Assignment 3

Student: Sorin Urcan

**Group: 30442**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The objective of this assignment is to allow students to become familiar with the client-server architectural style and the Observer design pattern.

# Functional Requirements

Use Java/C# API to design and implement a client-server application for managing the consultations of doctors in a clinic. The application has three types of users: the clinic secretary, the doctors and an administrator.

The clinic secretary can perform the following operations:

* Add/update patients (patient information: name, identity card number, personal numerical code, date of birth, address).
* CRUD on patients’ consultations (e.g. scheduling a consultation, assigning a doctor to a patient based on the doctor’s availability).

The doctors can perform the following operations:

* Add/view the details of a patient’s (past) consultation.

The administrator can perform the following operations:

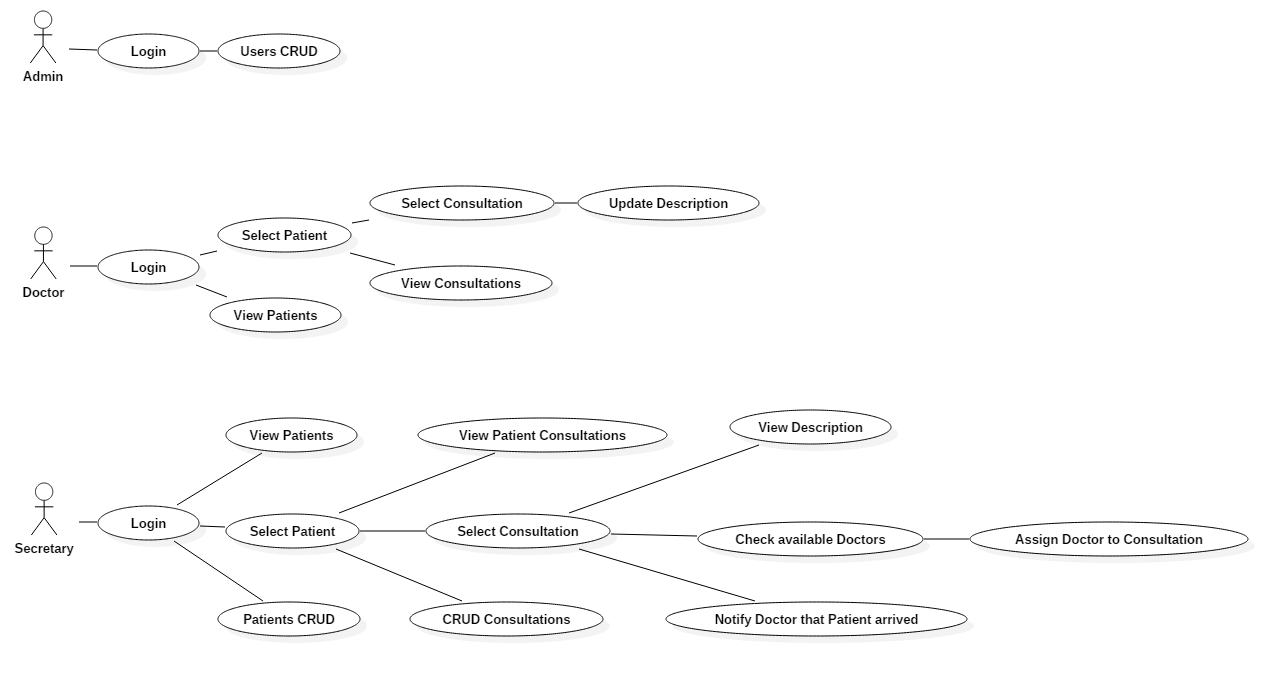
* CRUD on user accounts.

In addition, when a patient having a consultation has arrived at the clinic and checked in at the secretary desk, the application should inform the associated doctor by displaying a message.

# Non-functional Requirements

* The application should be client-server and the data will be stored in a database. Use the Observer design pattern for notifying the doctors when their patients have arrived.
* All the inputs of the application will be validated against invalid data before submitting the data and saving it.

