Assignment 3
Analysis and Design Document

Student: Bolba Raluca
Group: 30235

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Să se proiecteze și implementeze o aplicație client-server pentru administrarea consultatiilor doctorilor într-o clinică. Aplicația are trei tipuri de utilizatori : secretara, doctorii și administrator. Aplicatia ar trebui sa fie client-server iar datele sa fie salvate intr-o baza de date. Sa se utilizeze design pattern-ul Observer pentru a informa doctorul asociat prin afisarea unui mesaj.

1.2 Functional Requirements

Secreatara trebuie să efectueze următoarele operații:

- Să adauge sau să modifice date despre pacienți
- CRUD pe consultațiile pacientilor (să planifice o consultative, sa asigneze un doctor la un pacient in functie de programul doctorului)

Administratorul trebuie să efectueze următoarele operații:

- CRUD pe informațiile utilizatorilor

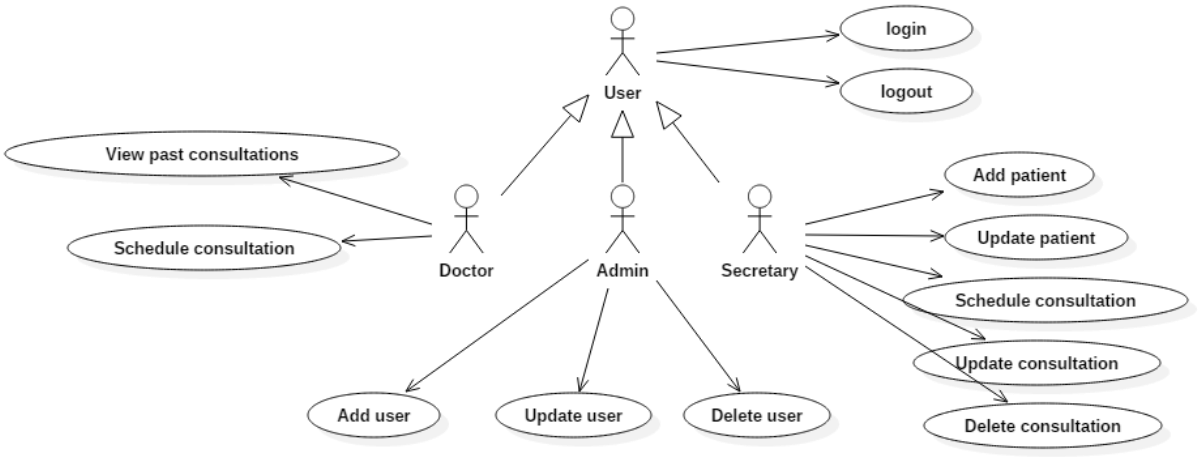
Doctorul trebuie să efectueze următoarele operații:

- Sa adauge si sa vizualizeze detalii despre consultatiile anterioare ale pacientilor

In plus, atunci cand un pacient avand o consultatie ajunge la clinica si se inregistreaza la secretara, aplicatia trebuie sa informeze doctorul asociat prin afisarea unui mesaj.

2. Use-Case Model

Use case general:



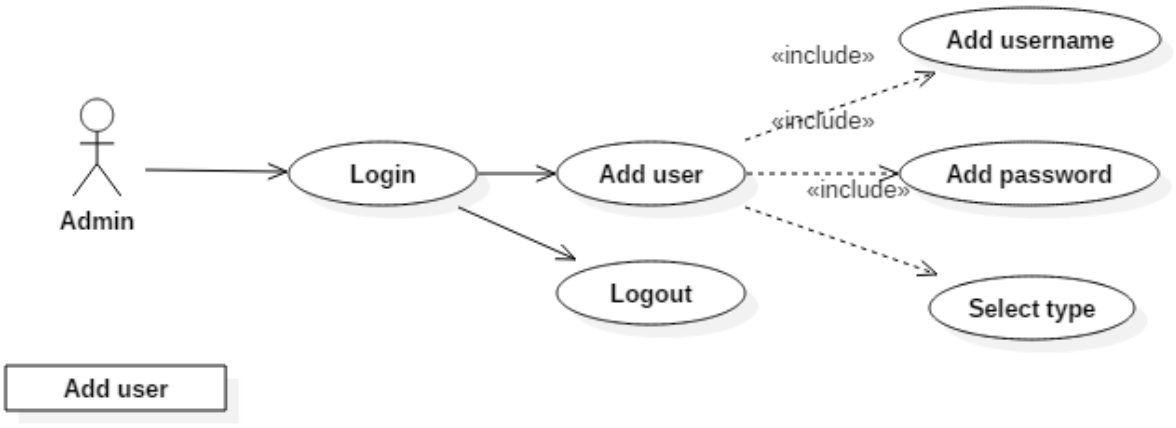
Use case: **Adăugare utilizator**
Level: User-goal

