Student:Galeata Bianca-Floriana

**Group:30235**

Table of Contents

Table of Contents 2

1. Requirements Analysis 3

.1 Assignment Specification 3

.2 Functional Requirement 3

.1 Non-functional Requirements 4

2. Use-Case Model 5

3. System Architectural Design 6

4. UML Sequence Diagrams 9

5. Class Design 9

6. Data Model 11

7. System Testing 12

8. Bibliography 12

1. Requirements Analysis

# Assignment Specification

The objective of this project is to design and implement an application for a pharmacy. The application is going to facilitate the way that an employee can search for medication and how he can make a sale.

Beside the employee, the application will have another user, named administrator. This will have the possibility to manage the employee’s and medications’ information.

The application should help employee and their supervisors (administrator) to manage the data more easily and to make the pharmacy that will use it more efficiently and organized.

# Functional Requirement

Using this application, an agent can do:

* Search medication by name, ingredients, manufacturer
* Sell medication

On the other hand, an administrator can do:

* CRUD on medication
* CRUD on regular users’ information
* Generate a report about the medication out of stock

The data will be divided between two storing places: XML files and a database. For the database an ORM will be needed in order to access/store/modify/delete data.

The information needed for a concept is:

***Medication:***

* Name
* Ingredients
* Manufacturer
* Quantity
* Price

***Employee/Admin:***

* Username
* Password

***Sale:***

* Sold medication’s name
* Quantity sold
* Employee that made the sale

***Report:***

* Medication’s name
* Medication’s ingredients
* Medication’s manufacturer

The reports generated has to be of two types: a pdf and a csv file. For constructing reports the Factory patterns will be used.

Both users have to use a username and a password in order to access the application.

# Non-functional Requirements

* *Availability* - This attribute counts the time the application is functional.

This project will be projected as a desktop application, and a result the application can be accessed anytime. But, there is a necessity to have an internet connection in order to have access the data from the database and to store new data in it. Even if we have data in XML files, not all of it saved on it. So in order see medication a database is used.

* *Performance* - indicates how responsive an application is when an action and an interval of time are given.

As a desktop application the performance can be reported to the system that is used and to the database connection.

On a decent system, the performance will be on its best behavior and the time response will be small. But all the responsiveness will be affected by the connection to the database. If the connection does not give a response in a short amount of time, this will influent the performance.

* *Security* - The capability of a system to prevent malicious or accidental actions outside of the designed usage.

Any person that have the application can access it. But in order to use the application and to have access to the database you have to login with a username and a password. This date must be correct or else the access will not be provided. In this manner just employees who have an account or the administrator can use the application.

* *Testability* - This measure show how easy is to create test for a system and how quickly a failure can be demonstrated.

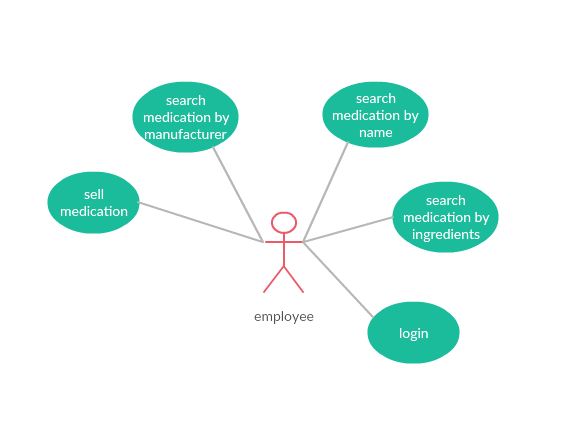
The application has been tested in the implementation process and after this process, as a system. Each component has been tested separately and in connection with others. With each new module, a small test was made in order to find the error. Each encountered problem has been solved.

* *Usability* - This quality measure how intuitive, easy and simple is the user’s interaction with the application.

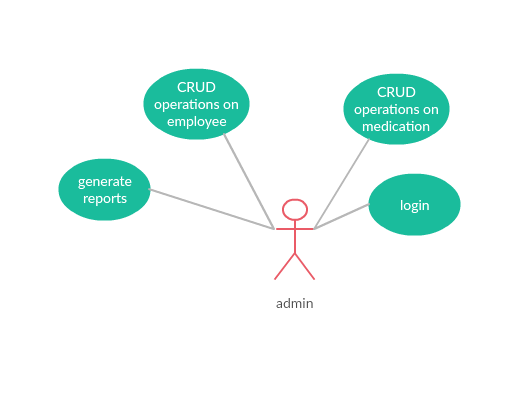
The UI is simple and intuitive. There are tables to show different data, each column having the name of the concept it represents. The buttons have suggestive names to indicate their role. There are messages that are shown in order to indicate if an action has been done wrong or it has success.

