

MEDBUDDY

Candidat: Cotulbea Adrian-Ionuț

Coordonator științific principal: Prof. dr. habil. Ing. Marius Marcu

Coordonator științific secundar: ing. Claudiu Groza

Sesiunea: Iunie/Iulie 2023

REZUMAT

Aplicația mobilă MedBuddy reprezintă o platformă creată cu scopul de a simplifica modul prin care fiecare dintre noi poate obține o părere calificată oferită de unul dintre medicii specialiști, care s-au înscris și ei, la rândul lor, în această aplicație.

Majoritatea dintre noi ajungem să neglijăm, de cele mai multe ori, sănătatea personală până când aceasta se agravează din diferite motive din viața de zi cu zi, cum ar fi lipsa timpului, orarul haotic de la muncă, traficul din oraș sau chiar și mai rău, comoditatea.

MedBuddy vine cu o soluție rapidă și convenabilă, deoarece cu ajutorul acestei aplicații poți obține un diagnostic și un plan de tratament pentru simptomele pe care le manifesti la un moment dat, indiferent de unde te afli, atât timp cât telefonul tău are conexiune la Internet.

În cazul în care consideri că starea ta de sănătate necesită un consult față în față cu un medic specialist, aplicația îți oferă posibilitatea de a cere acest lucru comunității de medici și să primești înapoi ca răspuns data, locul și numele medicului specialist care te va consulta, în momentul în care cererea ta a fost acceptată de medicul respectiv.

Proiectul este alcătuit din aplicația mobilă dezvoltată pentru platforma Android, care servește rolul de client, și dintr-un server dezvoltat în C++, care este conectat la o bază de date relațională.

1. CUPRINS

1.	CUPRINS.....	3
2.	INTRODUCERE.....	4
2.1	CONTEXT.....	4
2.2	DESCRIEREA PROIECTULUI.....	4
3.	ANALIZA DOMENIULUI.....	5
3.1	APLICAȚII ASEMĂNĂTOARE.....	5
4.	TEHNOLOGII FOLOSITE.....	7
4.1	LIMBAJE DE PROGRAMARE.....	7
4.1.1	KOTLIN.....	7
4.1.2	C++.....	7
4.2	MEDII DE DEZVOLTARE.....	8
4.2.1	ANDROID STUDIO.....	8
4.2.2	VISUAL STUDIO CODE.....	8
4.3	TEHNOLOGIE DE CONTAINERIZARE - DOCKER.....	9
4.4	GESTIUNEA BAZEI DE DATE - MYSQL.....	9
4.5	VERSIONAREA CODULUI - GIT.....	9
4.6	BIBLIOTECI FOLOSITE.....	10
4.6.1	RETROFIT.....	10
4.6.2	ASIO.....	10
4.6.3	MySQL CONNECTOR/C++.....	10
5.	PROIECTAREA APLICAȚIEI.....	11
5.1	DIAGrame UML A CAZURILOR DE UTILIZARE.....	12
6.	IMPLEMENTARE ȘI UTILIZARE.....	14
6.1	SERVER.....	14
6.2	APLICAȚIA MOBILĂ.....	24
6.2.1	UTILIZAREA DIN PERSPECTIVA PACIENTULUI.....	30
6.2.2	UTILIZAREA DIN PERSPECTIVA MEDICULUI.....	35
7.	CONCLUZIE ȘI DIRECȚII VIITOARE DE DEZVOLTARE.....	41
7.1	CONCLUZIE.....	41
7.2	DIRECȚII VIITOARE DE DEZVOLTARE.....	41
8.	LISTA FIGURILOR.....	42
9.	LISTA SECȚIUNILOR DE COD SURSA.....	43
10.	BIBLIOGRAFIE.....	44

2. INTRODUCERE

2.1 CONTEXT

MedBuddy reprezintă o platformă ce oferă două perspective, pentru rolul de medic și pentru rolul de pacient, care își propune să îmbunătățească comunicarea dintre aceștia.

Scopul principal este ca potențialii pacienți să economisească timp, scutind etapa de stat la cozile interminabile din fața cabinetului medicului specialist, printr-o simplă cerere de ajutor medical virtual, prin care acesta poate fi diagnosticat pe baza simptomelor expuse în cerere, iar acestuia îi este stabilită o medicație.

Odată începută perioada de tratament, medicul și pacientul dispun de o interacțiune bazată pe comunicarea lor, fiind expusă printr-un dialog între cei doi.

Utilizatorii au la dispoziție și un istoric al interacțiunilor cu rolul opus aferent, acesta fiind reprezentat fie de un istoric de pacienți, fie de un istoric de tratamente.

2.2 DESCRIEREA PROIECTULUI

Structura proiectului este alcătuită dintr-o aplicație mobilă dezvoltată în Kotlin, în mediul de dezvoltare Android Studio, care comunică direct, cu ajutorul bibliotecii Retrofit, cu un server HTTP dezvoltat în C++ cu ajutorul bibliotecii ASIO.

Serverul se conectează la rândul lui la o bază de date relațională MySQL, folosind biblioteca MySQL Connector/C++.

Aplicația dispune de date apelând API-ul implementat specific pentru acest proiect, manipulându-le în așa fel încât să se realizeze interacțiunile dintre medici și pacienți.

Am creat o imagine personalizată folosind platforma Docker pentru a include serverul C++. Deoarece serverul este dependent de baza de date MySQL, am pornit ambele servicii în mod sincron utilizând un fișier docker-compose. Am ales această abordare pentru a putea scala serviciile, permițându-ne să creăm servere de backup și să replicăm baza de date pentru a obține un sistem mai persistent.