Primary actor: Administrator
Main success scenario:

1. Utilizatorul se autentifică la sistem
2. Administratorul alege să efectueze operația de adăugare utilizator
3. Administratorul introduce informațiile despre utilizator
    3.1. Administratorul introduce numele de utilizator
    3.2. Administratorul introduce parola utilizatorului
    3.3. Administratorul introduce tipul utilizatorului
4. Administratorul adaugă utilizatorul
5. Administratorul se deconectează

Extensions:

1a. Datele de autentificare sunt incorecte
    1a1.Se revine la pasul 1
3a. Numele de utilizator este asociat unui utilizator existent in sistem
    3a1. Se revine la pasul 3



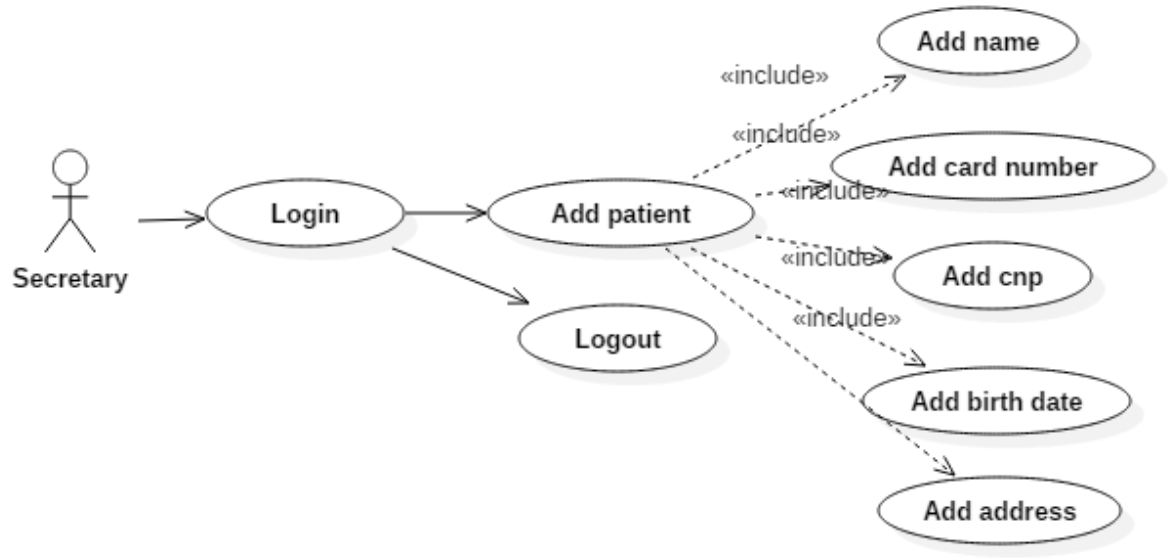
Use case: **Adăugare pacient**
Level: User-goal
Primary actor: Secretara
Main success scenario:

1. Utilizatorul se autentifică la sistem
2. Secretara alege să efectueze operația de adăugare pacient
3. Secretara introduce informațiile despre pacient
    3.1. Secretara introduce numele pacientului
    3.2. Secretara introduce numarul cardului de identificare
    3.3. Secretara introduce codul numeric personal

