UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Clinică medicală

Proiect la disciplina Ingineria Programării

Studenți:
Bordeian Marius
Coșmănescu Andreea Roxana
Ostafciuc Andrei
grupa 1407A

Iași, 2016

Iași, 2016

Cuprins

Capitolul 1. Descriere generală a aplicației

Aplicația își propune să faciliteze relația dintre pacient și clinică, să ușureze munca medicului și să ajute la gestionarea cu succes a celor mai importante elemente din clinică: pacienți și medici.

Există trei tipuri de utilizatori ai aplicației: pacient, doctor și administrator.

Din interfața aplicației există posibilitatea de logare a unui utilizator sau de creare a unui nou cont, posibilitatea de creare a unui cont nou este disponibilă doar pacienților, cu alte cuvinte se poate crea doar cont pentru pacient, contul unui doctor sau al unui alt administrator este creat de către un administrator, cel puțin un administrator existând în momentul de deploy a aplicației.

Pacientul care are un cont se poate loga, poate vedea toate departamentele clinicii și toți doctorii, precum și informații despre acestea, își poate vedea datele personale, poate edita datele personale, poate crea o nouă programare în funcție de disponibilitatea doctorului ales, poate vedea un istoric al programărilor sale sau un istoric al rezultatelor. Astfel pentru un pacient, este foarte simplu să vadă rezultatele date de medic la oricare dintre programările sale și e simplu să vadă medicația propusă de medic, în unele cazuri când nu sunt necesare alte discuții sau investigații, pacientul poate vedea rezultatele sale fără o deplasare în plus la clinică.

Doctorul se poate loga în aplicație și poate vedea toate programările pe care le are, de asemenea pentru fiecare programare poate asigna un rezultat, un diagnostic, medicație. El poate vedea și un istoric al pacienților săi, istoricul este foarte util deoarece toate afecțiunile și toate medicamentele prescrise sunt centralizate și pot fi accesate cu ușurință.

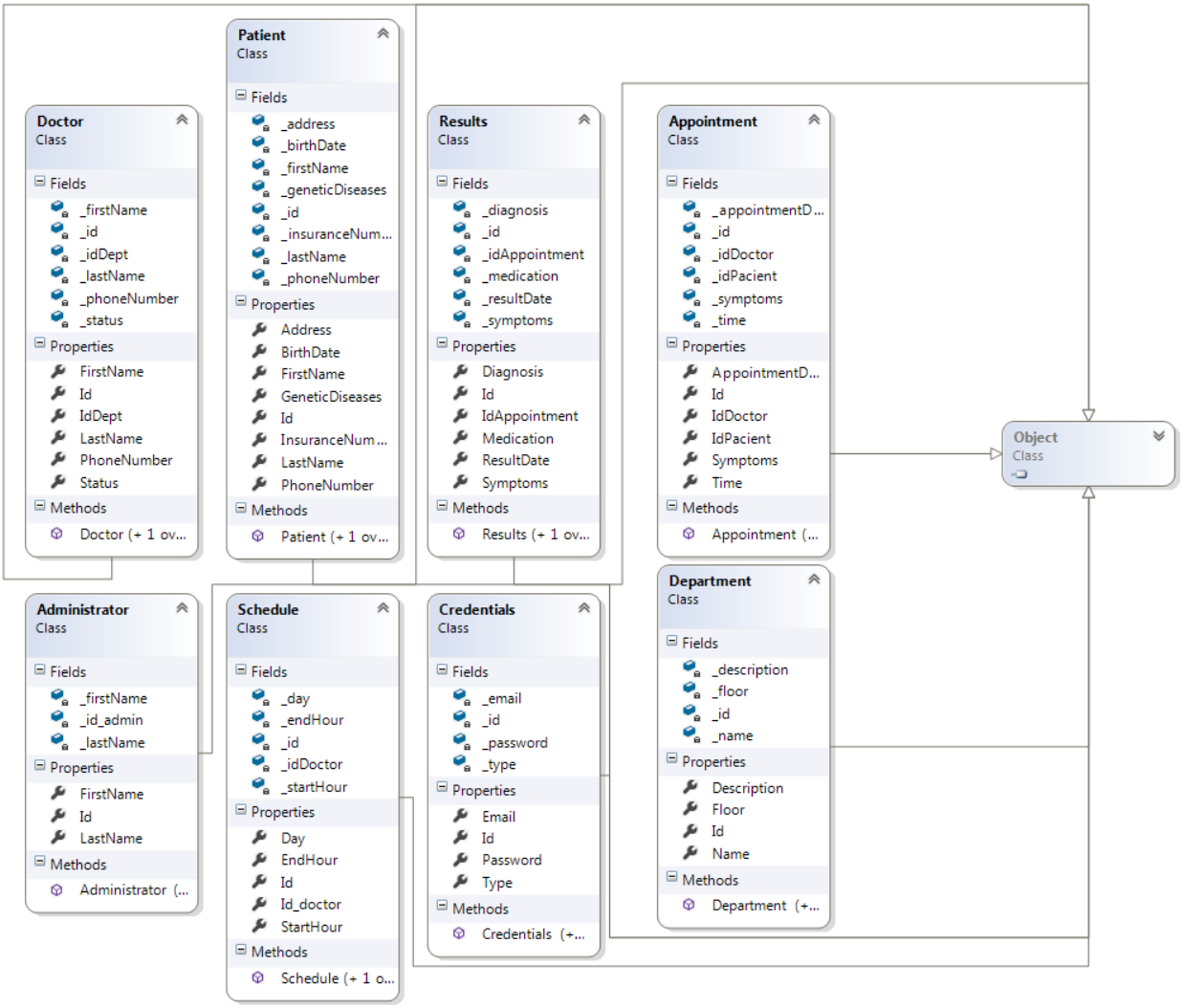
Administratorul se logează în aplicație și poate face modificări asupra personalului clinicii. El poate introduce un nou departament în cazul în care clinica se dezvoltă, poate introduce un nou medic sau poate crea un cont de administrator.

Capitolul 2. Prezentare generală a implementării

Aplicația este separată în mai multe module.

