Facultatea de Matematică şi Informatică

Universitatea Babeş-Bolyai



**Aplicaţie pentru gestionarea unui cabinet medical de familie**

**Introducere:**

Pe baza cerințelor aplicației, pentru început am proiectat o Diagramă a Cazurilor de Utilizare care cuprinde enunțarea funcționalităților majore ale aplicației.

Datorită faptului că sunt mulți utilizatori care vor folosi această aplicație și volumului mare de date utilizate de aceasta , am decis ca aplicația să fie de tip Web. De asemenea, proiectarea bazei de date este realizată în MySql pentru a fi compatibilă cu aplicațiile existente.

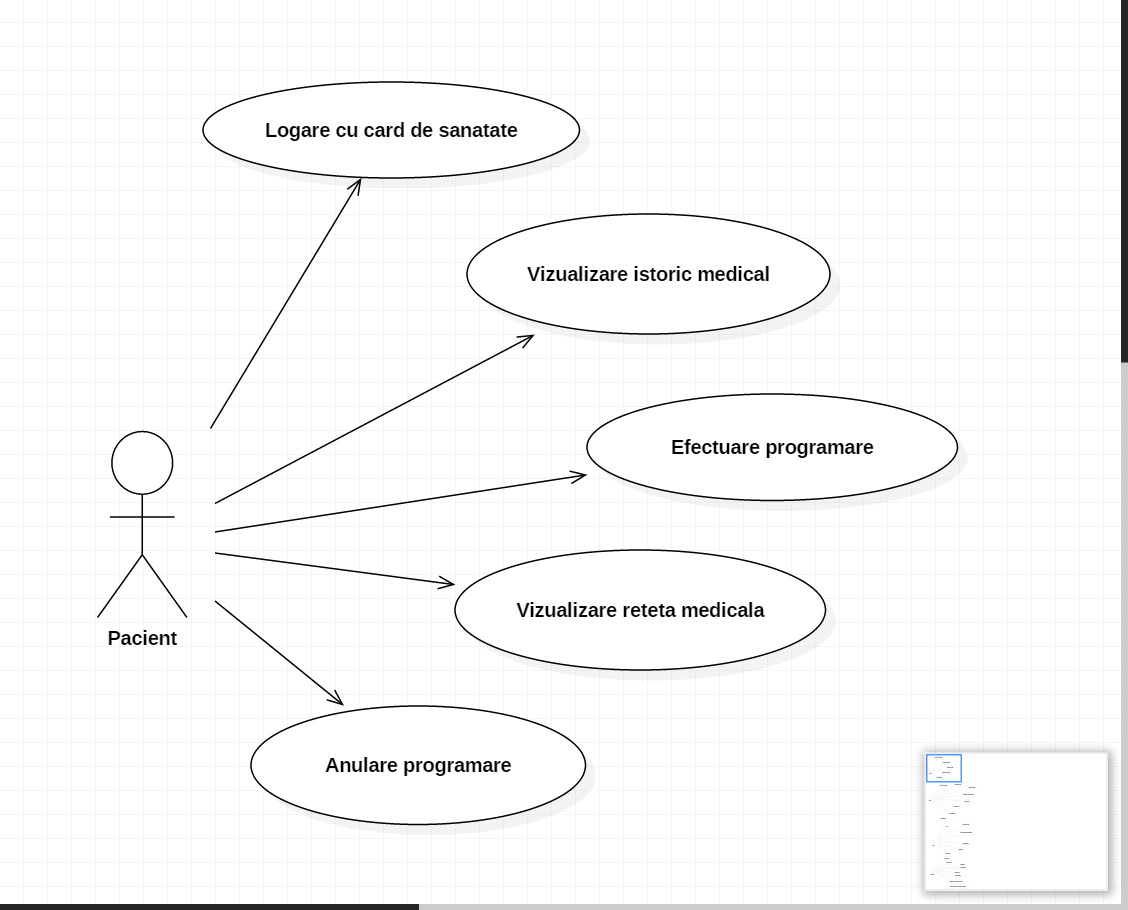
**Prezentarea Funcționalităților:**

Aplicatia cuprinde realizarea urmatoarelor functionalitati:

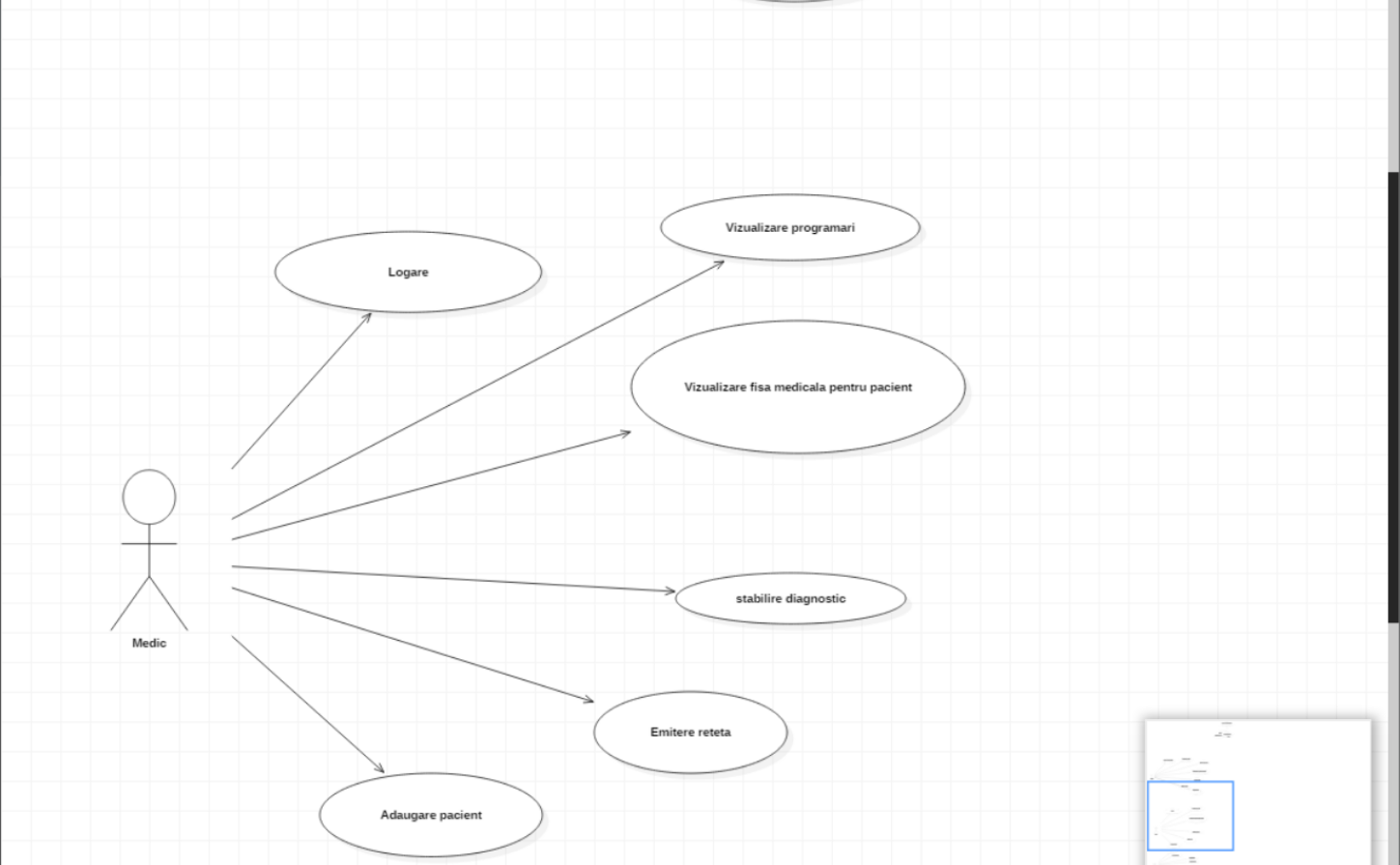
* Pentru **Pacient :**
* Logare în aplicație utilizând cardul național de sănătate
* Vizualizarea istoricului medical
* Efectuarea unei programari pentru o consultatie
* Vizualizarea rețetei medicale
* Ștergerea unei programări cu 24h înainte
* Pentru **Asistent :**
* Logare în aplicație
* Vizualizarea programărilor
* Modificare programare
* Trimite lista finală a programărilor către **Medic**
* Adăugare **Pacient**
* Modificare date **Pacient**
* Pentru **Medic :**
* Logare în aplicație
* Vizualizarea programărilor
* Vizualizarea unei fișe medicale pentru un anumit **Pacient**
* Stabilirea diagnosticului pentru un anumit **Pacient**
* Emitere rețetă
* Pentru **Administrator :**
* Adăugare/Ștergere **Medic**
* Vizualizarea tuturor programărilor pentru un anumit **Medic**
* Vizualizarea istoricului modificărilor pentru un **Pacient**

