<Doctor Web Application>
Analysis and Design Document

Student: Bogdan Stupariu
Group: 30432

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

[Application description]

This assignment requests to design and implement an application for a doctor's office and has as main objective making students familiar with sockets, communication between server and client. The application should have three types of users: an administrator whose main attributes will be to manage the database and other users accounts, a secretary, whose role will be to manage patients clients, patients consultations and the relation between patients and doctors, and doctor, whose main role will be to be notified when a consultation should start, add details to consultations, or even see all consults.

1.2 Functional Requirements

[Present the functional requirements]

The administrator should have the following possibilities: operate CRUD on the secretary and doctors accounts.
The secretary should have the following possibilities: operate CRUD on patients, operate on consultations – create new consult, edit existing consult, delete existing consult, assign patient's consult to a specific doctor, search for a specific doctor or a patient and finally announce a doctor that the patient arrived.
The doctor should have the following possibilities: be announced that a patient has arrived, set desk board to that consultation, search for patients and search for consultations, add description details to a specific consultation.
From those requirements shown above, there can obviously be deducted that the application should have a login functionality. Each account should have specific access rights related to its account type privileges. The direct access to specific pages without having permission should be restricted.
Also, all the data should be validated against invalid data before submitting any input by the user and so a response functionality for input data should also be required.
In most of the sides of the application, the input given by the user (each type) should generate a response as output, which will be generated in tables organizing data from the database in a readable and pleasurable way.
The most important new aspect of this assignment is related to the client-server aspect, a requirement being set around the notification the secretary sends to the doctor when a patient arrives. So, there should be implemented a real time notification system.

1.3 Non-functional Requirements

[Discuss the non-functional requirements for the system]

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. Accessibility should be provided by both implementing the web application as an online application with the obvious restriction of needing an internet connection, and also giving the possibility of using the application offline, by individually installing all the components on the computer.

2. Use-Case Model

[Create the use-case diagrams and provide one use-case description (according to the format below).
Use-Case description format:
Use case: <use case goal>
Level: <one of: summary level, user-goal level, sub-function>
Primary actor: <a role name for the actor who initiates the use case>
Main success scenario: <the steps of the main success scenario from trigger to goal delivery>
Extensions: <alternate scenarios of success or failure>
]

Use case:
Notify the doctor that the patient has arrived
Level:
Secretary account in the consultations panel. Notification trigger done on the client side, managed in the server side and with the end point again, in the client side -> but another account type / account this time.
Primary actor:
The primary actor is the secretary logged that performs the trigger situation.
Main success scenario:
The "Announce doctor" button is triggered by the secretary after opening a consultation. After that, a http request is sent to the backend in order to update the state of the consultation and to announce so that the consultation should give a notification on the specific doctor. On the doctor's page, when logged in, a notification will pop-up because when checking the consultations states, there will be a consultation with alert flag set. After clicking ok, the page of the doctor will be set to that specific consultation.
Extensions:
This use case may fail in the following case: the consultation it was already checked by the doctor. In this case, a message will pop-up and also the announce doctor button will be disabled.