3.ANALIZA DOMENIULUI

3.1 APLICAȚII ASEMĂNĂTOARE:

Trei dintre aplicațiile similare existente sunt "Practo", "Doctor on Demand" și "Talkspace".

"Practo" este o aplicație populară de sănătate disponibilă atât pe platforma iOS, cât și pe cea Android. Ea permite pacienților să caute medici și să își facă programări, să vizualizeze înregistrările medicale și să comande medicamente online. Una dintre caracteristicile cheie ale Practo este posibilitatea ca pacienții să caute și să își facă programări cu medici, precum și să își vizualizeze înregistrările medicale. Aplicația oferă, de asemenea, pacienților posibilitatea de a comanda medicamente online, ceea ce este o facilitare convenabilă.

"Doctor on Demand" este o aplicație de telemedicină disponibilă atât pe platforma iOS, cât și pe cea Android. Permite pacienților să aibă consultații virtuale cu medici, să vizualizeze înregistrările medicale și să primească tratament la distanță.

"Talkspace" este o aplicație de sănătate mentală care oferă pacienților acces la terapie virtuală cu terapeuți licențiați. Utilizatorii pot accesa aplicația pentru a programa și gestiona sesiunile de terapie, primi notificări și urmări progresul lor.

Un potențial avantaj al aplicației MedBuddy este includerea funcțiilor de chat și de feedback între pacienți și medici, ceea ce ar putea permite o comunicare mai eficientă și o mai bună gestionare a nevoilor de sănătate.

MedBuddy își propune să ofere un punct unic de gestionare a nevoilor de sănătate pentru pacienți și medici, și prin valorificarea avantajelor oferite de celelalte aplicații existente, să creeze o experiență completă pentru utilizatori.

Caracteristici	<u>Practo</u>	<u>Doctor On Demand</u>	<u>Talkspace</u>	MedBuddy
<i>Adresa site-ului web</i>	Practo.com	Doctorondemand.com	Talkspace.com	-
<i>Link magazin</i>	Practo	Doctor On Demand	Talkspace	-
<i>Rating Magazin</i>	4.4 / 5	4 / 5	2 / 5	-
<i>Număr de instalări</i>	10M+	1M+	500k+	-
<i>Număr de instalări</i>	300k	40k	6k	-
<i>Anunțuri/achiziții în aplicație</i>	x	x	x	-
Login/user	x	x	x	x
Chat	-	-	-	x
<i>Acces la înregistrările medicale</i>	x	x	x	x
<i>Funcții de feedback</i>	-	x	-	x
<i>Editare profil</i>	-	-	-	x
<i>Programare de consultații</i>	x	x	-	x
<i>Gestionarea medicamentelor</i>	x	-	x	x
<i>Notificări</i>	-	x	x	x
<i>Cerere virtuală</i>	-	-	x	x

4. TEHNOLOGII FOLOSITE

4.1 Limbaje de programare

4.1.1 Kotlin

MedBuddy fiind o aplicație mobilă pentru platforma Android era necesar un limbaj de programare cât mai avantajos pentru dezvoltarea acesteia, așa ca am ales Kotlin.

Kotlin este un limbaj de programare modern, concis și static, care rulează pe JVM (Java Virtual Machine) și este complet interoperabil cu Java. A fost dezvoltat de JetBrains și a câștigat popularitate rapidă în comunitatea dezvoltatorilor datorită sintaxei clare și expresive, siguranței la tip și caracteristicilor moderne. Kotlin oferă o sintaxă concisă și expresivă, reducând cantitatea de cod necesar pentru a realiza aceeași funcționalitate comparativ cu Java. De asemenea, aduce îmbunătățiri în siguranța la tip, eliminând erorile de tip la timpul de execuție și gestionând mai eficient valorile null prin conceptul de tipuri nullable și non-nullable. Un alt avantaj major al lui Kotlin este interoperabilitatea completă cu Java, permițând integrarea ușoară a codului Kotlin în proiectele Java existente. De asemenea, oferă suport pentru funcții moderne, precum funcții de ordin superior și lambda expressions, facilitând dezvoltarea de cod clar și concis[1][2].

4.1.2 C++

Partea de server a fost realizată în C++, deoarece acest limbaj de programare este renumit ca fiind eficient din punct de vedere al costului de timp și al resurselor, folosit deseori în dezvoltarea de servere.

C++ este un limbaj puternic de programare de nivel scăzut și oferă acces la operațiuni de același nivel, cum ar fi manipularea directă a memoriei și controlul precis al resurselor. De asemenea, oferă suport pentru programarea orientată pe obiecte, permițând dezvoltatorilor să structureze și să organizeze codul în clase și obiecte. Limbajul C++ este cunoscut pentru performanța sa ridicată și este adesea utilizat în dezvoltarea de aplicații care necesită eficiență maximă și control granular[3][4].

4.2 Medii de dezvoltare

4.2.1 Android Studio

Mediul de dezvoltare ales pentru această aplicație este Android studio, deoarece acesta ofera foarte multe utilități încorporate în interfața lui, cum ar fi dezvoltarea fișierelor XML care reprezintă scenele și design-ul aplicației sau emulatoarele de dispozitive mobile virtuale, unde ai de ales dintr-o varietate de modele de telefoane sau tablete, pe care poți rula și testa aplicația.