2. Use-Case Model



*Use case: Add User*

*Level: user-goal level*

*Primary actor: Admin*

*Main success scenario:*

- Complete Username and Password fields

- Successfully log in into the Assignment 3 application

- A table containing user fields to complete will appear

- Click the Add User button

- User is created and displayed

*Extensions:*

- Wrong Username or Password will not allow you to enter the application

- Empty user fields will not allow you to add new user

- Used new username will result in a pop-up dialog

*Use case: Remove User*

*Level: user-goal level*

*Primary actor: Admin*

*Main success scenario:*

- Complete Username and Password fields

- Successfully log in into the Assignment 3 application

- A table containing user fields to complete will appear

- Click the Remove User button

- User is deleted and no longer displayed

*Extensions:*

- Wrong Username or Password will not allow you to enter the application

- Empty user fields will not allow you to delete user

*Use case: Assign Doctor*

*Level: user-goal level*

*Primary actor: Secretary*

*Main success scenario:*

- Complete Username and Password fields

- Successfully log in into the Assignment 3 application

- A table containing patient’s fields will appear

- Insert patient ID and press *Select Patient*

- A table containing patient’s consultation will appear

- Insert consultation ID and press *Select Consultation*

- Press *Check for available doctors* button

- Insert a available doctor from the new table with available doctors and press *Assign Doctor* button

- A doctor will be assigned to the selected Consultation

*Extensions:*

- Wrong Username or Password will not allow you to enter the application

- No Patient ID selected will stop for proceeding

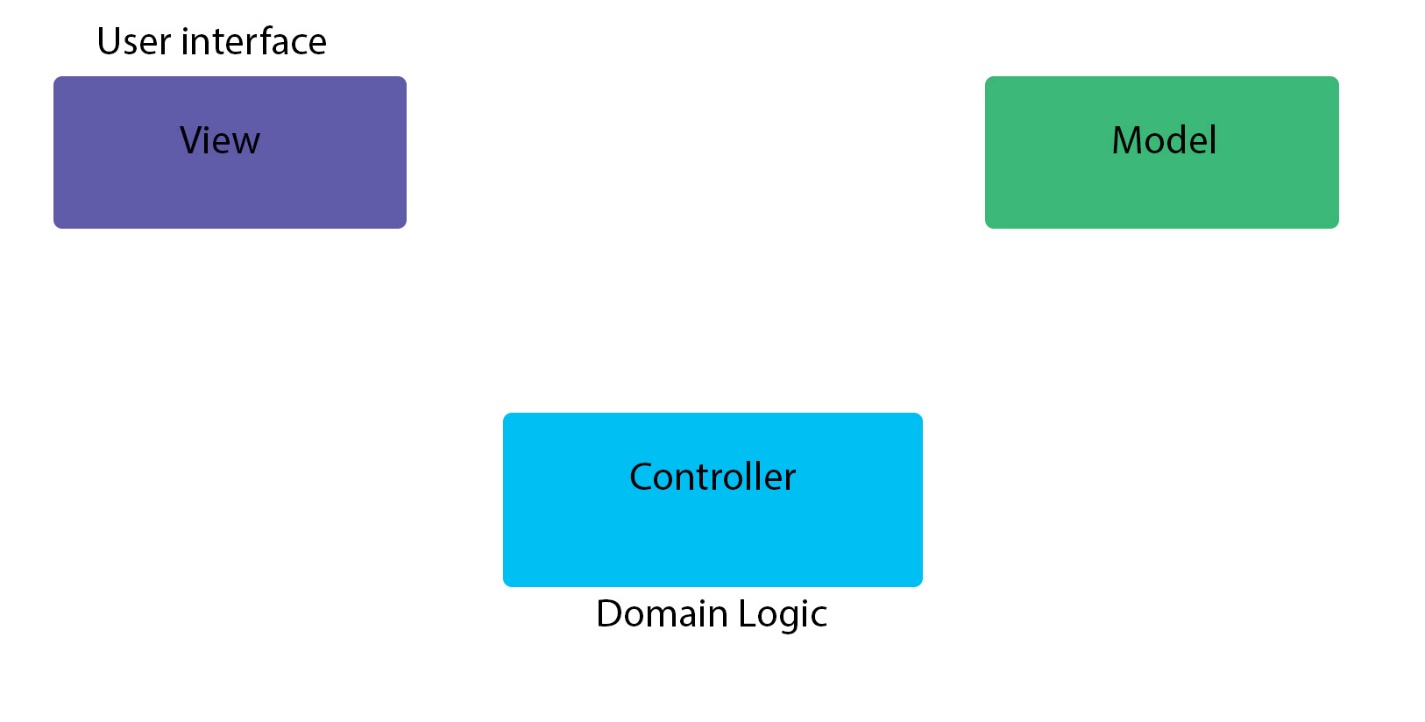
- No Consultation ID selected will stop for proceeding

