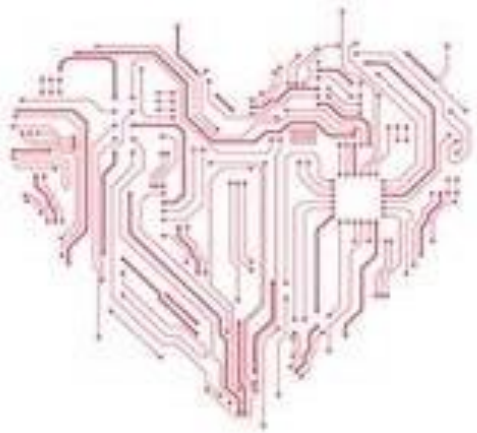
Biomedics

-SISTEM PURTABIL DE SUPRAVEGHERE A STĂRII DE SĂNĂTATE-



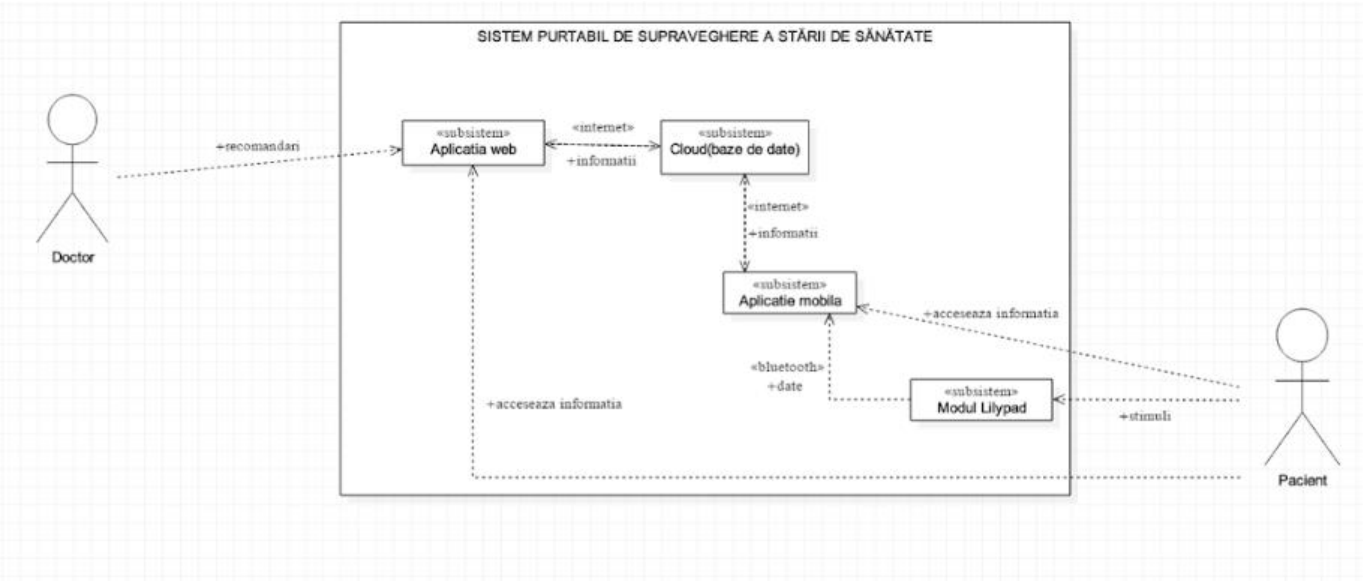
BIOMEDICS

Colectivul de elaborare

- Capăta Cristina(Coordonator sef/ Programator sef/Modul Web)
- Bociort Dinu Iulian(Coordonator adjunct/ Modul Mobile)
- Burlacu Camelia Adelina(Secretar/Modul Mobile)
- Bușe Dragoș(Modul Web)
- Bălaș Victor(Modul Web)
- Casapu Mircea-Adrian(Modul Cloud)
- Bujor Daniela(Modul Cloud)
- Bunceanu Gabriel-Daniel(Modul Inteligent)
- Bîldea Gabriel-Ioan(Modul Inteligent)

Cuprins

Arhitectura programului



1. Tehnologii și medii de dezvoltare

Proiectul Lilycord se bazează pe mai multe tehnologii și medii de dezvoltare pentru fiecare modul în parte:

- **Modulul Inteligent**

Mediul de dezvoltare: **Arduino IDE**, este o aplicație folosită pentru a scrie și încărca programe pe plăci Arduino și care suportă limbajele de programare C și C++ folosind reguli speciale de structurare a codurilor. Codul scris de utilizator necesită doar două funcții de bază: funcția pentru setări(void setup()) și funcția principală a programului(void loop()).

Tehnologii: C/C++ cu library embedded.

- **Modulul Mobile**

Mediul de dezvoltare: **Android Studio** , este aplicația oficială de la Google pentru sistemul lor de operare Android. Aceasta este bazat pe software-ul IntelliJ IDEA de la JetBrains.

Tehnologii: Java

- **Modulul Web**

Mediul de dezvoltare: Microsoft Visual Code, editor de cod sursă ce include suport pentru IntelliSense, Git, Snippets.

Tehnologii: HTML5, CSS3, JavaScript, Boostrap

- **Modulul Cloud**

Mediul de dezvoltare: Microsoft Visual Studio, ce include un set complet pentru dezvoltarea unui backend cu ajutorul serviciilor oferite de Azure.

Tehnologii: C#

2. Comunicarea Client-Server

Sistemul celor 4 module comunică între ele sub forma a mai multor relații client server.

Client	Server	Descriere
MobileApp	Modul Inteligent	Trimite datele citite de modulul inteligent la aplicația mobila
Cloud	MobileApp	Datele trebuie accesate bidirecțional: din aplicația mobila încărcam un mesaj text asociat unei alarme/avertizări
MobileApp	Cloud	Din server trimitem recomandări/avertizări in aplicația mobila
Web	Cloud	Aplicația web ne permite vizualizarea datelor din serverul Cloud

3. Modul în care funționează fiecare modul

Sistemul Lilycord este împărțit pe 4 module care sunt strâns legate.

- **Modulul inteligent**

Modulul Inteligent se bazează pe o componentă hardware (LilyPad) care, cu ajutorul senzorilor de puls, umiditate, temperatură și ECG, preia datele pacientului în timp real, având ca scop principal trimiterea acestora la un smartphone prin intermediul unei conexiuni Bluetooth.

- **Aplicația mobilă**

Aplicația mobilă reprezintă aplicația ce preia datele de la modulul inteligent pe care apoi le procesează pentru a le afișa în cadrul aplicației și pentru a le trimite mai departe în cadrul unui cont personal care ține evidența acestora pe un server cloud. De asemenea, aceasta realizează avertizări dacă datele primite nu sunt în limita normala și afișează recomandările si activitățile date de către un medic ce se ocupă de starea pacientului asociat aplicației.

- **Sistemul Cloud**

Cloud reprezintă o instanța de server Azure care conține:

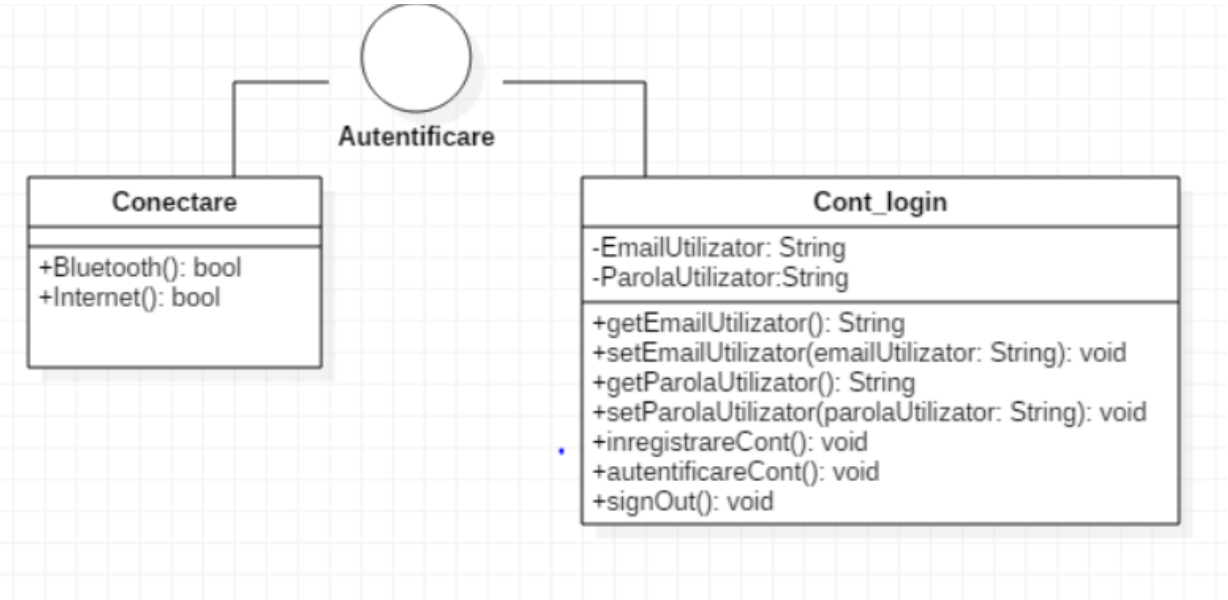
- ✓ baza de date care conține toate datele citite de la modulul inteligent, transmise prin intermediul aplicației mobile.
- ✓ tabela separata in baza de date care conține datele de autentificare ale medicilor si pacienților
- ✓ Aplicația web

