Diagnostic Centre Client Coordination System

Analysis and Design Document

Student: Danila Lucia-Diana

Group: 30433

Revision History

Date	Version	Description	Author
<30/mar/18>	<1.0>	<made the first version>	<Danila Lucia-Diana>

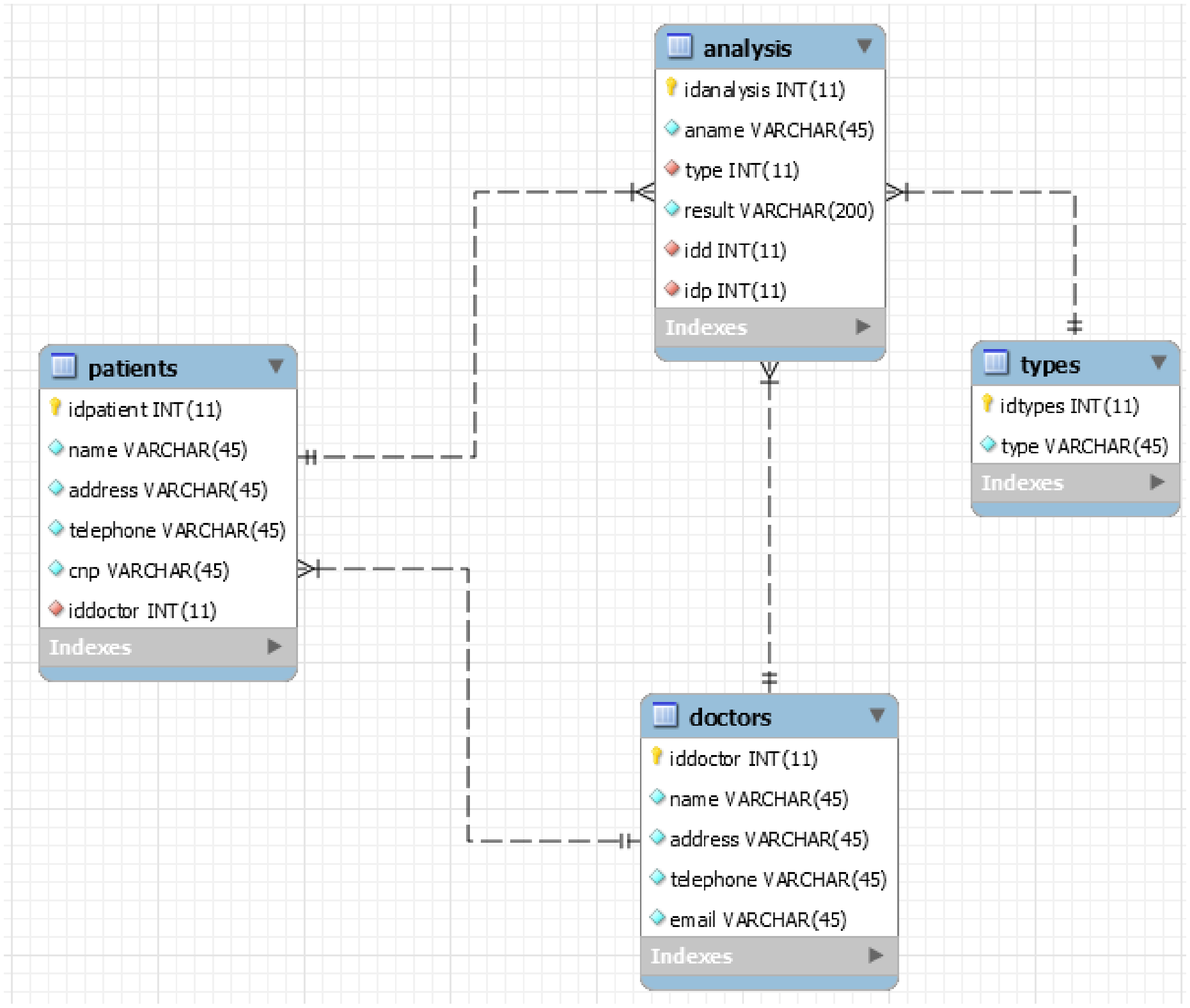
Table of Contents

I. Project Specification

Diagnostic Centre Client Coordination System helps the diagnostic centres to maintain good relations with their clients. The clients may conduct various tests at the diagnostic center and the system must be capable of valuating patient bills and providing them in printable format.

II. Elaboration – Iteration 1.1

1. Domain Model



2. Architectural Design

2.1 Conceptual Architecture

For implementing this project, the architectural patterns I have chosen to implement are: a client server architecture with the following design patterns: table module pattern, and a dao pattern. The user interface will be implemented using an MVC architecture.

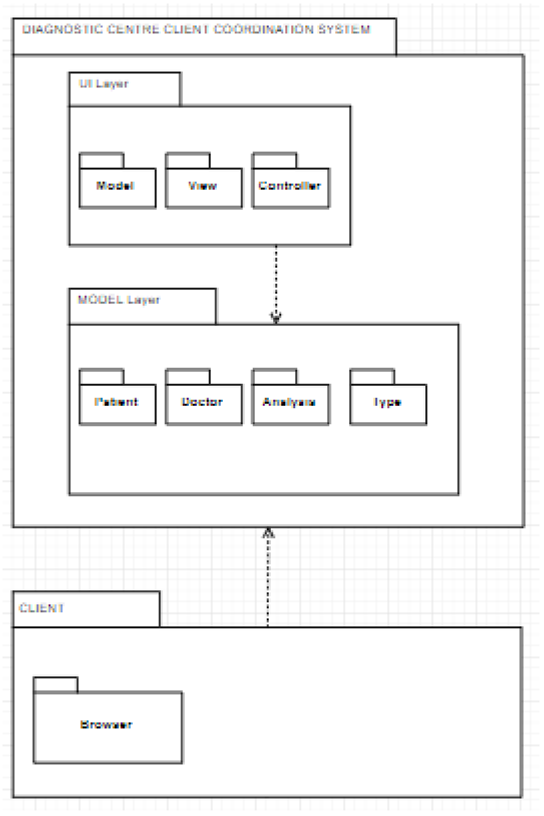
Client Server: Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources.