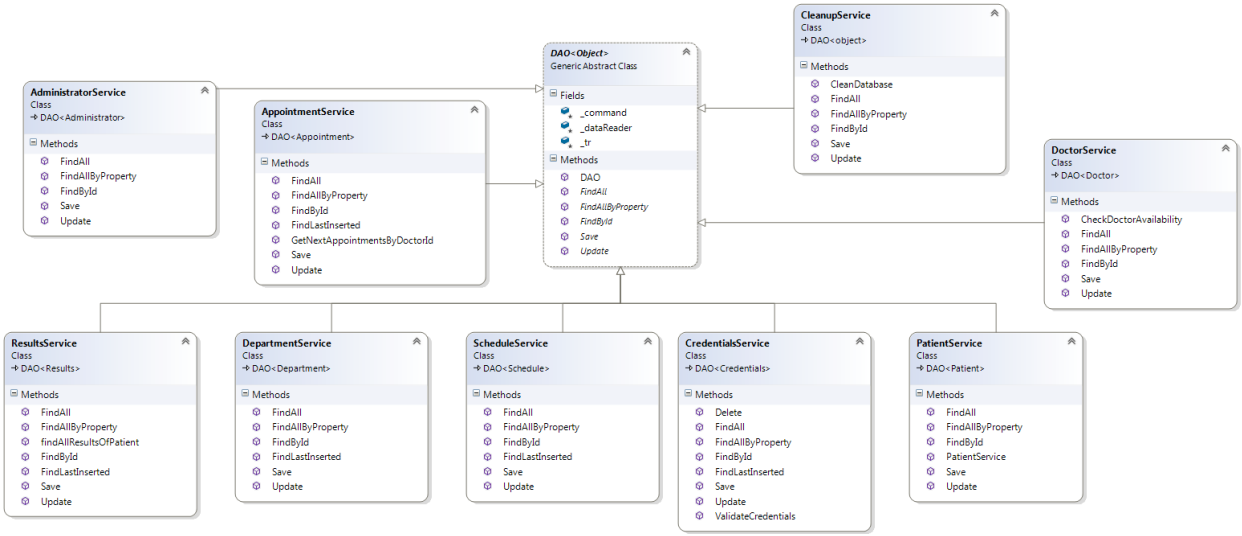
2.1. Modulul Entity

Conține clasele care mapează tabelele din baza de date. Structura este prezentată în diagrama de mai jos.



2.2. Modulul DAO

Conține servicii ce fac ca interacțiunea cu baza de date să fie ușoară. Fiecare clasă serviciun conține metode pentru cele mai utilizate operații asupra bazei de date.



2.3. *Modulul UIViews- descriere Template Method*

În acest modul am implementat șablonul Template Method (Metoda Tipar).

Am observat că interfața noastră are tot timpul trei elemente principale a căror poziție trebuie să fie mereu aceeași. Astfel am distins trei secțiuni ale interfeței: header (conține logo-ul aplicației), footer (conține specificații de copyright) și body (care este un conținut ce se schimbă în funcție de utilizatorul logat și acțiunile pe care le face).

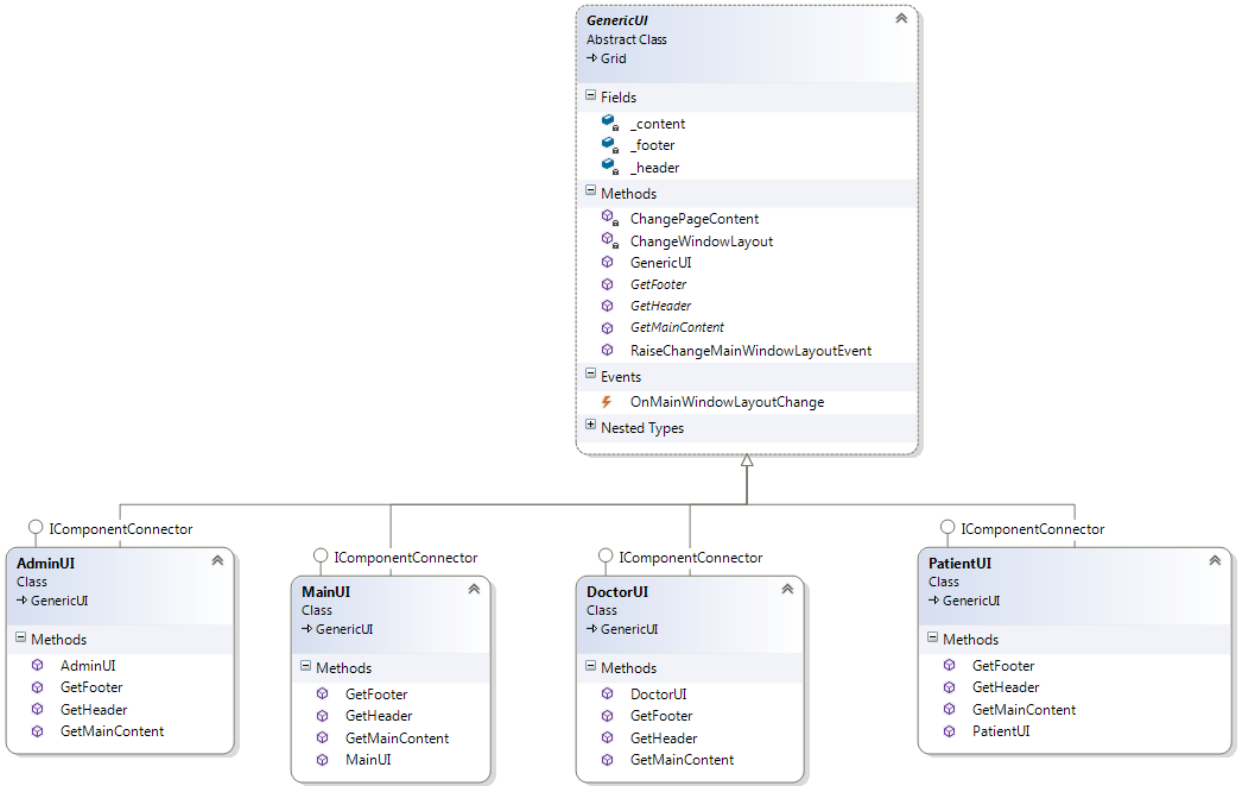
Toate aceste componente: header, footer, body sunt de tip CustomUserControl (o clasă derivată din `System.Windows.Controls.UserControl` , care suportă evenimente custom create de noi pentru a putea naviga mai ușor de pe o pagină pe alta, în funcție de acțiunile utilizatorului).

Pentru fiecare tip de utilizator am creat câte o interfață diferită, pentru că acțiunile lor și conținutul de care au nevoie diferă. Au rezultat următoarele tipuri: MainUI, DoctorUI, PatientUI, AdminUI. Toate acestea au nevoie de header, footer, body, deci respecă tiparul.

Astfel ne-am dat seama că aici se respectă o „rețetă" : indiferent de conținutul header-ului, footer-ului, body-ului ele sunt elemente care au la bază CustomUserControl și trebuiesc poziționate după cum urmează: header-ul sus, având înălțimea fixată, footer-ul jos având înălțimea fixată, iar body intre acestea două, ocupând restul spațiului disponibil.

Am creat clasa abstractă de bază care este GenericUI (un element de tip Grid) și care conține trei elemente de tip CustomUserControl și care în constructor poziționează aceste elemente după rețeta descrisă mai sus.

Celelalte clase MainUI, DoctorUI, PatientUI, AdminUI sunt derivate din GenericUI și implementează metodele abstracte GetHeader, GetFooter, GetMainPage astfel încât să conțină elemente specifici fiecărei interfețe utilizator în parte, dar poziția lor este stabilită după rețeta descrisă în constructorul clasei.