2. Use-Case Model

* Use case for employee:



* Use case for admin:



* Detailed use case:

*Use case:* Sell medication

*Level:* user-driven goal

*Primary actor:* employee

*Main success scenario:*

The first step is to access the application. Once it is open, a window will be seen. There are 2 text field where the username and password must be inserted. Let’s suppose that those are correct. Once the button “login” is clicked, a new window will appear. There is an empty table with medications and buttons that help to make searches or a sale. In order to sell something, first we need to have that something. To do that, we first need to do a search by name, ingredients or manufacturer. Let’s suppose that we make a search by name and a medication appear in the table. Next we have to click on the row from the table where the medication is. After that, we introduce the quantity for sale in the textbox near the button “Sell”. Supposing that the quantity that is intended for selling is smaller or equal with the available one, clicking the button “Sell”, the sale will be made.

*Extensions:*

From the above scenario, let’s suppose that the quantity intended for sell is bigger than the one available. In this case when the button “Sell” is clicked, an error message will appear and the sale won’t be made*.*

3. System Architectural Design

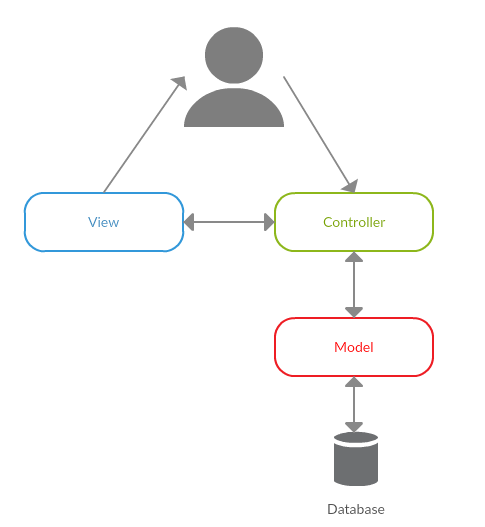
**3.1 Architectural Pattern Description**

The used Architectural Pattern is named Model-View-Controller (MVC). The concept is simple explain by the division of the program in three layers: model (similar with a domain logic), view (UI), and controller (handles the input).

The view is responsible to handle the user interface, the model represents concepts that are worked with (Medication for example) and has to execute specific business rules associated with a request and controller has to handle the input from the user, and to make the connection between model and view (these are not directed connected, neither know about each other, but the controller makes the updates of the UI when the model changes and send information to the model when a request is made).

**3.2 Diagrams**

* Conceptual architecture:



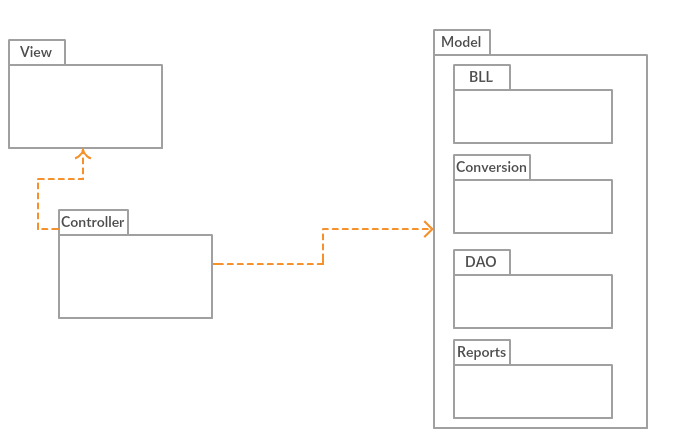
This diagram shows how the application is structured and how it’s components communicate. As it can be seen, there are three layers: Model, View and Controller.

The View has components as View classes which compose the UI. This is responsible for user interactions.

The Model has classes which represent the model (the data manipulated) and the classes for business logic (ConvertorSale, MedicationsServices etc.). Also here are the classes which offer the database and XML manipulations.