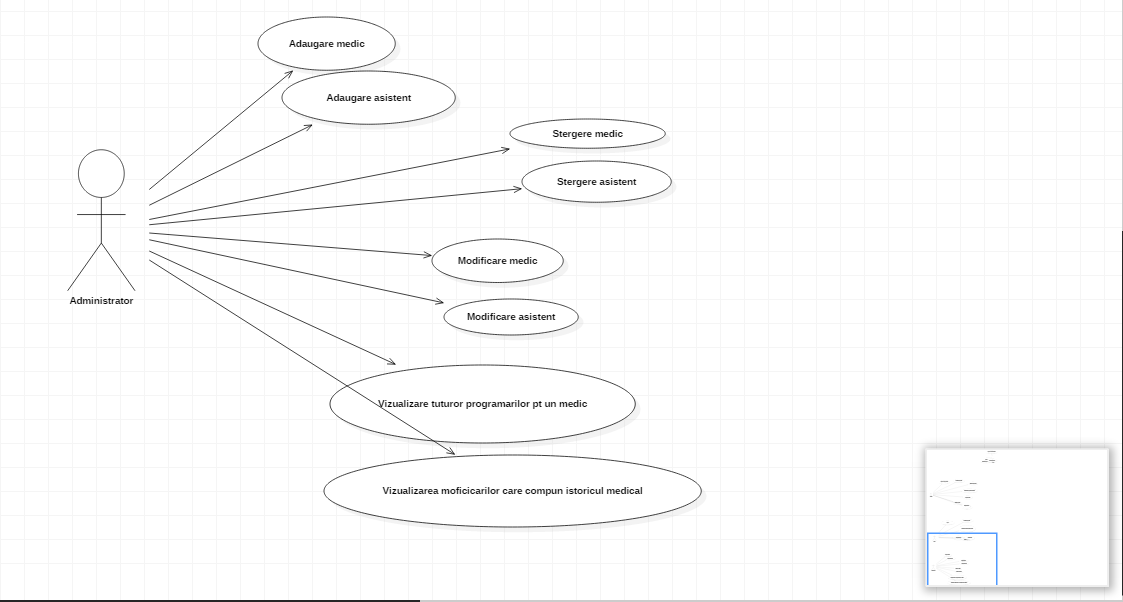
1. **Etapele proiectării aplicației**
   1. **Diagrama cazurilor de utilizare**

În continuare prezentăm diagrama cazurilor de utilizare care conține funcționalitățile majore ale aplicației, menționate mai sus.(În ordine, diagrama este prezentată astfel : **Pacient, Asistent, Medic** și **Administrator**).