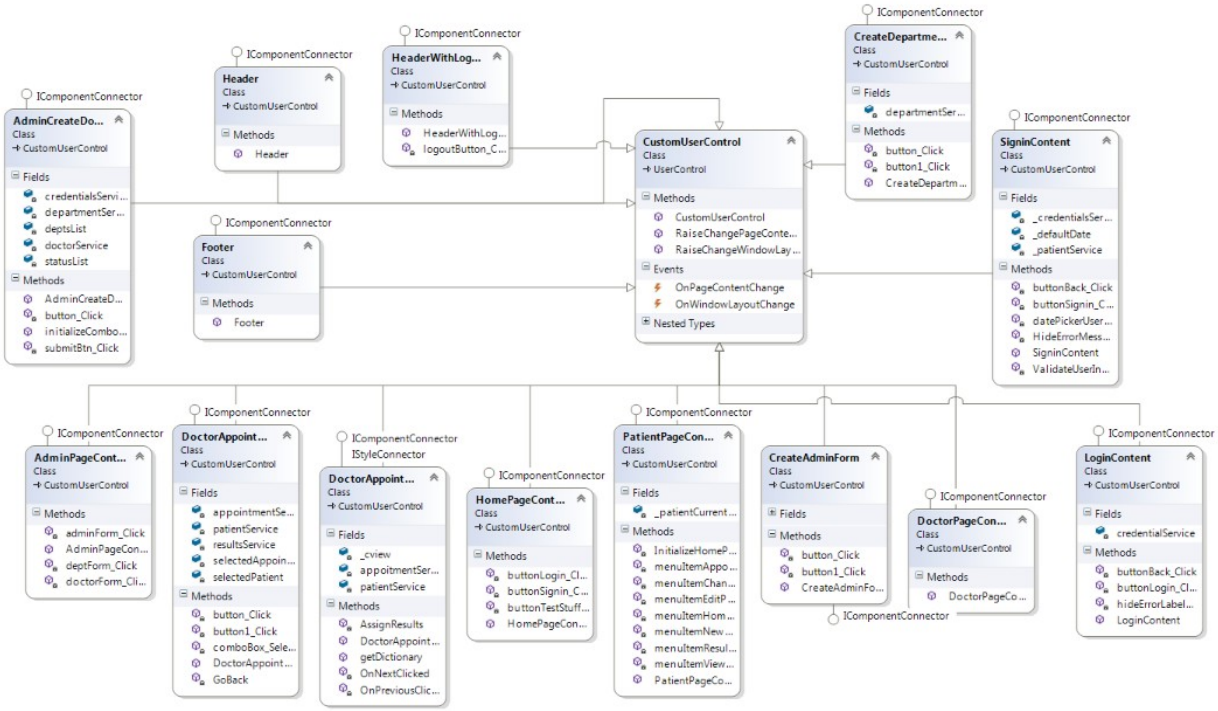
Rețeta referitoare la poziționarea elementelor este descrisă în următoarele secvențe de cod

```
private CustomUserControl _header, _footer, _content;
        public delegate void ChangeMainWindowLayoutHandler(object sender,
WindowLayoutEventArgs e);
        public event ChangeMainWindowLayoutHandler OnMainWindowLayoutChange;


        public abstract CustomUserControl GetHeader();
        public abstract CustomUserControl GetFooter();
        public abstract CustomUserControl GetMainContent();
        public GenericUI() : base()
        {
            this.VerticalAlignment = System.Windows.VerticalAlignment.Top;
            _header = GetHeader();
            _footer = GetFooter();
            _content = GetMainContent();
            _content.HorizontalAlignment = HorizontalAlignment.Stretch;
            _content.VerticalAlignment = VerticalAlignment.Stretch;

            _content.OnPageContentChange += new
CustomUserControl.ChangePageContentHandler(ChangePageContent);
            _content.OnWindowLayoutChange += new
CustomUserControl.ChangeWindowLayoutHandler(ChangeWindowLayout);
            _header.OnWindowLayoutChange+= new
CustomUserControl.ChangeWindowLayoutHandler(ChangeWindowLayout);
            this.RowDefinitions.Add(new RowDefinition());
            this.RowDefinitions.Add(new RowDefinition());
            this.RowDefinitions.Add(new RowDefinition());

            this.RowDefinitions[0].Height = new GridLength(50);
            this.RowDefinitions[1].Height = new GridLength(1, GridUnitType.Star);
            this.RowDefinitions[2].Height = new GridLength(30);
            Grid.SetRow(_header, 0);
            Grid.SetColumn(_header, 0);
            Grid.SetRow(_content, 1);
            Grid.SetColumn(_content, 0);
            Grid.SetRow(_footer, 2);
```

```
            Grid.SetColumn(_footer, 0);

            this.Children.Add(_header);
            this.Children.Add(_content);
            this.Children.Add(_footer);
        }
```