- No Doctor ID selected will stop for proceeding

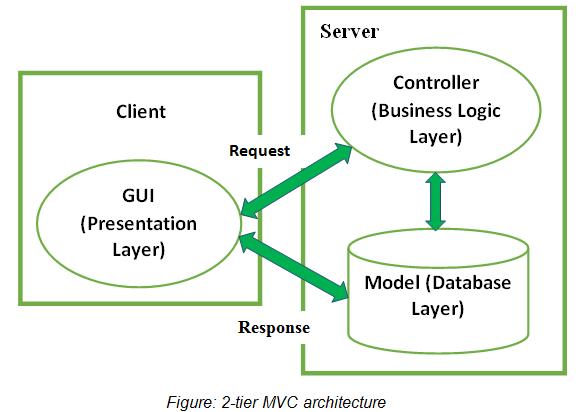
3. System Architectural Design

**3.1 Architectural Pattern Description**

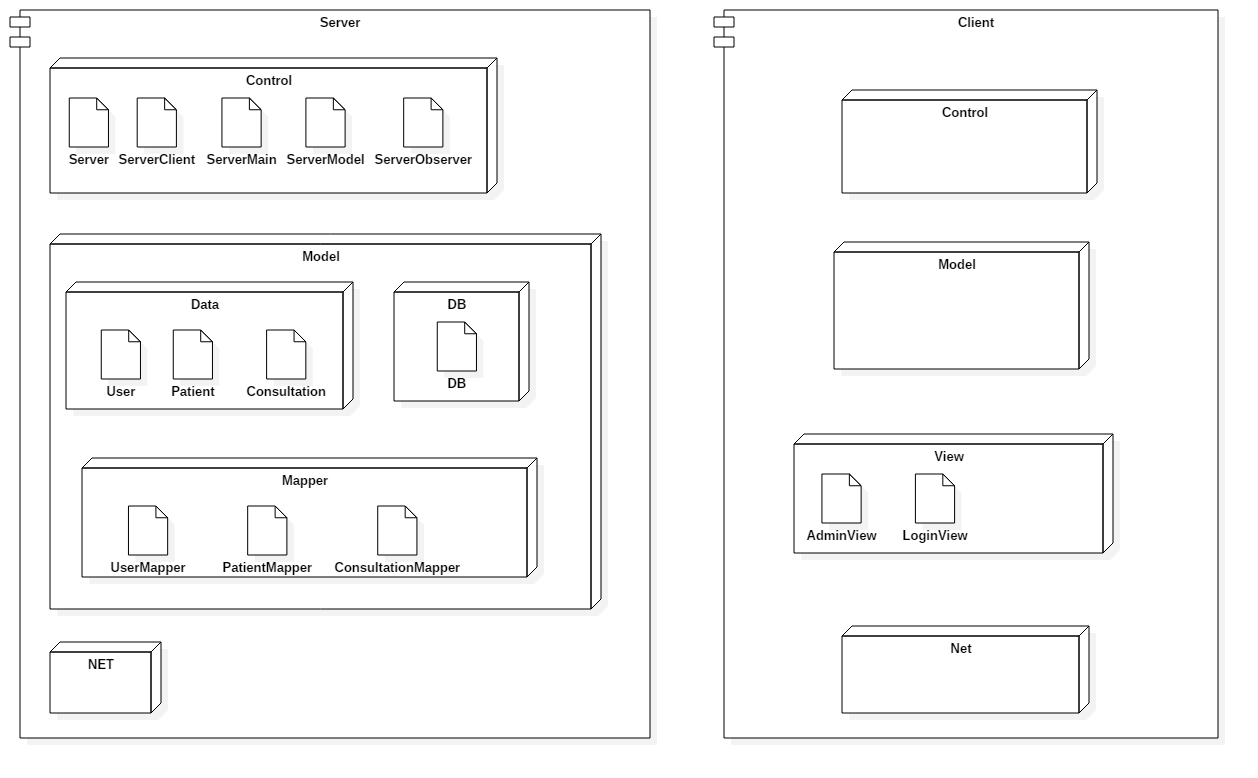
The chosen architectural pattern is the model-view-controller. It fits the requirements very well, since it divides the software into three main parts, the model, the view and the controller. For our system, the model is represented by the User, Patient, Consultation, DB, UserMapper, PatientMapper, ConsultationMapper classes, the view is represented by the user interface and the controller is the domain logic of the system.