****

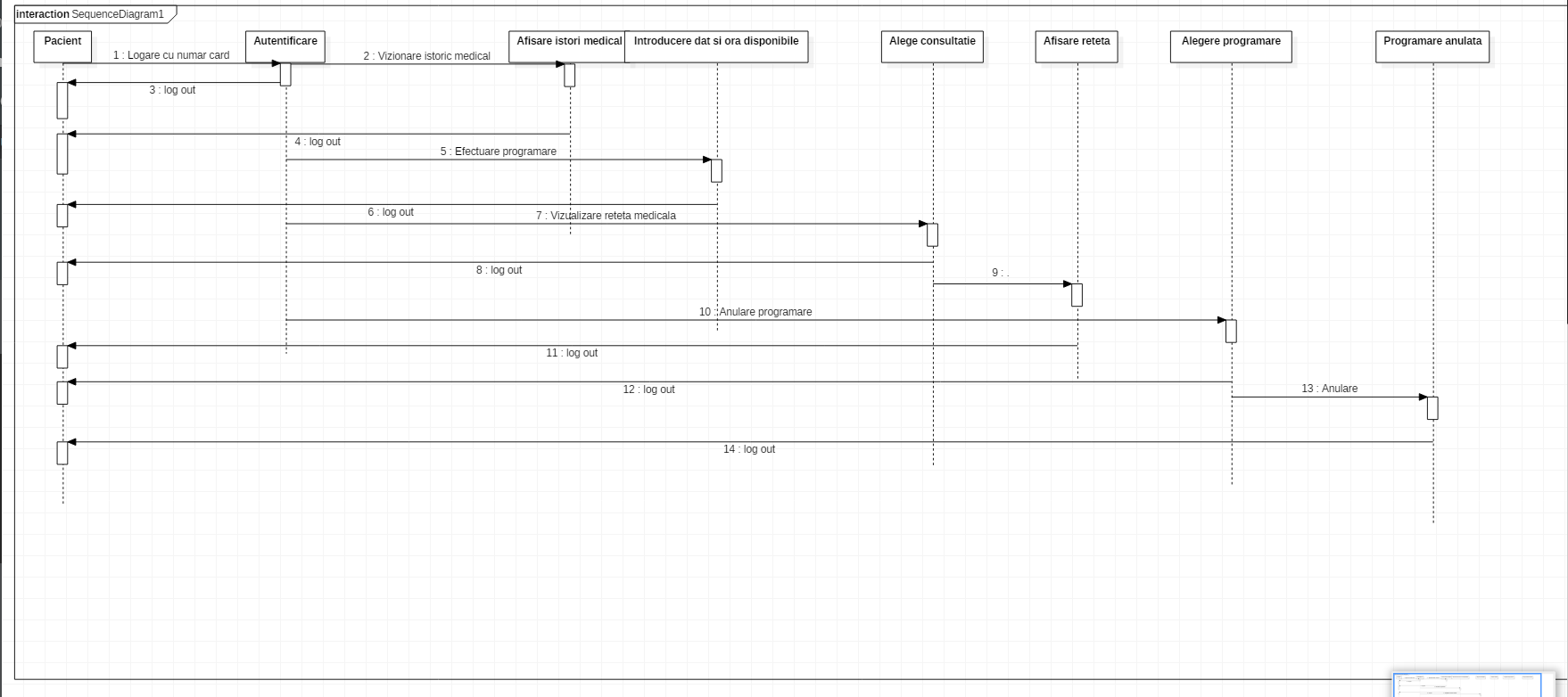
****

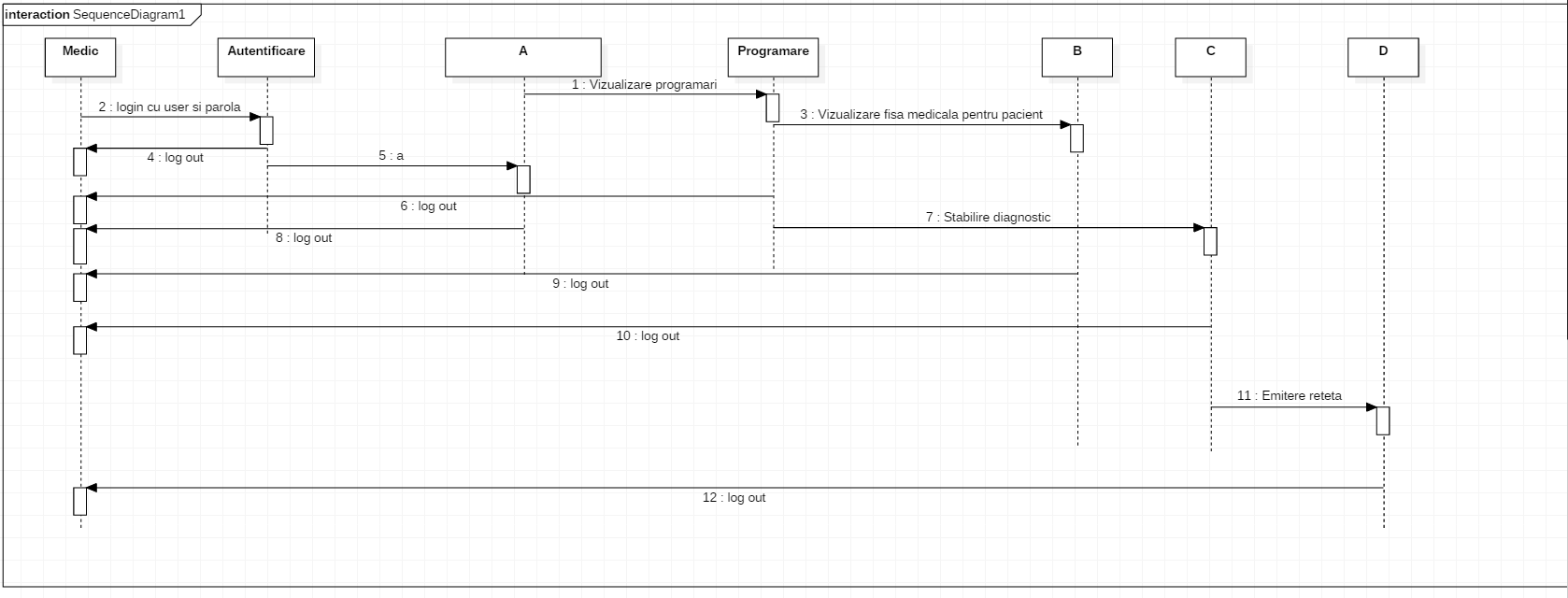
****

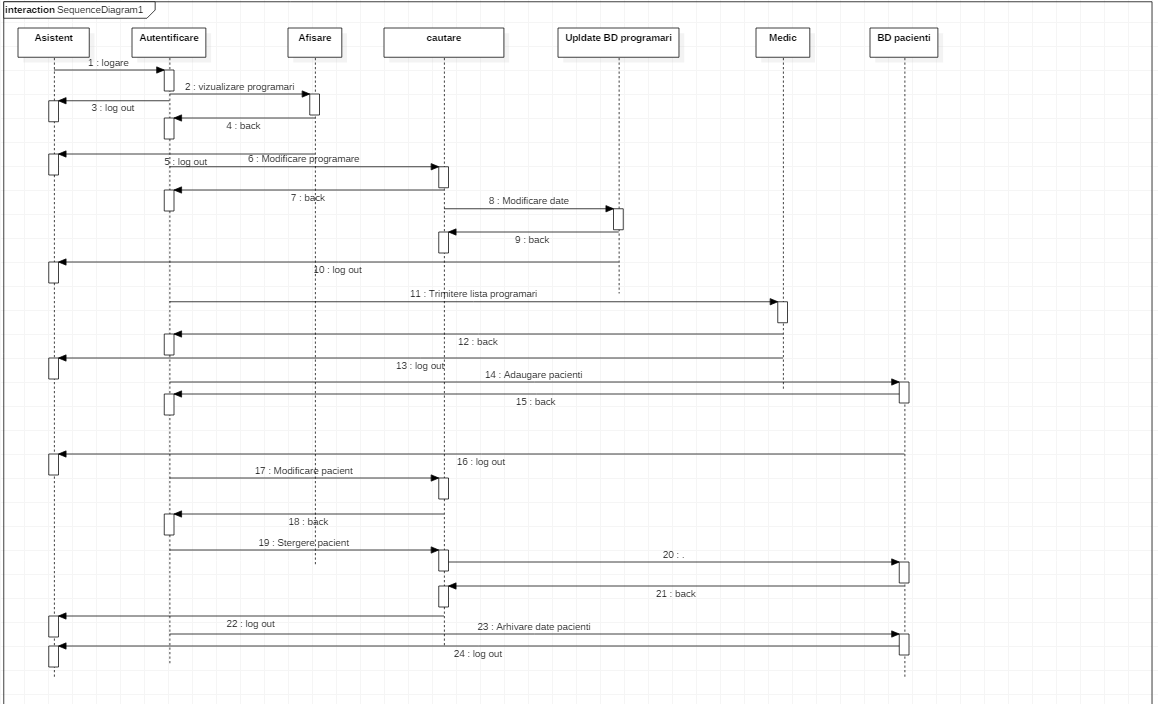
Aplicația va fi folosită de către mai multe tipuri de utilizatori, respectiv **Pacient, Asistent, Medic si Administrator**. Aplicația nu permite acces la informații decat după ce utilizatorii se loghează.