- **Aplicația Web**

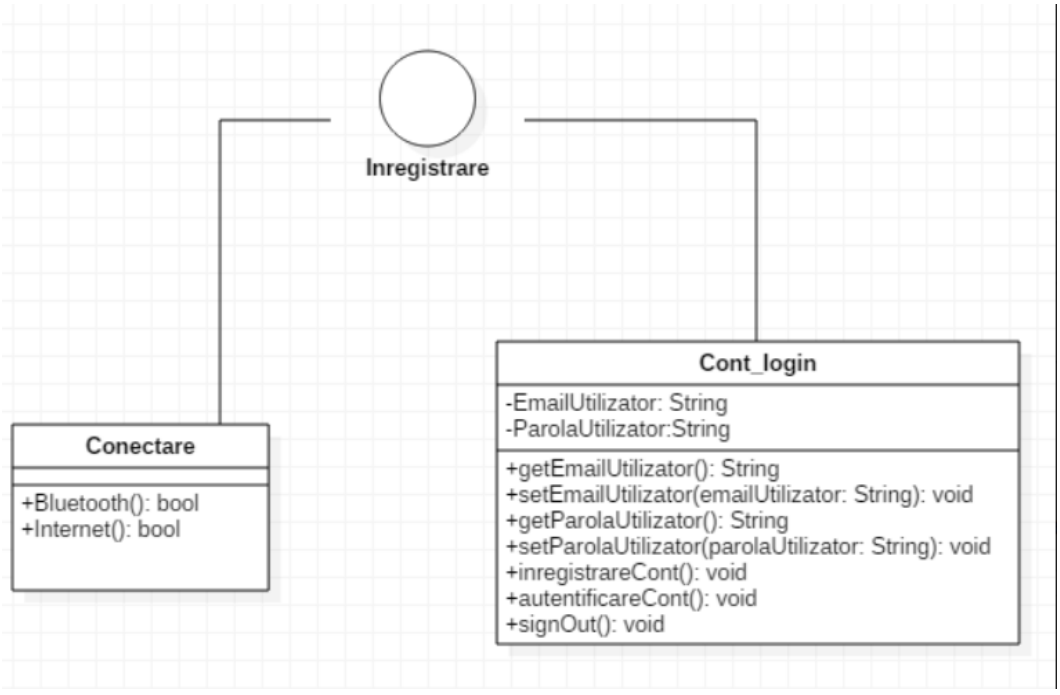
În cadrul aplicației WEB avem interfața pentru medici de unde aceștia pot monitoriza pacienții și fișa medicală a acestora, precum și trimite avertizări sau recomandări prin intermediul serviciului de Cloud înapoi la aplicația mobilă. Interfața pentru pacienți le va permite să vadă un istoric cu starea lor primită de la senzori, cât și istoricul avertizărilor și recomandărilor.