3.4. Secretara introduce data de nastere a pacientului
3.5. Secretara introduce adresa pacientului

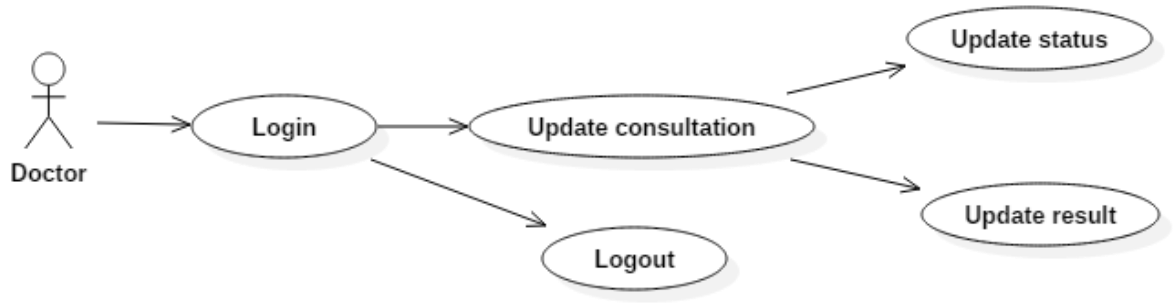
4. Secretara adaugă pacientului
5. Secretara se deconectează

Extensions:
1a. Datele de autentificare sunt incorecte
1a1.Se revine la pasul 1
3a. Pacientul a fost deja adaugat la sistem
3a1. Se revine la pasul 3



Use case: **Actualizare consultație**
Level: User-goal
Primary actor: Doctor
Main success scenario:
1. Utilizatorul se autentifică la sistem
2. Doctorul alege să efectueze operația de actualizare consultatie
3. Doctorul actualizeaza detalii despre consultatie
3.1. Doctorul actualizeaza starea consultatiei
3.2. Doctorul actualizeaza rezultatul consultatiei
4. Doctorul actualizeaza consultatia
5. Doctorul de deconecteaza

Extensions:
1a. Datele de autentificare sunt incorecte
1a1.Se revine la pasul 1
3a. Stare consultatiei nu este una din : *checked in, scheduled, complete*
3a1. Se revine la pasul 3



3. System Architectural Design

3.1 Architectural Pattern Description

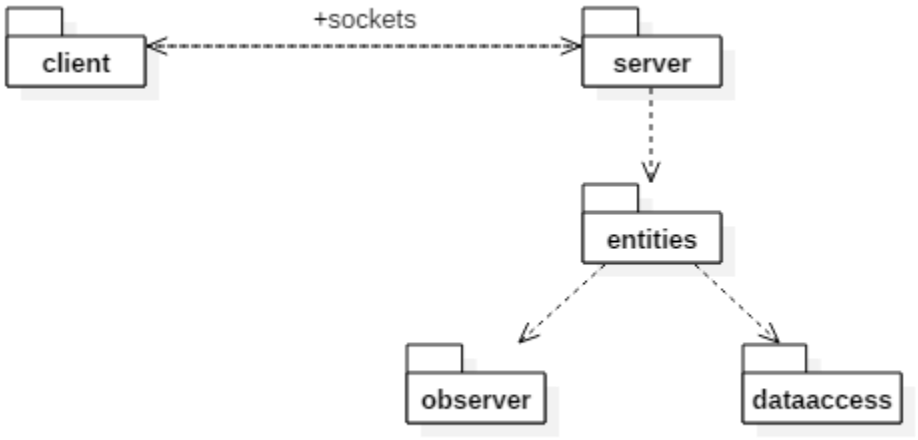
Pattern-ul arhitectural folosit pentru aceasta aplicatie este pattern-ul *Client-Server*, care împarte aplicația în două aplicatii, una care ruleaza pe un server, iar una pe client. Cele doua componente trebuie sa faca schimb de informatii pentru ca aplicatia sa fie functionala. O modalitate de comunicare este prin intermediul *Socket-urilor* furnizate de Java. Prin intermediul acestora se pot transfera date bidirectional, dupa ce au fost setate socket-urile pentru client si server folosind un port. Pentru a stii ce date trebuie trimise sau primite, cele doua componente trebuie sa stabileasca o conventie. In acest caz, atunci cand clientul vrea sa trimita date spre server, prima data trimite numele comenzii (de exemplu *login, adduser, updatepatient)* dupa care se trimit datele (de exemplu in cazul autentificarii se trimite username-ul si parola) iar serverul va trimite prima data un raspuns de succes sau esec, iar ulterior alte date de interes.

