Hospital Application

Student: Gliga-Hambet Bogdan Andrei

**Group:30431**

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 designed application is designed for the front desk employees of a hospital. It is designed to allow secretary/doctors to manage the consultations of the hospital’s patients.

# Functional Requirements

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 is a web-app which can be accessed via the browser, as long as the user has an active interned connection. It must have persistent storage in which the data is stored such that it is not lost.

2. Use-Case Model

C:\Users\GligaBogdan\Downloads\Untitled Diagram.png

Use case: Add patient consultation

Level: User-goal level

Primary actor: Application Secretary

Main success scenario: Login with user having permission rights => View All Patient List => Select desired patient => Click “Add consultation” button = > Input with valid data

Extensions: Failure in case of corrupted data in the Database

3. System Architectural Design

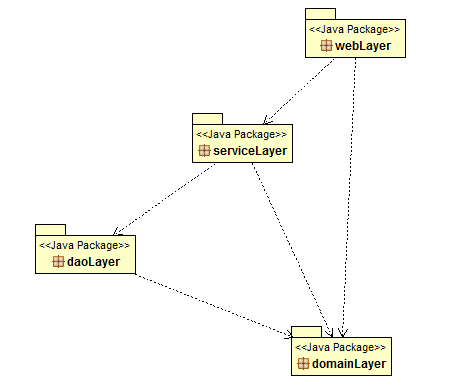
**3.1 Architectural Pattern Description**

Client-server architecture (client/server) is a [network](http://www.webopedia.com/TERM/N/network.html)[architecture](http://www.webopedia.com/TERM/A/architecture.html) in which each [computer](http://www.webopedia.com/TERM/C/computer.html) or process on the network is either a [*client*](http://www.webopedia.com/TERM/C/client.html) or a [*server*](http://www.webopedia.com/TERM/S/server.html). [Servers](http://www.webopedia.com/TERM/S/server.html) are powerful computers or processes [dedicated](http://www.webopedia.com/TERM/D/dedicated.html) to managing [disk drives](http://www.webopedia.com/TERM/D/disk_drive.html) ([*file servers*](http://www.webopedia.com/TERM/S/server.html)), [printers](http://www.webopedia.com/TERM/P/printer.html) ([*print servers*](http://www.webopedia.com/TERM/S/server.html)), or network [traffic](http://www.webopedia.com/TERM/T/traffic.html) ([*network*](http://www.webopedia.com/TERM/N/network.html) [servers](http://www.webopedia.com/TERM/S/server.html)). Clients are [PCs](http://www.webopedia.com/TERM/P/PC.html) or [workstations](http://www.webopedia.com/TERM/W/workstation.html) on which users run [applications](http://www.webopedia.com/TERM/A/application.html). Clients rely on servers for [resources](http://www.webopedia.com/TERM/R/resource.html), such as [files](http://www.webopedia.com/TERM/F/file.html), [devices](http://www.webopedia.com/TERM/D/device.html), and even processing power.

**3.2 Diagrams**

More specific, the layers used in this application were the: presentation-layer (consisting of JSPs), web-layer (containing all the Spring Controllers), service-layer (containing the Spring Services handling the business logic), the dao-layer (containing DAOs for database interaction) and the domain-layer ( being consisted of simple POJOs)

As it can be noticed from the package diagram, layers only talk to the layer below them in order to keep a clear separation of functionality in the application