4. Diagrame clase

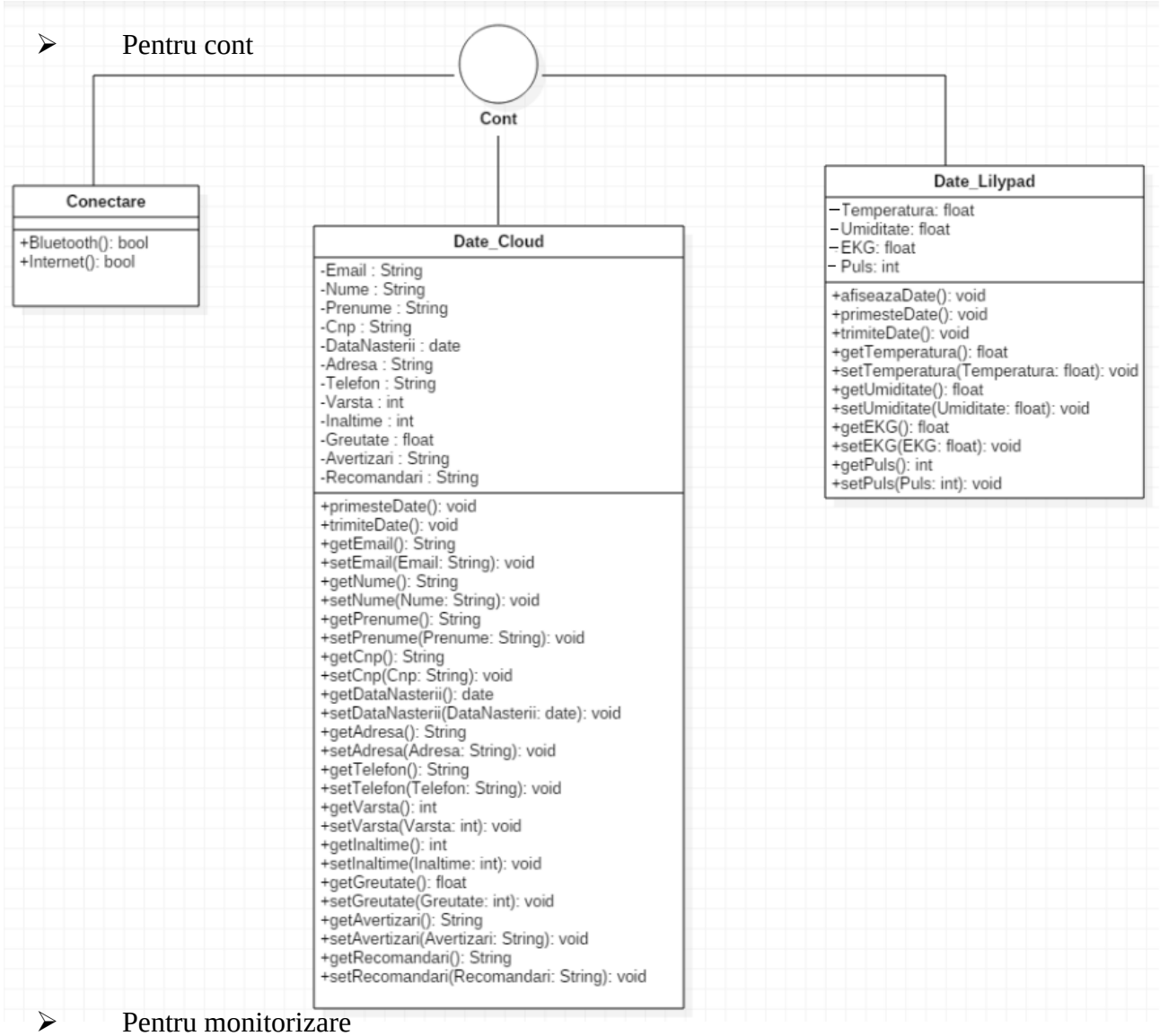
- Pentru Autentificare



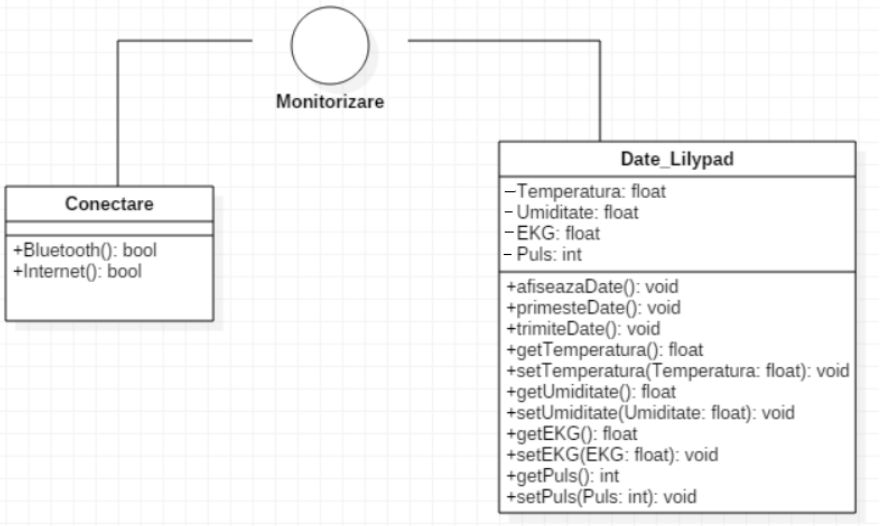
- Pentru Inregistrare



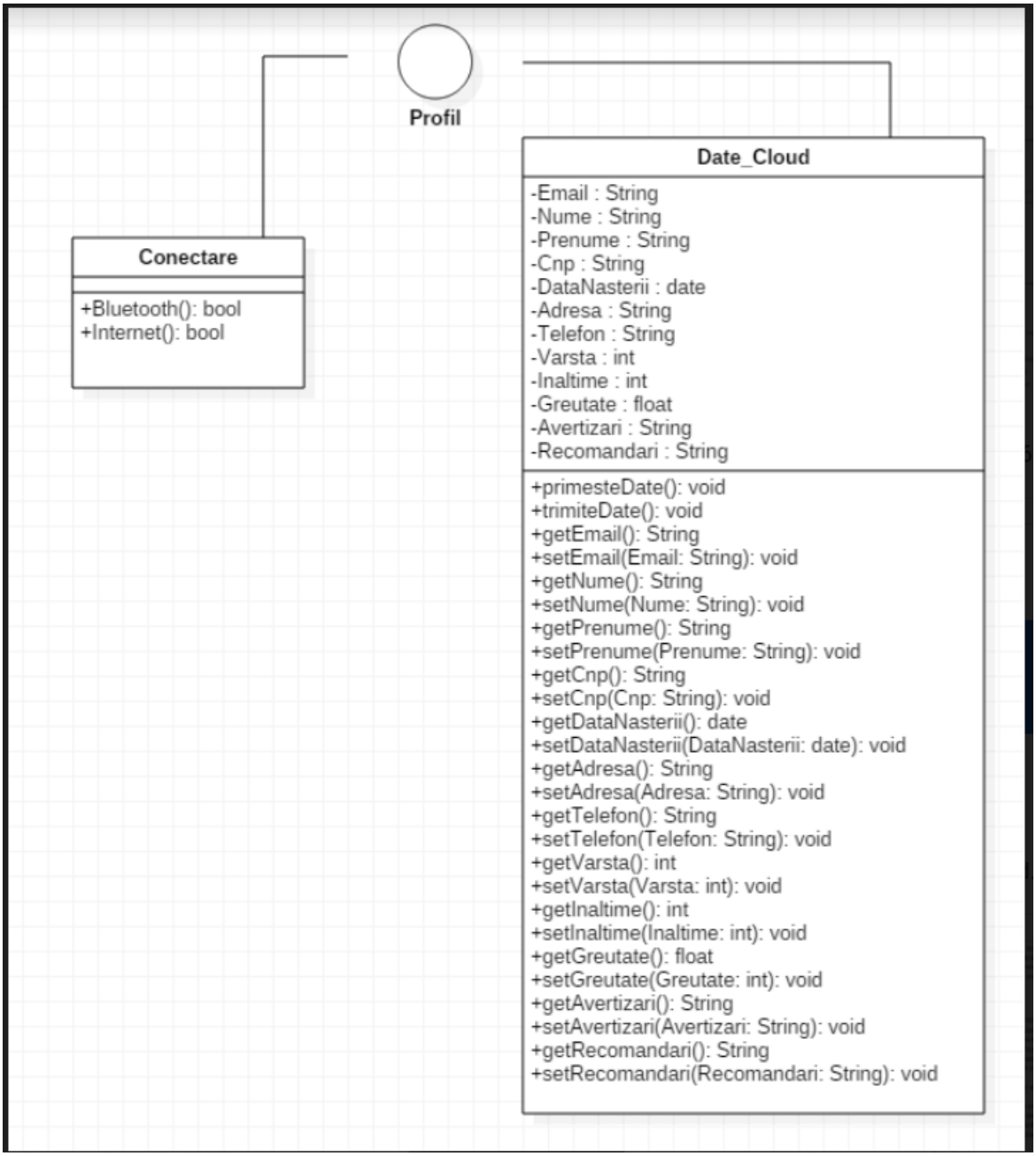
➢ Pentru cont



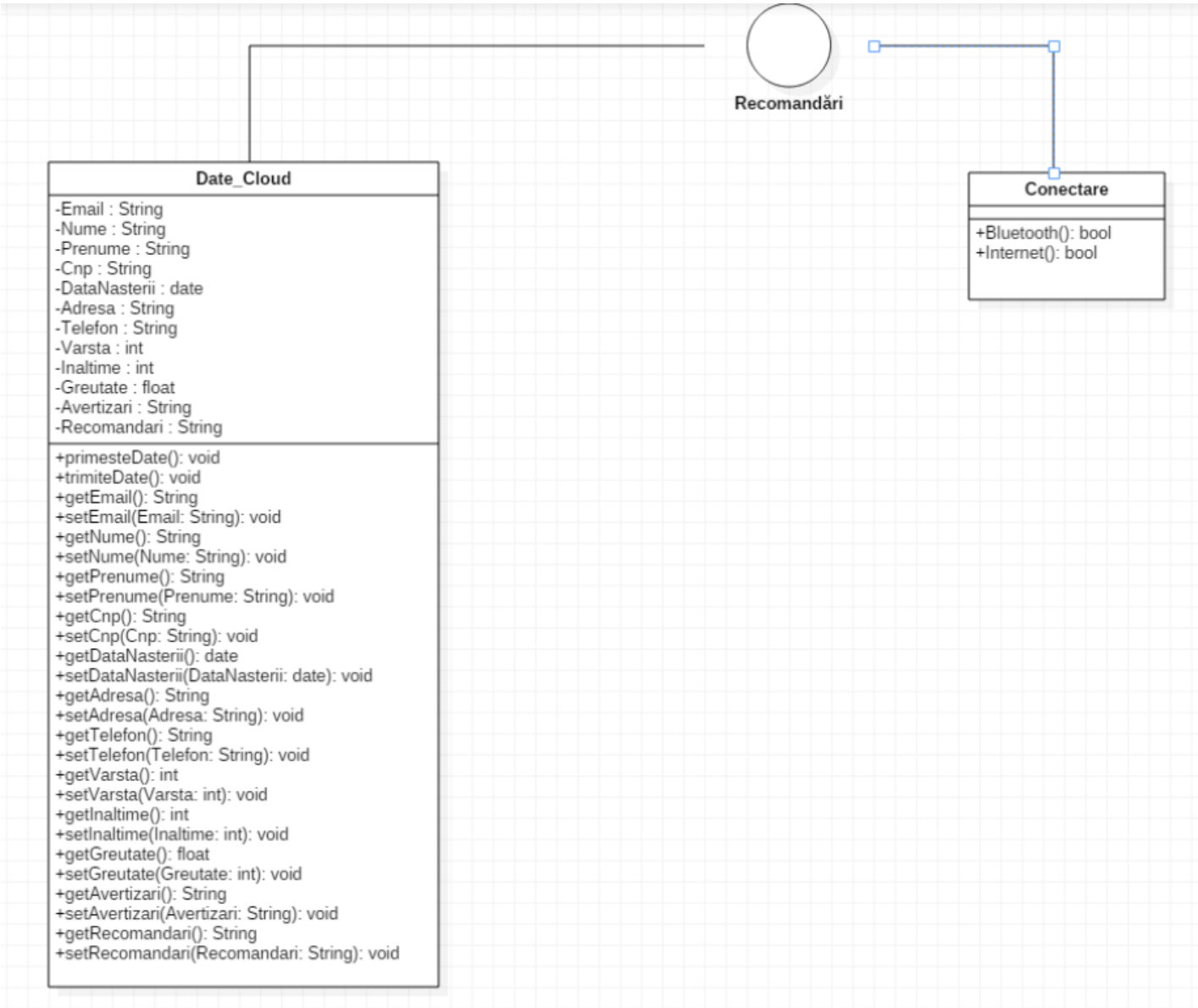
➢ Pentru monitorizare