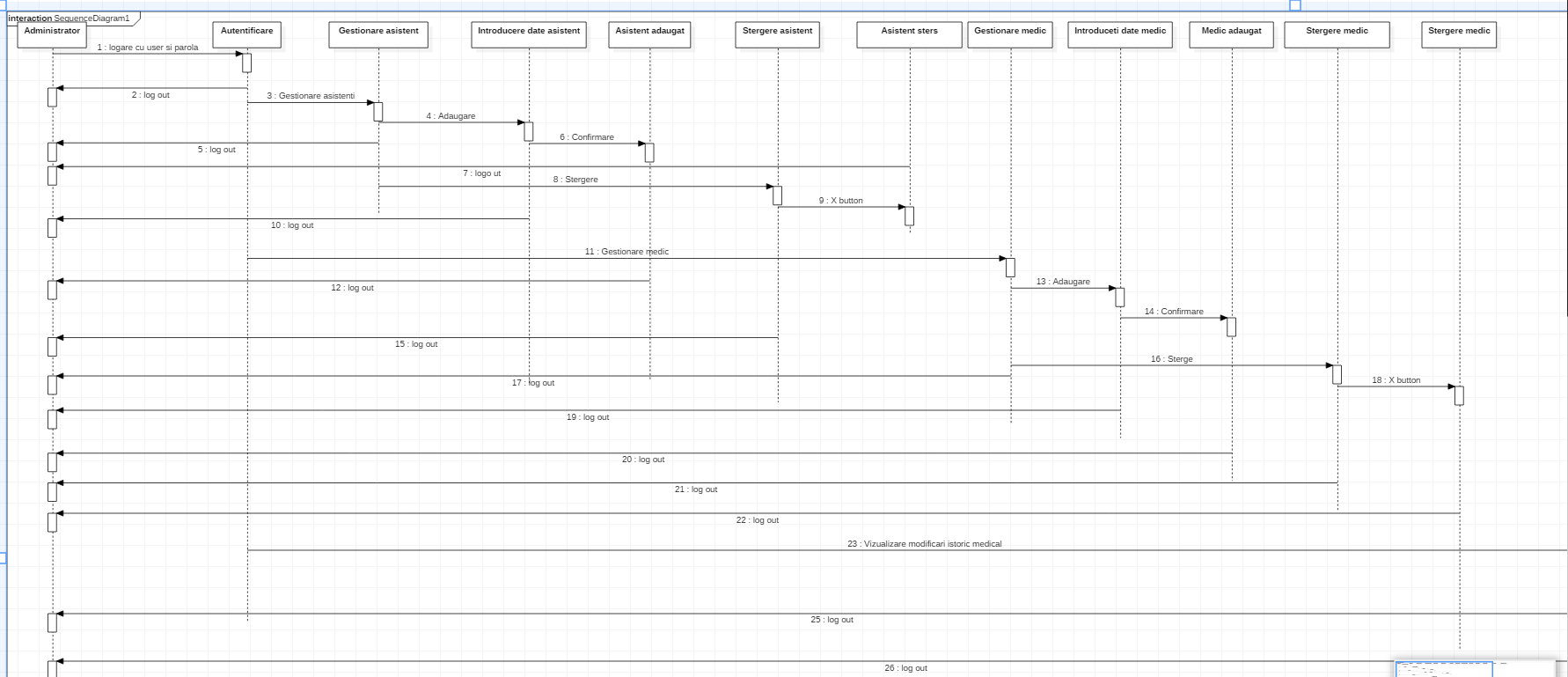
* 1. **Diagrama de Interacțiune**

Următorul pas în realizarea aplicației este crearea **Diagramei de Interacțiune** pentru cazurile de utilizare.









La pornirea aplicației de către utilizator, pagina de Home obligă la înregistrarea sau logarea utilizatorului(ca **Pacient, Medic,** **Asistent** sau **Administrator).**

În cazul în care autentificarea se termină cu succes, in funcție de utilizator, se afișează în bara de navigație o listă de opțiuni.

Bara de navigație va conține următoarele :

**Pacient**

* My profile, book appointment, my medical record, my appointments și logout.(Profilul meu, creează programare, istoricul meu medical, programările mele și ieșire din cont).

**Medic**

* Vizualizare pacienți, programări zilnice, ieșire din cont.

