Blood Bank

Analysis and Design Document

Student: Danila Vlad-Mihai

**Group: 30432**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 04/04/2018 | 1.0 | Preliminary domain model, architectural design and deployment diagram | Dănilă Vlad-Mihai |
| 25/04/2018 | 1.1 | Design model, data model | Dănilă Vlad-Mihai |
| 14/05/2018 | 1.2 | Design patterns added, required changes made | Dănilă Vlad-Mihai |
| 15/05/2018 | 1.3 | Data model updated | Dănilă Vlad-Mihai |
| 21/05/2018 | 1.4 | Class diagram and conceptual architecture improved | Dănilă Vlad-Mihai |

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 5

2.3 Deployment Diagram 6

6

III. Elaboration – Iteration 1.2 6

1. Design Model 6

1.1 Dynamic Behavior 6

1.2 Class Design 7

2. Data Model 8

3. Unit Testing 9

IV. Construction and Transition 9

1. System Testing 9

2. Future improvements 9

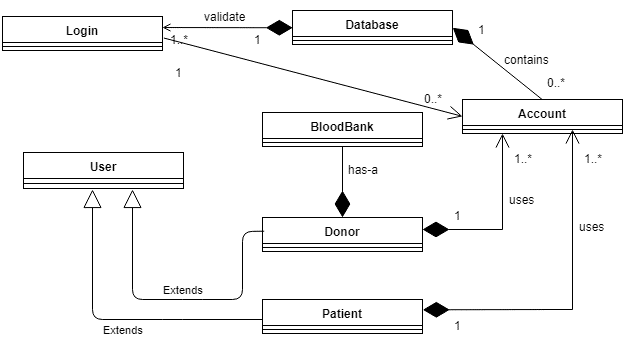
V. Bibliography 9

# Project Specification

The Blood Bank project is a system whose intent is to create a windows-based tool designed for people involved or willing to be involved in blood transfusion events. The users details will be stored in a Relational Database Management System.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

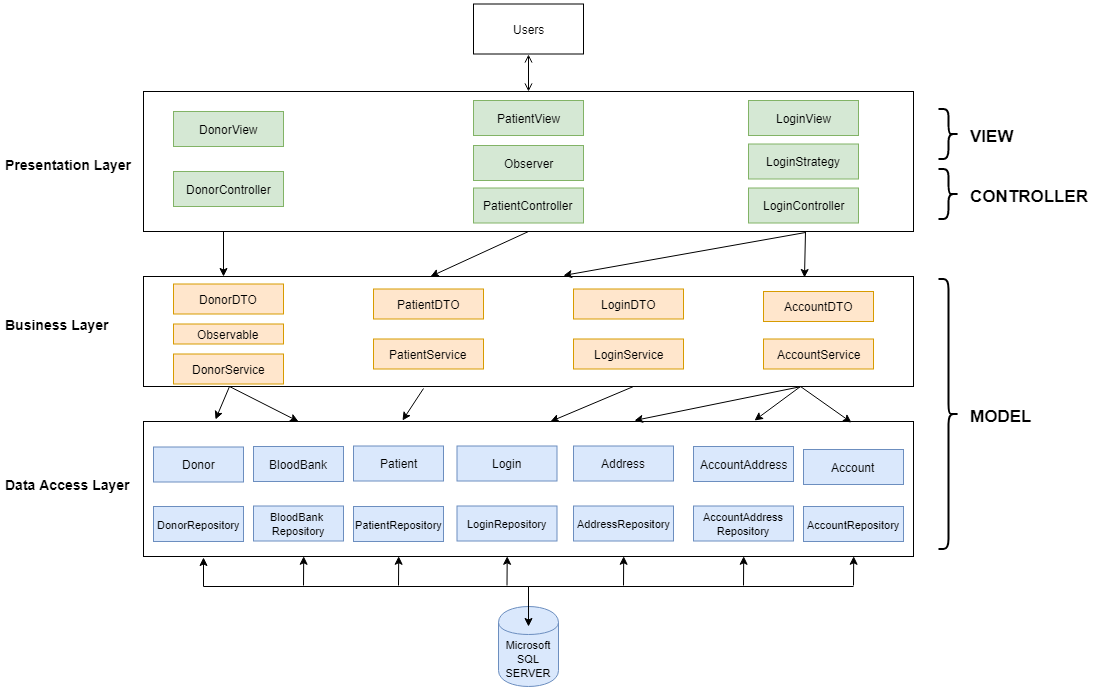
## Conceptual Architecture

For this system, the Layers Architectural Pattern is going to be used. We will group the logical functionalities of the application from the technical point of view as follows: Presentation Layer, Business Layer and Data Layer. This division is performed to increase maintainability, readability, reusability and to minimize the number of overlapping functionalities across the entire application.