These three components interact with each other. The controller can send commands to the model to change its state, or send commands to the view in order to update its presentation of the model. The view requests information from the model through the controller. By interacting with the view, the user sends commands through the controller, which can modify the model or the view. 

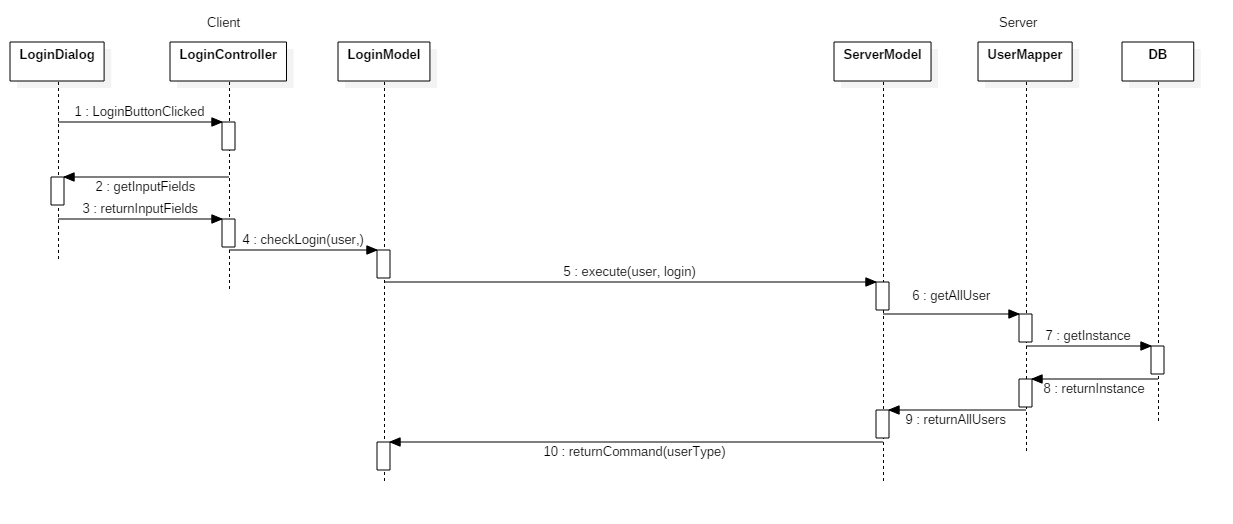
The architectural style is client-server based. This facilitates storage of large amounts of data, data which can be accessed remotely. The database would contain several tables, for holding user account information, client information, hotel rooms booking, client history. Patterns used in the domain logic are: DataMapper. In our implementation, it is useful in instantiating only once the Database connection. The DataMapper creates or modifies an instance of a model, communicating with the database in an indirect way.