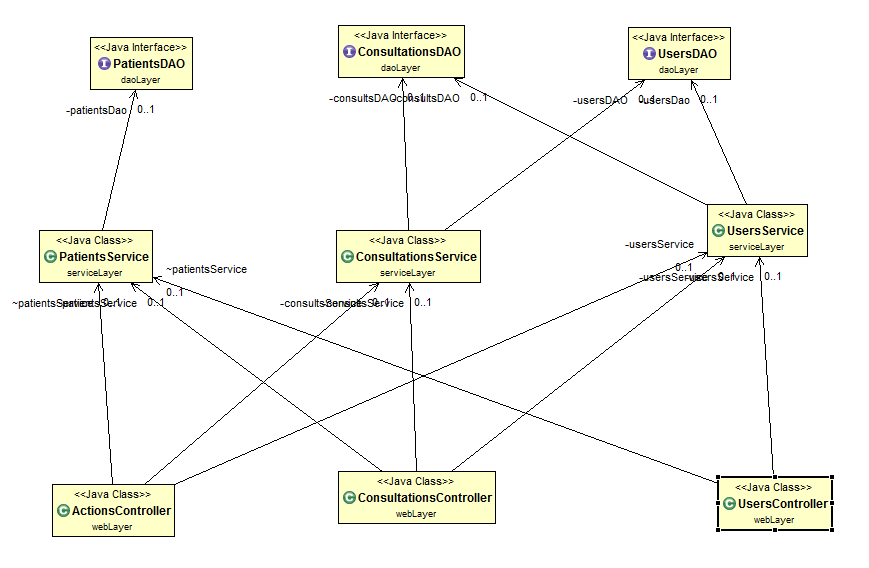
4. UML Sequence Diagrams

This is the sequence diagram for generating a report by the secretary to get all the consultations that a specific patient has.

C:\Users\GligaBogdan\Downloads\Untitled Diagram (1).png

5. Class Design

**5.1 UML Class Diagram**



6. Data Model

**6.1 User**

This is used to model a real time user of the application. It has different fields on it such as the name, username, password, etc. It is the main model associated with each application user and it is by its attributes that the level of operations he can perform is decided.

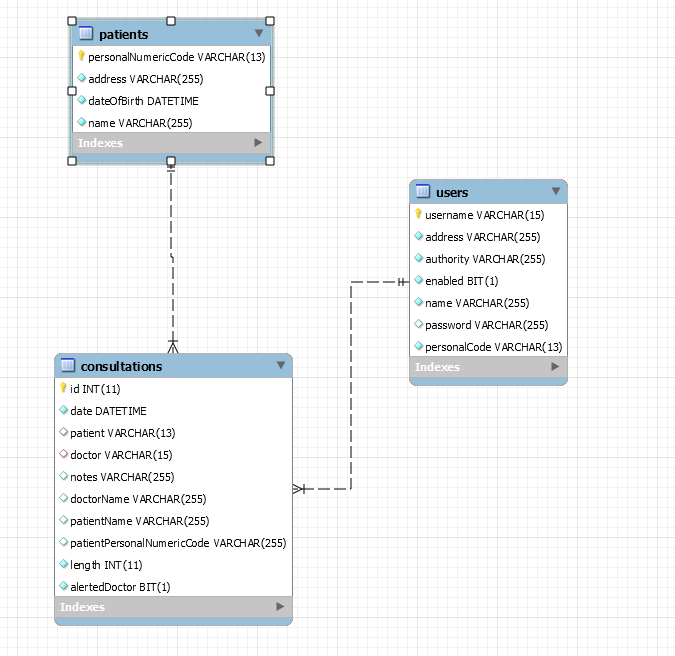
**6.2 Patient**

The patient class models a real life patient of this particular hospital. It is used to keep track of what consultations he had and to which doctors over time. It has fields such as name, CNP, date of birth and address.

**6.3 Consultation**

The consultations are modeled by the Consultation class which keeps information about the time and date when the patient visited the hospital and about the doctor which treated him, as well as information about the diagnostic of this visit.

**6.4 Schema Model**



7. System Testing

The application has been tested using JUnit tests, as they provide an easy, robust and reliable method to test your application. The main features that have been testes are the classes in the DAO Layer, as they handle the database operations and they are the most prone to fail.

A Test Suite has been designed for each of the 3 DAO classes, namely: ConsultationsDAOTests, PatientDAOTests and UsersDAOTests. Each one of these has methods which independently test CRUD operations which are performed by the application. Before each test, the database is cleaned and populated with validation data and after each test the database is cleaned again in order to leave it in a stable state and not affect other Junit tests

8. Bibliography

<http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/htmlsingle/spring-framework-reference.html>

<http://hibernate.org/orm/documentation/5.1/>

<https://tiles.apache.org/framework/tutorial/configuration.html>

<https://spring.io/guides/gs/messaging-stomp-websocket/>

<https://developer.mozilla.org/en/docs/AJAX>