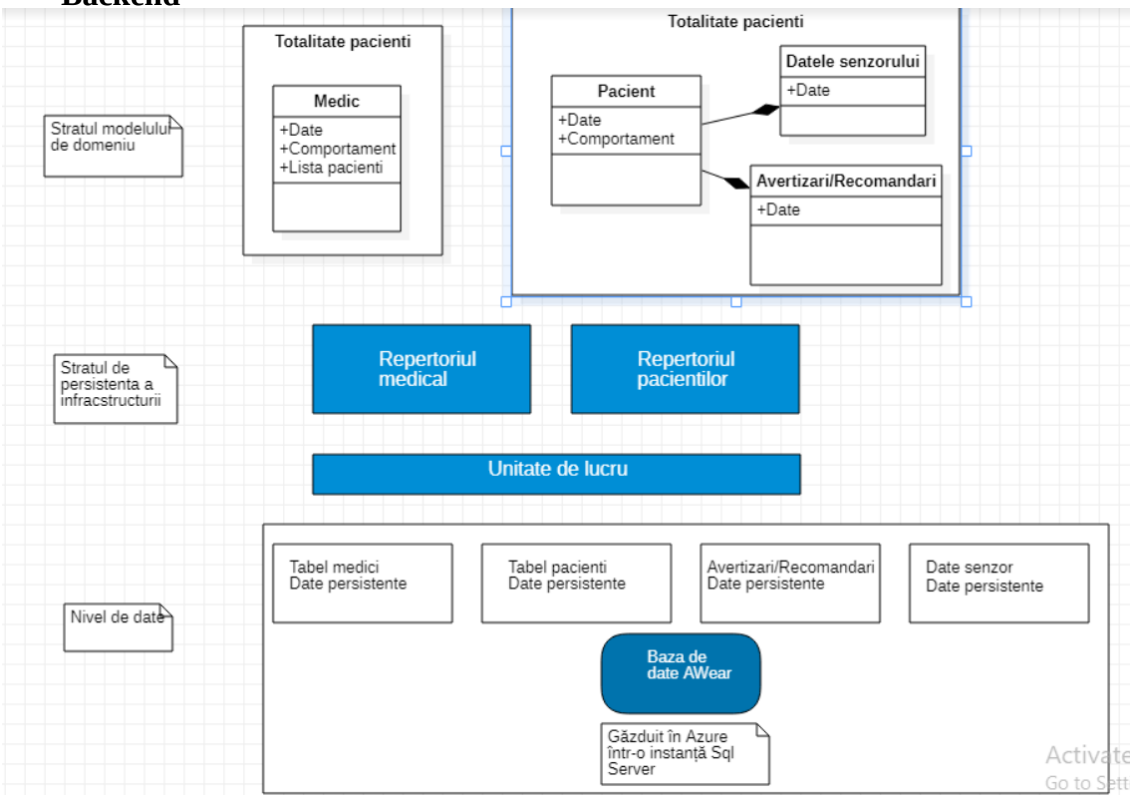
- Pentru profil



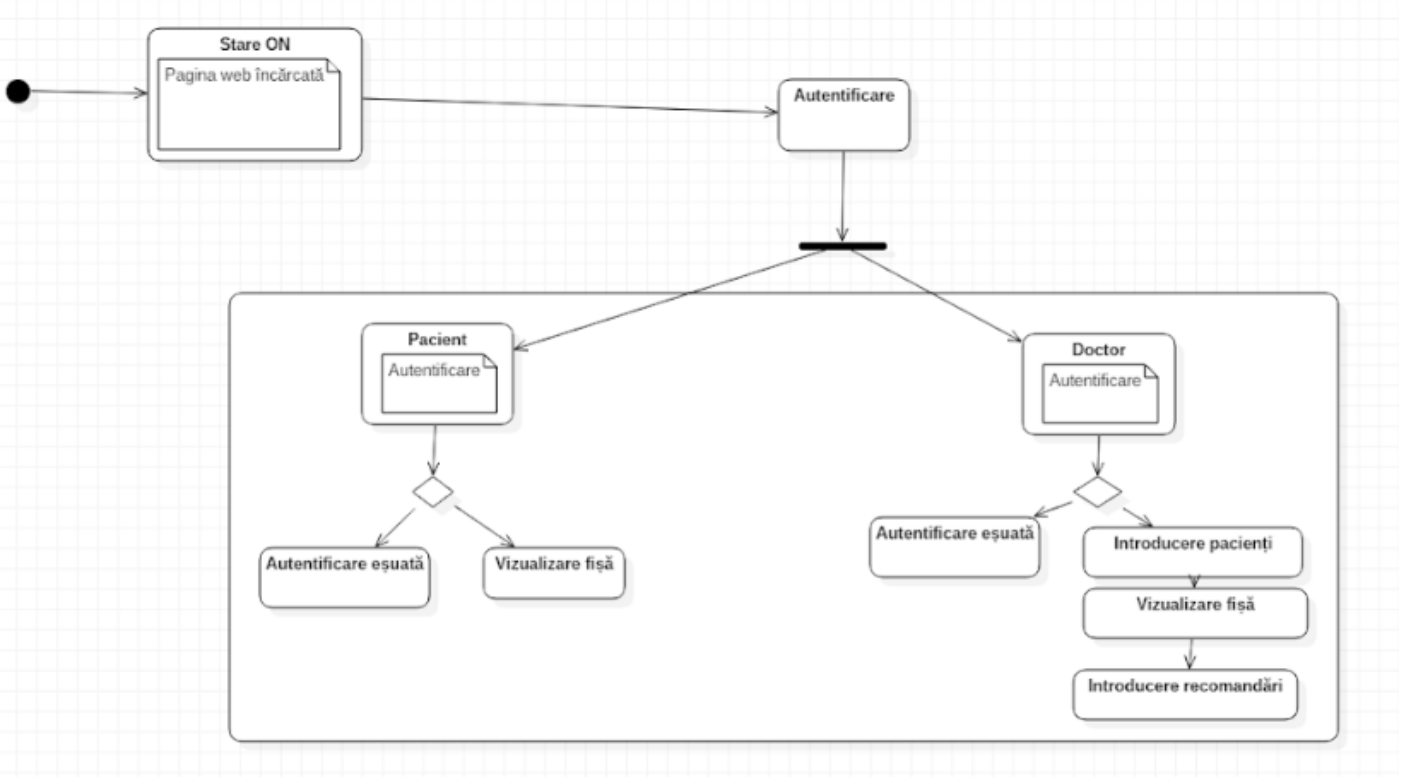
- Pentru recomandari



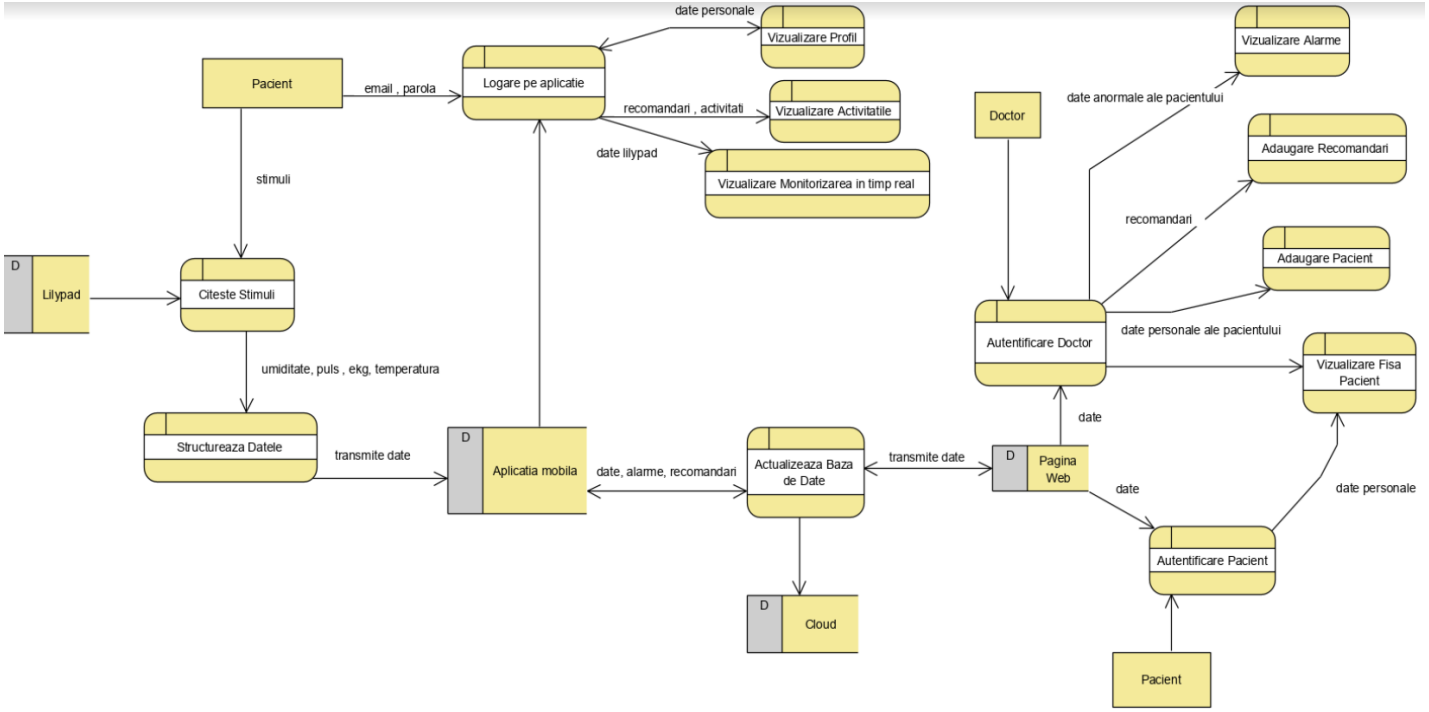
5. Backend



6. Diagrama de stare



7. Diagrama de flux de date



Descrierea componentelor(modulelor)