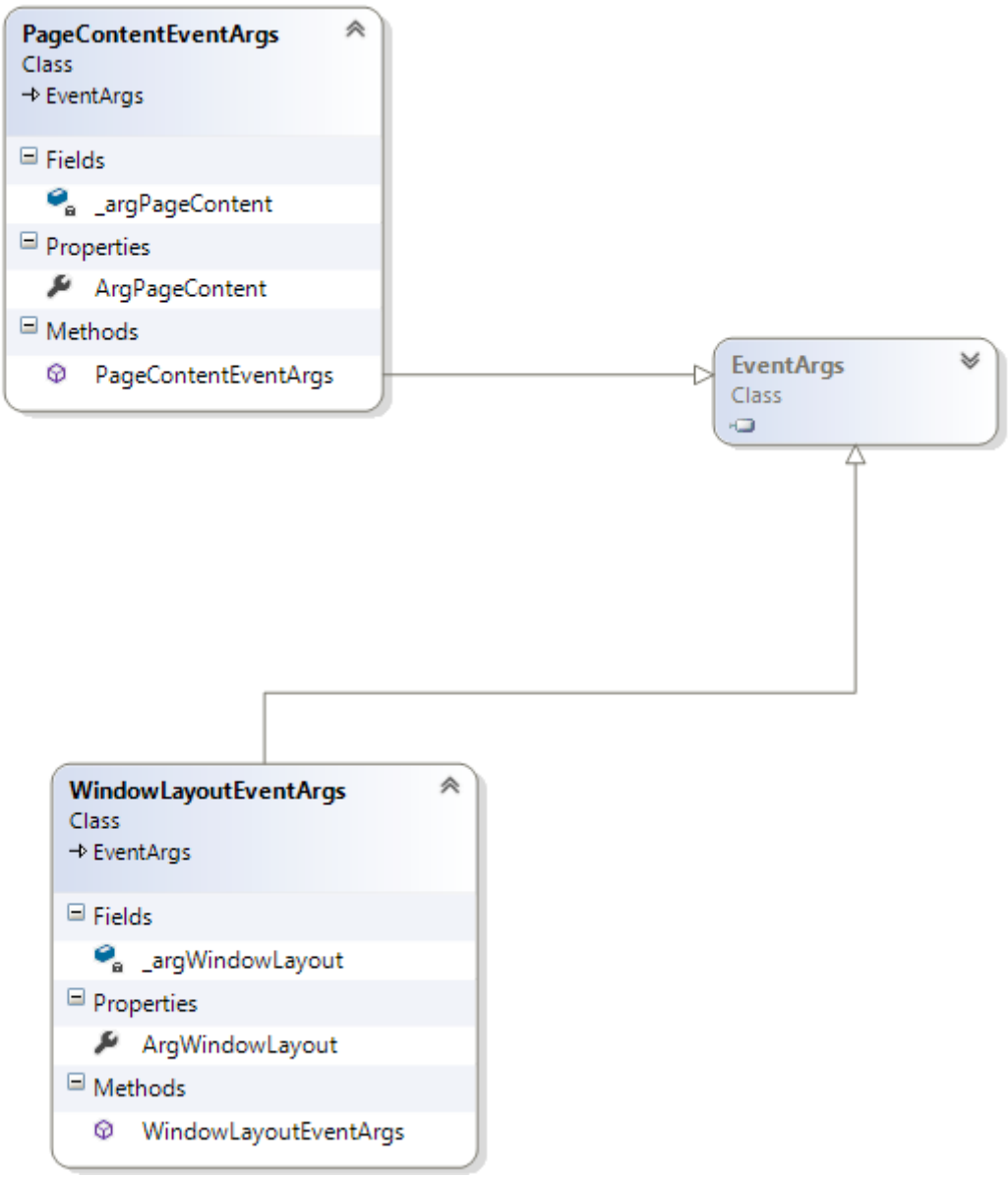
Pe lângă poziționarea elementelor, se asignează și handlere pentru evenimentele ce pot să apară (schimbare conținut *mainPage* și schimbare layout- schimbă tipul de interfață).

2.4. Modulul GenericControls



Acest modul conține toate elementele de interfață folosite pentru această aplicație.

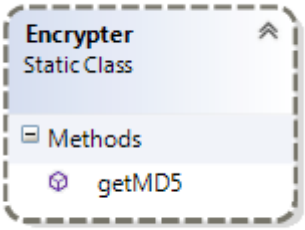
2.5. Custom EventArgs



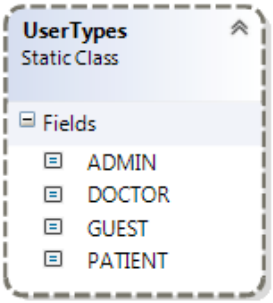
Aceste argumente sunt transmise atunci când sunt declanșate evenimente de tip: schimbă pagina (mainPage) și este folosit argumentul *PageContentMainArgs* care conține un obiect de tip CustomUserControl ce reprezintă noul conținut al body-ului, schimba layout-ul aplicației, ceea ce înseamnă că se schimbă tipul interfeței. (de exemplu atunci când un utilizator logat, vrea să părăsească sesiunea curentă și dă logout, el este redirecționat din interfața specifica lui AdminUI sau PatientUI sau DoctorUI spre MainUI, care este pagina principală a aplicației). În ultimul caz se transmite un argument de tip *WindowLayoutEventArgs,* care conține tipul noii interfețe ce urmează a fi afișată.

2.6. Modulul Utils

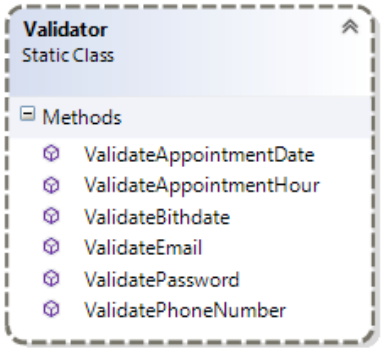
Acest modul conține clase care sunt neceare în majoritatea modulelor. Definiții de tipuri utilizatori, clase pentru encriptarea datelor, clase care definesc numele coloanelor din tabele din baza de date, aceste proprietăți sunt folosite în serviciile ce se găsesc în modul DAO și faptul că au fost organizate și definite aici, face ca orice modificare asupra unui numelor coloanelor al unui tabel din baza de date să nu modifice întregul proiect ci modificările să fie făcute doar în această clasă.



Clasa Encrypter conține o metodă folosită pentru codificarea parolei utilizatorului.



Clasa UserTypes conține definiții pentru cele patru tipuri de interfețe utilizator.

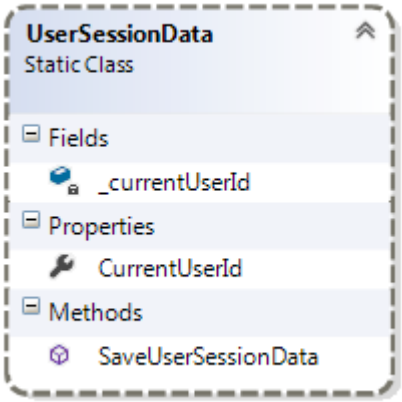


Clasa Validator conține funcții utilizate pentru validarea câmpurilor completate de utilizator.

Principalele validări care se face: email-ul trebuie să aibă un format valid de forma example@example.exa, numărul de telefon trebuie să conțină numai cifre și să aibă exact 10 caractere, data nașterii nu trebuie să fie mai mare sau egală cu ziua curentă, data programării nu trebuie să fie mai mică decât ziua curentă deoarece nu se poate face o programare în trecut, parola trebuie să aibă minim 6 caractere, ora programării trebuie să se regăsească în intrevalul 00-24.

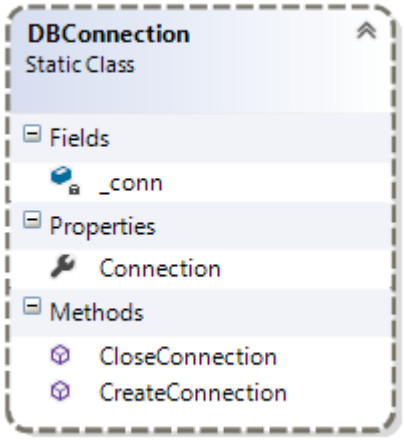
2.7. Modulul SessionData

Conține clasa *UserSessionData* care stochează informații necesare indentificării utilizatorului logat la un moment dat.



2.8. Modulul DBConnection

Conține o clasă care se ocupă cu conexiunea la baza de date, conexiune fără de care aplicația nu poate rula corect. Conexiunea la baza de date se realizează o singură dată, la pornirea aplicației, fiind persistentă pe tot parcursul rulării aplicației, conexiunea cu baza de date fiind distrusă la închiderea ferestrei principale a aplicației.