Partea de *Client* in cazul acestei aplicatii o reprezinta partea de interfata grafica cu utilizatorul. Aceasta transmite datele furnizate de utilizator prin intermediul interfetei catre server, iar in functie de raspunsul primit, furnizeaza la randul lui raspunsuri catre utilizator. Partea de client revine in cazul acestei aplicatii pachetului *client* cu clasele *ClientMain, ClientController și Window.*

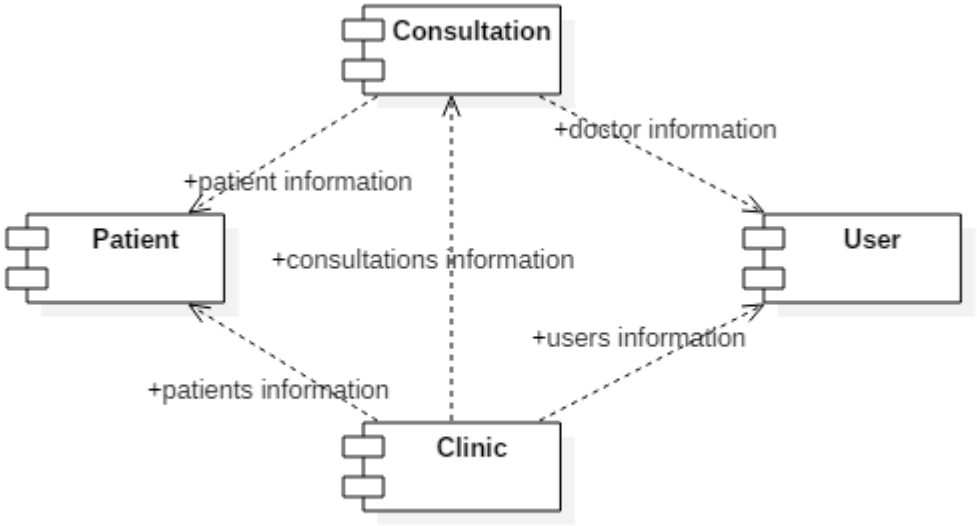
Partea de *Server* cuprinde cea mai mare parte din aplicatie, incluzand partea de model a aplicatiei, de furnizare de servicii și de acces la baza de date. Serverul asteapta ca un client sa ceara anumite servicii si creaza cate un fir de executie pentru fiecare client, astfel incat aplicatia poate avea mai multi clienti inregistrati la acelasi server. In functie de comenzile si datele primite de la client, serverul efectueaza modificari asupra modelului aplicatiei si in functie de rezultat, trasmite un mesaj de succes sau esec catre client. Pentru aceasta aplicatie, partea de server revine pachetelor *server, entities, dataaccess și observer.*