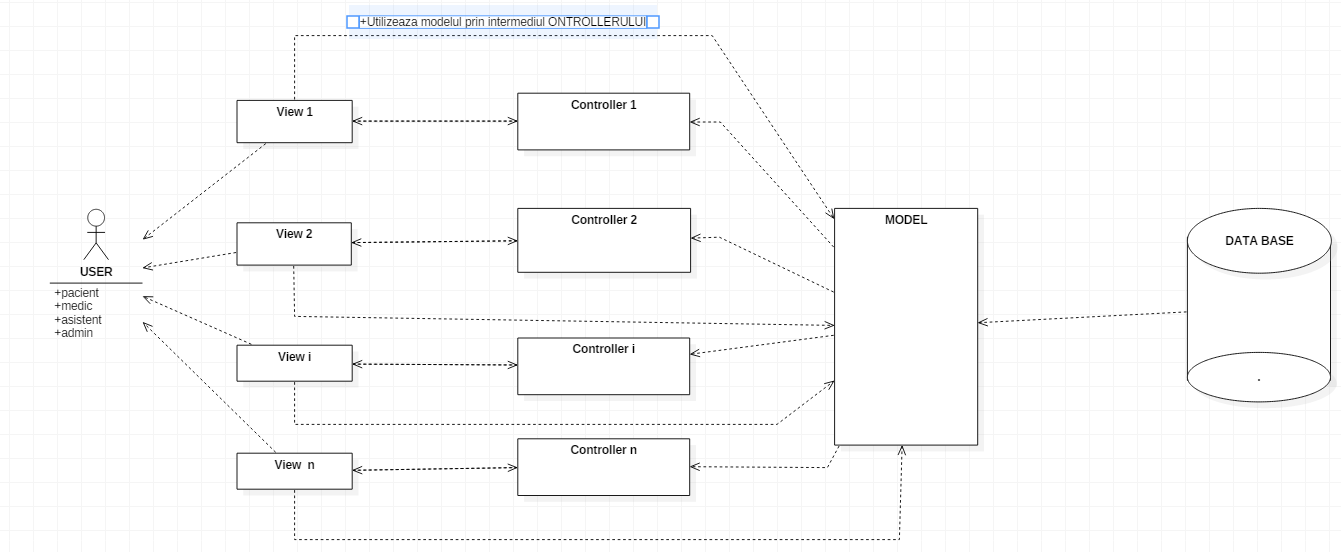
**Asistent**

* Manage appointments, pacients and prescriptions, logout.(Gestionare programări, pacienți și prescripții medicale, ieșire din cont.

**Administrator**

* Gestionare medici si asistenți.
  1. **Diagrama de Arhitectură**

Pentru implementarea proiectului, am folosit arhitectura **MVC(Model View Controller)** pentru partea de front-end, iar pentru partea de back-end am folosit **.Net**.



//// de facut !!!

* 1. **Diagrama de clase**

Diagrama de clase conține entitățile utilizate în aplicație . Acestea sunt mapate pe baza de date. (Fig. 1.4, Fig. 1.5)

Entitatea Persoana este o clasă generică din care vor moșteni următoarele clase: Decedat, Detinator, DecedatFaraApartinator, deoarece cele trei clase contin niste campuri identice, dar unele sunt specifice unui anumit tip de Persoana. Fiecare tip de persoana are un rol diferit, si legaturi cu diferite clase.

Dupa proiectarea diagramei de clase, ținând cont de modul de implementare a legaturii dintre entitatea Pesoană și celelalte clase subordonate entității, am decis să eliminăm clasele Decedat și Detinator, și sa le înlocuim cu clasa Persoana.

Interfața cu utilizatorul conține o fereastră pentru logare, precum și o fereastră pentru a accesa registrele. MainCatalogWindow conține butoane pentru fiecare registru în parte.

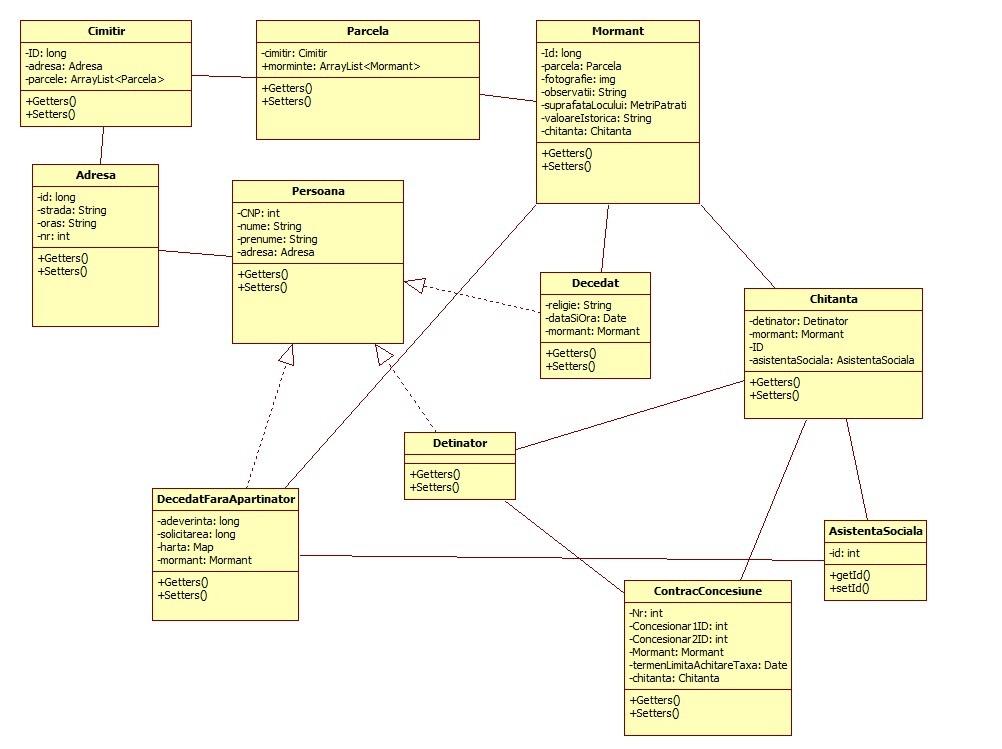
Am ales sa implementam o tabela separata pentru AsistentaSociala deoarece pe chitanta emisa trebuie sa apara un reprezentant care a achitat-o, pentru decedatul fara apartinator.

De asemenea exista un tabel separat pentru adrese deoarece o adresa poate sa apara de mai multe ori pentru o persoana sau pentru cimitire, si pentru a nu exista inconsistente in baza de date.