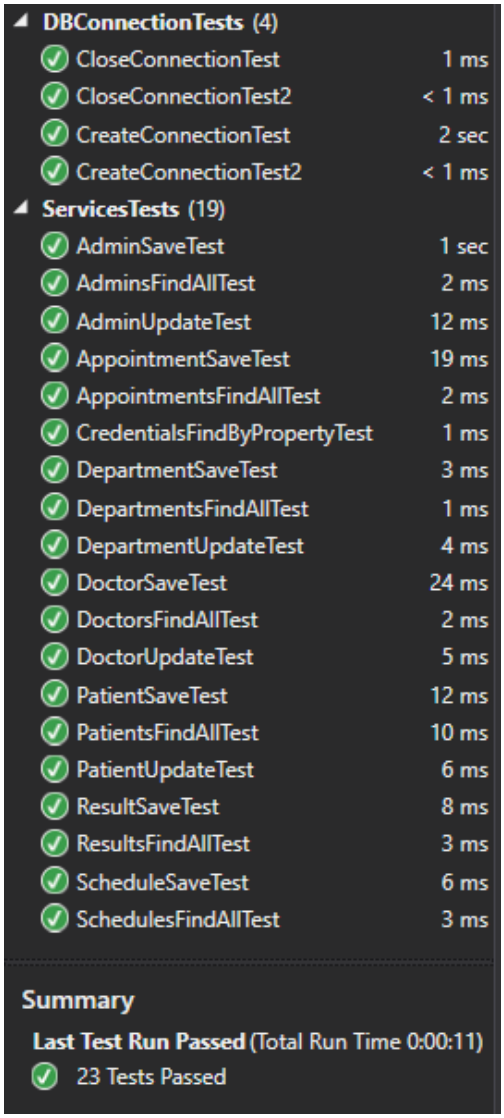
2.9. Modulul UnitTests

Testarea unităților aplicației este o parte extrem de importantă în special pentru o aplicație în stilul non-academic.

Sunt prezente 23 cazuri de test ce acoperă o parte semnificativă a funcționalității aplicației.

MedicalClinic fiind o aplicație ce se bazează pe accesul si operații asupra unei bază de date delimiteaza 4 cazuri de test pentru partea de conexiune intre aplicație si Baza de Date și 19 cazuri pentru partea de servicii ce gestionează datele servind drept interfață de comunicare între MedicalClinic Desktop Application și MedicalClinic Database Server.

```
    [TestClass()]
    public class ServicesTests
    {
       [TestMethod()]
        public void DoctorSaveTest()
        {
            OpenConnection();
   Credentials c = new Credentials(lastName + firstName + "@testDoc.com", "0000", User-
Types.DOCTOR);
            Doctor d = new Doctor(credentialsService.Save(c), lastName, firstName, de-
partmentService.FindAll()[0].Id, "85643213", Utils.DoctorStatus.ACTIVE);
            doctorService.Save(d);
      Assert.IsNotNull(doctorService.FindAllByProperty(DoctorTableProperties.LastName,
d.LastName));
            CloseConnection();
            }
```

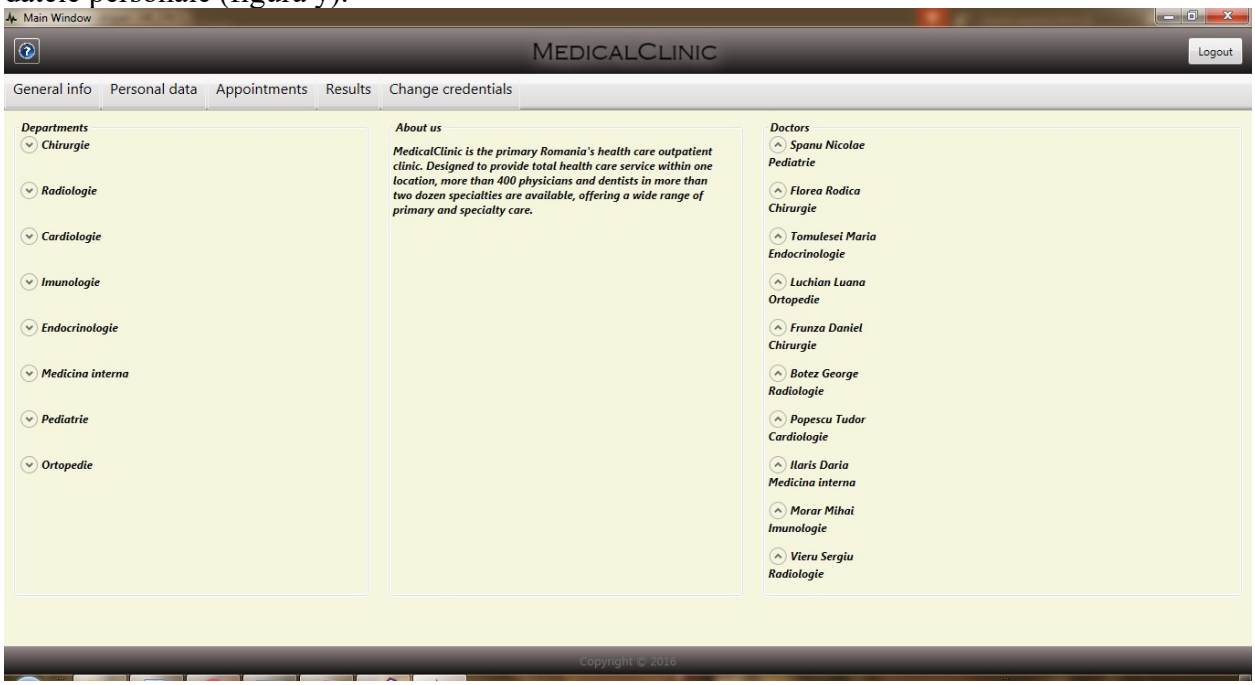


Capitolul 3. Interacțiunea utilizatorilor cu aplicația

Pagina principală a aplicației este reprezentată în imaginea de mai jos, un utilizator se poate loga dacă are un cont sau își poate crea unul apăsând butonul ***Sign up.***