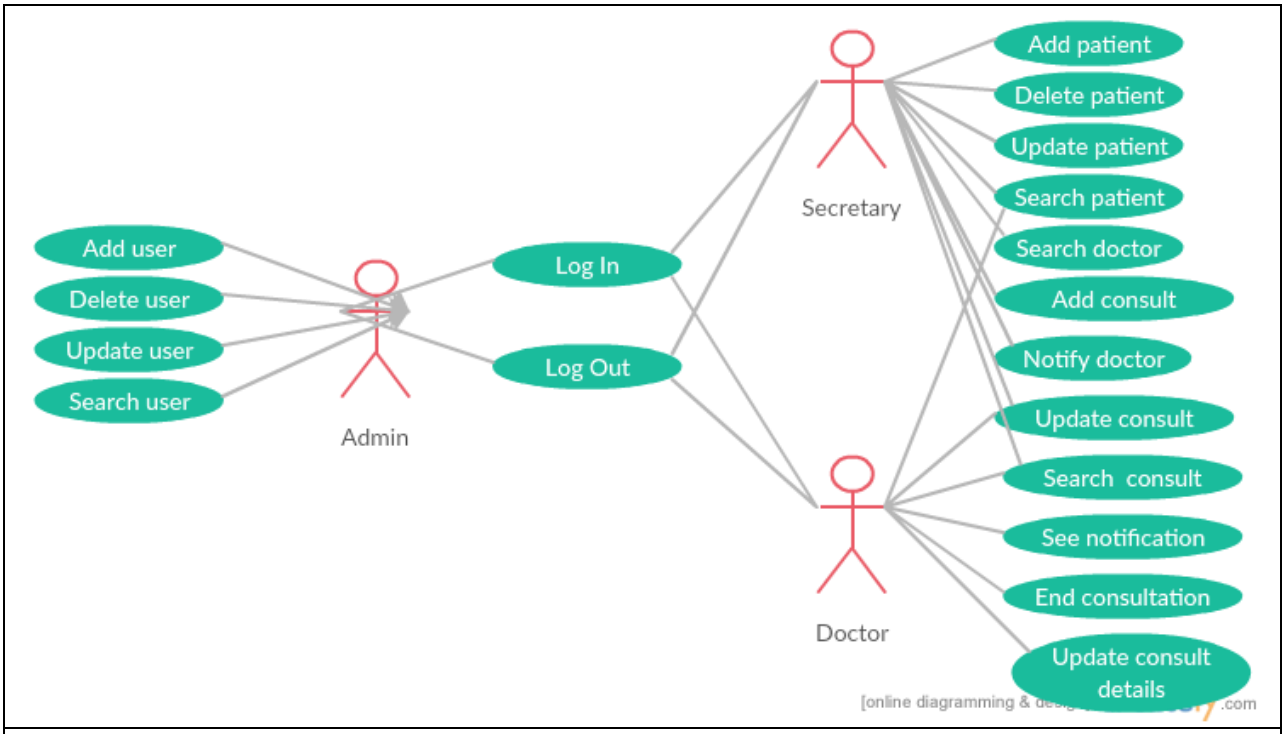

Fig. 1 - USE CASE MODEL

3. System Architectural Design

3.1 Architectural Pattern Description

[Describe briefly the used architectural patterns.]

Layered Architecture - Partitions the concerns of the application into stacked groups (layers). Usage in the backend of Controllers, Services, Entities and repositories. Usage in the frontend of Views, Services and Controllers.
Domain Business logic patters - An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain. Usage of the entities in the backend and

making the business logic in the services side.

Table Data Gateway - The first Data Source pattern is the Table Data Gateway. This pattern is sometimes also referred to as a Data Access Object or Data Access Component (although that terminology is also used to refer to any class used in the Data Access Services layer). Basically, the idea is that the class encapsulates the SQL (statements or stored procedures) for accessing sets of related tabular data in a database. These sets can be queried from multiple tables, views, or joins and so needn't, and typically won't, map to single tables.

Request–response, or request–reply, is one of the basic methods computers use to communicate with each other, in which the first computer sends a request for some data and the second computer responds to the request. Usually, there is a series of such interchanges until the complete message is sent; browsing a web page is an example of request–response communication. Request–response can be seen as a telephone call, in which someone is called and they answer the call.

3.2 Diagrams

[Create the system's conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]