Android Studio este un mediu de dezvoltare integrat (IDE) creat special pentru dezvoltarea aplicațiilor Android. Dezvoltat de Google, Android Studio oferă un set complet de instrumente și funcționalități necesare pentru a crea, testa și distribui aplicații Android de înaltă calitate. IDE-ul dispune de o interfață prietenoasă și ușor de utilizat, precum și de funcționalități avansate, cum ar fi un editor de cod, depanator și emulator Android integrat. Unul dintre avantajele sale majore este suportul nativ pentru limbajul Kotlin, permițând dezvoltatorilor să utilizeze acest limbaj pentru dezvoltarea aplicațiilor Android într-un mod mai concis și mai sigur. Android Studio integrează, de asemenea, serviciile Google Play, facilitând implementarea funcționalităților precum autentificarea cu contul Google și integrarea cu serviciile Google[5].

4.2.2 Visual Studio Code

Serverul a fost dezvoltat în editorul Visual Studio Code, deoarece am avut la dispoziție diferite unelte care au ajutat la livrarea unui progres considerabil și corect, din punct de vedere funcțional.

Visual Studio Code (VS Code) este un editor de cod cu sursă publică, dezvoltat de Microsoft. Este unul dintre cele mai populare și versatile instrumente de dezvoltare disponibile, potrivit pentru dezvoltarea aplicațiilor într-o gamă largă de limbaje de programare. VS Code se remarcă prin simplitatea și flexibilitatea sa, oferind o interfață intuitivă și o gamă largă de extensii și plugin-uri care pot fi personalizate pentru a se potrivi nevoilor fiecărui dezvoltator[6].

4.3. Tehnologie de containerizare - Docker

Am ales să folosesc Docker pentru a defini un container (un Dockerfile) pentru serverul C++, deoarece acesta este inclus alături de o imagine docker mysql server într-un fișier yml (docker-compose), cu ajutorul căruia pot să pornesc toată funcționalitatea serverului cu o singură comandă.

Docker este o platformă de virtualizare la nivel de sistem de operare, care permite împachetarea și distribuirea aplicațiilor în containere portabile și ușoare. Un container Docker conține toate componentele necesare pentru rularea unei aplicații, inclusiv codul sursă, bibliotecile și dependențele, într-un mod izolat de sistemul gazdă. Aceasta facilitează reproducerea și distribuirea consistentă a aplicațiilor pe diferite medii, indiferent de configurația sistemului gazdă[7].

4.4. Gestiunea bazei de date - MySQL

Server-ul C++ se conectează la o bază de date MySQL, unde sunt definite mai multe tabele care servesc nevoile aplicației client MedBuddy. Acest server MySQL este configurat și pornit cu ajutorul lui Docker și o imagine de MySQL, cum am menționat și mai sus.

MySQL este un sistem de gestiune a bazelor de date relaționale, open-source și popular, care oferă o platformă robustă pentru stocarea și gestionarea datelor. Acesta utilizează modelul relațional, organizând datele în tabele cu rânduri și coloane. MySQL oferă performanță ridicată, scalabilitate și funcționalități avansate de securitate. Este susținut de o comunitate puternică și oferă acces la resurse și suport pentru dezvoltatori[8].

4.5. Versionarea codului – GIT

Codul aplicației și al serverului au fost versionate cu ajutorul platformei GitHub, unde am făcut urmărirea problemelor și am folosit ca mod de dezvoltare GitHub Flow, unde fiecare issue se rezolvă/implementează pe un branch separat față de branch-ul main.

GitHub este o platformă de gestionare a dezvoltării software bazată pe sistemul de control al versiunilor Git. Aceasta permite dezvoltatorilor să colaboreze la proiecte, să urmărească modificările de cod și să gestioneze fluxul de lucru al dezvoltării software[9].

4.6. Biblioteci folosite

4.6.1 Retrofit

Retrofit a fost folosit pentru a consuma API-ul construit pentru a face schimbul de date dintre server și aplicație, apelând astfel request-urile HTTP.

Retrofit este o bibliotecă populară pentru dezvoltarea de aplicații Android în limbajul de programare Kotlin. Aceasta oferă o modalitate simplă și eficientă de a realiza cereri HTTP și de a consuma API-uri web în aplicații[10].

4.6.2 Asio

Biblioteca Asio de sine stătătoare(non-boost) a fost folosită pentru a rezolva manevrarea cererilor pe partea de server, astfel implementând logica server-ului.

Asio Standalone este o bibliotecă C++ cu sursă publică, care oferă un suport eficient pentru programarea asincronă și gestionarea evenimentelor de rețea. Aceasta este proiectată pentru a fi ușor de utilizat și scalabilă, oferind performanțe ridicate și o abordare modernă în dezvoltarea aplicațiilor de rețea[11].

4.6.3 MySQL Connector/C++

Biblioteca MySQL Connector/C++ a fost folosită în implementarea server-ului. Cu ajutorul ei am conectat server-ul la baza de date SQL și am efectuat operațiile necesare pentru a insera, actualiza sau a prelua date.

MySQL Connector/C++ este o bibliotecă C++ care oferă suport pentru conectarea și interacțiunea cu bazele de date MySQL. Acesta facilitează dezvoltarea aplicațiilor C++ care utilizează MySQL ca sistem de gestiune a bazelor de date[12][13].

5. PROIECTAREA APLICAȚIEI

Datele persistente stocate în baza de date SQL ajung la aplicația mobilă astfel:



Figura 1. Diagrama UML Relație Server-Client

Aplicația client MedBuddy este împărțită în două perspective diferite, astfel utilizatorii având posibilitatea de a alege, în momentul înregistrării, între două roluri, medic sau pacient.