Dao: access to data varies depending on the source of the data. Access to persistent storage, such as to a database, varies greatly depending on the type of storage (relational databases, object-oriented databases, flat files, and so forth) and the vendor implementation.

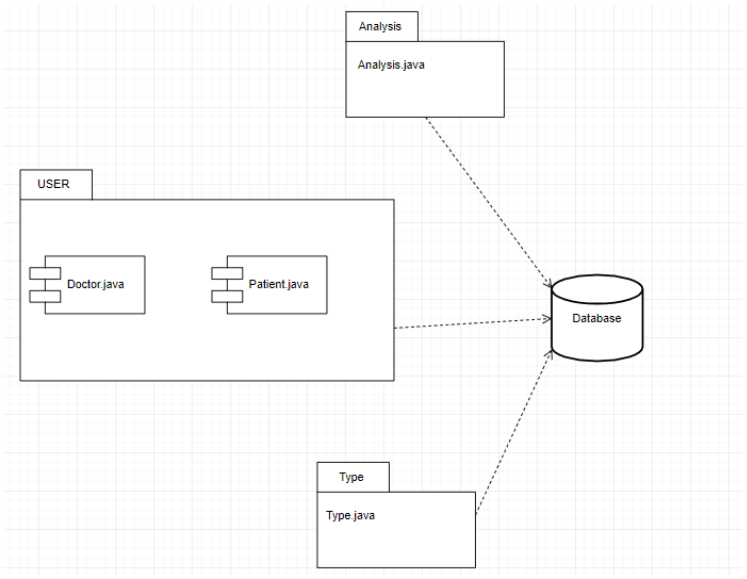
Table Module: A single instance that handles the business logic for all rows in a database table or view.

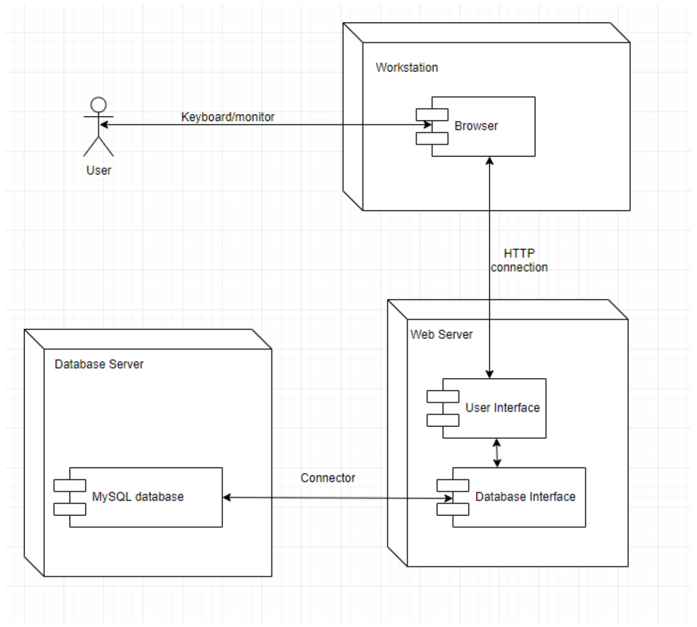
I choose them because the DCCCS will be client server system, dao and table module patterns will deal with the database needed to store the doctors, clients and their analysis (accessing these informations).