**3.2 Diagrams**



4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

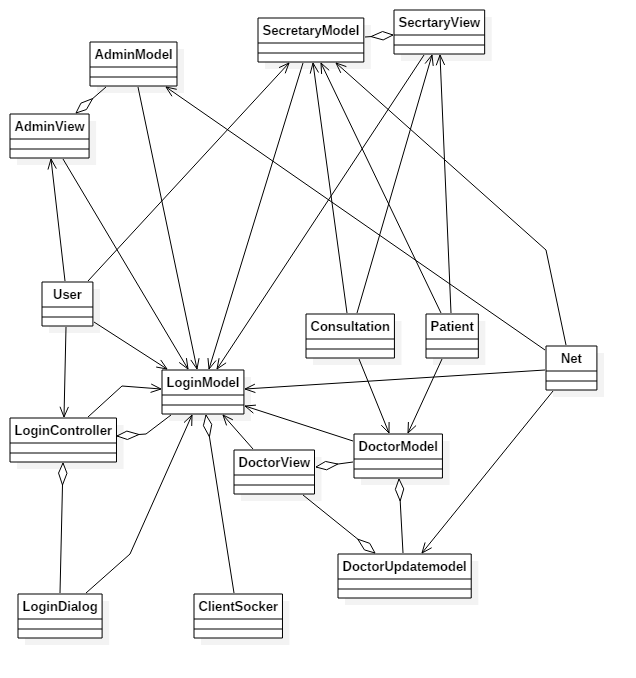
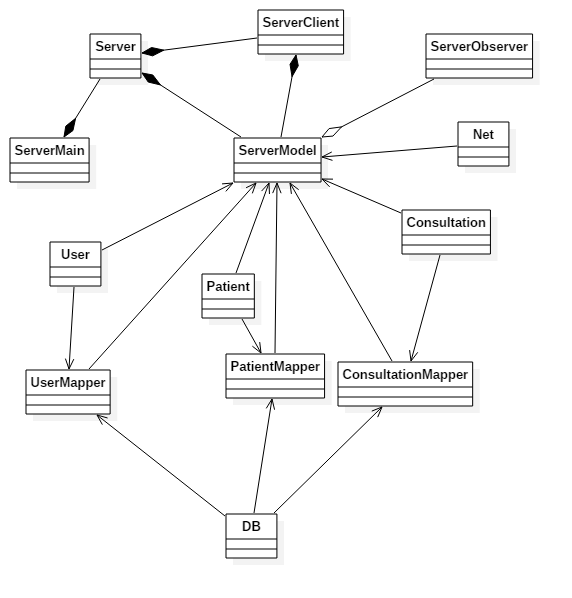
Define an object that is the "keeper" of the data model and/or business logic (the Subject). Delegate all "view" functionality to decoupled and distinct Observer objects. Observers register themselves with the Subject as they are created. Whenever the Subject changes, it broadcasts to all registered Observers that it has changed, and each Observer queries the Subject for that subset of the Subject's state that it is responsible for monitoring.

This allows the number and "type" of "view" objects to be configured dynamically, instead of being statically specified at compile-time.

The protocol described above specifies a "pull" interaction model. Instead of the Subject "pushing" what has changed to all Observers, each Observer is responsible for "pulling" its particular "window of interest" from the Subject. The "push" model compromises reuse, while the "pull" model is less efficient.

Issues that are discussed, but left to the discretion of the designer, include: implementing event compression (only sending a single change broadcast after a series of consecutive changes has occurred), having a single Observer monitoring multiple Subjects, and ensuring that a Subject notify its Observers when it is about to go away.

**5.2 UML Class Diagram**



6. Data Model

User stores username, password and userType.

CREATE TABLE `users` (

`username` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

`type` varchar(45) NOT NULL,

PRIMARY KEY (`username`))

Patients stores an ID, name, cnp, birthday, address and Consultation stores ID, PatientID, DoctorID, Date and description.

All are stored in a database.

7. System Testing

Input validation.

8. Bibliography

<http://docs.oracle.com/javase/tutorial/uiswing/>