3.2 Diagrams

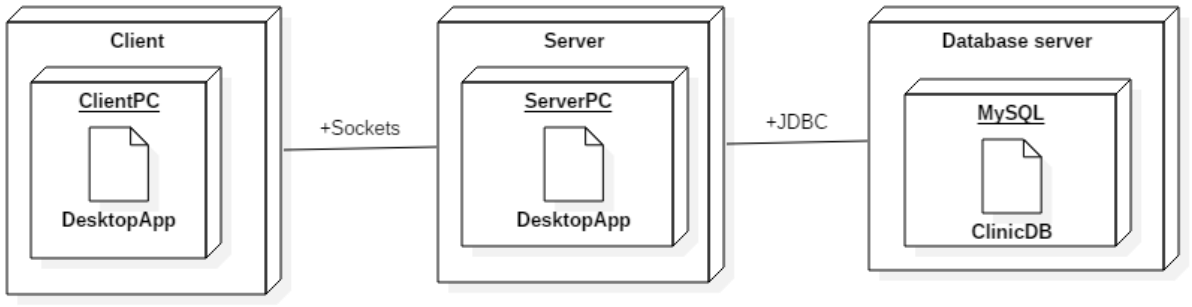
Diagramă de pachete



Diagramă de componente

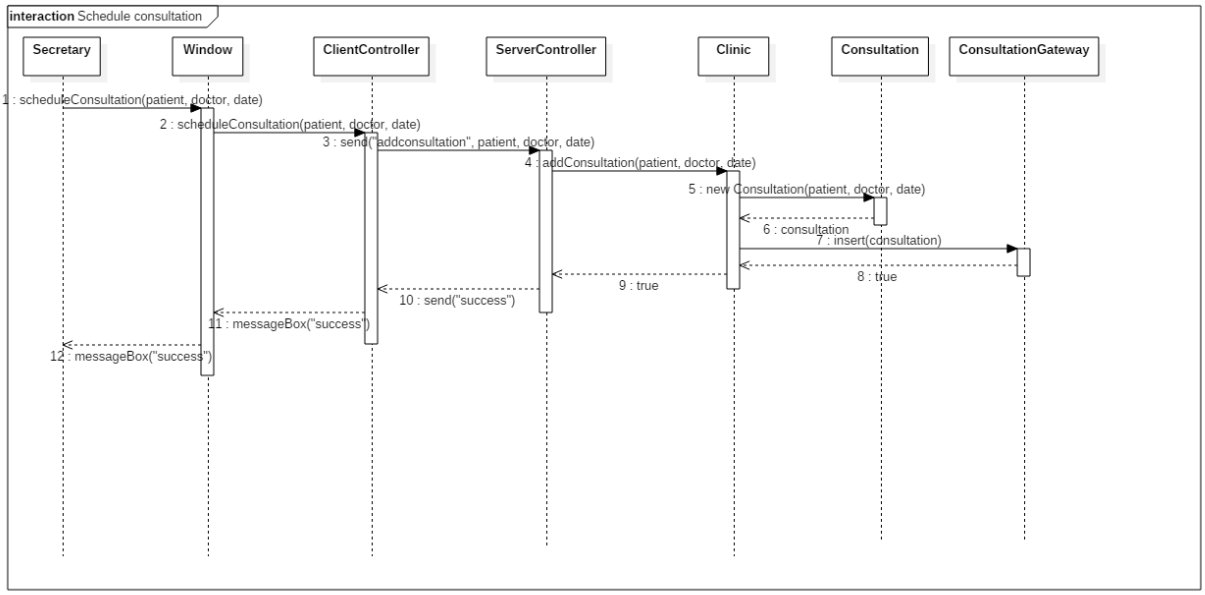


Deployment diagram



4. UML Sequence Diagrams

Diagramă de secvență pentru *Schedule consultation.*



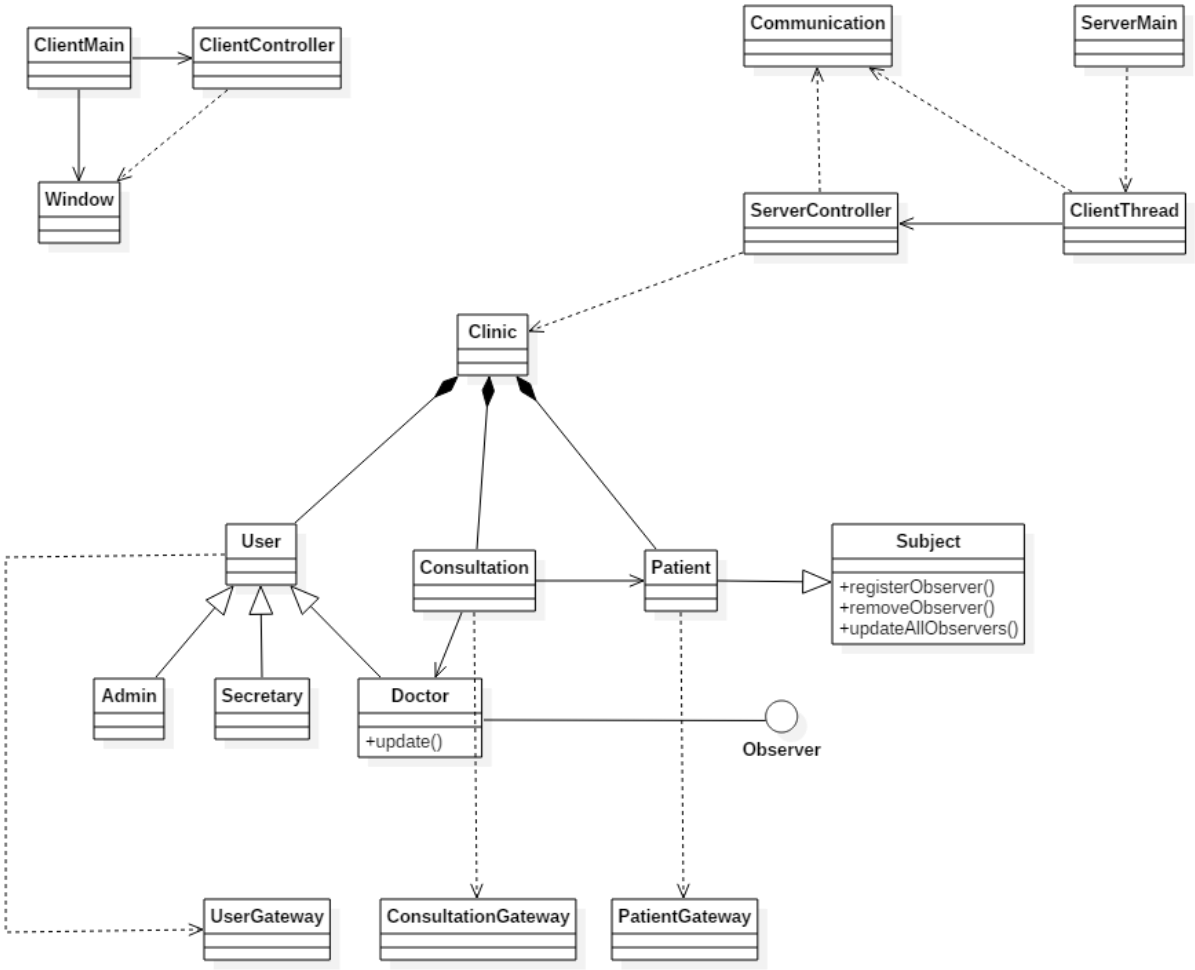
5. Class Design

5.1 Design Patterns Description

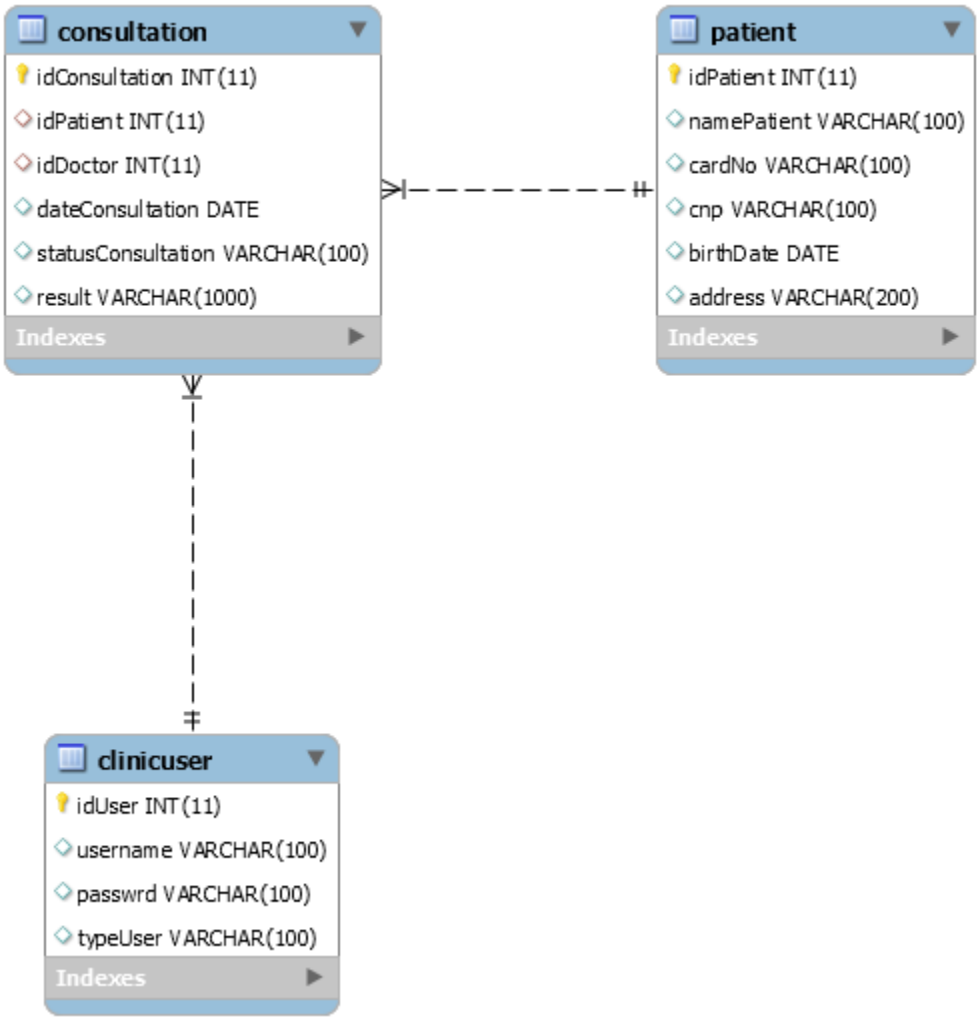
Design pattern-ul utilizat în această aplicație este *Observer*, un pattern care se incadreaza in categoria de *behavioral patterns* și este un pattern care este utilizat atunci cand exista o relatie one-to-many intre obiecte astfel incat cand un obiect este modificat, obiectele dependente sunt notificate automat. Obiectele care „observa" alte obiecte sunt instante ale unei clase, sau interfete, numita in acest caz *Observer*, care contin o metoda *notify() (sau update())*, prin care se realizeaza notificarea in cazul modificarii obiectelor ce sunt observabile. Aceste obiecte observabile sunt instante ale unei clase *Subject* care are ca metode: *registerObserver(Observer o), removeObserver(Observer o), notifyAllObservers()* care adauga, elimina un observer sau notifica toti observerii si are ca atribut o colectie de obiecte de tip *Observer.*