2.2 Package Design



2.3 Component and Deployment Diagrams



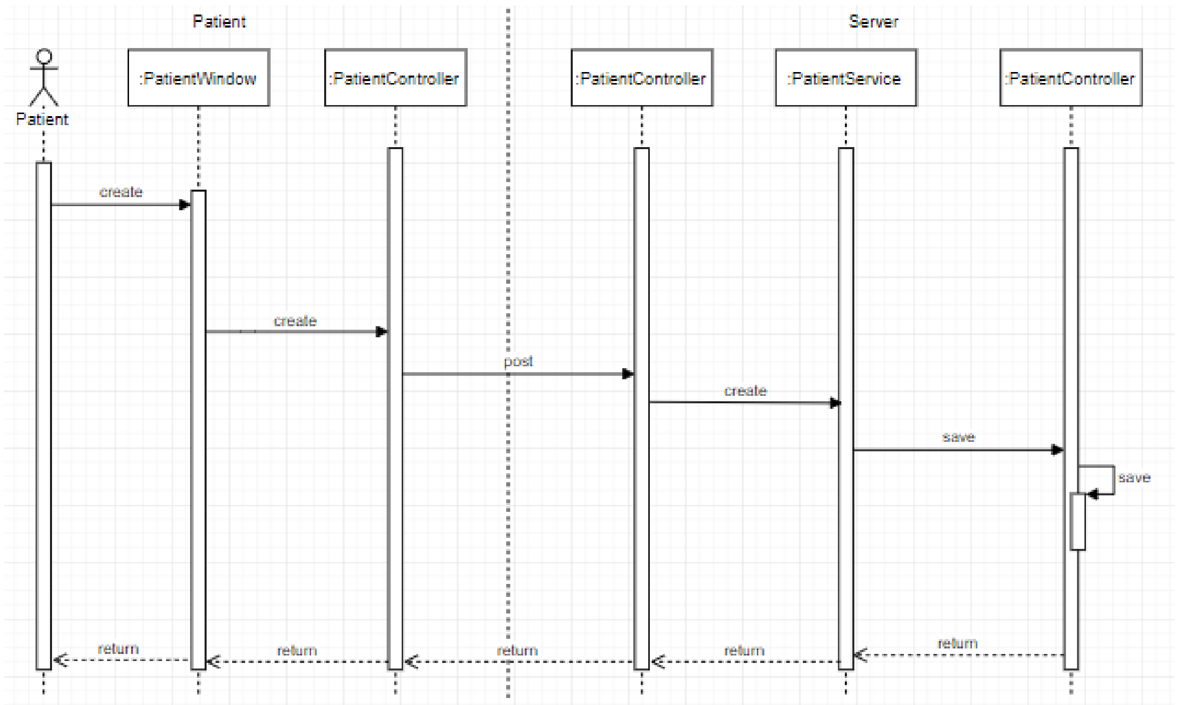


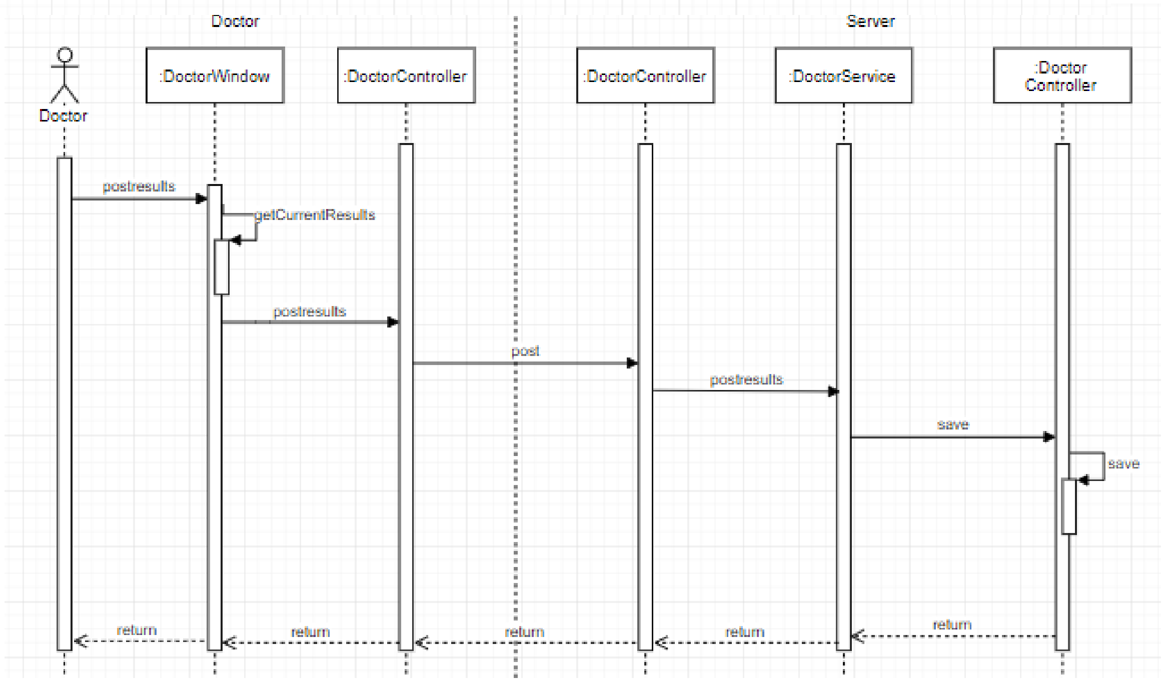