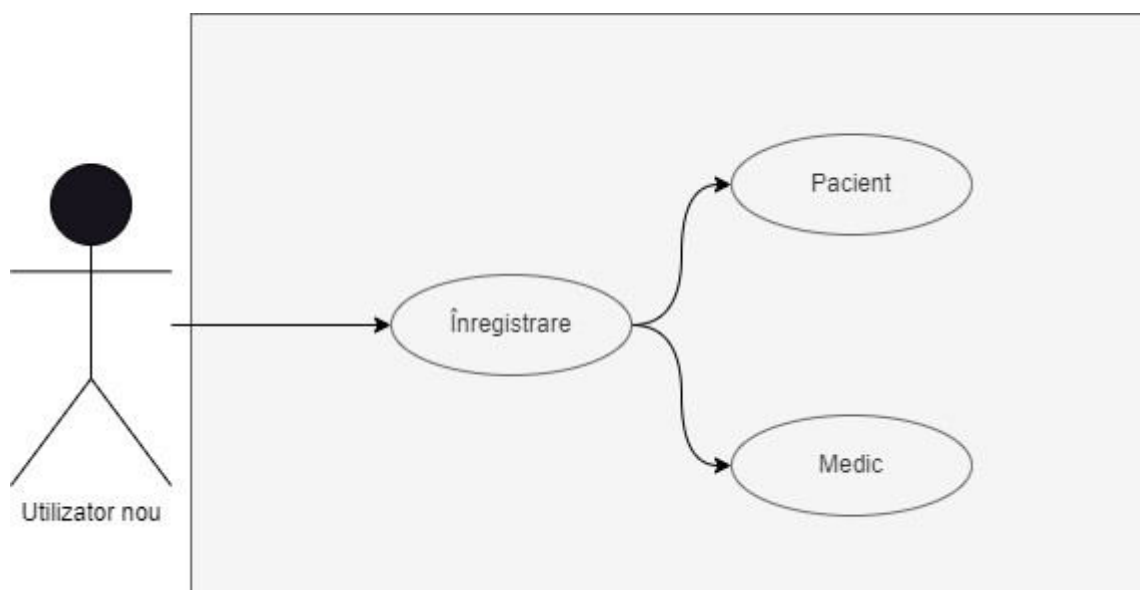


Figura 2. Diagrama UML Roluri

În funcție de rolul utilizatorului, scenariile și funcționalitățile sunt distincte.