5.2 UML Class Diagram

În cazul acestei aplicații, pattern-ul este utilizat pentru notificarea doctorului asociat cu un mesaj atunci cand un pacient de-al sau se prezinta la consultatie si da check in la biroul secretarei. Asadar obiectele care sunt observabile sunt obiectele de tip *Patient*, astfel incat aceasta clasa va extinde clasa *Subject*, iar obiectele care observa sunt cele de tip *Doctor*, astfel incat aceasta clasa implementeaza interfata *Observer*. Atunci cand un pacient se prezinta la consultatie, secretara va apela metoda check-in pentru pacientul respective, care la randul ei apeleaza metoda de notificare a toti observatorilor, iar fiecare dintre acestia va fi notificat printr-un mesaj.


ClientMain
ClientController
Window
Communication
ServerMain
ServerController
ClientThread
Clinic
User
Consultation
Patient
Subject
+registerObserver()
+removeObserver()
+updateAllObservers()
Admin
Secretary
Doctor
+update()
Observer
UserGateway
ConsultationGateway
PatientGateway

6. Data Model



In cazul acestei aplicatii, datele au fost stocate intr-o baza de date. Tabele din cadrul bazei de date sunt : *Patient, Consultation si ClinicUser.*

Tabela *Patient* contine date referitoare la pacientii clinicii, informatii precum numele pacientului, numarul cardului de identificare, codul numeric personal, data nasterii si adresa acestuia.

Tabela *ClinicUser* conține date referitoare la utilizatorii aplicației și anume un identificator, date de autentificare (username și parolă) și tipul acestuia. Întrucât utilizatorul poate fi administrator, doctor sau secretara, am decis să păstrez această informație în câmpul *typeUser* în detrimentrul creării a trei tabele, câte una pentru fiecare utilizator, întrucât datele ce trebuie păstrate pentru aceștia nu diferă.

Tabela contine date referitoare la consultatiilor pacientilor clinicii, si anume un identificator, o cheie straina catre tabela *Patient* care indica pentru ce pacient a fost facuta consultatia, o cheie straina catre tabela *ClinicUser* care indica ce doctor este asignat pacientului respective pentru consultatie, data la care are loc consultatia, statusul ei (*scheduled, checked in, complete)* si rezultatul acesteia.

7. System Testing

Pentru testarea aplicației aceasta s-a rulat și s-a verificat dacă operațiile sunt realizate în mod corect. De exemplu pentru operația de autentificare s-a verificat dacă datele introduse sunt cele prezente și în baza de date, iar în caz contrar s-a afișat un mesaj de eroare.

8. Bibliography

http://www.tutorialspoint.com/design_pattern/observer_pattern.htm
https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html
https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html