III. Elaboration – Iteration 1.2

1. Design Model

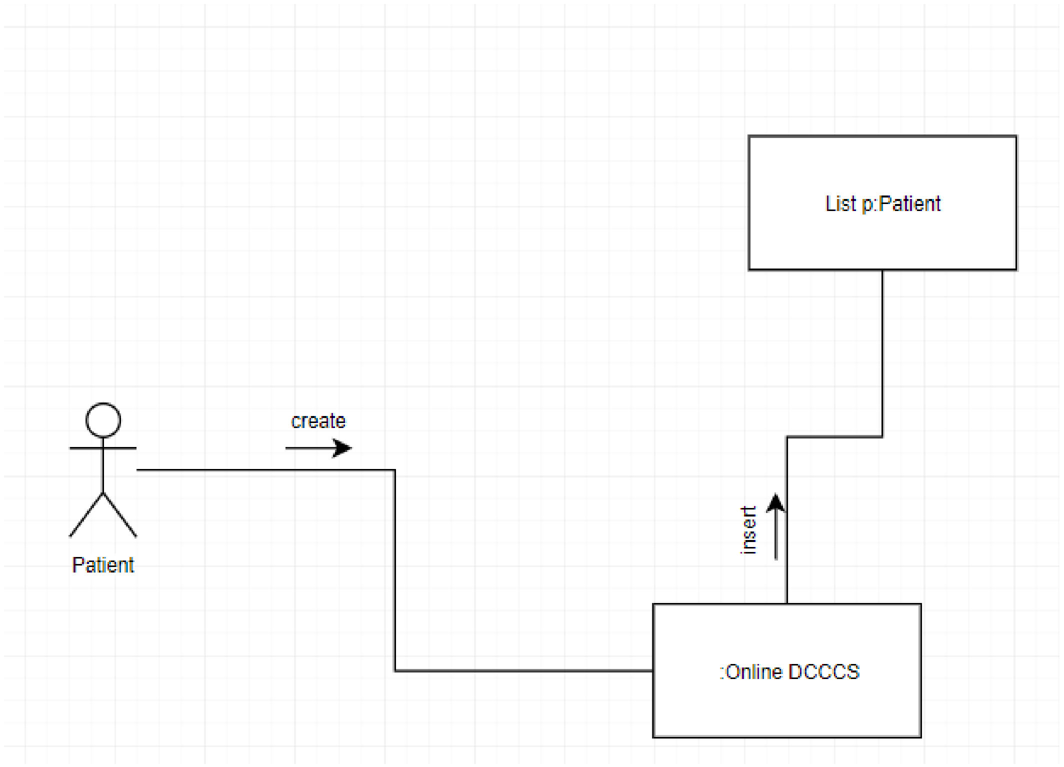
1.1 Dynamic Behavior

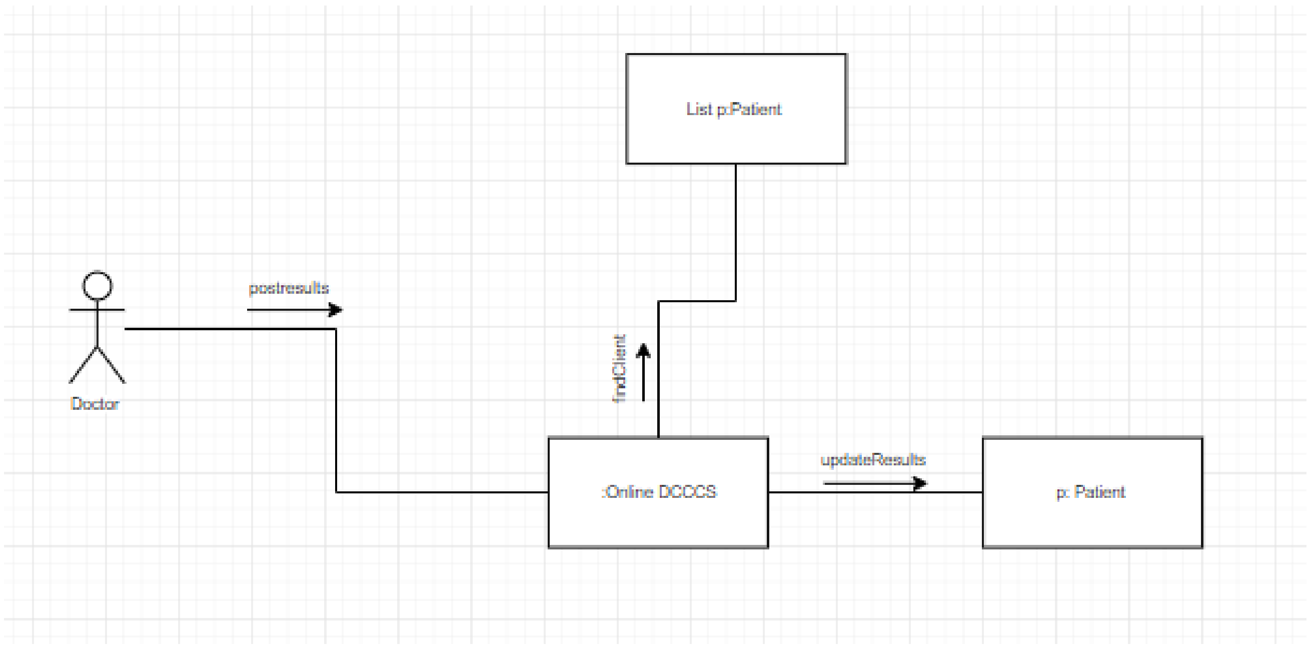
Sequence diagrams:



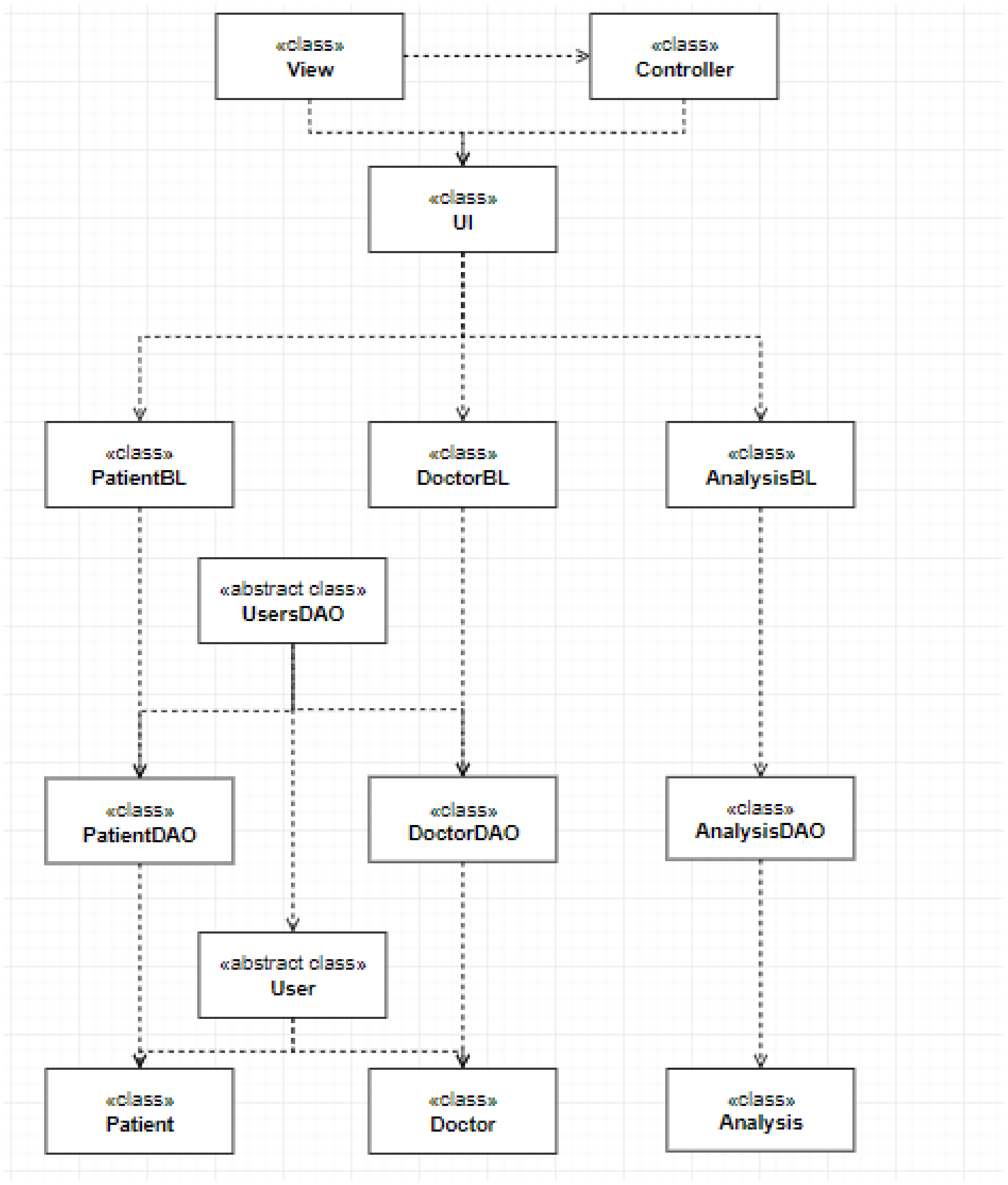


Comunication diagrams:

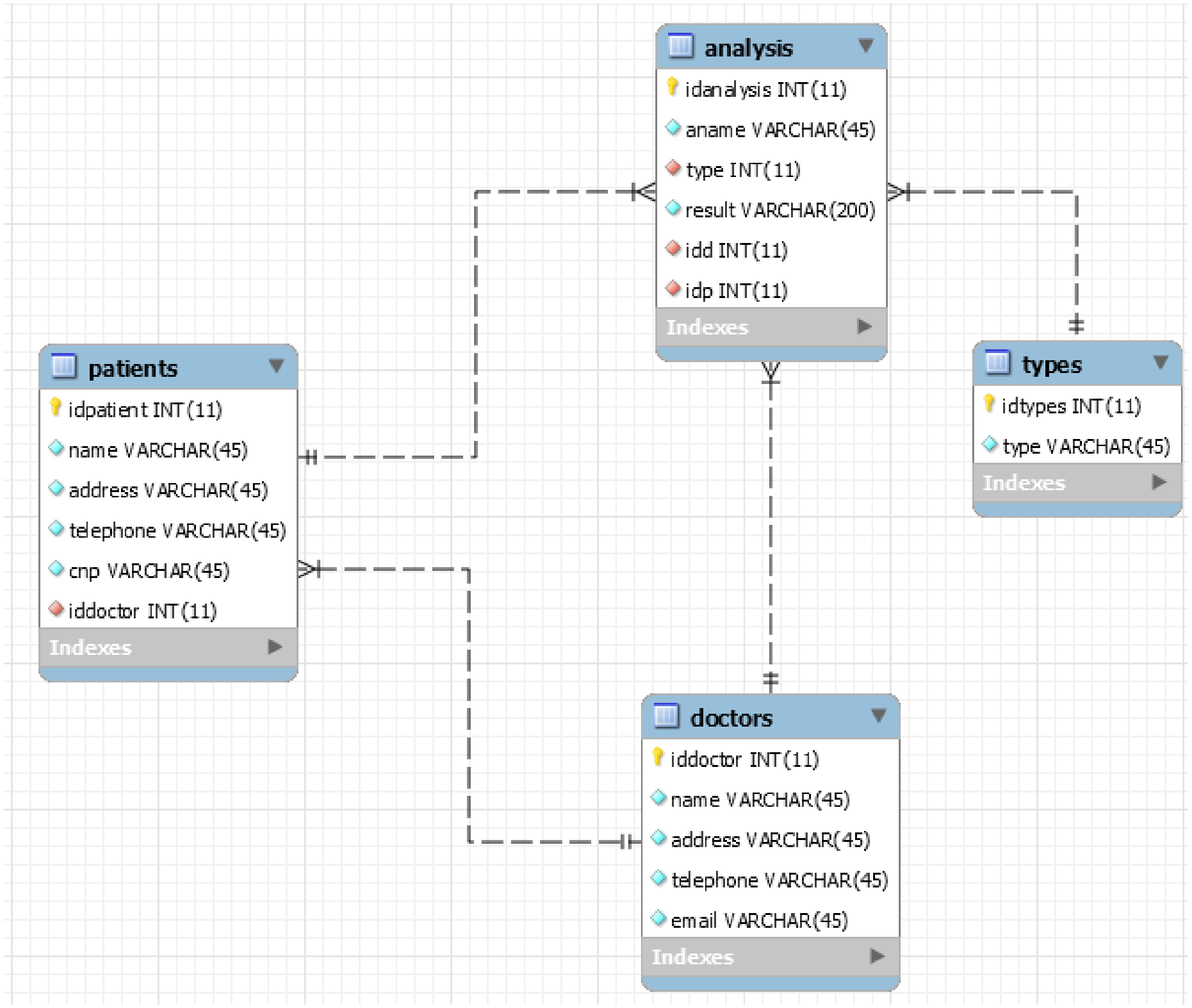




1.2 Class Design



2. Data Model



3. Unit Testing

The Unit Testing done in this project consisted of writing methods/classes and then testing them. Whenever an error appeared or something didn't go as planned System.out.println() were inserted in the code to resolve the errors or inconsistencies.

IV. Elaboration – Iteration 2

1. Architectural Design Refinement

1.1 Conceptual architecture

After documenting myself more, (and doing the 3 assignments and seeing the requirements), the architectural patterns that I have chosen to use in this project are: a client-server architecture with the following design patterns: table module pattern, and a Dao pattern, for creating the entities I also have used a Builder pattern. The user interface will be implemented using an MVC architecture. For better managing the repositories I used the Abstract Factory design pattern.

Client Server: 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. This system shares computing resources.

Dao: access to data varies depending on the source of the data. Access to persistent storage, such as to a database, varies greatly depending on the type of storage (relational databases, object-oriented databases, flat files, and so forth) and the vendor implementation.

Table Module: A single instance that handles the business logic for all rows in a database table or view.

Builder: Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. A Builder class builds the final object step by step. This builder is independent of other objects.

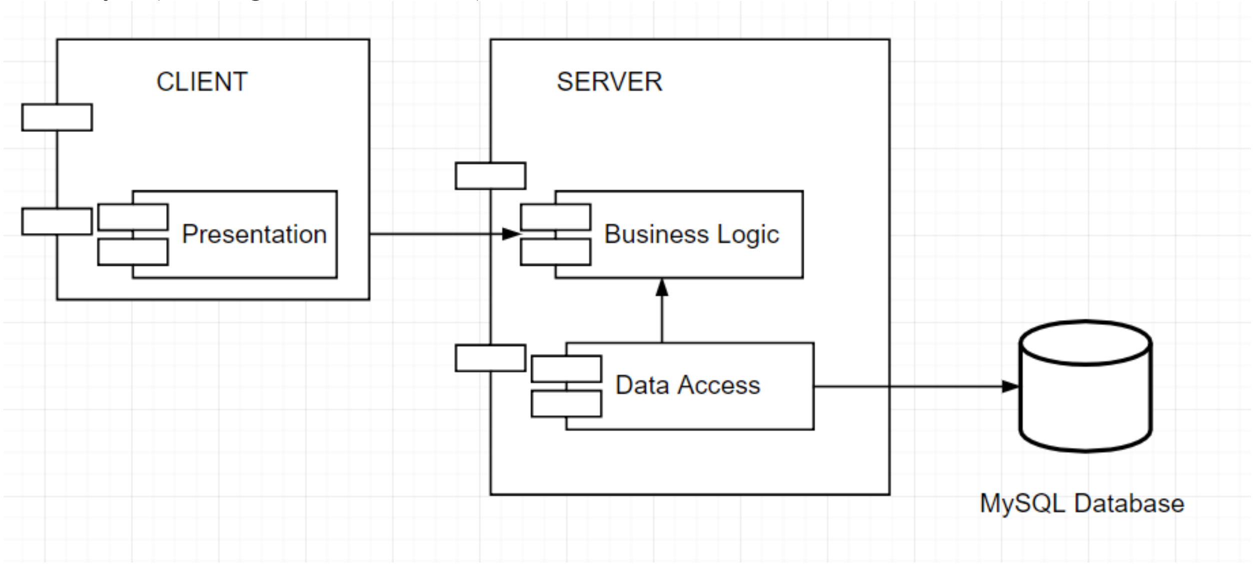
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

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.