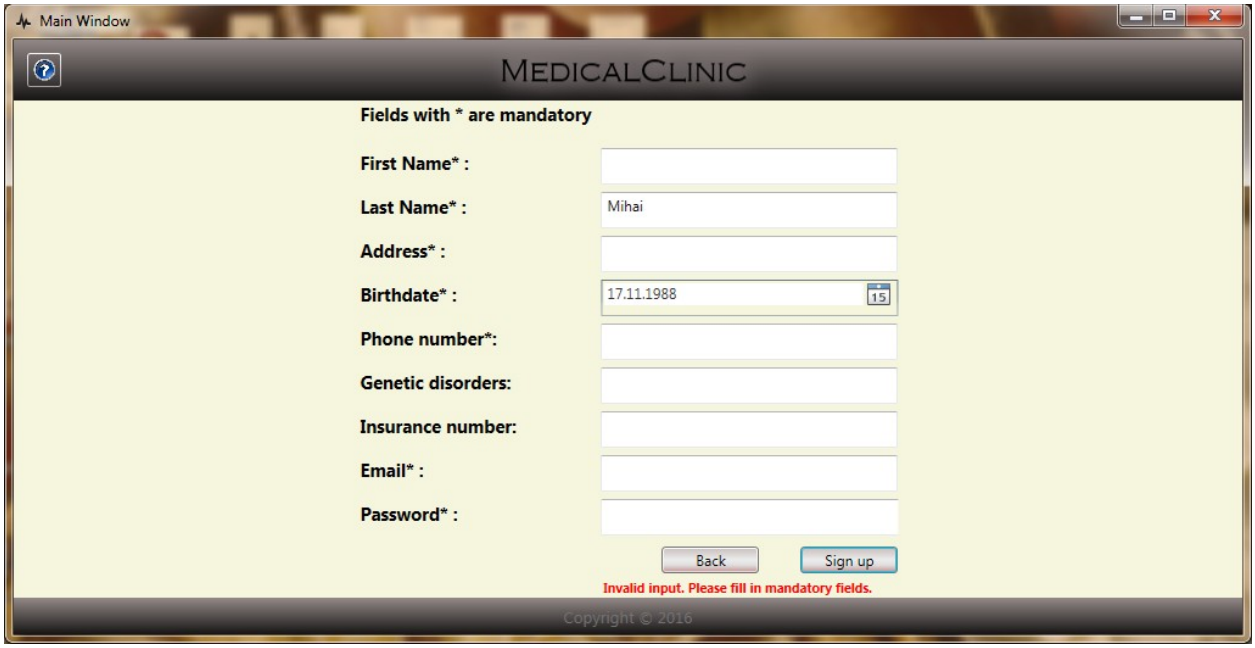
3.1. Interacțiunea pacientului

Pacientul care are un cont se poate loga, poate vedea toate departamentele clinicii și toți doctorii, precum și informații despre acestea (așa cum se poate vedea în figura x), își poate vedea datele personale (figura y).

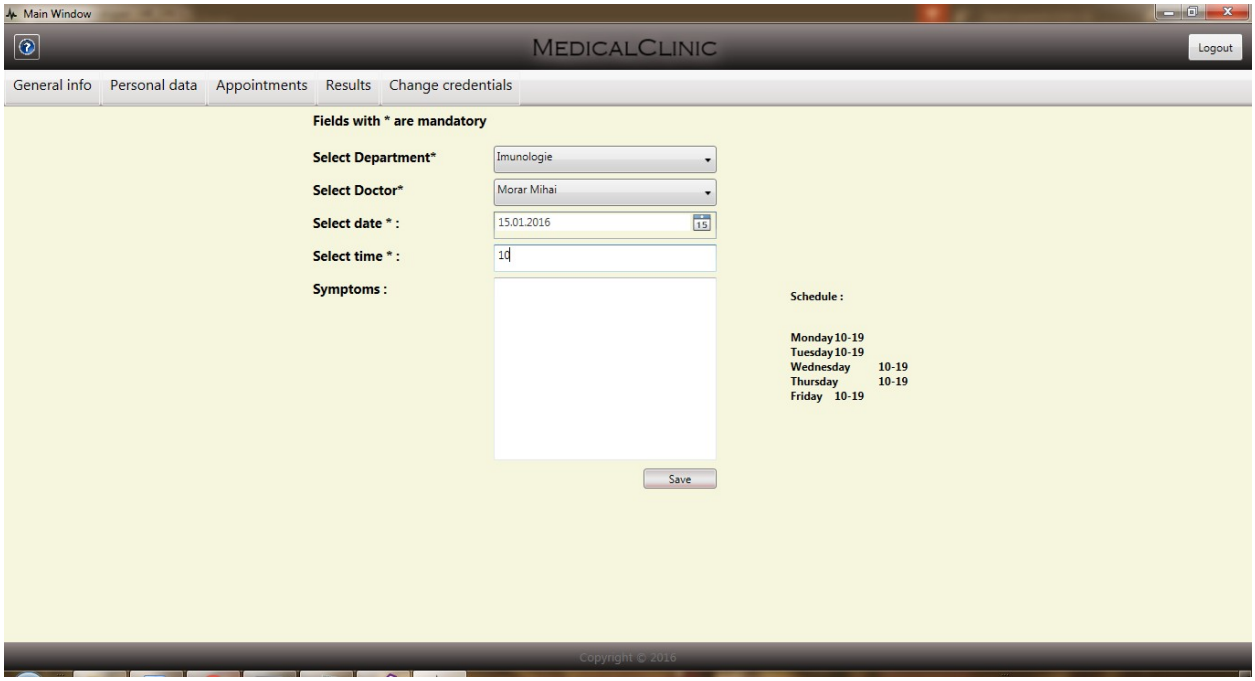


Pacientul poate edita datele personale și poate să-și modifice datele de logare: parola și emailul.

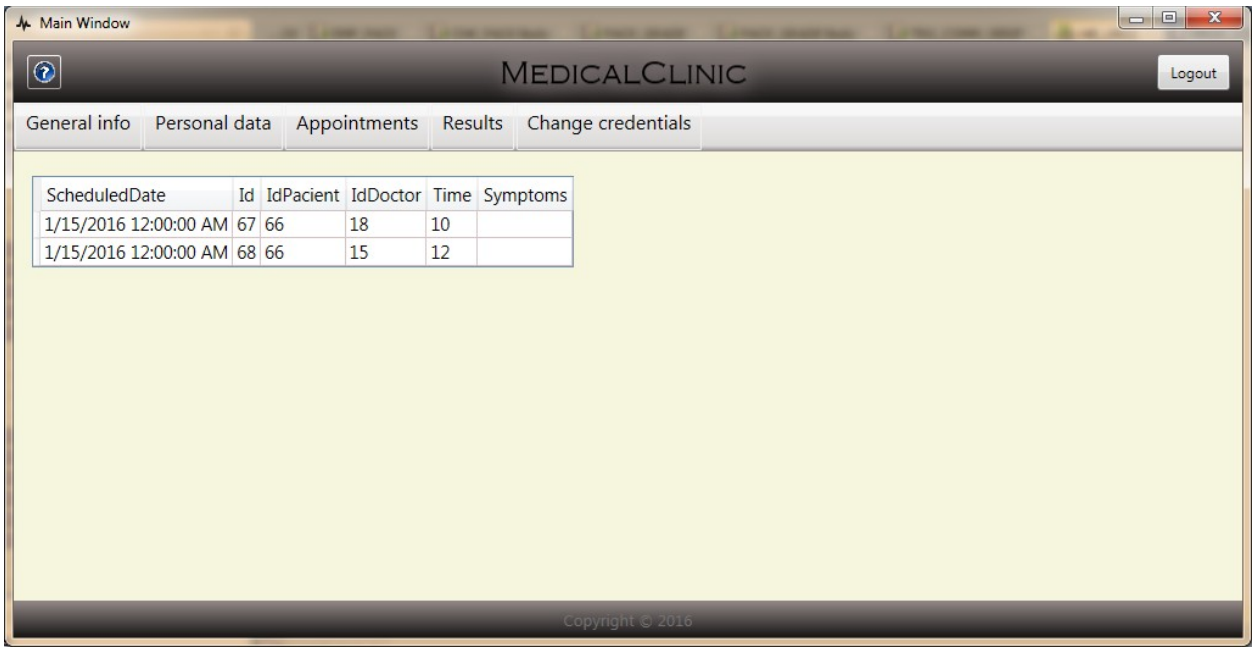
Dacă datele pe care le introduce utilizatorul sunt invalide, atunci el primește un mesaj de eroare pentru a corecta greșelile.