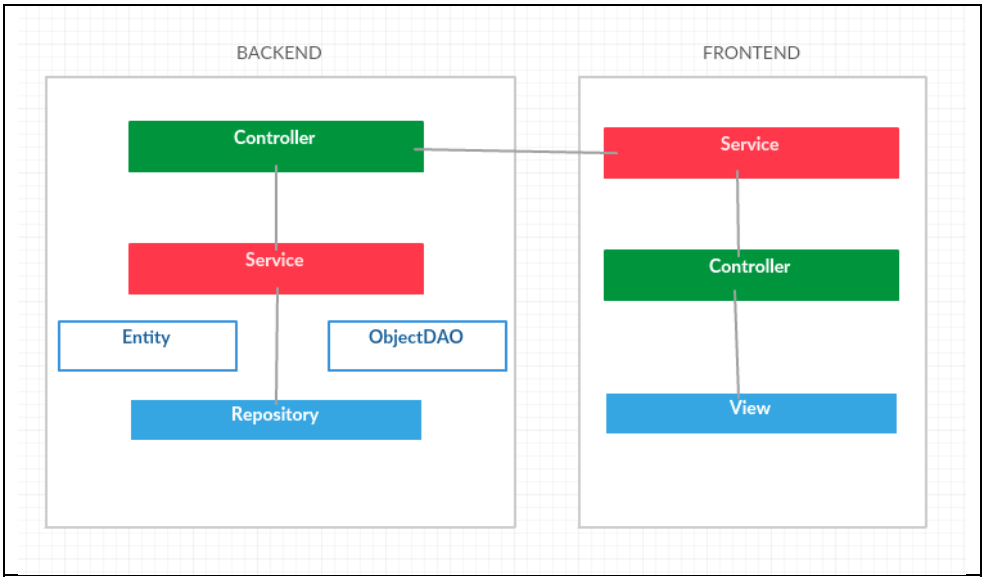

Fig. 2 – MVC Pattern

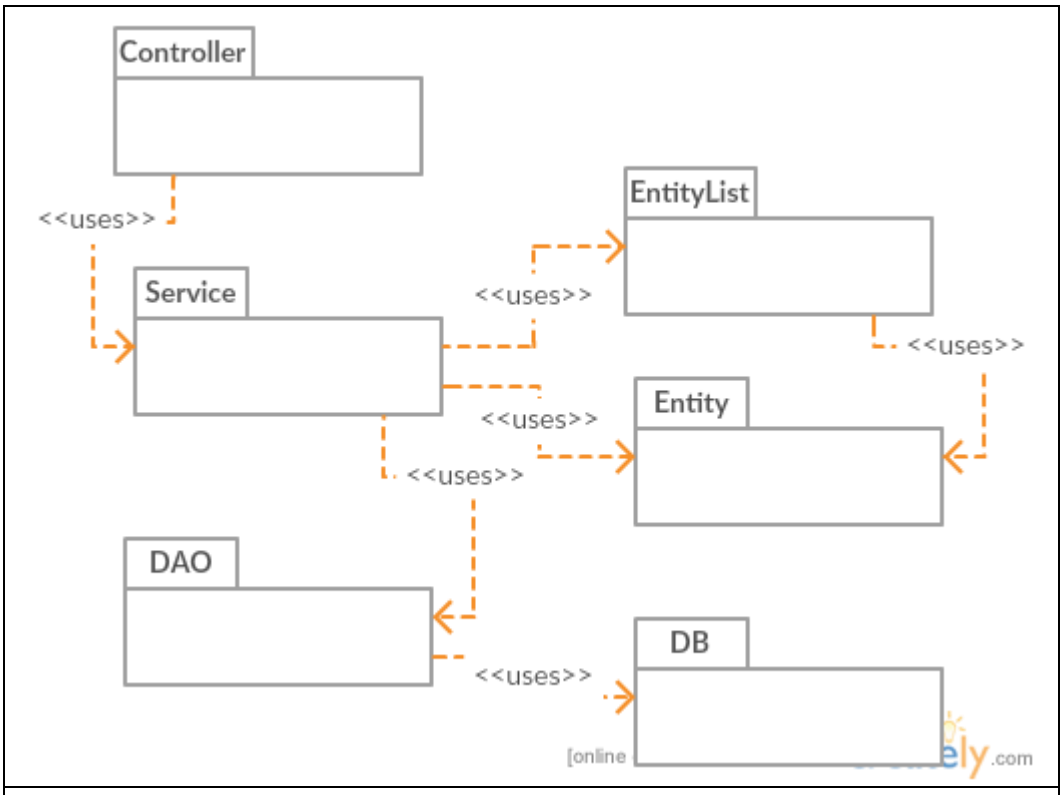

Fig. 3 – Package Diagram Backend

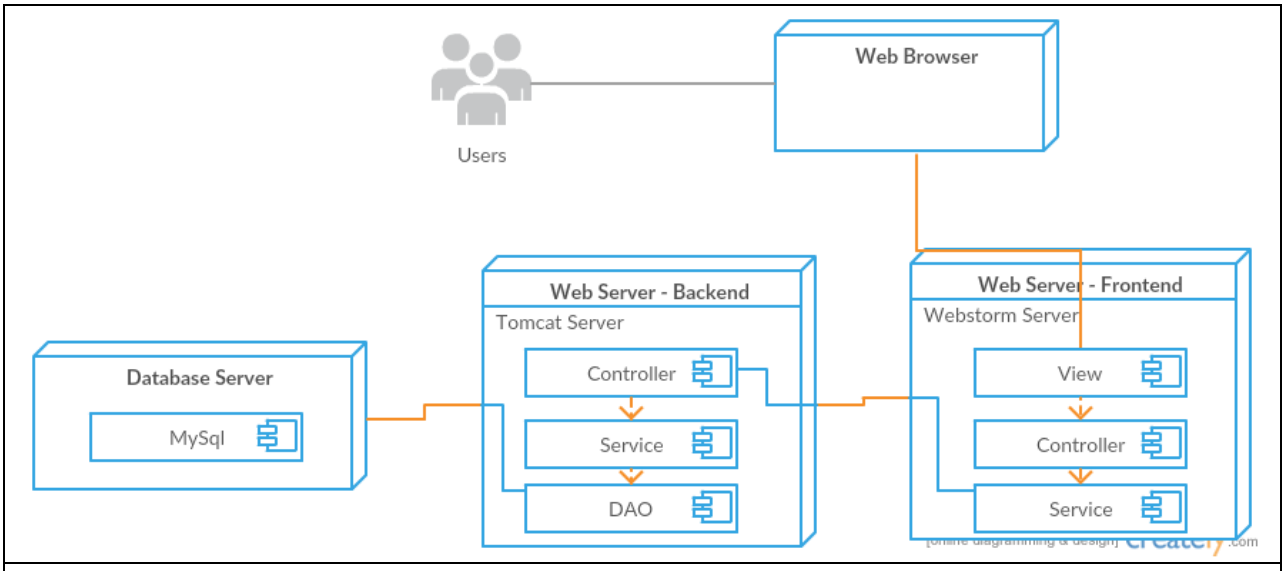

Fig. 4 – Deployment Diagram

4. UML Sequence Diagrams

[Create a sequence diagram for a relevant scenario.]