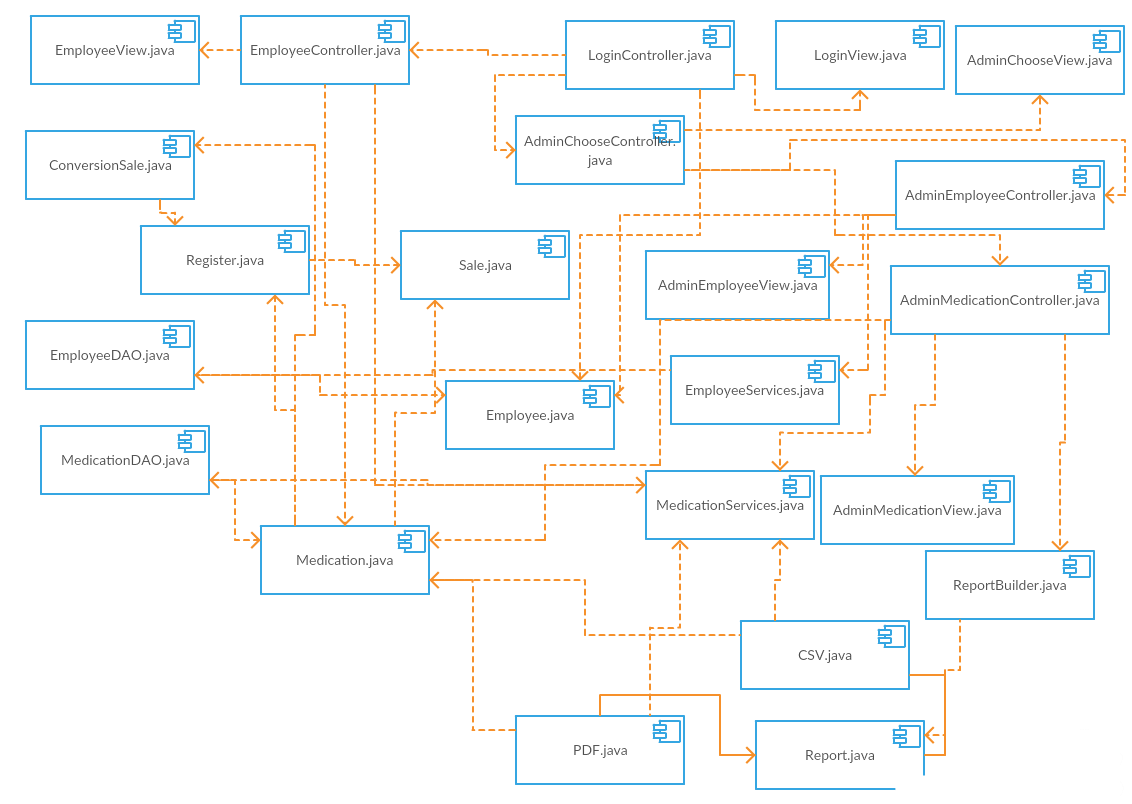
The Controller has a class for the connection between model and view. It receives from the view an input and transmit is to the model. After the model possess the data, the controller sends it back to the view in order to update it.

* Package diagram:



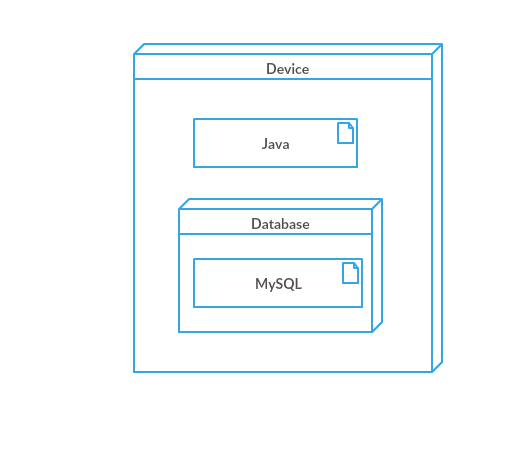
Here is presented the packages diagram. The structure of packages highlights the Model-View-Controller architecture whit 3 main layers and their inside structured on layers.

* Component diagram:



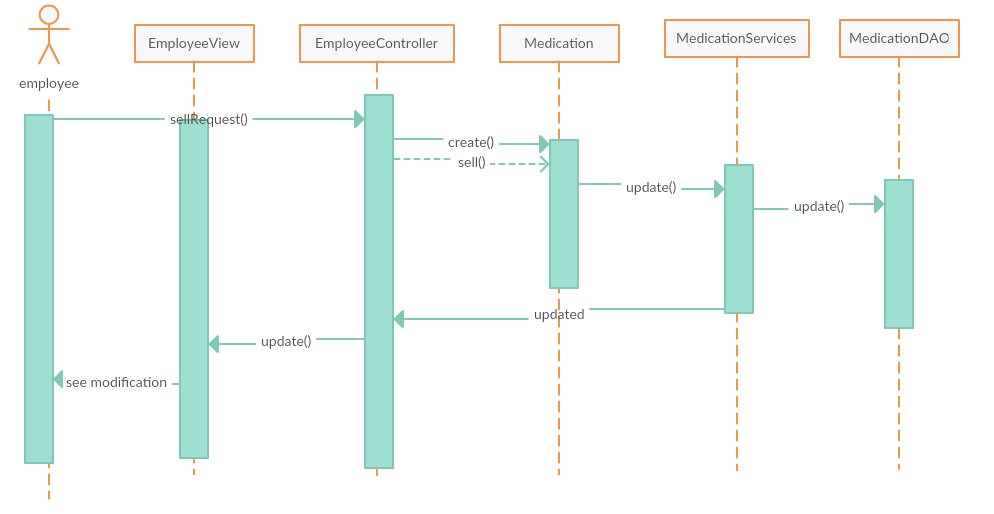
This component diagram shows how classes communicate. As is can be seen the view classes communicates with its controller classes and from there these communicates with models (Employee for example) and with classes which provides services. Also the are communications between controller classes. The services classes represent the connection between database classes and others. The model classes also use this services classes and other models.

* Deployment diagram



This diagram present how the application is deployed. As it can be seen the application and the database runs on the same device.

4. UML Sequence Diagrams



This sequence diagram shows how the components connect in order to make a sale. The request leaves from user to the controller by the view. From there it is transmitted to the model. The model (medication) is updated and from there a request for the class responsible with the conversion to XML file is send. After the model is updated, a message is sent in order to update the interface. This message is send for the controller and this send the update message to the UI.

5. Class Design

**5.1 Design Patterns Description**

The used design patterns are:

* ***Factory Pattern***

With this pattern we create object without exposing the creation logic to the client and refer to newly created object using a common interface. In our case it has been used to create two types of reports (CSV file and PDF).

* ***Domain Model Pattern***

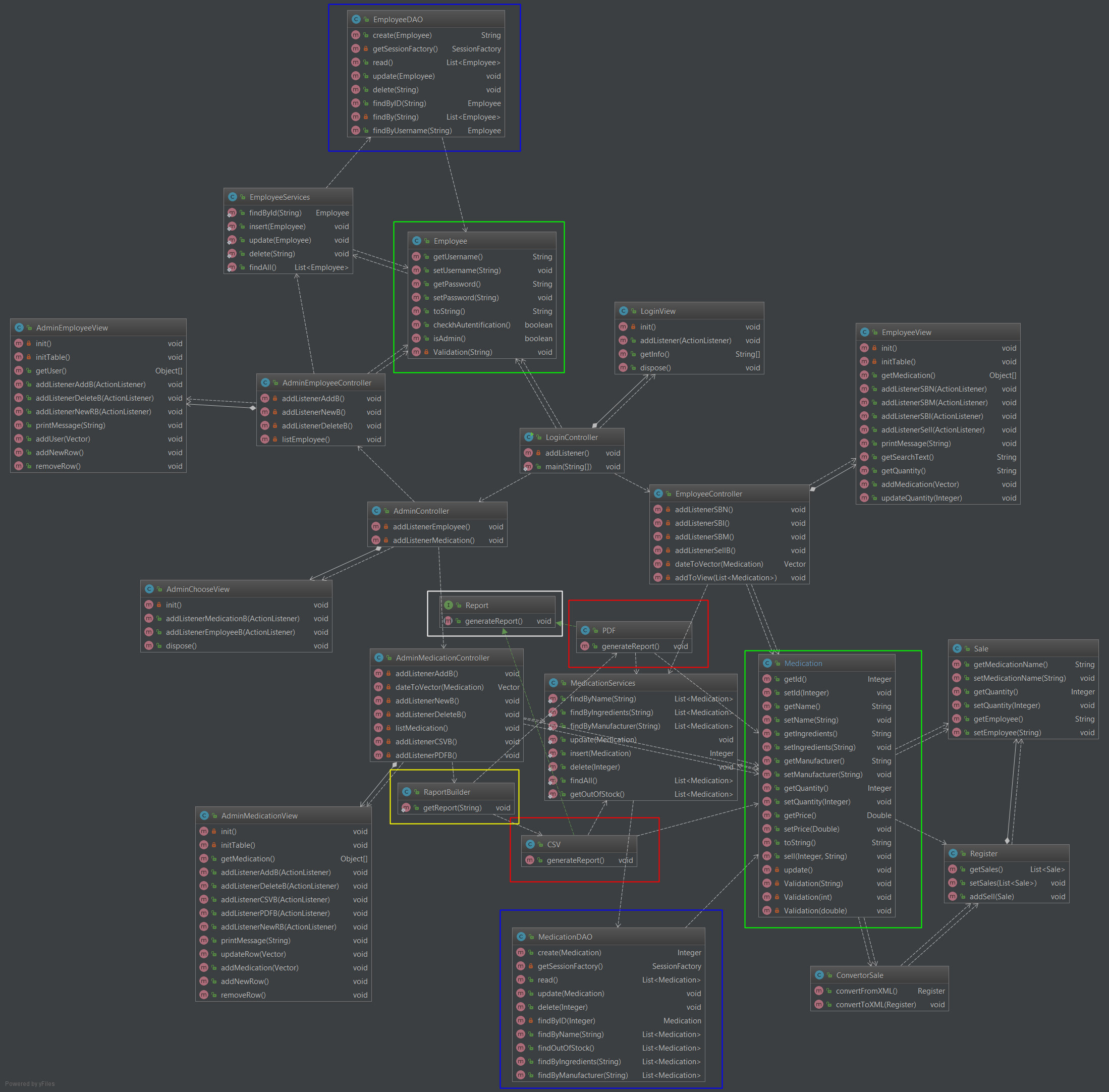
The *Domain Model* is a pattern for the domain logic. It represents the model of the domain that has behavior and data. In our case, is used for classes which represents the concepts and these incorporates different methods (for making a sell for example).