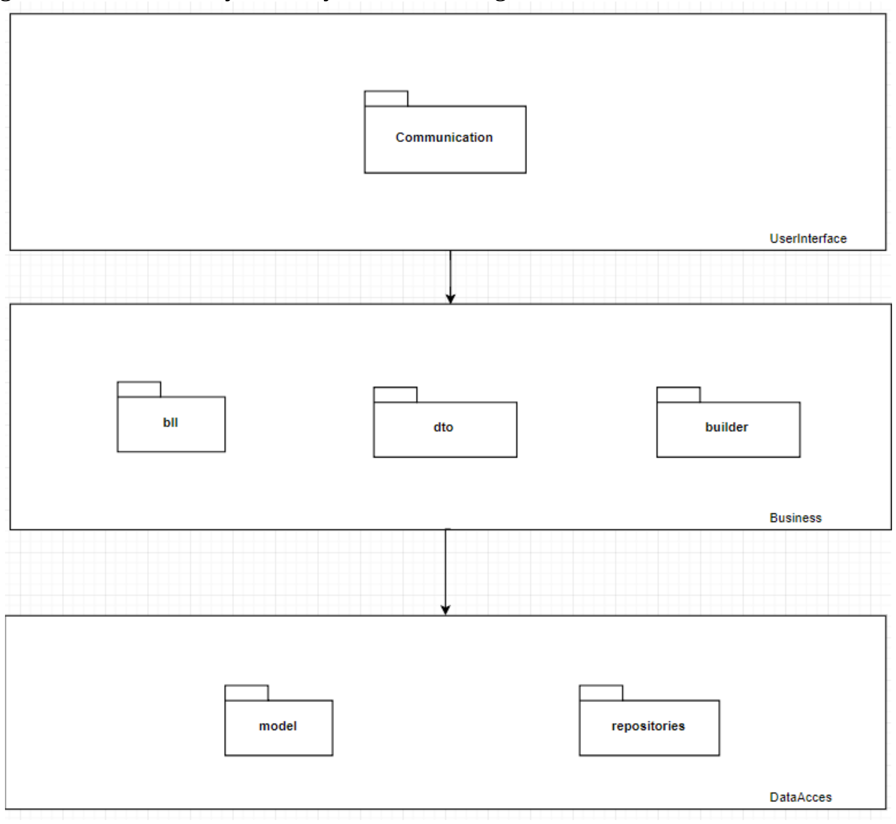
Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

I choose them because the DCCCS will be client server system, Dao and table module patterns will deal with the database needed to store the doctors by using the Builder to create the entities, clients and their analysis (accessing these information's).



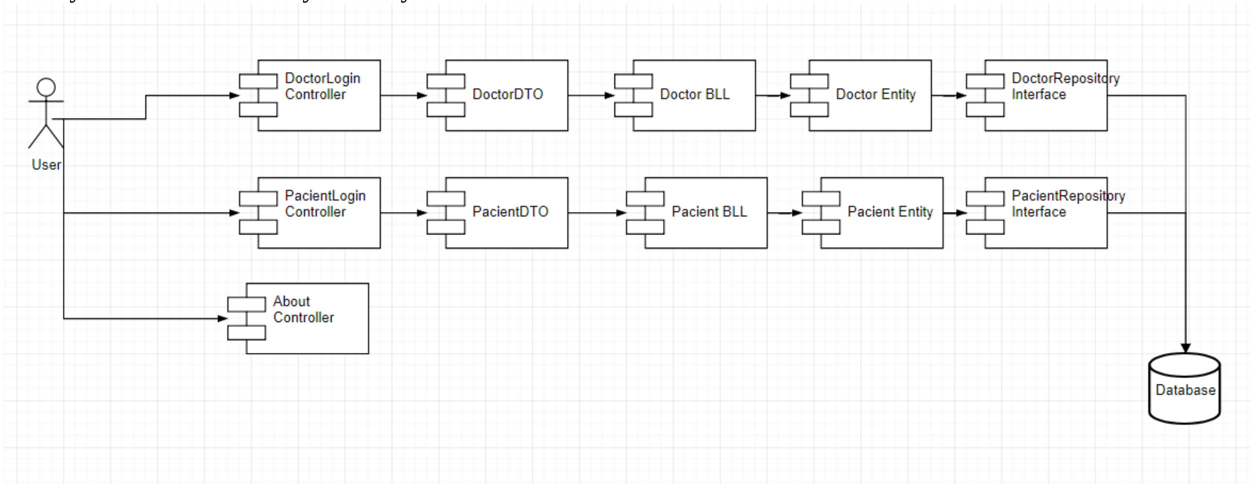
1.2 Package design

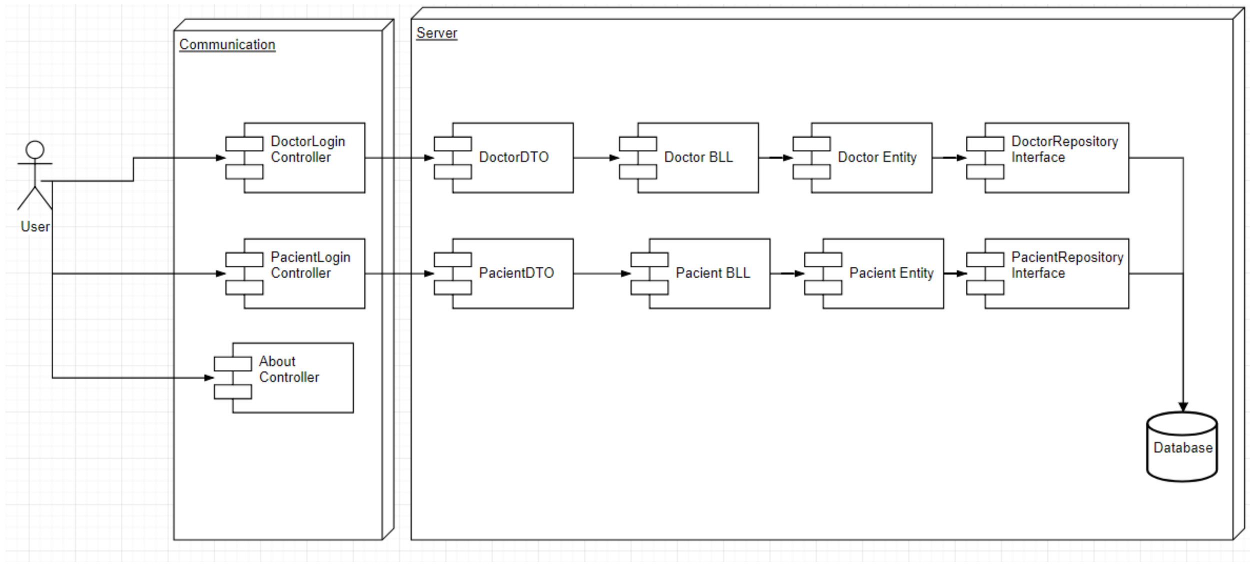
The older diagram was not showing enough implementation of the system, the new one is a more complex one, achieving the full functionality of the system at this stage.