Odată autentificat, acesta poate crea o nouă programare în funcție de disponibilitatea doctorului ales, poate vedea un istoric al programărilor sale sau un istoric al rezultatelor.



Astfel pentru un pacient, este foarte simplu să vadă rezultatele date de medic la oricare dintre programările sale și e simplu să vadă medicația propusă de medic, în unele cazuri când nu sunt necesare alte discuții sau investigații, pacientul poate vedea rezultatele sale fără o deplasare în plus la clinică.



3.2. Interacțiunea administratorului

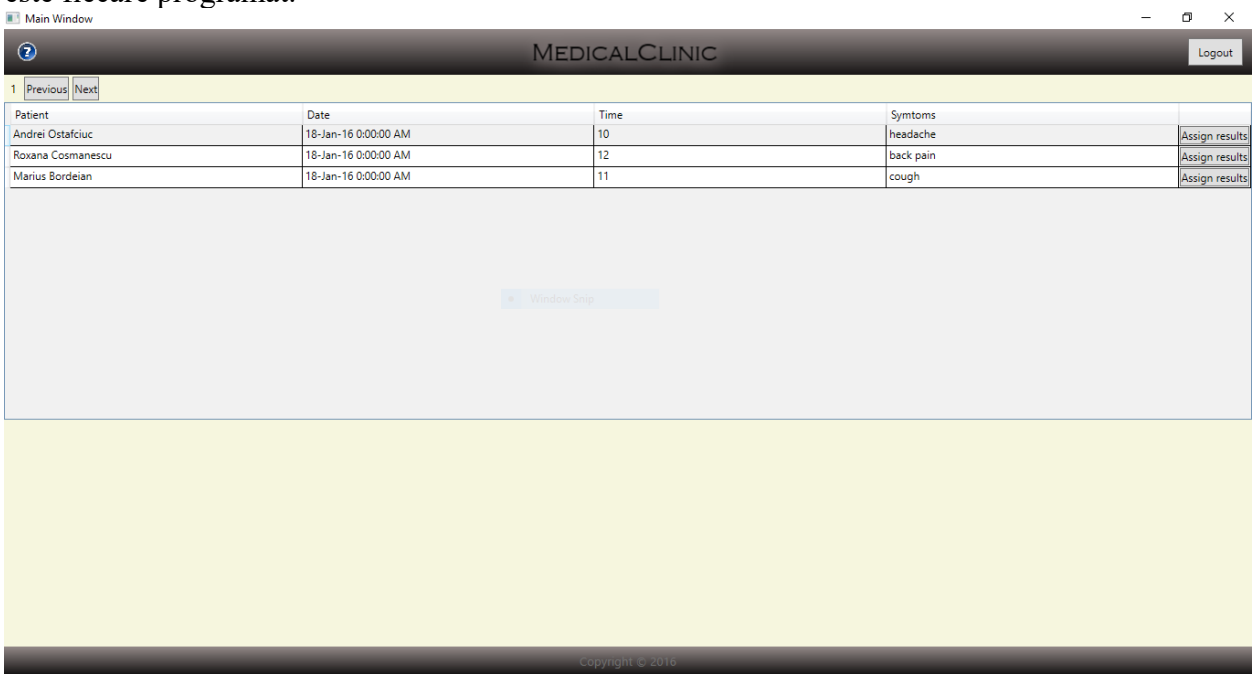
Un administrator logat are 3 opțiuni :

- Creare de account pentru un doctor
- Creare de account pentru un administrator
- Creare de departamente

În aceasta maniera se deține un control foarte stric asupra accesibilitatii la baza de date. Un user nu se poate „autoproclama" doctor și nici administrator, motiv pentru care conturile acestor pot fi create doar de către un administrator.

3.3. Interacțiunea doctorului

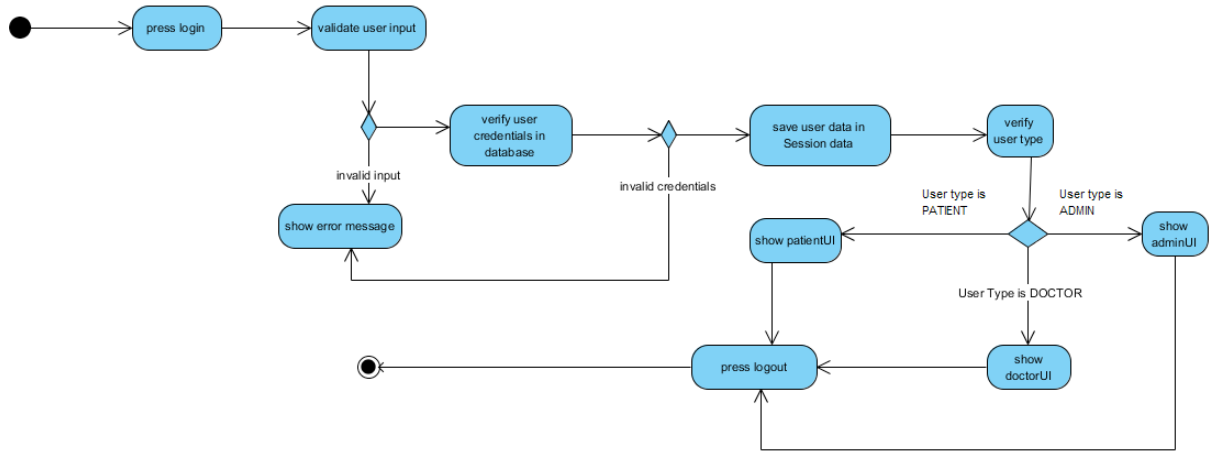
Doctorul, odată logat, este intampinat de o interfata ce cuprinde o lista de programari cu pacienții din viitor. El poate vedea cu ușurința câte programari are în ziua curenta și ora la care este fiecare programat.