1. Modulul inteligent

Componenta principală a modulului inteligent este reprezentată de o structură:

sensorData

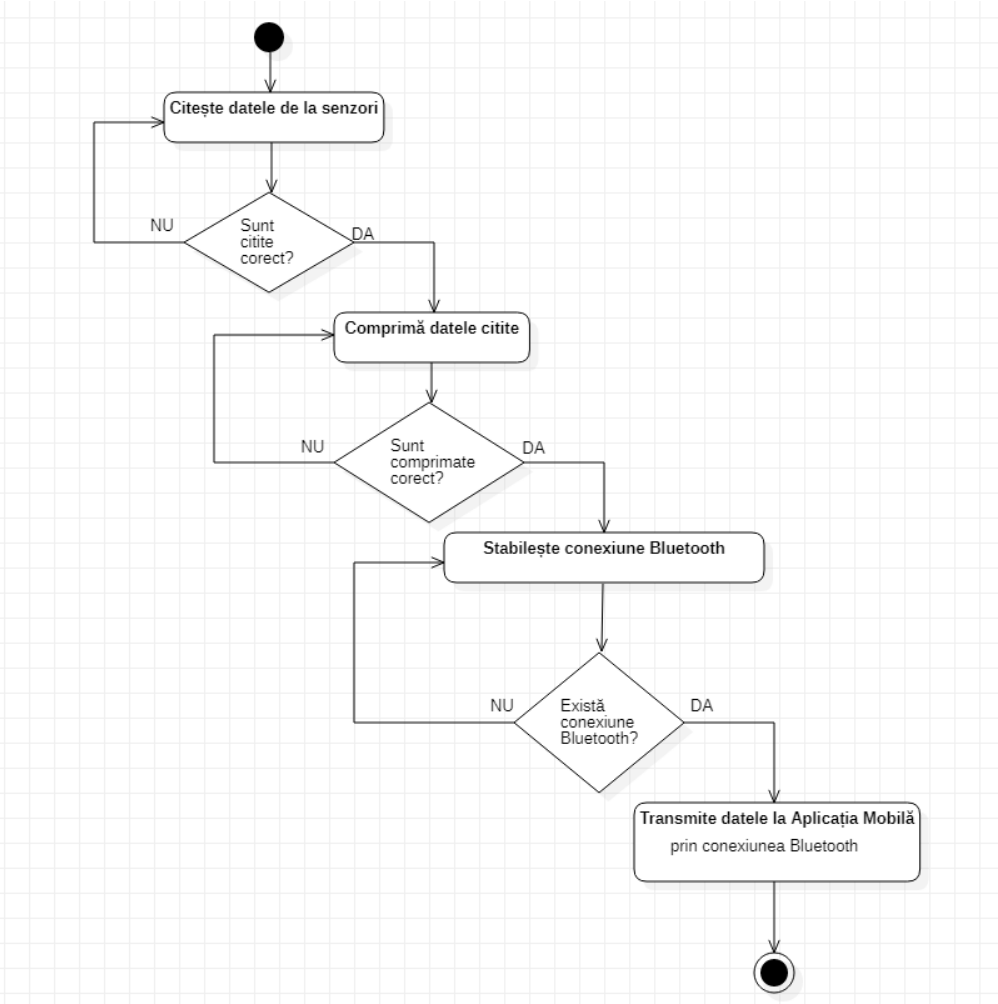
- temperData - float (temperatura exprimată!!! în grade Celsius)
- pulseData - int (valoarea pulsului în bătăi pe minute)
- humidData - int (procentul umidității)
- ECGData – int [2] (valoarea sistolei și a diastolei)

Alte componente:

- **citirea datelor**: readTemper, readPulse, readHumid, readECG (folosite pentru citirea datelor preluate de la senzori);

- **actualizarea datelor din structura principală (sensorData)**: updateStatistics(sensorData)
- **transmiterea datelor**: sendData(sensorData)

Schemă logică:



2. Modulul Aplicație mobilă

Componentele aplicației mobile sunt împărțite pe activități:

- **Autentificare**(prima interfața care apare în specificați): aici verificăm cu baza de date dacă exista parola și numele utilizatorului. Dacă exista intră în meniul aplicației , iar dacă nu se va afișa o eroare.
- **Înregistrare** : aici adaugăm în baza de date: parola , emailul
- **Cont**: pentru a reține toate datele din cont introduse de utilizator și cele primite de la modulul intelligent
- **Monitorizare**: pentru a salva și afișa datele care vin în timp real de la modulul inteligent
- **Profil**:pentru a salva datele de profil personal ale utilizatorului
- **Recomandări**: pentru a salva datele pe care le primește utilizatorul din cloud referitoare la recomandările medicului.

Fiecare dintre acestea au următoarele clase:

- Cont_Login
- Date_Cloud
- Date_Lilypad
- Conectare

3. Modulul Aplicație Cloud

Componentele modului cloud se leagă de căile de acces la date:

- **/Api/Autentificare** - calea unde se verifică datele de autentificare ale unui utilizator
- **/Api/Pacienti/{id_pacient}** - calea de unde se iau informații legate de fișa medicală a pacienților
- **/Api/Pacienti/{id_pacienti}/Diagnostic** - calea de unde se iau informații legate de diagnosticul pacienților
- **/Api/Pacienti/{id_pacienti}/Tratament** - calea de unde se iau informații legate de tratamentul pacienților
- **/Api/Pacienti/{id_pacienti}/DateSenzori** - calea de unde se iau informații legate de datele primite de la senzorii modulului inteligent al pacientului
- **/Api/Medici/{id_medic}/Pacienti** - calea de unde se iau pacienții asociați unui medic
- **/Api/Medici/{id_medic}/Pacienti/{id_pacient}** - calea spre un pacient anume asociat unui medic

4. Modulul Aplicație Web

Componentele modulului se leagă de căile de access la datele din Cloud:

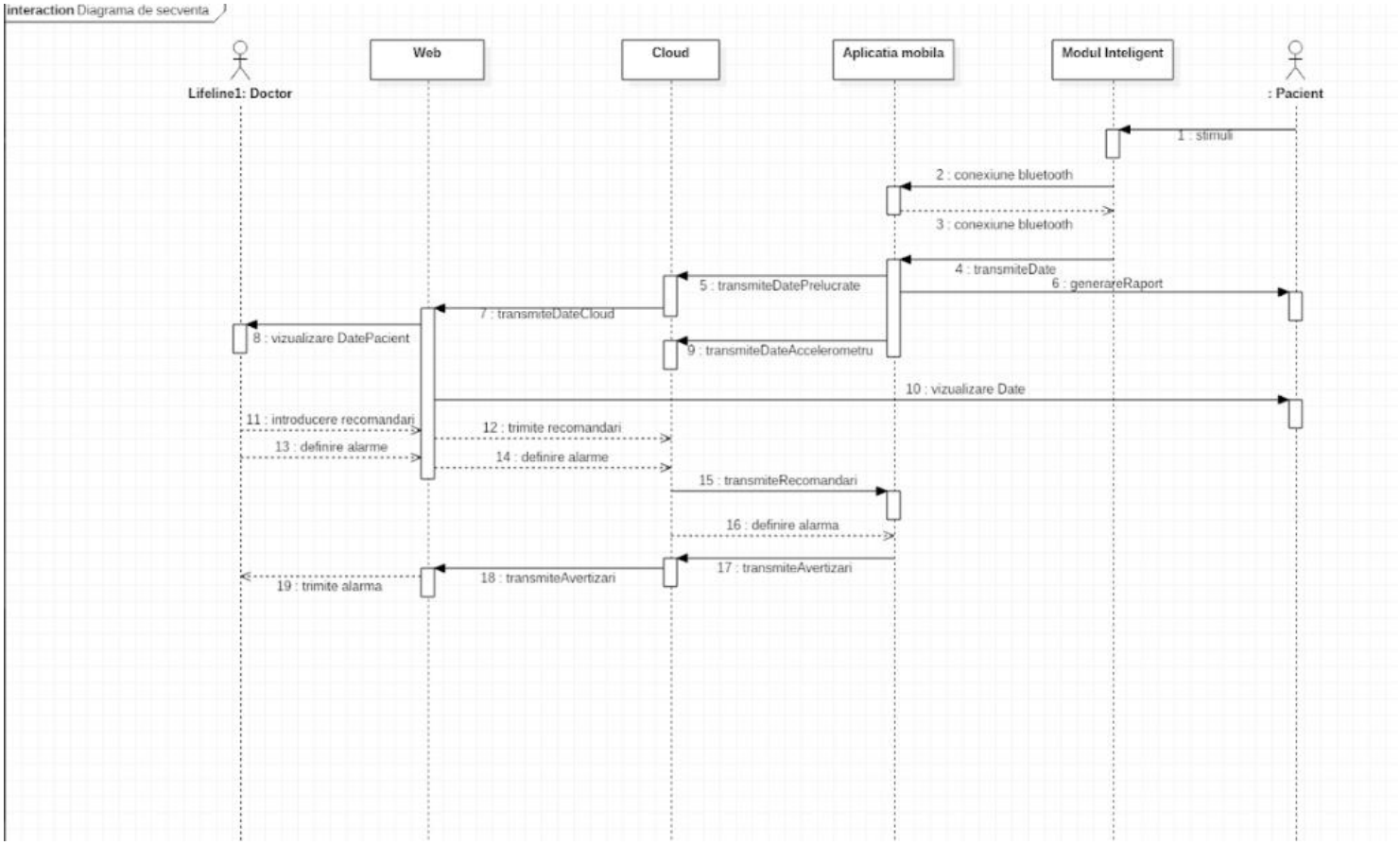
- **fetch('/API/Autentificare')** – fetch-ul folosit pentru datele de logare ale unui pacient sau ale unui medic
- **fetch('/API/Pacienti/id_pacient') –** fetch-ul folosit pentru datele din fișa medicală a pacienților
- **fetch('/API/Pacienti/id_pacient/Diagnostic') –** fetch-ul folosit pentru datele legate de diagnostic
- **fetch('/API/Pacienti/id_pacient/Tratament') –** fetch-ul folosit pentru datele legate de tratamentul pacienților
- **fetch('/API/Pacienti/id_pacient/DateSenzori') –** fetch-ul folosit pentru datele legate de datele primate de la senzorii modulului Inteligent
- **fetch('/API/Medici/id_medic/Pacienti') –** fetch-ul folosit pentru preluarea informațiilor despre pacienți ai unui medic
- **fetch('/API/Medici/id_medic/Pacienti/id_pacient') –** fetch-ul folosit pentru preluarea informațiilor despre un anumit pacient al unui medic