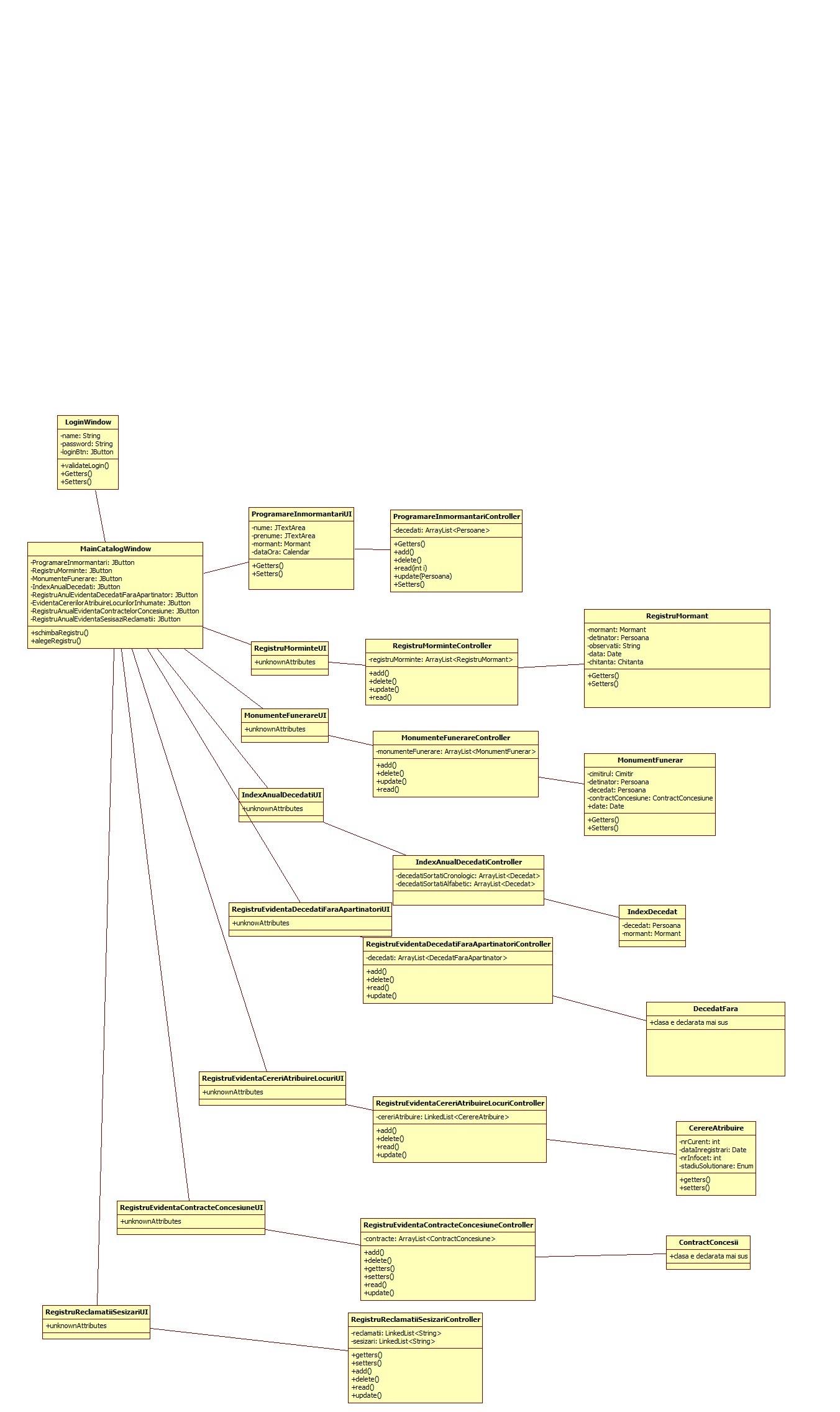
Informația este gestionată de Controllere care au structură identică (au operatii similare). Dependințele dintre Controlleri sunt similare în ceea ce privește interfața cu utilizatorul.

Fiecare Controller conține o listă de entitați de tip Registru.

Din diverse motive diagrama de clase s-a modificat, clasa AsistentaSociala fiind desființata, iar in clasa DecedatFaraApartinator am inclus un atribut asistentaSociala. Din acest motiv, nici în baza de date nu va mai exista tabela asistentaSociala.



**Fig. 1.4 - Diagrama de Clase**

****

**Fig. 1.5 – Diagrama de clase**

**1.5 Baza de date**

Baza de date a aplicației conține cate un tabel pentru entitățile definite în Diagrama de Clase.

Tabela **Medic** reprezintă Medicul și conține datele personale ale acestuia.

Tabela **Assistant** are în plus cheia primară a tabelei **Medic,** pentru a defini apartenența unui Asistent de un Medic.

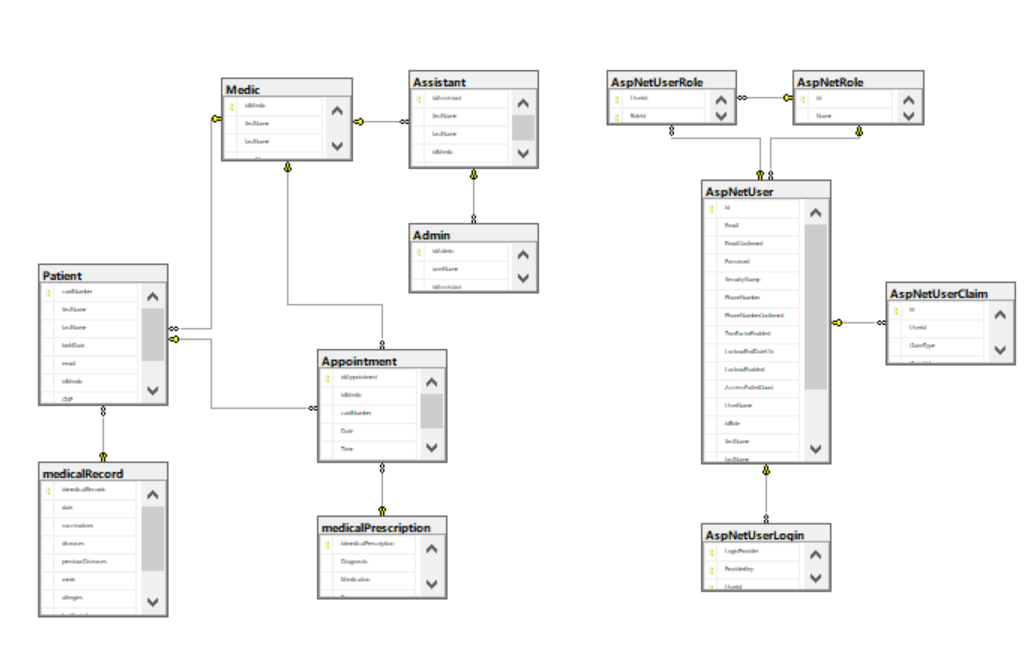
Entitatea **Admin** este legată de **Assistant** printr-o cheie străina. Analog sunt legate și tabelele **Assistant** cu **Medic,** astfel ca Administratorul are drept de gestionare a Medicilor si Asistentilor.

Entitatea **Patient** reprezinta Pacientul si poate suferi modificari sau vizualizari din partea Medicului si a Asistentului. De asemenea, prin tabela **medicalRecords**(istoric medical), se face legatura intre Pacient si istoricul medical al acestuia.

Tabela **medicalPrescription** definește atributele unei prescripții medicale : diagnostic, tratament medical și tipul acesteia (compensată sau necompensată).

Tabela **Appointment** conține detaliile programării : data, timpul si numărul cardului pacientului. Asistenul este cel care gestionează programările și le leagă de Medic printr-o cheie străină.

Tabelele **AspNet** sunt cele predefinite de **MCV** și au rolurile de înregistrare și logare, fiind modelate pentru cerința primită.