* ***Table Gateway Pattern***

The *Table* *Data Gateway* handles all the logic for database access (insert, delete etc.) and all the rows from a table. It is a data source pure pattern. In our situation is located in the model and is used by other classes through classes that provide services.

**5.2 UML Class Diagram**

In this section is presented a class diagram. The classes are grouped by the concept they belong to. The design pattern used are highlighted.



Green – *Domain Model Pattern*

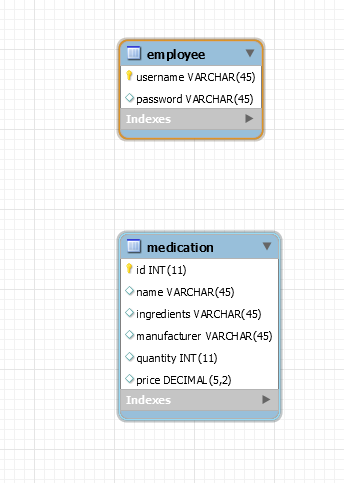
Blue – *Table Gateway Pattern*

White (interface), Yellow (Builder), Red (classes that implement the interface) – *Factory Pattern*

6. Data Model

The user Data Model in this project is *Relational model*. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. There also are data saved in a XML file.

The next diagram shows how the database for this project looks:

**

In the database have been stored two tables: one for medication and one for employee.

Next we can see the structure of the XML file:

<register>

<sale>

<employee></employee>

<medicationName></medicationName>

<quantity></quantity>

</sale>

</register>

7. System Testing

In each of the strategies that will be presented the data-flow method has been used. Even if it started from a user interface or from a class test that has triggered a series of event, information has been printed in order to detect if the data have anomalies.

* *Unit testing*

Each class has been tasted once implemented. Tests have been made for the view package, for the model package and for controller package and theirs classes.

The models’ classes have been tested, making inserts, deletes and updates into the database using a test class for different tables. Also the methods have been tested by printing the result and manually verify it.

The view’s classes have been tested by inserting data and then using controller it has been printed and manually verified.

The controller’s classes have been tested by printing the result and manually verify it.

* *Integration testing*

The modules have been tested in a group in order to obtain the functionality required. First classes from the model package have been integrated with those in the controller package, and then the ones from view package layer have been added.

* *Validation testing*

This testing strategy has been made by users by testing the functionality provided by the application.

8. Bibliography

[1] *Domain Model pattern example,* from <https://stackoverflow.com/questions/41335249/domain-model-pattern-example> [2018-04-18]

[2] *Everything You Need to Know About Software Testing Methods,* from <https://www.thebalance.com/all-you-need-to-know-about-software-testing-methods-4019921> [2018-04-18]

[3] *Design Pattern - Factory Pattern, from* <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm> [2018-04-18]

[4] *Design Patterns - MVC Pattern*

, from <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm> [2018-04-18]