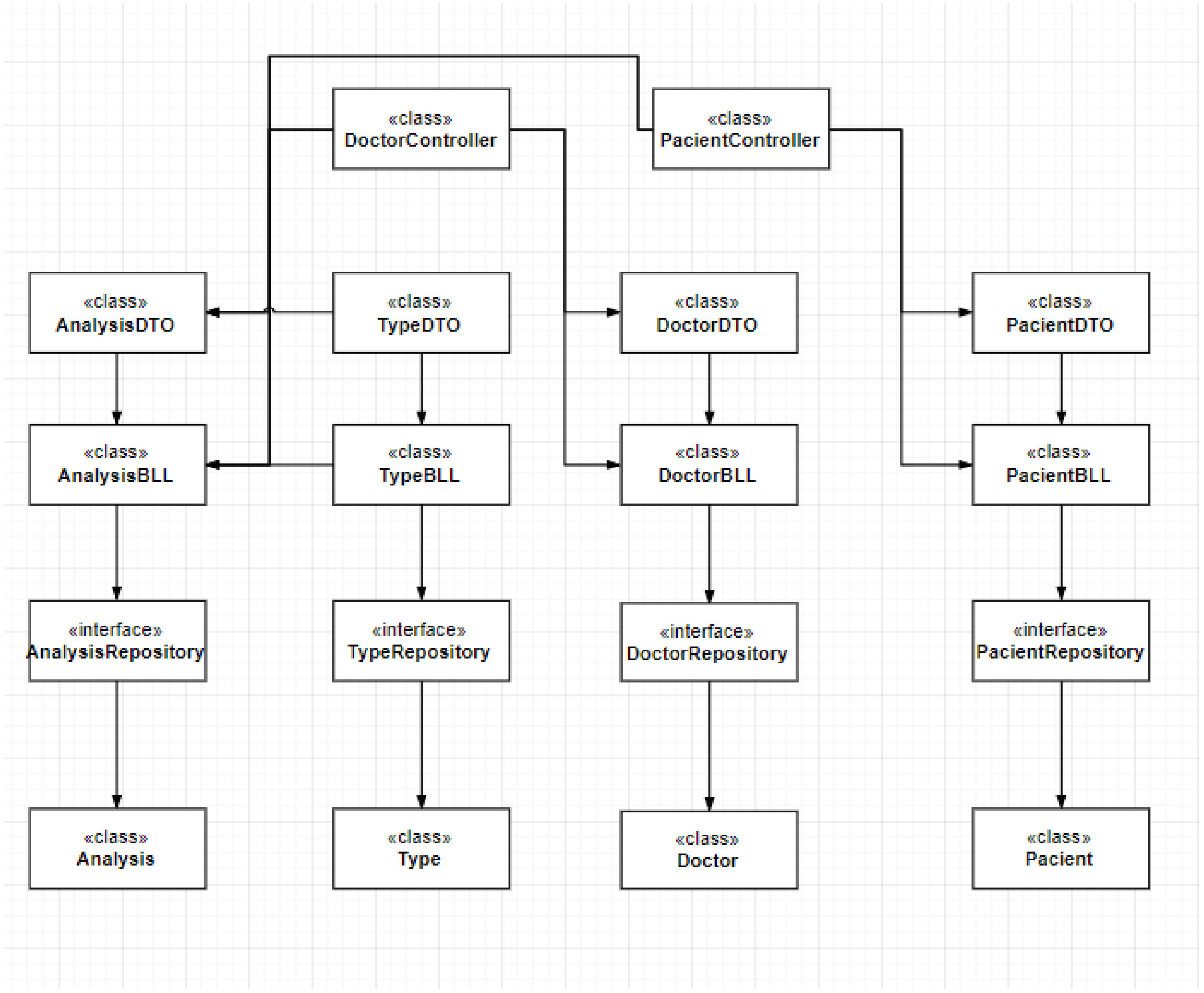
1.3 Component and Deployment Diagrams

As mentioned above, the new diagrams reflect better and more completely the functionality of the system, one can say the new functionality of the system.





2. Design Model Refinement



I chose to implement these classes because of the requirements I had to do for this project and because of what I have learned doing the assignments for this laboratory.

V. Construction and Transition

1. System Testing

The System Testing done in this project consisted of writing methods/classes and then testing them, after that testing if the higher part (who used the method/class) worked properly. Whenever an error appeared or something didn't go as planned System.out.println() were inserted in the code to resolve the errors or inconsistencies.

2. Future improvements

Some possible future improvements possible for my project are: enabling the doctor to add multiple types when creating an analysis. Also one big improvement would be to do a secure login.

VI. Bibliography

https://www.w3schools.com
https://imgbb.com/
https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html
https://www.tutorialspoint.com
https://bootsnipp.com