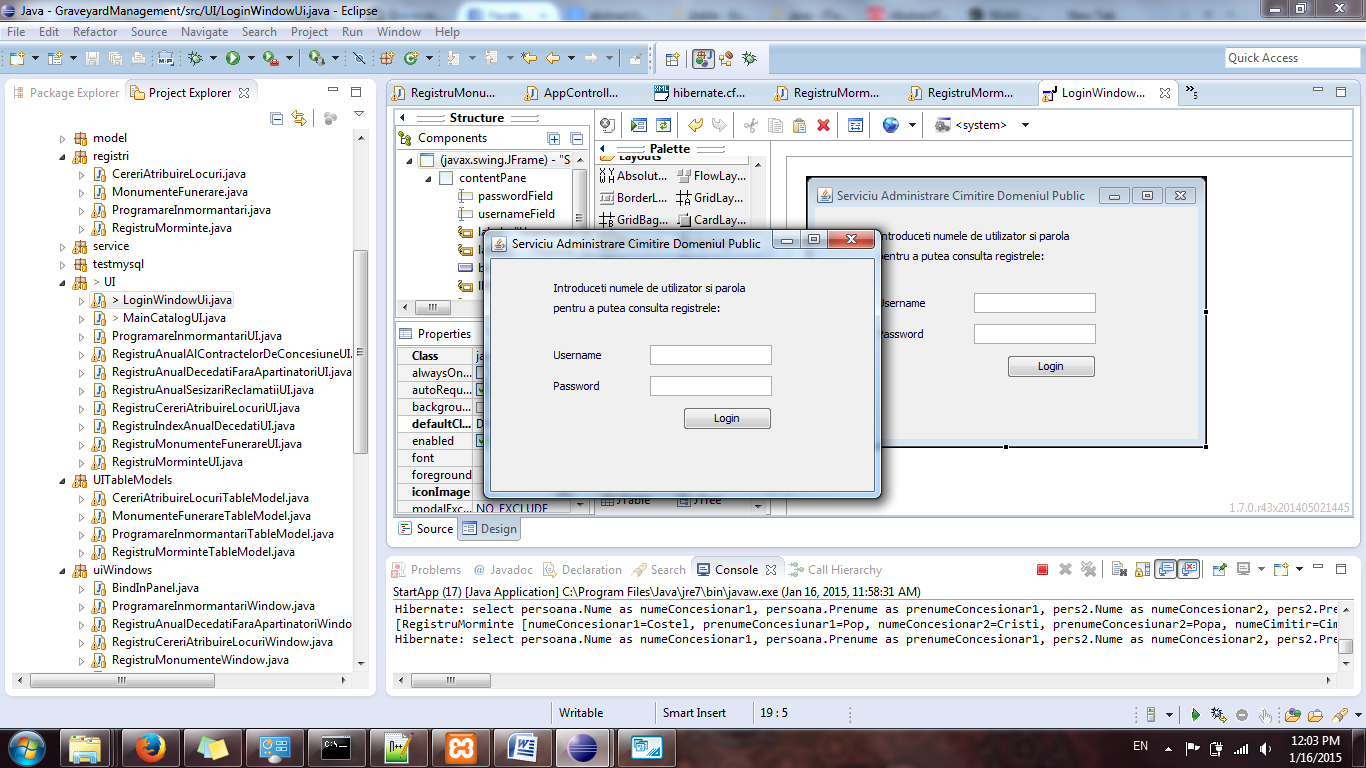


**2.Modul de funcționare al aplicației**

**2.1 Interfața cu utilizatorul**

La pornirea aplicației pe ecran va aparea o fereastra de Login (Fig. 2.1).

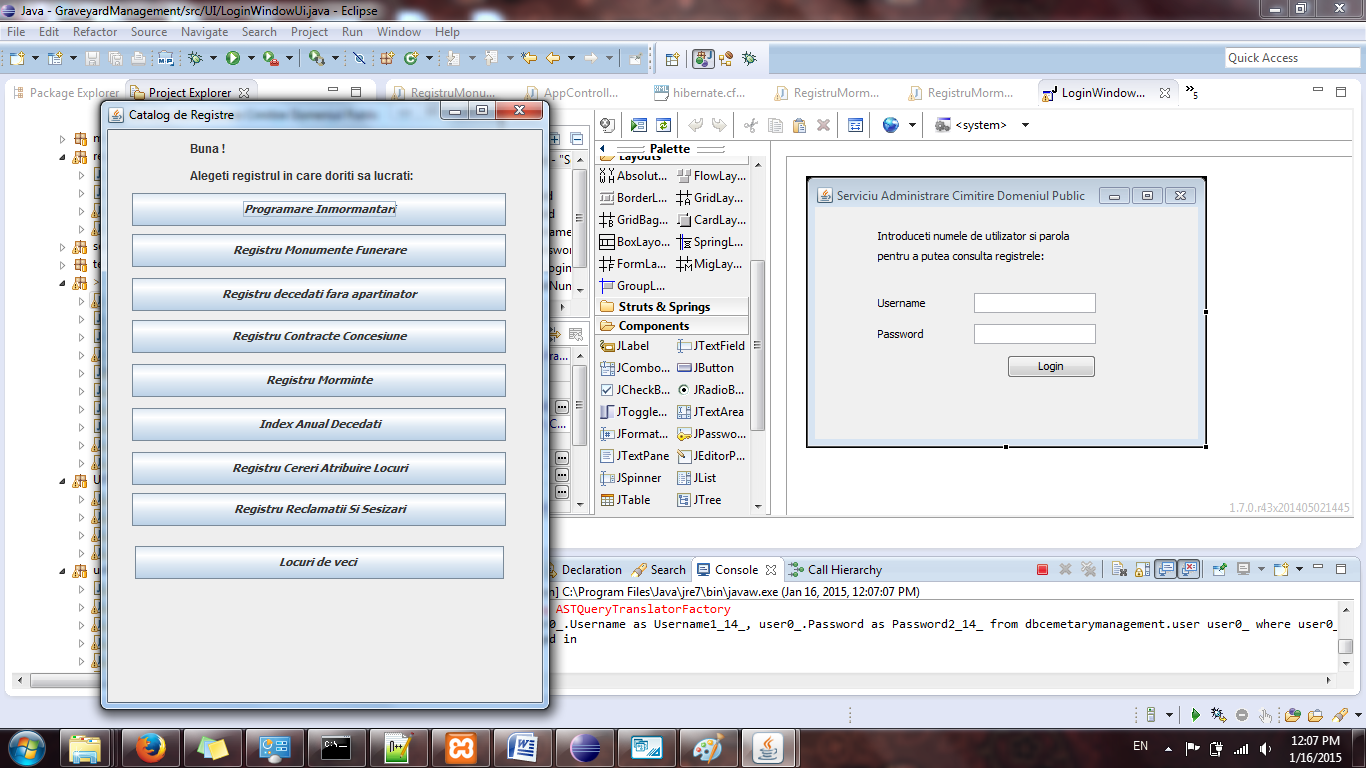
Utilizatorului i se cere sa introducă numele de utilizator și parola, după care va trebui sa acționeze butonul de Login situat în partea de jos a ferestrei.