5.1 DIAGrame UML A CAZURILOR DE UTILIZARE

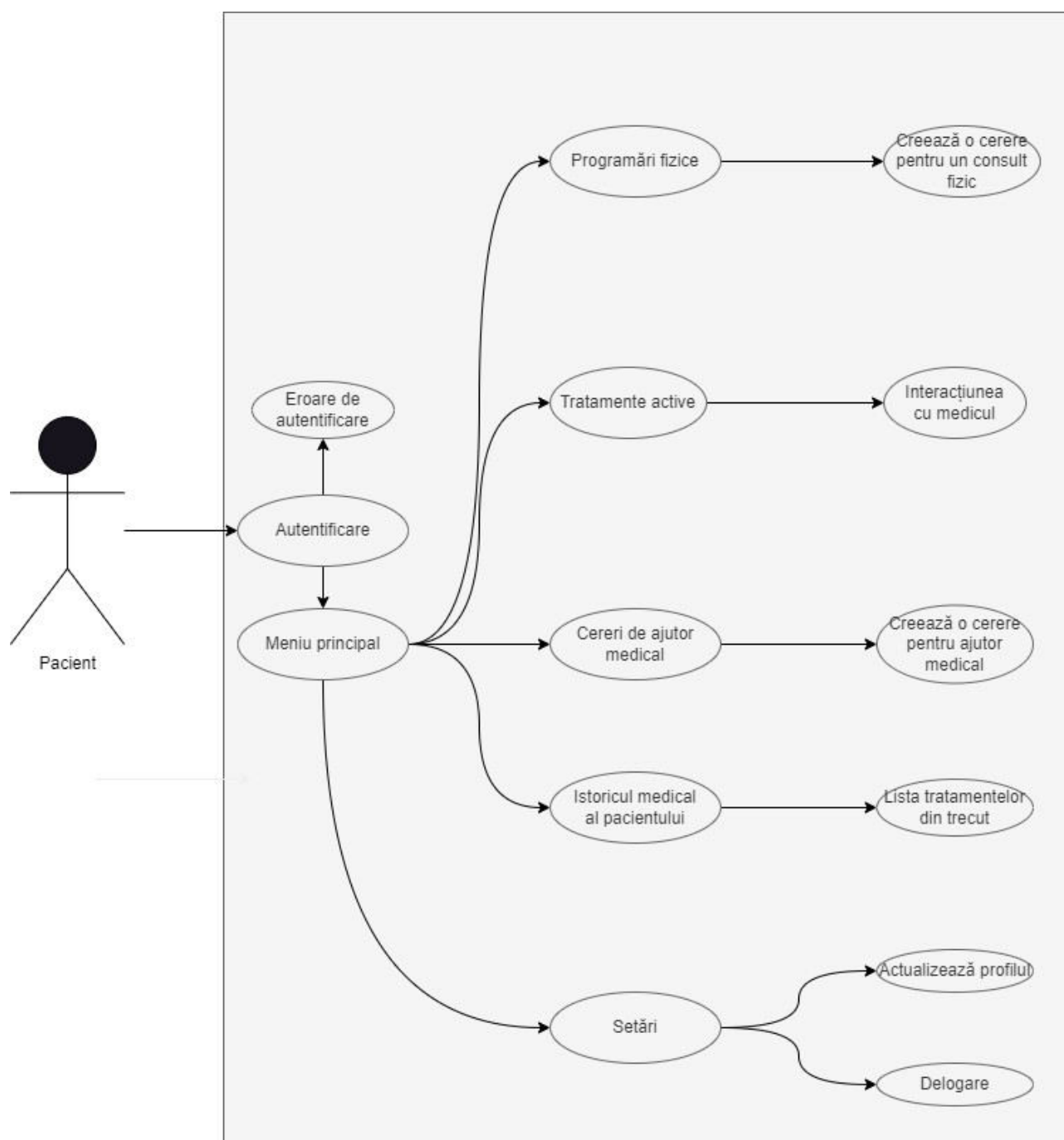


Figura 3. Diagrama UML a cazului de utilizare al pacientului

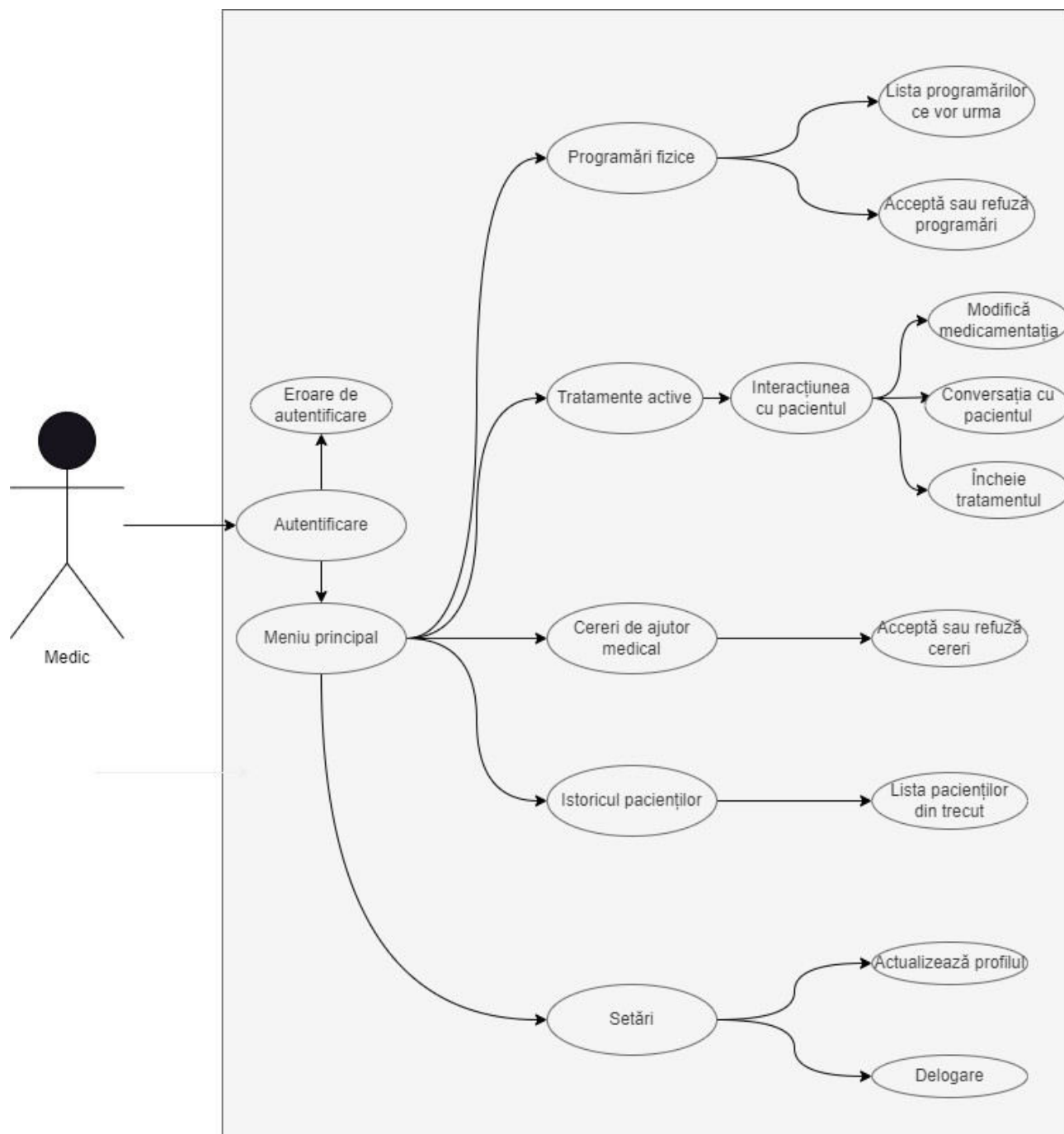


Figura 4. Diagrama UML a cazului de utilizare al medicului

6. IMPLEMENTARE ȘI UTILIZARE

6.1 Server

Partea de backend a acestei lucrări constă într-un server dezvoltat în C++ configurat într-un Dockerfile pentru a crea o imagine de docker personalizată.

```
FROM gcc:latest

COPY ../server.cpp /app/

WORKDIR /app

RUN apt-get update && apt-get install -y libasio-dev libmysqlcppconn-dev

RUN g++ -o server server.cpp -lmysqlcppconn

EXPOSE 8080

CMD ["/server"]
```

Cod sursă 1. Dockerfile pentru fișierul C++

Voi descrie în linii mari fișierul de configurare:

- 'FROM gcc:latest' - aceasta linie ne indică faptul că imaginea de bază pentru construirea imaginii personalizate este 'gcc', iar 'latest' indică ultima versiune pentru acea imagine disponibilă pe Dockerhub.
- 'COPY ../server.cpp /app/' - indică instrucțiunea prin care se copiază fișierul 'server.cpp' în directorul /app/ din interiorul imaginii de Docker.
- 'WORKDIR /app' – stabilește directorul de lucru curent în interiorul imaginii, prin urmare toate instrucțiunile ulterioare se vor executa din această locație.
- 'RUN apt-get update && apt-get install -y libasio-dev libmysqlcppconn-dev' - cu ajutorul comenzii 'RUN' executăm două instrucțiuni consecutive prin care se actualizează lista de pachete și se instalează 'libasio-dev' și 'libmysqlcppconn-dev', pachete necesare pentru bibliotecile folosite în fișierul C++.
- 'RUN g++ -o server server.cpp -lmysqlcppconn' – compilează fișierul c++, specificând numele cu ajutorul parametrului '-o', iar apoi specificăm bibliotecile ce trebuie link-uite pentru a putea rula.
- 'EXPOSE 8080' - expune portul '8080' pentru a permite altor servicii din interiorul sau din exteriorul container-ului să se conecteze la server.
- 'CMD ["/server"]' - pornește executabilul 'server'.