Clasele și variabilele din JavaScript ce stochează informațiile sunt următoarele:

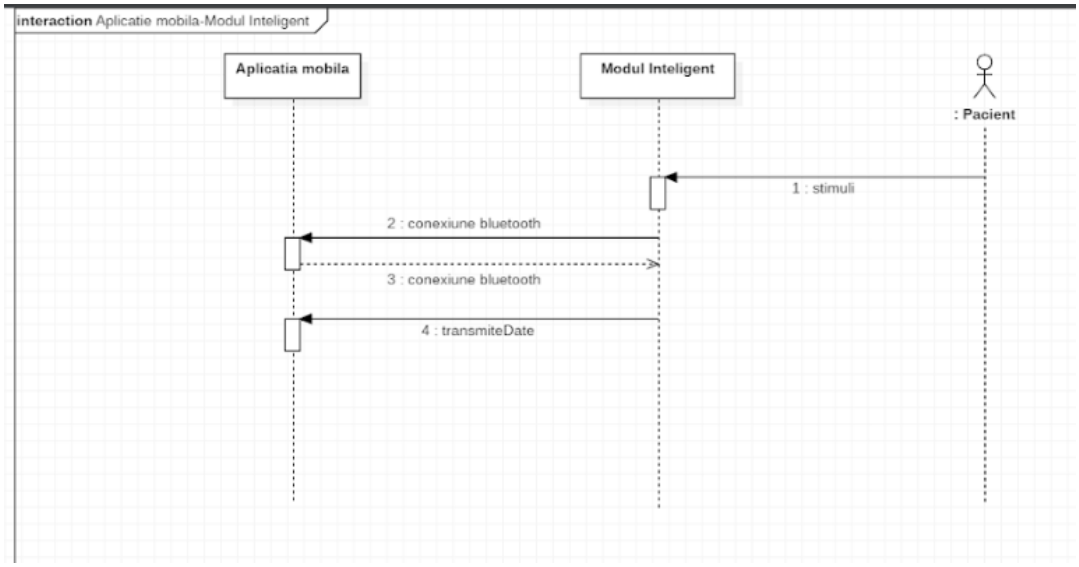
- **log_info**
- **recomandare_info**
- **sensor_info**

Descrierea comunicării între module

Diagrama de secvență



1. Modulul inteligent



Folosind drept dispozitiv un LilyPad, avem nevoie de **biblioteca SoftwareSerial** pentru a stabili o conexiune serială între LilyPad și Android:

#include "SoftwareSerial.h";

Folosind biblioteca SoftwareSerial ne este permis să folosim comunicația serială pe alți pini digitali ai plăcii de dezvoltare Arduino, pe lângă comunicația nativă pe pinii PIN0 și PIN1.

Biblioteca SoftwareSerial va replica această funcționalitatea de UART.

SoftwareSerial bluetooth_connect(bluetoothTx, bluetoothRx);

Modulul Bluetooth folosit are un chipset cu un **Baud Rate** (rată de transfer) de 115.200, iar pentru a-l folosi la o rată de transfer mai scăzută (9.600 baud) vom folosi un cod ce utilizează o conexiune de tip SoftwareSerial între Bluetooth și Arduino, pentru a face configurația sesiunii înainte de a fi utilizată.

Funcții folosite:

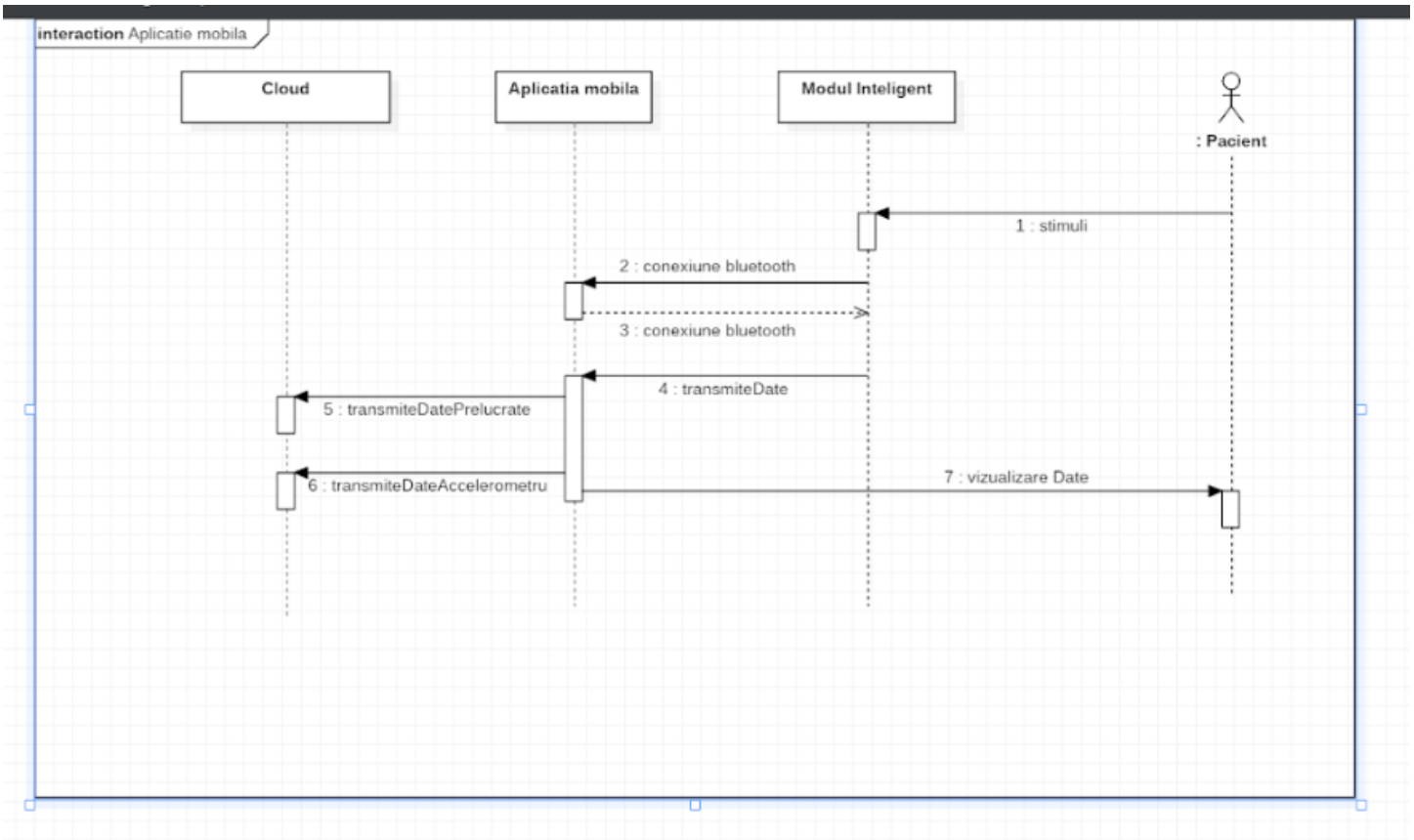
- SoftwareSerial()
- available()
- begin()
- isListening()
- peek()
- read()
- println()
- print()
- listen()
- write()
- overflow()

