Analysis and Design Document

Student:Cristea Andrei

**Group:30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

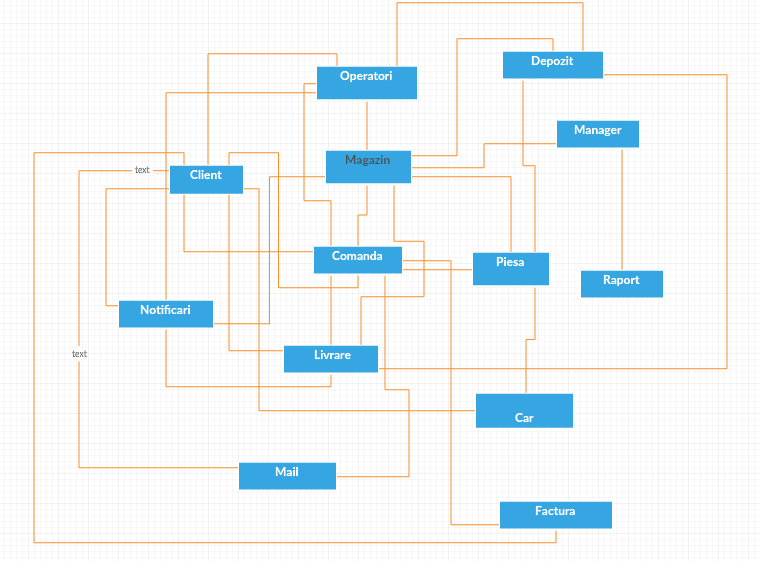
VI. Bibliography 5

# Project Specification

The application is a desktop application that helps users to search for and order the desired pieces. The main functionality of the application is to find the desired product, which can be found by several search methods designed to help users. The system is designed to could see product information and finally order it.

# Elaboration – Iteration 1.1

# Domain Model



**Relation:**

* Client 1..\* Comanda
* Client 1..\* Livrare
* Client 1..\* Notificare
* Client 1..1 Operatori
* Client 1..1 Car
* Magazin 1..\* Comanda
* Magazin 1..\* Piesa
* Magazin 1..1 Manager
* Magazin 1..\* Operatori
* Magazin 1..\* Livrare
* Comanda 1..1 Client
* Comanda 1..\* Piesa
* Comanda 1..1 Livrare
* Manager 1..\* Raport
* Masina 1..\* Piesa
* Operator 1..\*Comanda
* Operator 1..\* Client
* Operatori 1..\* Notificari
* Livrare 1..1 Notificare
* Comanda 1..1 Factura
* Client 1..\* Factura
* Comanda 1..1 Mail
* Client 1..\* Mail

# Architectural Design

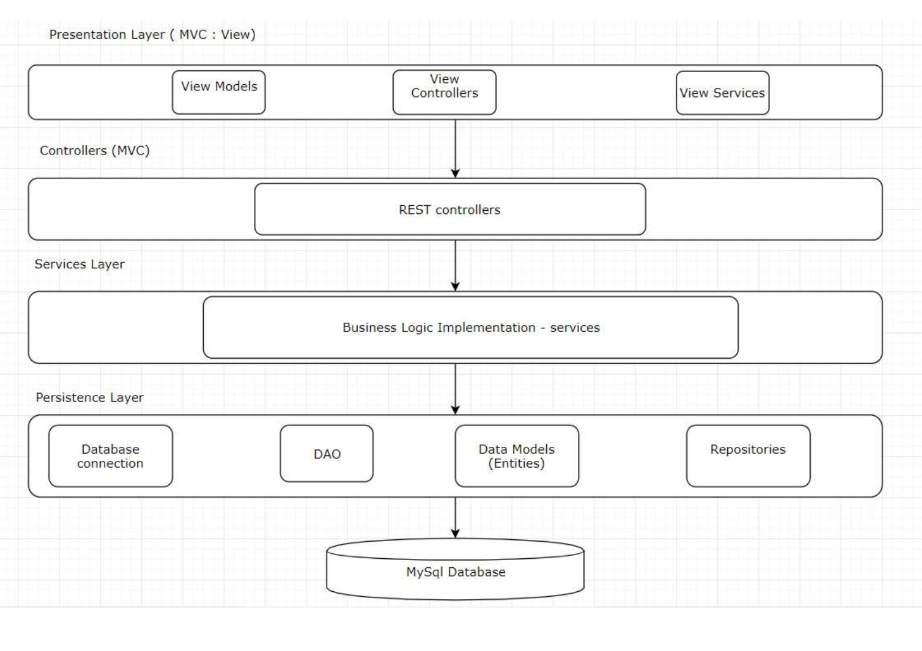
## Conceptual Architecture

The architectural patters used are the following:

* Model-View-Controller architectural pattern
* Layer Architectural pattern

The used design patterns will be:

* Observer Design Pattern
* Iterator



1. **Layers Architectural Pattern**

This pattern can be used in order to structure the application such that it can be decomposed into groups of subtasks. Each layer implements subtasks at a particular level of abstraction and each layer provides services to the next higher level. The following 4 layers will be used in order to offer the application the accurate structure:

* Presentation Layer
* Controller Layer
* Business Logic Layer
* Data Access Layer

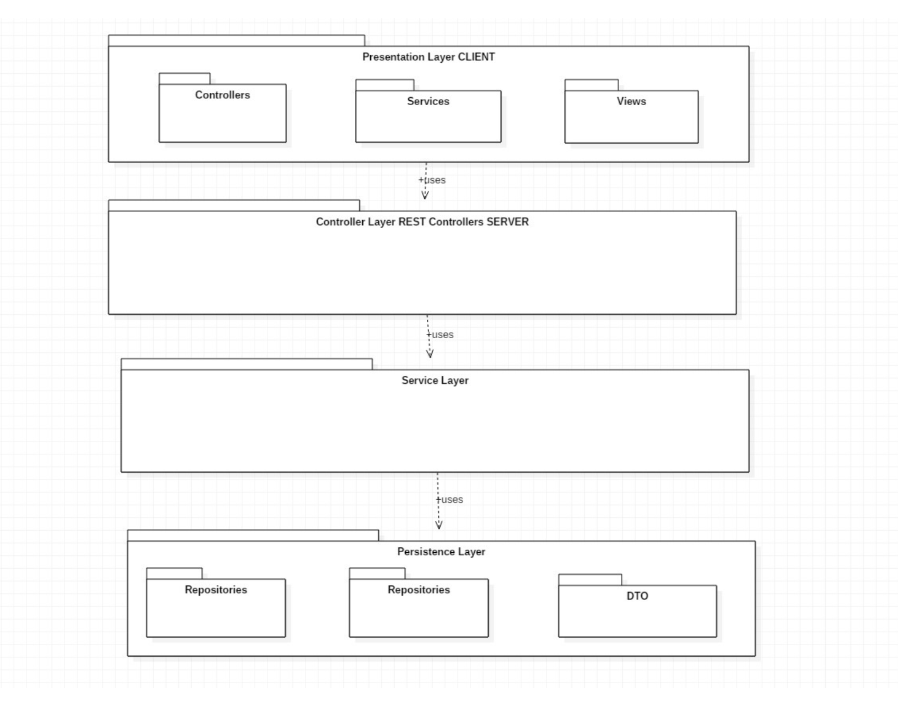
These layers are modeled using different packages inside the application.

1. **The Model-View-Controller Architectural Pattern**

Model–view–controller (MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development

* Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* View - View represents the visualization of the data that model contains.
* Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