Aceasta imagine personalizată este accesată în sincron cu o imagine mysql în următorul fișier:

```
version: '3'

services:
  # Serviciul pentru server
  medbuddy-sv:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8080:8080
    depends_on:
      - medbuddy-db
    restart: always # Repornește serviciul dacă acesta se oprește neașteptat

  # Serviciul pentru baza de date
  medbuddy-db:
    image: mysql:latest
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: medbuddy
      MYSQL_USER: admin
      MYSQL_PASSWORD: admin
      MYSQL_USER_HOST: "%"
    volumes:
      - mysql-data:/var/lib/mysql
    restart: always # Repornește serviciul dacă acesta se oprește neașteptat

volumes:
  # Volum gol pentru persistența datelor MySQL
  mysql-data:
```

Cod sursă 2. Fișierul docker-compose pentru pornirea serverului

Acest fișier descrie o configurație Docker Compose pentru cele două servicii necesare: 'medbuddy-sv', imaginea personalizată de mai sus, și 'medbuddy-db', o imagine mysql.

Voi descrie în linii mari fișierul:

- Se specifică versiunea '3' pentru Docker Compose.
- Serviciul 'medbuddy-sv' este definit astfel: se construiește o imagine folosind fișierul Dockerfile din directorul curent și se va mapa portul local 8080 pe portul containerului 8080, iar acest serviciu depinde de 'medbuddy-db', însemnând că dacă acel serviciu se va încheia neașteptat, se va încheia și acesta.
- Serviciul 'medbuddy-db' folosește ultima versiune de imagine 'mysql' disponibilă pe Dockerhub, se va mapa portul local 3306 pe portul 3306 al containerului, iar apoi sunt specificate variabilele de mediu.
- Volumul gol 'mysql-data' este utilizat pentru persistența datelor MySQL.

Baza de date „medbuddy” creată în interiorul instanței conține următoarele tabele pentru organizarea datelor:

- users

```
-- Crearea tabelii 'users'
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  fullName VARCHAR(255),
  email VARCHAR(255) UNIQUE,
  phoneNumber VARCHAR(255),
  password VARCHAR(255),
  role VARCHAR(255)
);
```

Cod sursă 3. Definiția tabelii „users”

În această tabelă se salvează datele de autentificare și de identificare ale fiecărui utilizator ce s-a înregistrat în aplicația mobilă.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabele și numărul unic de identificare a fiecărui utilizator
- 'fullName' = numele complet al utilizatorului
- 'email' = adresa de email al utilizatorului
- 'phoneNumber' = numărul de telefon al utilizatorului
- 'password' = parola cu care acesta se autentifică în aplicație
- 'role' = rolul acestuia în această aplicație, medic sau pacient

- userDetails

```
• -- Crearea tabelii 'UserDetails'  
• CREATE TABLE userDetails (  
•   id INT AUTO_INCREMENT PRIMARY KEY,  
•   userId INT,  
•   age INT,  
•   gender VARCHAR(255),  
•   weight VARCHAR(10)  
• );
```

Cod sursă 4. Definiția tabelii „userDetails”

În această tabelă se salvează datele fizice ale unui pacient, luate în considerare de către medic în momentul diagnosticării și în oferirea medicației.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabele
- 'userId' = ID-ul prin care este identificat pacientul în tabela 'users'
- 'age' = vârsta utilizatorului
- 'gender' = sexul utilizatorului
- 'weight' = greutatea utilizatorului

- doctorSpecialty

```
• -- Crearea tabelii 'doctorSpecialty'  
• CREATE TABLE doctorSpecialty (  
•   id INT AUTO_INCREMENT PRIMARY KEY,  
•   doctorID INT,  
•   doctorSpecialty VARCHAR(255),  
• );
```

Cod sursă 5. Definiția tabelii „doctorSpecialty”

În această tabelă se salvează datele care specifică specialitatea respectivului medic.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabele
- 'doctorID' = ID-ul prin care este identificat medicul în tabela 'users'
- 'doctorSpecialty' = specialitatea medicului

- medicalRecords

```
-- Crearea tabelii 'medical_records'  
CREATE TABLE medical_records (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  active INT default '1',  
  accepted INT default '0',  
  diagnostic VARCHAR(255),  
  doctorID INT,  
  medication VARCHAR(255),  
  patientID INT,  
  symptom VARCHAR(255),  
  specialty VARCHAR(255)  
);
```

Cod sursă 6. Definiția tabelii „medical_records”