**Fig. 2.1. – Fereastra de Login**

În cazul în care autentificarea se efectuează cu succes, pe ecran va apărea fereastra CatalogRegistre (Fig. 2.2) iar utilizatorul poate alege pe care registru dorește să lucreze.

În cazul ân care autentificarea eșuează, pe ecran va apărea o fereastră de tip pop-up cu un mesaj corespunzător: „User failed to log in”, utilizatorul urmând să introducă din nou numele de utilizator și parola.



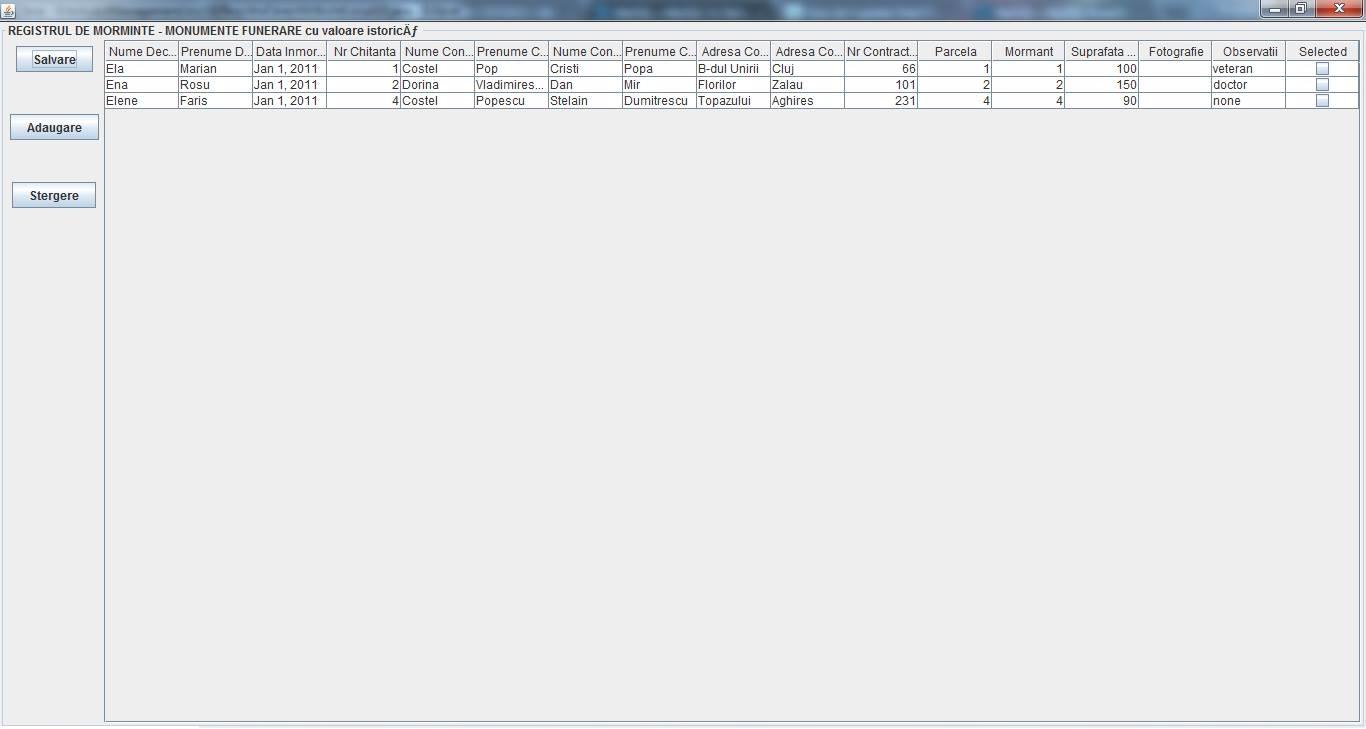
**Fig. 2.2 Fereastra Catalog Registre**

După alegerea registrului în care utilizatorul dorește sa facă modificări (ex. Registru Monumente Funerare, Fig. 2.3.) pe ecran este afișată o fereastră ce conține campurile aspecifice registrului ales.

Pentru a putea introduce date in registru, utilizatorul le poate introduce direct în tabel completând câmpul corespunzător, iar apoi acționând butonul de Adăugare.

Pentru modificarea datelor din tabel, utilizatorul va modifica doar câmpul dorit, urmând să acționeze butonul de Salvare. În cazul în care utilizatorul nu dorește sa salveze modificările, trebuie doar sa debifeze campul Selected și să acționeze butonul de Salvare.

Pentru a șterge o intrare din registru, se va activa campul Selected, apoi se acționează butonul Sterge.



**Fig. 2.3 – Registrul de Morminte – Monumente Funerare cu valoare istorică**

**3.Tehnologii folosite în realizarea aplicației (Detalii de implementare)**

În realizarea acestei aplicații am folosit MySql pentru realizarea bazei de date, aplicatia este implementata în Java, și folosește framework-ul Hibernate pentru a facilita legatura cu baza de date.

Fiecare tabelă din baza de date are asociată o clasă care se regăsește în pachetul Entity. În DAO se regasește câte o clasă pentru fiecare tabel, care conține interogările utilizate în realizarea operațiilor CRUD. Pachetul Service apelează DAO-urile.

Există o fereastră de tip LoginWindow care este creeată de un Controller; după logare, se creează un nou Controller pentru gestionarea registrelor, și care la rândul lui conține Controlleri care gestioneaza fereastra principală.

Fiecare registru este afișat într-o fereastră care conține un tabel de tip JTable care are asociat o clasă care extinde AbstractTableModel, acest model se regăsește pentru fiecare registru în parte, și se află în pachetul UITableModels.

Fiecare TableMode contine o listă de intrări în care se regăsesc toate coloanele specifice fiecărui registru. De asemenea, fiecare rând al registrului reprezinta o clasă, care se găsește în pachetul Registrii. În aceste clase există interogări pentru a construi fiecare registru in parte, conțin și un check-box isModified care dă utilizatorului opțiunea de a salva sau nu modificările făcute de acesta în baza de date, acest flag fiind setat automat să salveze modificările, dar userul are posibilitatea de a debifa check-box-ul în cazul în care nu dorește să salveze modificările. TableModel mai conține și butoane pentru salvarea, actualizarea sau ștergerea modificărilor.