## Package Design



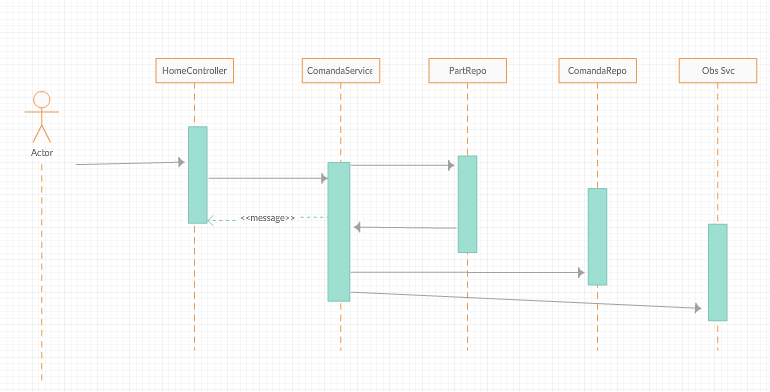
## Component and Deployment Diagrams

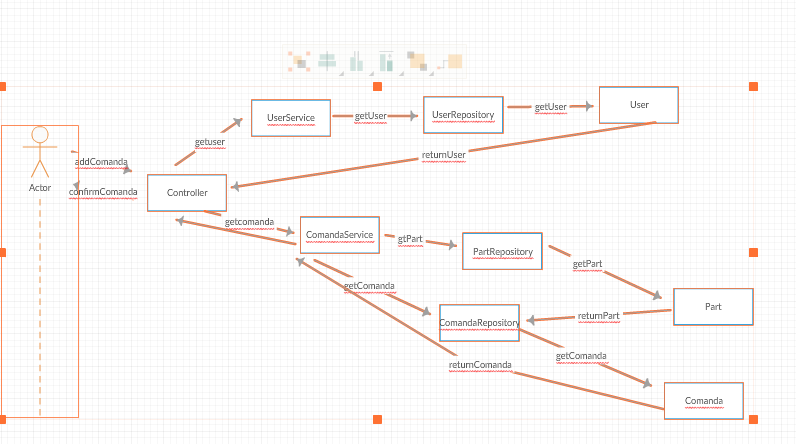
# 

# Elaboration – Iteration 1.2

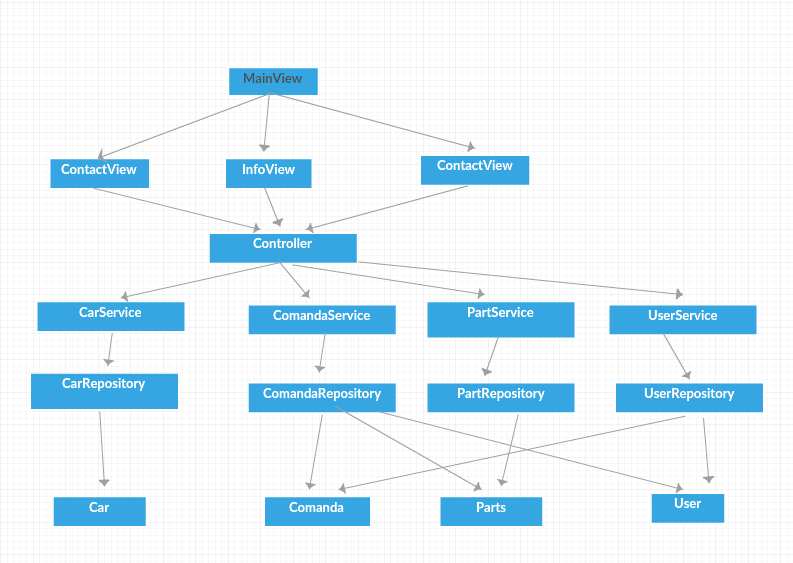
# Design Model

## Dynamic Behavior

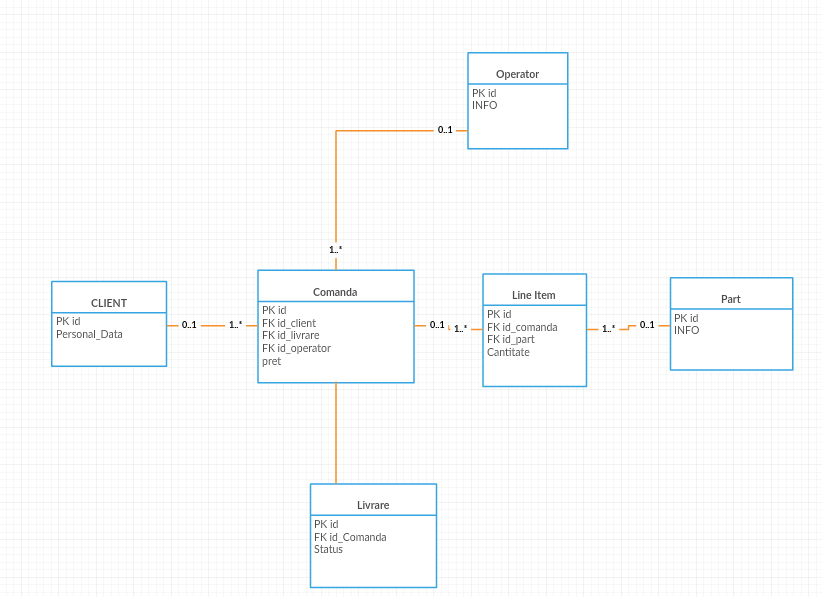
**

**

## Class Design

**

# Data Model



# Elaboration – Iteration 2

# Architectural Design Refinement

Documentation modifications:

* Domain Model
* Data Model
* Class design
* Dynamic behavior

# Unit Testing

I will use Mokito to test my application. Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behaviour of the object we want to test we replace the other objects by mocks that simulate the behavior of the real objects. In short, mocking is creating objects that simulate the behaviour of real objects. I will write small tests in order to mainly test the functions implemented in the service layer.

# System Testing

The system has been tested using the client-side application. The user, which can be an admin or a regular user interacts directly with the application through the friendly user interface.

The following usual and particular cases have been tested:

* The user log in, the username and password validation.
* The distinction between an user account and an admin account.
* The creation of a new user account and user account validation.
* The system response when the provided data for an user account is not valid.
* The placing of an order containing the products.
* Add new products , remove products, increase or decrease the quantity of any product.
* The operations the admin can perform on the data stored in the database.
* The update of product stock after an order is placed and the stock must be updated.
* The action of deleting a user account.

# Future improvements

Future improvements could be the following ones:

* The user can get a recommendation of a part
* The user can get notified when a product in his/her wish list gets available.
* The navigation between the views of the application can be made easier.
* The application security can be implemented using Spring Security.
* The user log in part of the client-side application is implemented using cookies.

# Bibliography

<http://www.mkyong.com/tutorials/spring-boot-tutorials/>

<https://www.javaworld.com/article/2077674/java-web-development/a-standardized-object-relational-mapping>

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

<https://zeroturnaround.com/rebellabs/spring-framework-annotations-cheat-sheet/>

<https://spring.io/guides/tutorials/bookmarks/>