În această tabelă se salvează datele pentru cererea de ajutor medical virtuală.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabeli
- 'active' = statusul cererii din punct de vedere al activității, valoarea inițială fiind 1, iar aceasta devine 0 în momentul în care medicul încheie perioada de tratament
- 'accepted' = statusul cererii din punct de vedere al validării, valoarea în momentul în care cererea a fost creată este 0, iar aceasta devine 1 în momentul în care un medic specialist acceptă cererea
- 'diagnostic' = diagnosticul oferit de medicul specialist după analizarea simptomelor și a datelor fizice
- 'medication' = medicația stabilită de medicul specialist
- 'symptom' = descrierea pacientului despre simptomele pe care le manifestă
- 'specialty' = specialitatea cerută în cererea de ajutor
- 'doctorID' = ID-ul medicului specialist responsabil din momentul acceptării cererii
- 'patientID' = ID-ul pacientului care a creat cererea

- messages

```
-- Crearea tabelii 'messages'  
CREATE TABLE messages (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  roomID INT,  
  senderID INT,  
  receiverID INT,  
  message VARCHAR(255)  
);
```

Cod sursă 7. Definiția tabelii „messages”

În această tabelă se salvează datele schimburilor de mesaje dintre medic și pacient pe perioada tratamentului virtual.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabele
- 'roomID' = ID-ul unic al cererii de ajutor, reprezentând perioada de tratament
- 'senderID' = ID-ul celui care a trimis mesajul, medic sau pacient
- 'receiverID' = ID-ul celui care a primit mesajul, medic sau pacient
- 'message' = mesajul propriu-zis

- appointment

```
-- Crearea tabelii 'appointment'
CREATE TABLE appointment (
  id INT AUTO_INCREMENT PRIMARY KEY,
  active INT default '1',
  accepted INT default '0',
  date VARCHAR(255),
  location VARCHAR(255),
  doctorID INT,
  patientID INT,
  specialty VARCHAR(255)
);
```

Cod sursă 8. Definiția tabelii „appointment”

În aceasta tabelă se salvează datele pentru cererea de programare fizică la un consult de către un pacient și datele acesteia după primirea răspunsului de la un medic.

Câmpurile reprezintă următoarele lucruri:

- 'id' = cheia primară a acestei tabele
- 'active' = statusul cererii din punct de vedere al activității, valoarea inițială fiind 1, iar aceasta devine 0 în momentul în care consultul a avut loc.
- 'accepted' = statusul cererii din punct de vedere al validării, valoarea în momentul în care cererea a fost creată este 0, iar aceasta devine 1 în momentul în care un medic specialist acceptă cererea
- 'date' = data și ora consultului, stabilite de medic
- 'location' = adresa unde va avea loc consultul
- 'specialty' = specialitatea cerută în cererea de consult
- 'doctorID' = ID-ul medicului specialist responsabil din momentul acceptării cererii
- 'patientID' = ID-ul pacientului care a creat cererea

Fișierul C++ (server.cpp) în care a fost dezvoltat serverul pornește și este configurat astfel:

```
// Procează cererile rimate
void startServer(asio::io_context &io_context, short port, sql::mysql::MySQL_Driver
*driver)
{
    tcp::acceptor acceptor(io_context, tcp::endpoint(tcp::v4(), port));

    while (true)
    {
        tcp::socket socket(io_context);
        acceptor.accept(socket);

        std::string request;

        // Citește cererea
        char data[1024];
        size_t bytes_read = socket.read_some(asio::buffer(data, sizeof(data)));

        // Concatenează datele primite la șirul request
        request.append(data, data + bytes_read);

        // Procează cererea
        handleRequest(socket, request, driver);
    }
}

int main()
{
    try
    {
        // Inițializează driver-ul MySQL
        sql::mysql::MySQL_Driver *driver;
        driver = sql::mysql::get_mysql_driver_instance();

        // Create unui io_context ASIO.
        Asio::io_context io_context;

        // Pornește serverul pe portul 8080
        startServer(io_context, 8080, driver);
    }
    catch (std::exception &e)
    {
        std::cerr << "Server error: " << e.what() << std::endl;
    }
}
```

Cod sursă 9. Pornirea serverului

În funcția main, punctul de intrare al programului, se încearcă pornirea serverului în blocul 'try', după ce se creează un obiect io_context din biblioteca Asio și un driver pentru conexiunea la baza de date MySQL, iar în cazul în care aceasta eșuează, eroarea va fi prinsă și printată spre standard output în blocul 'catch'.

Apelând funcția 'startServer' cu parametrii corespunzători, se creează un obiect 'acceptor' care ascultă pentru conexiuni pe portul specificat folosind Ipv4. Aceasta intră într-un ciclu infinit pentru a accepta noi conexiuni și a procesa cereri, iar la fiecare iterație se va crea un nou obiect socket pentru a reprezenta conexiunea către un client.

Cererea se gestionează apelând funcția 'handleRequest', unde fiecare tip de cerere este procesat diferit.

```
std::string success = "HTTP/1.1 200 OK\r\n";  
std::string unauthorized = "HTTP/1.1 401 Unauthorized\r\n";  
std::string internalError = "HTTP/1.1 500 Internal Server Error\r\n";  
std::string badRequest = "HTTP/1.1 400 Bad Request";
```

Cod sursă 10. Șirurile pentru răspunsuri

Variabilele de tip std::string, succes, unauthorized, internalError și badRequest, reprezintă începutul variabilei response_header, care se trimite ca răspuns către client după procesarea request-ului.

```
// Proceșează o cerere HTTP  
void handleRequest(tcp::socket &socket, const std::string &request,  
sql::mysql::MySQL_Driver *driver)  
{  
    std::string response_header;  
    std::string response_body;  
  
    // Extrage calea și metoda cererii  
    std::stringstream request_stream(request);  
    std::string method, path, http_version;  
    request_stream >> method >> path >> http_version;  
  
    // Verifică și generează răspunsul aferent căii  
  
    // ...
```

Cod sursă 11. Declararea variabilelor necesare pentru procesarea cererii

La începutul funcției handleRequest, declarăm variabilele de tip std::string response_header și response_body. Aceste două variabile se vor trimite ca răspuns, după procesarea request-ului, către aplicația client.