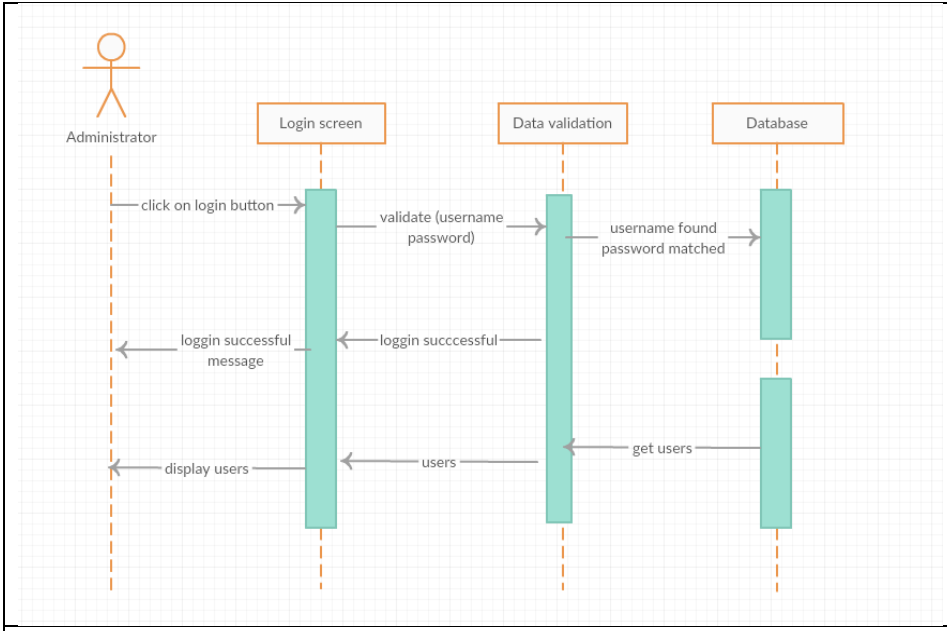

Fig.5 – Sequence Diagram - Login

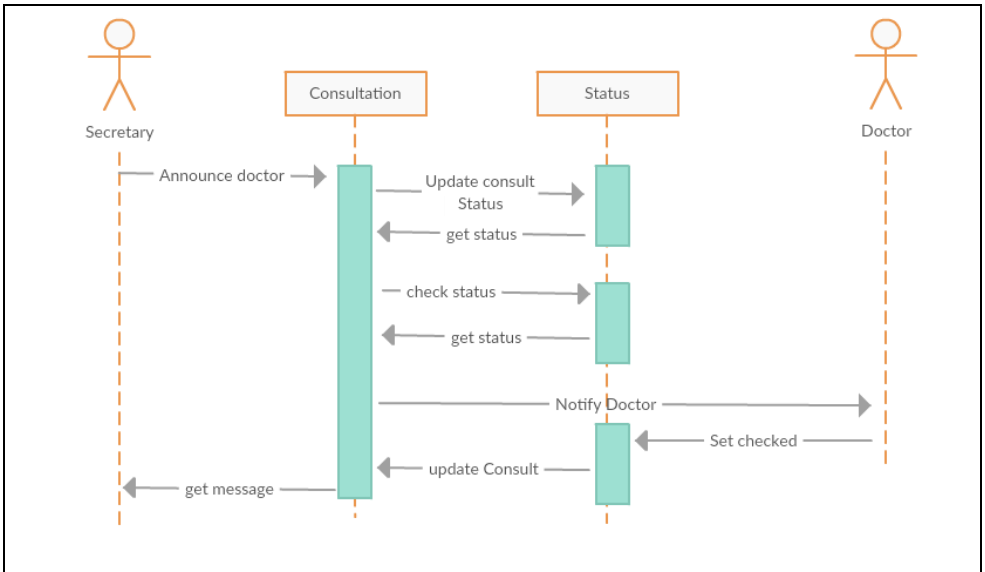

Fig.5b – Sequence Diagram - Notify

5. Class Design

5.1 Design Patterns Description

[Describe briefly the used design patterns.]

