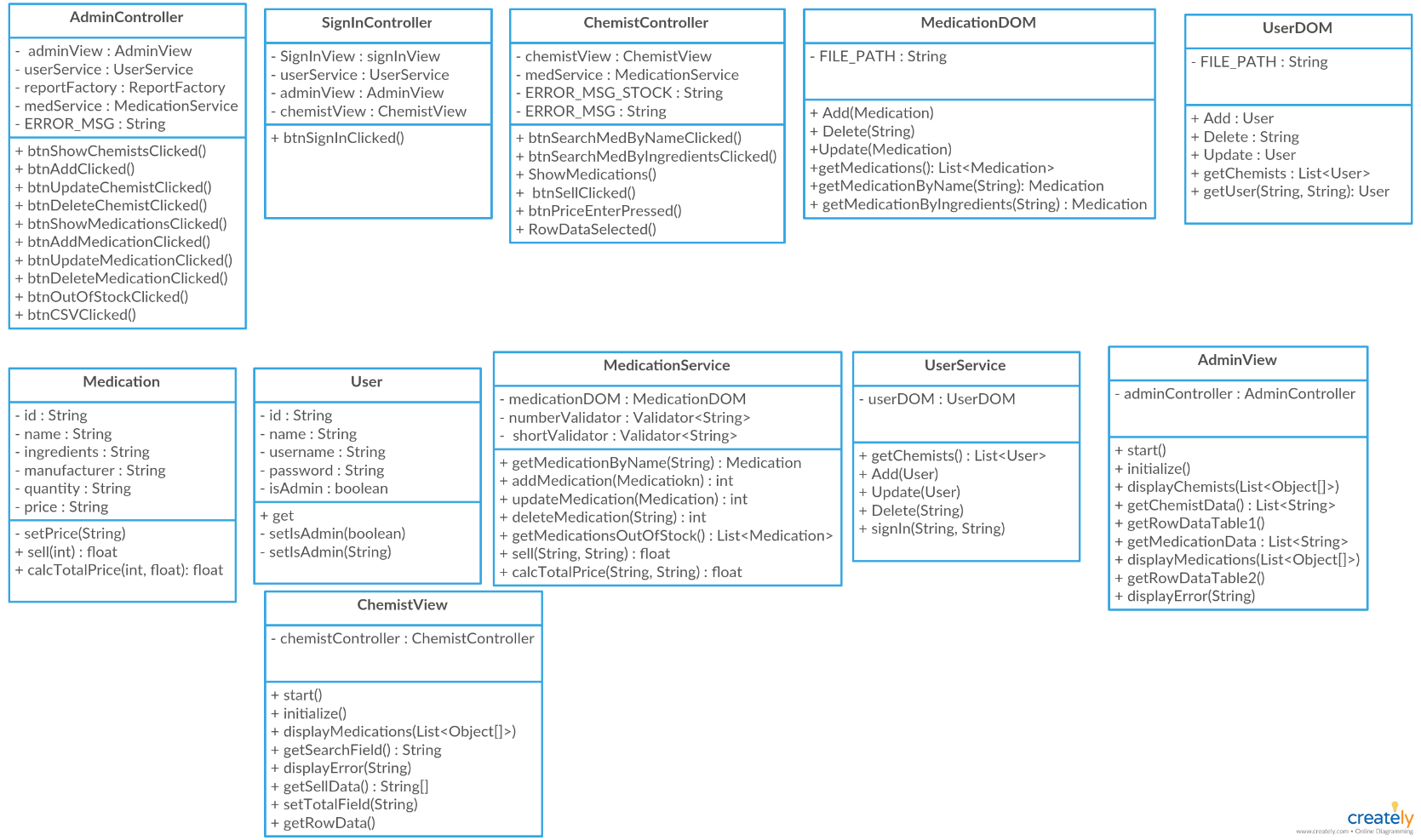
There is a class to represent a data type from the xml file, which contains attributes, setters, getters and domain logic methods.

**Factory pattern**

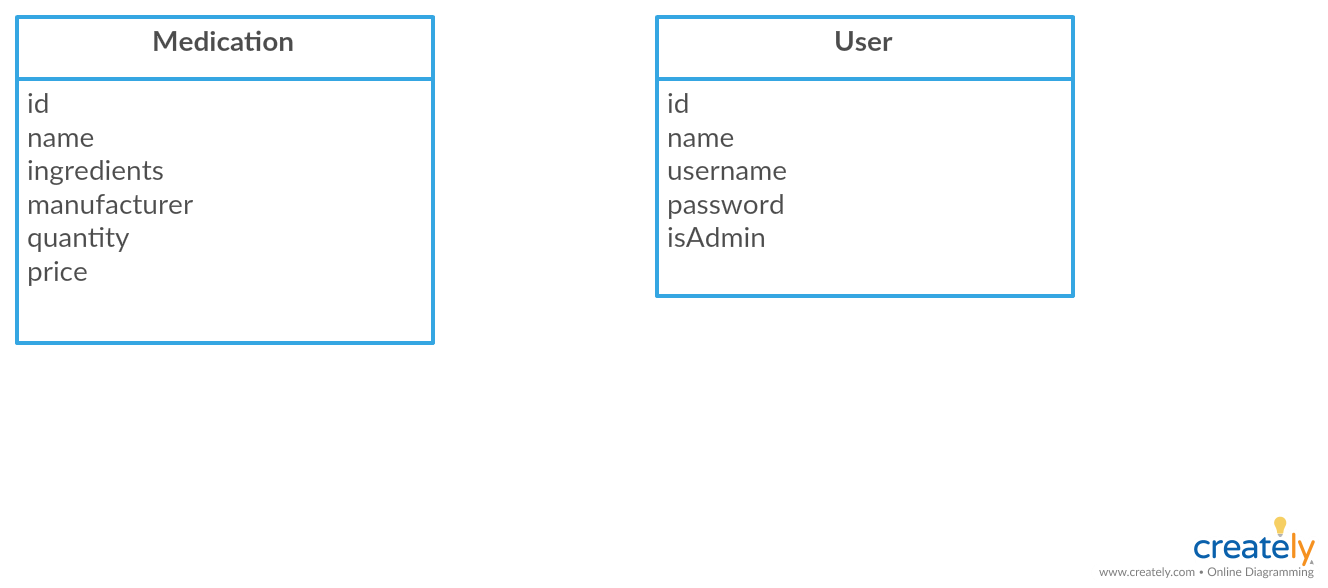
Is used to create the report objects, without exposing the creation logic. In this way the creation is better controlled.



## 5.2 UML Class Diagram



# 6. Data Model

**

<users>

<user>

<id>*1*</id>

<name>*Derek*</name>

<username>*derek*</username>

<password>*derek*</password>

<isAdmin>*true*</isAdmin>

</user>

</usrs>

<medications>

<medication>

<id>*1*</id>

<name>*Aspirin*</name>

<ingredients>

*hypromellose, lactose monohydrate, methacrylic acid copolymer*

</ingredients>

<manufacturer>*ASE*</manufacturer>

<quantity>*108*</quantity>

<price>*12.7*</price>

</medication>

</medications>

# 7. System Testing

The strategy used for testing was unit testing, using Junit Test

Also testing was made directly on the application, verifying the validations for each field.

# 8. Bibliography

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

<https://www.tutorialspoint.com/hibernate/orm_overview.htm>

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

https://www.youtube.com/watch?v=diSwriAWwvs