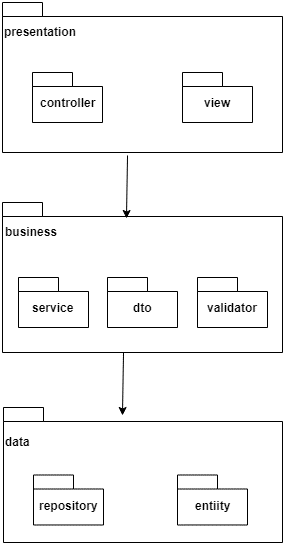
* Presentation Layer: provides the application’s user interface.
* Business Layer: implements the business functionality of the application.
* Data Layer: provides access to database.

MVC (Model-View-Controller) architectural pattern will also be used to divide the system into three interconnected parts:

* Model represents the data and business logic of the application.
* View module is responsible to display data i.e. it represents the presentation.
* Controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.



## Package Design



## Deployment Diagram

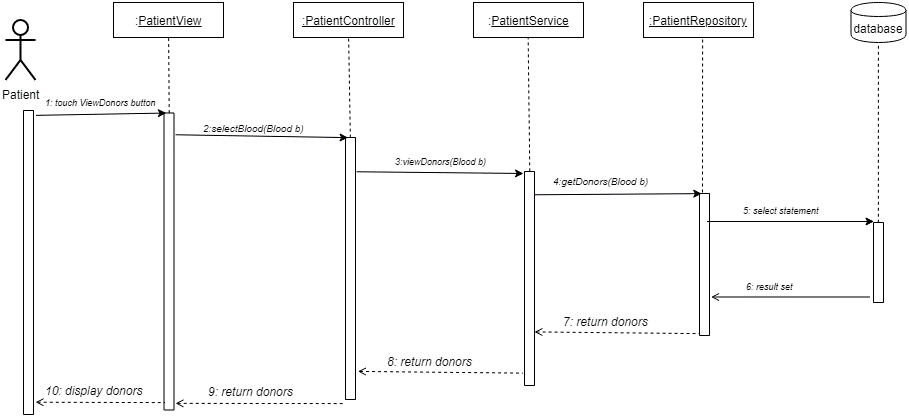
# 

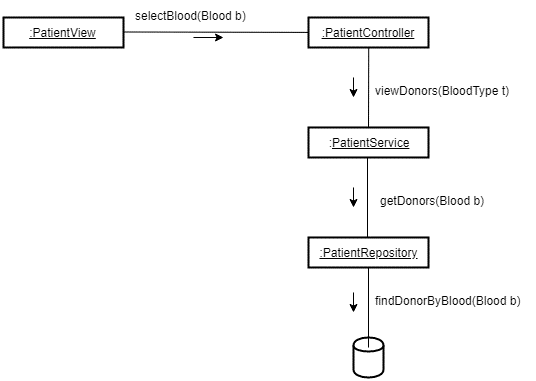
# Elaboration – Iteration 1.2

# Design Model

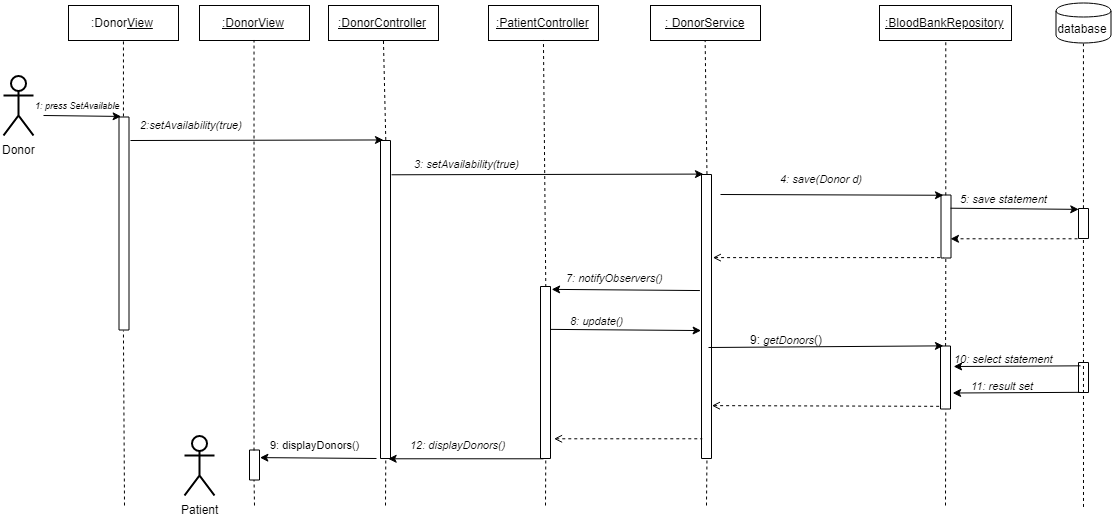
## Dynamic Behavior

Scenario: Search for a compatible donor

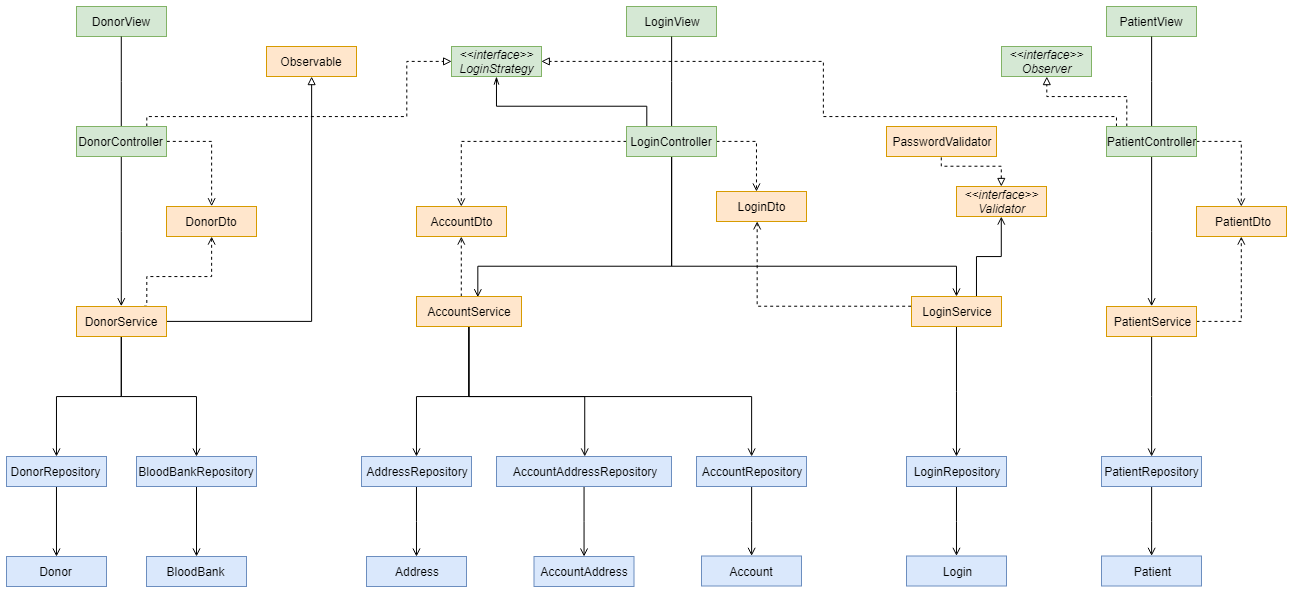




Scenario: Set donor as available



## Class Design



**Observer Design Pattern** will be used. This defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

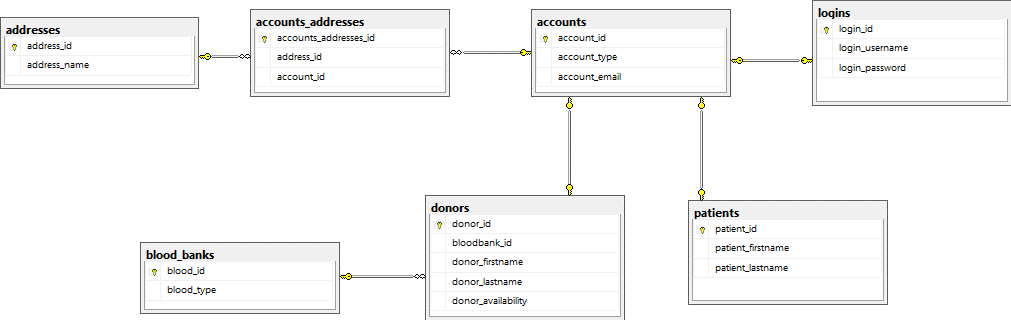
A specific change in a donor(observable) details will be notified to the patient(observer) waiting for that specific change(i.e. a new donor appears with the needed blood group).

**Strategy Design Pattern** encapsulates an algorithm inside a class, letting the algorithm vary independently from clients that use it. In our situation, it’s necessary to process login event according to the type of user: patient or donor. The donor and patient controller classes will be responsible for encapsulating the ‘algorithm’ of handling the flow following the login confirmation.

**Builder Design Pattern** separates the construction of a complex object from its representation so that the same construction process can create different representations. Using this for some entities, we will make their creation more flexible.

# Data Model

The persistence data will be modeled using Relational Database Model. Microsoft SQL Server Database Management System will be used.



# System Testing

Unit testing will be performed using JUnit and Mockito frameworks. Tests will be written to verify that a relatively small individual piece of code is doing what it is intended to do. Individual methods belonging to repository and business classes will be tested individually, by checking that the actual output matches the expected output.

Integration testing will be performed using JUnit for the data layer. The repositories will be tested without altering the database.

# Future improvements

The application can be improved by:

* More intuitive user interface
* Direct communication between patients and donors
* Username and password encryption
* Check in donation
* Block the donors availability for 6 months after a blood donation was performed
* Add application rewards based on performing donations, on check in.

# Bibliography

<https://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial>

<https://springframework.guru/configuring-spring-boot-for-microsoft-sql-server/>

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

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

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