5.2 UML Class Diagram

[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]

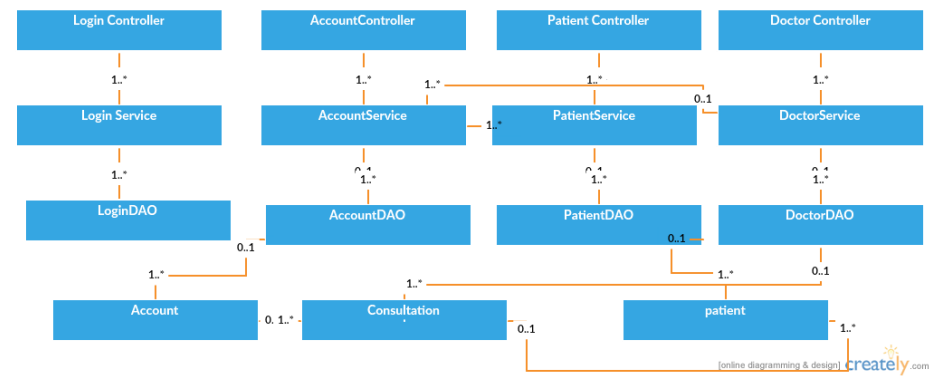

Fig. 6 – Class Diagram

6. Data Model

[Present the data models used in the system's implementation.]

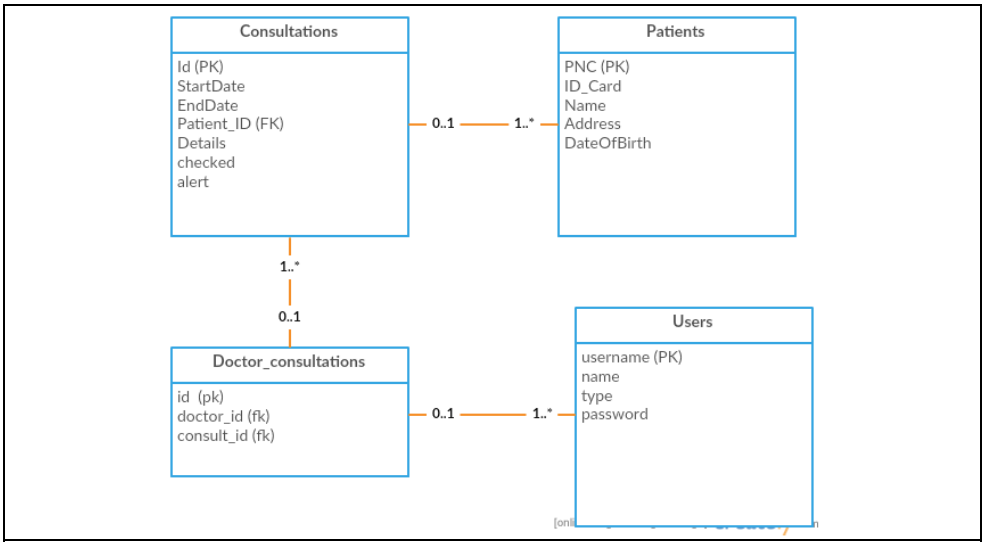

Fig. 6 – Data Model

7. System Testing

[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]

All input data is tested against invalid data by using regex data validation. Regex Pattern are supposed to be matching the input data and so to validate the input. Otherwise, the input will be rejected and a message will occup.
Examples of patterns:
CNP Validation:

```
var pattern = /^(1|2|5|6)(0[0-9]|1[0-7]|[3-9][0-9])(0[1-9]|[10]|[11]|[12])([0-2][1-
9]|[20]|3[0-1])\d{6}$/im;
```

8. Bibliography

1. http://stackoverflow.com/
2. https://docs.angularjs.org/api
3. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
4. http://getbootstrap.com/components/
5. https://www.codecademy.com/learn/javascript
6. Youtube tutorials