Conexiuni Hardware:

conector *Bluetooth Mate Silver pentru Arduino clasa 2*

- Bluetooth GND - Arduino GND
- Bluetooth CTS-I – nu se conectează
- Bluetooth VCC - Arduino 5V
- Bluetooth TX-O - Arduino PIN Digital 2
- Bluetooth RX-I - Arduino PIN Digital 3
- Bluetooth RTS-O - nu se conectează

2. Modulul Aplicație mobile



➢ **<u>Bluetooth(pentru comunicarea cu Modulul Inteligent)</u>**

Platforma Android include suport pentru Bluetooth care permite unui dispozitiv să facă schimb de date wireless cu alte dispozitive Bluetooth, în cazul nostru cu Lilypad.

Pachetele importate pentru Android și Bluetooth:

- **import android.app.Activity;** (se folosește pentru activități, o activitate reprezintă un singur ecran cu o interfață de utilizator)
- **import android.bluetooth.BluetoothAdapter;** (se folosește pentru Bluetooth)
- **import android.bluetooth.BluetoothDevice;** (se folosește pentru Bluetooth)
- **import android.content.Intent;** (se folosește pentru **Intent ,** acestea sunt mesaje care leagă componentele împreună)
- **import android.os.Bundle;** (se folosește pentru **Bundle** , acesta este o clasă care permite să stocăm variabile)
- **import android.view.View;** (se folosește pentru **Views,** acestea sunt componente ale interfeței utilizator care sunt desenate pe ecran)

Android oferă API-ul Bluetooth pentru a efectua diferite operațiuni. Din acest pachet de baza sunt:

- **BluetoothAdapter** - reprezintă adaptorul local Bluetooth

```
private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter(); //se creeaza un obiect de tipul BluetoothAdapter și se apelează metoda statică getDefaultAdapter ()
Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(turnOn, 0); // pentru a se activa Bluetooth-ul
_bluetooth.enable(); // verifica dacă este sau nu active
```

- **BluetoothDevice** permite să creem o conexiune cu alte dispozitive și sa extragem informații despre acestea, cum ar fi numele, adresa, clasa și starea de legare

Pentru a primi informații despre dispozitivul găsit, aplicația trebuie să înregistreze un BroadcastReceiver pentru ACTION_FOUND.Următorul fragment de cod arată cum creem un BroadcastReceiver:

```
private final BroadcastReceiver receiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
             if (BluetoothDevice.ACTION_FOUND.equals(action)) {
BluetoothDevice device = intent.getParcelableExtra (BluetoothDevice.EXTRA_DEVICE);
                String deviceName = device.getName();
```

```
                String deviceHardwareAddress = device.getAddress(); // adresa MAC
            }
        }
    };
```

- **BluetoothClass** descrie caracteristicile generale și capabilitățile unui dispozitiv Bluetooth.
- **BluetoothSocket** reprezintă interfața pentru un socket Bluetooth (un socket este un fișier special care ajuta la transmiterea datelor)
- **BluetoothServerSocket** reprezintă un socket de server deschis care ascultă pentru cererile primite (similar cu un TCP ServerSocket). El ajută la conectarea a două dispozitive.

creatingServerSocketConnection();// initializarea la conectare
creatingClientSocketConnection();//initializarea la conectare

```
Server
    final BluetoothServerSocket _serverSocket=
bluetoothAdapter.listenUsingRfcommWithServiRecord
(My_BLUETOOTH_SERVER_NAME,UUID.fromString("a60f35f0-b93a-11de-8a39-
08002009c666"));
Client
    final BluetoothServerSocket _clientSocket =
device.createRfcommSocketToServiceRecord (UUID.fromString("a60f35f0-b93a-
11de-8a39-08002009c666"));
```

Pentru a avea acces la Bluetooth în aplicație folosim 2 permisiuni:

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

- **<u>Internet si HTTP(pentru comunicarea cu Cloud)</u>**

Înainte de a efectua operațiuni în rețea, trebuie mai întâi să verificăm dacă suntem conectați la internet . Pentru acest lucru, Android oferă clasa ConnectivityManager.

ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);

Odată ce inițializăm obiectul clasei ConnectivityManager, putem utiliza metoda getAllNetworkInfo pentru a obține informațiile despre toate rețelele.

NetworkInfo[] info = check.getAllNetworkInfo();

Pentru a trimite cereri trebuie adaugată prima data o permisiune pentru internet:

<uses-permission android:name="android.permission.INTERNET" />

Pachetele necesare activității pentru a trimite "GET request”:

- **import com.android.volley.Request;**
- **import com.android.volley.RequestQueue;**
- **import com.android.volley.Response;**
- **import com.android.volley.VolleyError;**
- **import com.android.volley.toolbox.StringRequest;**
- **import com.android.volley.toolbox.Volley;**

HTTP (Hypertext Transfer Protocol) este un protocol de comunicație responsabil cu transferul de hipertext (text structurat ce conține legături) dintre un client (de regulă, un navigator) și un server web.

Volley este o bibliotecă HTTP care ajuta rețeaua pentru aplicațiile Android să fie mai rapidă.

În Android, comunicația dintre un server web și un client poate fi gestionată prin intermediul clasei **HttpURLConnection.** O aplicație ce utilizează această clasă presupune implementarea următoarelor etape:

1. instanțierea unui obiect URL
2. deschiderea unei conexiuni

HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

3. utilizarea unui flux de intrare / flux de ieșire pentru transferul de informații

URLConnection.getOutputStream ().urlConnection.setDoOutput(true);

OutputStream out=new BufferedOutputStream(urlConnection.getOutputStream();

4. citirea răspunsului din stream-ul returnat de URLConnection.getInputStream ()

InputStream in = new BufferedInputStream(urlConnection.getInputStream());

5. închiderea conexiunii, prin intermediul metodei **disconnect()**.Deconectarea eliberează resursele deținute de o conexiune, astfel încât acestea să poată fi închise sau reutilizate.

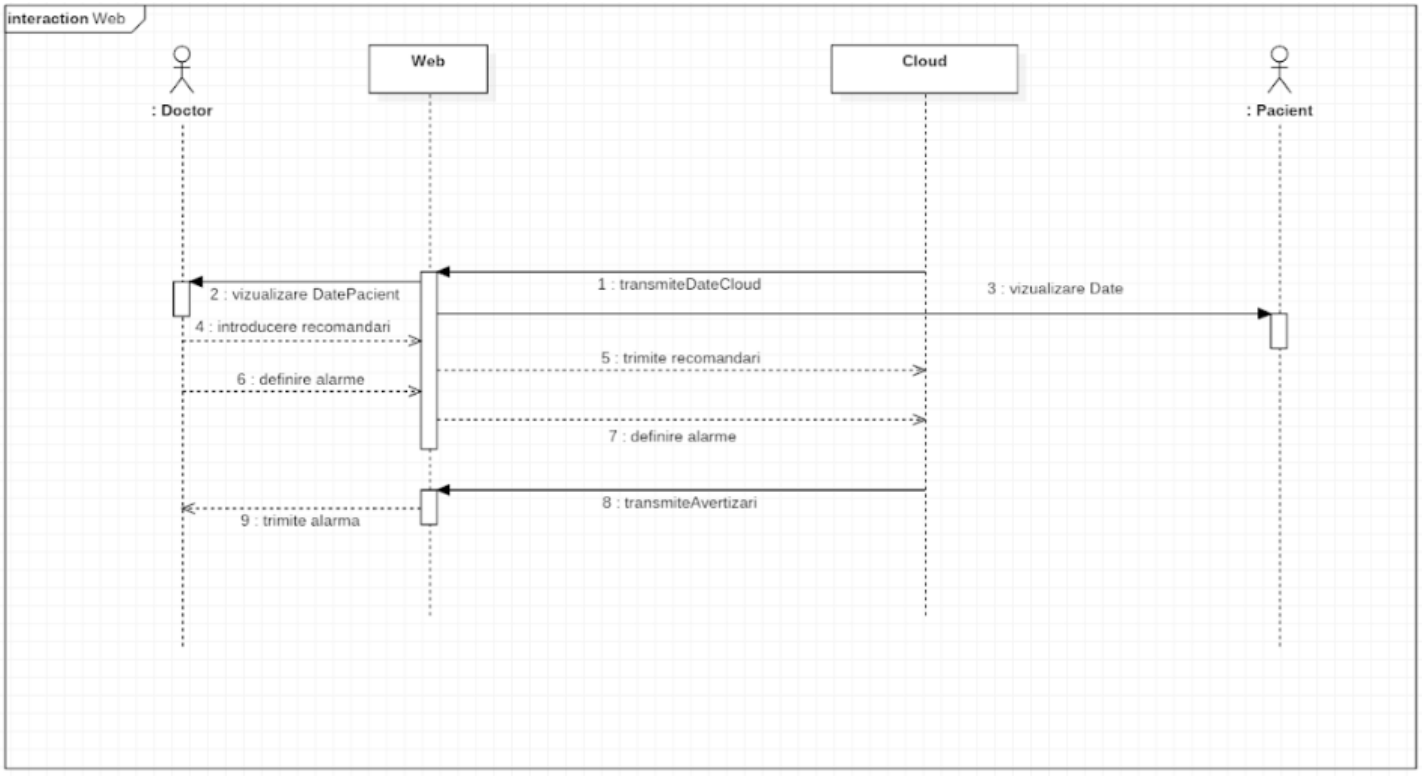
Pentru a transmite un Request se folosesc următoarele funcții:

- Inițierea unui request:

RequestQueue ExampleRequestQueue = Volley.newRequestQueue(this);

- Crearea unui StringRequest:
  StringRequest stringRequest = new StringRequest(Request.Method.GET, url,new Response.Listener<String>()
- Pentru crearea unui nou request se folosește funcția **add ()**
- Pentru anularea unui Request **cancel (),** iar pentru anularea tuturor request-urilor se folosește **onStop ()**
- Se executa funcția **onResponse()** dacă serverul răspunde
- Crearea unui Error Listener pentru posibilele erori **new Response.ErrorListener()**
- Funcția se execută în cazul apariției unei erori **onErrorResponse(VolleyError error)**

3. Modulul Aplicație Web



Pentru protocolul HTTP se folosește un model prin care clientul, reprezentat aici de browser, trimite un mesaj către server, sub formă de **HTTP request,** iar apoi serverul trimite un mesaj-răspuns, **HTTP response.**

Mesajele HTTP sunt alcătuite din headere care au următoarea structură:

nume_header: valoare_header

O cerere HTTP conține pe prima linie, trei elemente:

1. **Metoda HTTP folosită.**
2. **URL-ul cerut**

   URL-ul are rol de nume pentru resursa care a fost solicitată, împreună cu un șir de interogări introdus de client, reprezentat de semnul „?”.

3. **Versiunea HTTP folosită**

Structuri de baze de date si fișiere

Modulul Aplicatie Cloud deține o evidentă a datelor sistemului în interiorul unei baze de date Azure SQL DB.

Am stabilit ca structura acesteia sa aiba urmatoarele tabele:

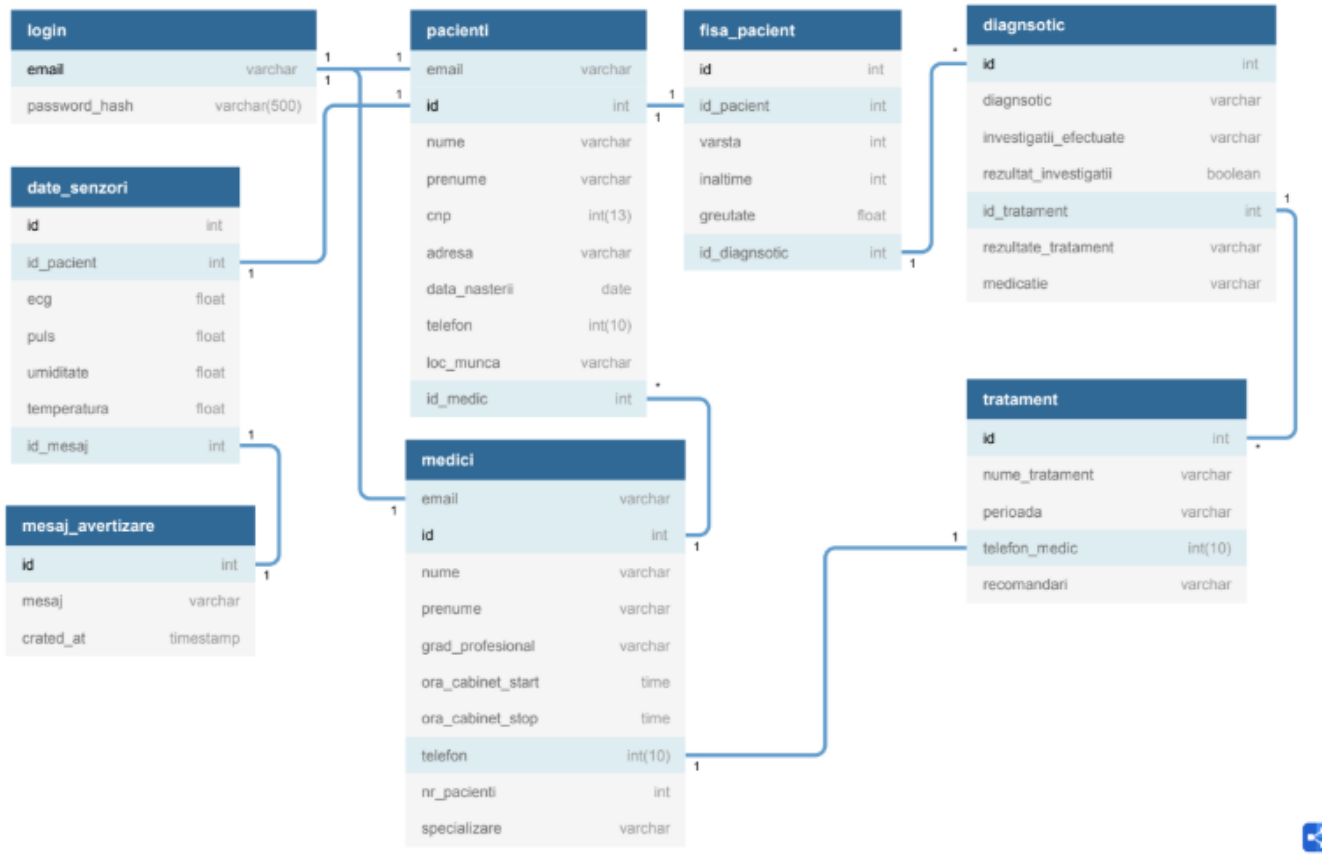


Tabela	Camp	Descriere
Login	email	Varchar(100), PK
	Password_hash	Varchar(500) – parola este tinuta

		ca un hash pe server, nu parola ad-literam
Pacienti	*email*	*Varchar(100), FK*
	id	Int, PK
	nume	Varchar(50)
	prenume	Varchar(50)
	cnp	Int(13)
	adresa	Varchar(200)
	Data_nasterii	date
	telefon	Int(10)
	Loc_munca	Varchar(200)
	Id_medic	Int, FK
Medici	*email*	*Varchar(100), FK*
	id	Int, PK
	nume	Varchar(50)
	prenume	Varchar(50)
	Grad_profesional	Varchar(50)
	Ora_cabinet_start	time
	Ora_cabinet_stop	time
	telefon	Int(13)
	specializare	Varchar(50)
Fisa_pacient	id	Int, PK
	Id_pacient	*Int, FK*
	varsta	Int(3)
	inaltime	Int(3)
	greutate	Float(5,2)
	Id_diagnsotic	Int, FK
Diagnostic	id	Int,PK
	diagnostic	Varchar(300)
	Investigatii_efectuate	Varchar(1000)
	Rezultat_investigatii	Bloolean (daca s-a gasit ceva neinregula cu pacientul sau nu)
	Id_tratament	*Int, FK*
	Rezultate_tratament	Varchar(1000)
	medicatie	Varchar(500)
Tratament	id	Int PK
	Nume_tratament	Varchar(200)
	perioada	Varchar(50) (ex: 2 saptamani/3 luni)
	Telefon_medic	Int(10)

	recomandari	Varchar(100)
Date_senzori	id	Int, PK
	Id_pacient	*Int,FK*
	ecg	float
	puls	Float(5,2)
	umiditate	Float(4,2)
	temperatura	Float(3,1)
	Id_mesaj	*Int, FK*
Mesaj_avertizare	id	Int,PK
	mesaj	Varchar(1000)
	Created_at	Datetime/timestamp