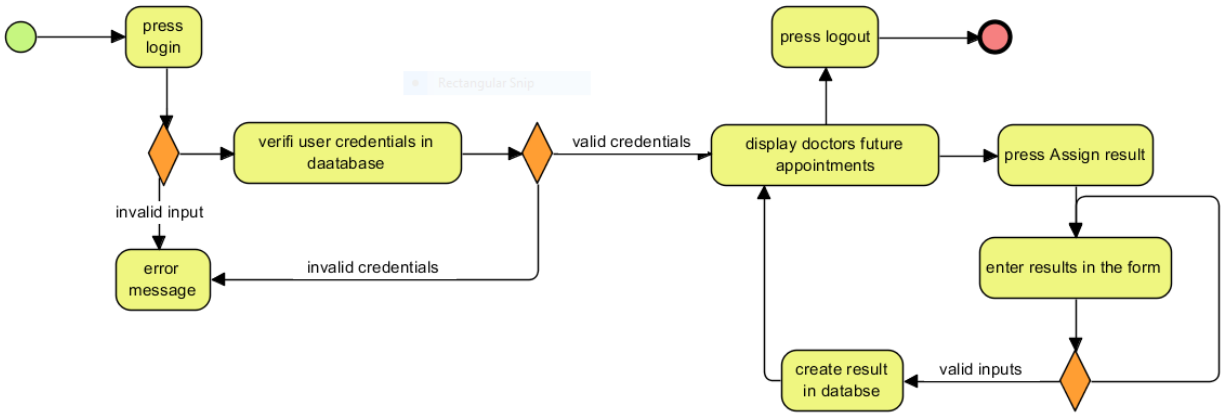
În urma consularii, doctorul poate asigna programării un resultat, ce conține simptomele, diagnosticul și medicatia data de acesta. Totodata el poate vedea în aceeași fereastra și istoricul pacientului, cât și rezultatele acestuia.

3.4. Diagrame de activitate

3.4.1. Diagrama pentru autentificarea utilizatorului

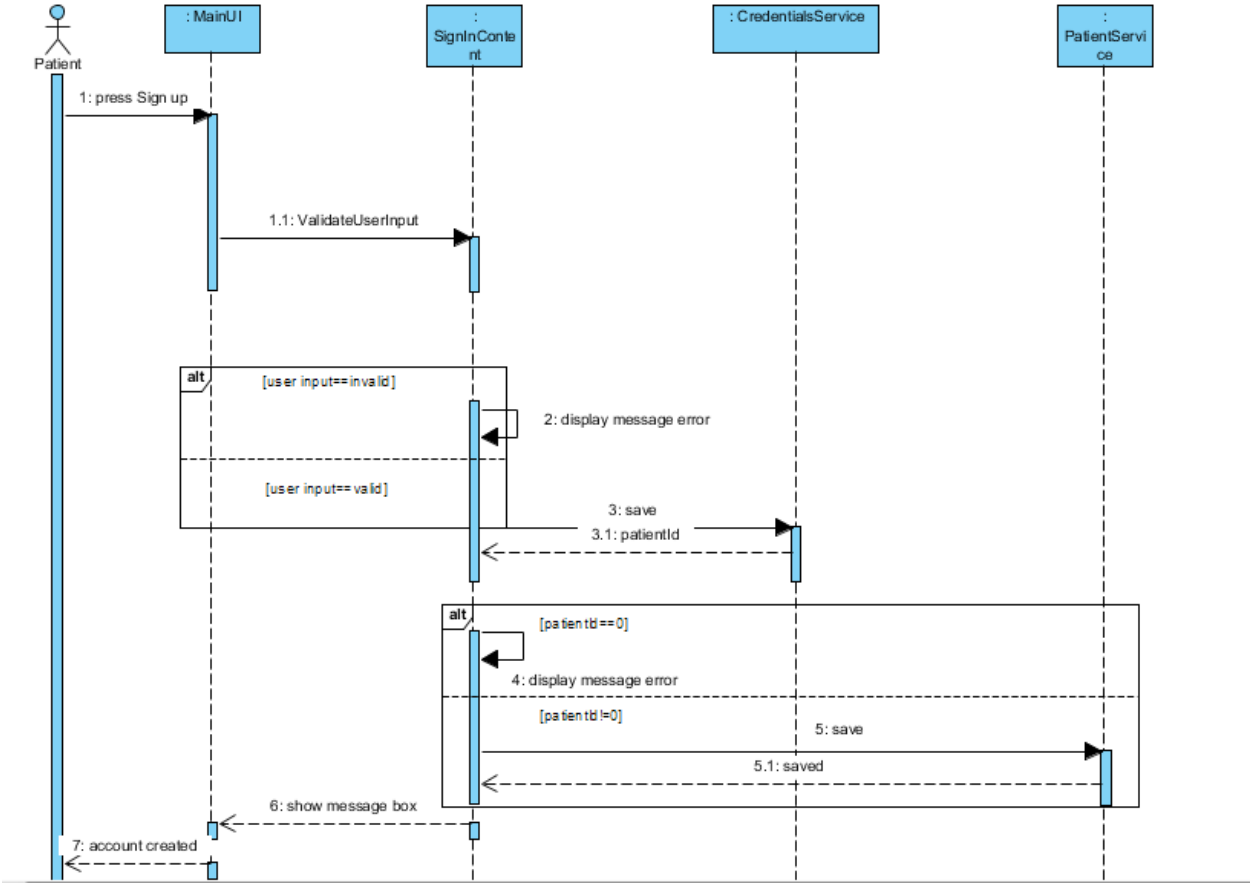


3.4.2. Diagrama pentru atibuire a unui rezultat unei consultatii



3.5. Diagrame de secvență

3.5.1. Diagrama de creare cont pacient



Capitolul 4. Concluzii

În concluzie această aplicație este utilă atât doctorilor cât și pacienților, doctorii au acces la programările curente, pot asigna rezultate, având o interfață ușor de utilizat, nu mai este nevoie de rapoarte scrise manual, sau de dosare pline cu datele pacienților, istoricul acestora fiind mult mai ușor de accesat.

Pacienții își pot accesa istoricul medical oricând este nevoie, datele lor sunt păstrate în siguranță, confidențialitatea rezultatelor este asigurată deoarece pacientul le accesează pe baza unei parole setată de el însuși.

Administratorii sunt cei care introduc date în sistem și sunt responsabili de corectitudinea lor, tot ei crează conturile doctorilor.

Avantajele tuturor utilizatorilor sunt evidente, clinica medicală beneficiind astfel de date centralizate, istoricele pacienților, a doctorilor și astfel relația pacient-clinică-medic este facilitată.

Capitolul 5. Anexe

Anexa 1. Codul sursă

Link GitHub : https://github.com/AndreiOstafciuc/MedicalClinic