Parametrul request este transformat într-un std::stringstream pentru a extrage cu ușurință datele necesare pentru identificarea request-ului.

Pentru a explica cum se procesează un request, o să dau ca o exemplu un caz mai mic și anume request-ul 'getName', care returnează numele întreg al utilizatorului, fie medic, fie pacient, primind ca și parametru ID-ul respectivului.

```
// ### GET NAME ###

else if (method == "POST" && path == "/getName")
{
    std::string line;

    while (std::getline(request_stream, line))
    {
        // Iterează până la ultima linie, care conține parametrii cererii
    }
    // Decodifică conținutul cererii
    std::string decoded_line = urlDecode(line);

    size_t idStart = decoded_line.find("id=") + 3; // Verifică poziția de start
    // si adaugă lungimea șirului "id="
    std::string id = decoded_line.substr(idStart); // Extrage subșirul id

    // Verifică dacă subșirul id nu este gol
    if (!id.empty())
    {
        try
        {
            // Creează o conexiune MySQL
            sql::Connection *con = driver->connect("tcp://medbuddy-db:3306",
"admin", "admin");
            con->setSchema("medbuddy");

            // Pregătește interogarea
            std::unique_ptr<sql::PreparedStatement> pstmt(con-
>prepareStatement("SELECT fullName FROM users WHERE id = ?"));
            pstmt->setString(1, id);

            sql::ResultSet *res = pstmt->executeQuery();

            // Verifică dacă utilizatorul a fost găsit
            if (res->next())
            {
                do
                {
                    // Obține ID-ul si numele utilizatorului
                    std::string fullName = res->getString("fullName");

                    response_header = success;
                    response_body = "fullName=" + fullName + "\n";
                } while (res->next());
            }
        }
        catch (...)
        {
            // Eroare la conectare la baza de date
        }
    }
}
```

```
        } while (res->next());
    }
    else
    {
        // Utilizatorul nu a fost găsit sau datele sunt greșite
        response_header += unauthorized;
        response_body = "Invalid id";
    }

    delete res;
    delete con;
}
catch (std::exception &e)
{
    std::cerr << "Data request failed: " << e.what() << std::endl;
    response_header += internalError;
    response_body = "Data request failed";
}
}
else
{
    response_header += badRequest;
    response_body = "Invalid data request";
}
}
```

Cod sursă 12. Exemplu de procesare al unei cereri

La început se iterează stream-ul 'request_stream' pentru a ajunge la șirul care conține datele pentru procesare, în acest caz fiind doar id-ul utilizatorului, apoi se decodează acel șir și se extrage valoare id-ului.

În cazul în care valoare id-ului nu este nulă, se realizează conexiunea la MySQL în interiorul container-ului de docker și se setează ca baza de date pe care o interogăm 'medbuddy', iar dacă valoarea ID-ului nu corespunde, server-ul va trimite răspunsul pentru request invalid. Se interoghează tabela users pentru a obține câmpul 'fullName' și în cazul în care aceasta se execută cu succes, se trimite răspunsul către client, iar dacă ID-ul nu există în baza de date sau apare o problemă la execuția interogării, se vor trimite răspunsurile aferente.

În această manieră se procesează fiecare tip de request trimis de către client, dar diferă numărul de interogări executate și de parametrii.

6.2 Aplicația mobilă

În momentul pornirii aplicației MedBuddy, utilizatorul este întâmpinat cu un splash screen, care conține logo-ul și numele aplicației.



Figura 5. Fereastra Splash

Acest scenariu este lansat în MainActivity și este legat trecerea la scenariul de autentificare al aplicației, iar tranziția la aceasta se realizează cu un delay de 4 secunde, astfel:

```
class MainActivity : AppCompatActivity() {  
  
    private val splashScreen = 4000  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        ContextCompat.checkSelfPermission(this,  
        android.Manifest.permission.POST_NOTIFICATIONS)  
  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.splash)  
        val image = findViewById<ImageView>(R.id.medbuddy_image)
```



```
val title = findViewById<TextView>(R.id.medbuddy_title)
val tagline = findViewById<TextView>(R.id.medbuddy_tagline)
val topAnim = AnimationUtils.loadAnimation(this, R.anim.top_animation)
val botAnim = AnimationUtils.loadAnimation(this, R.anim.bot_animation)

image.animation = topAnim
title.animation = botAnim
tagline.animation = botAnim

Handler(Looper.getMainLooper()).postDelayed({
    val intent = Intent(this, Login::class.java)
    val options = ActivityOptions.makeSceneTransitionAnimation(
        this, UtilPair.create(title, "logo_text"),
        UtilPair.create(image, "logo_image")
    )
    startActivity(intent, options.toBundle())
}, splashScreen.toLong())
}
```

Cod sursă 13. Implementarea ferestrei Splash și trecerea la autentificare

Pentru a îmbunătății design-ul de pornire a aplicației, am adăugat și animații în momentul în care apare SplashScreen-ul și apoi, în momentul tranziției la ecranul de autentificare.

Odată executată tranziția, utilizatorul ajunge la ecranul de login. Acesta este compus din imaginea de logo a aplicației, titlul acesteia, două câmpuri pentru introdus text, unul pentru introducerea adresei de email și unul pentru introducerea parolei, aferente contului utilizatorului, și de un buton destinat tranziției la scenariul de înregistrare a unui cont nou, în cazul în care este vorba despre un utilizator nou.

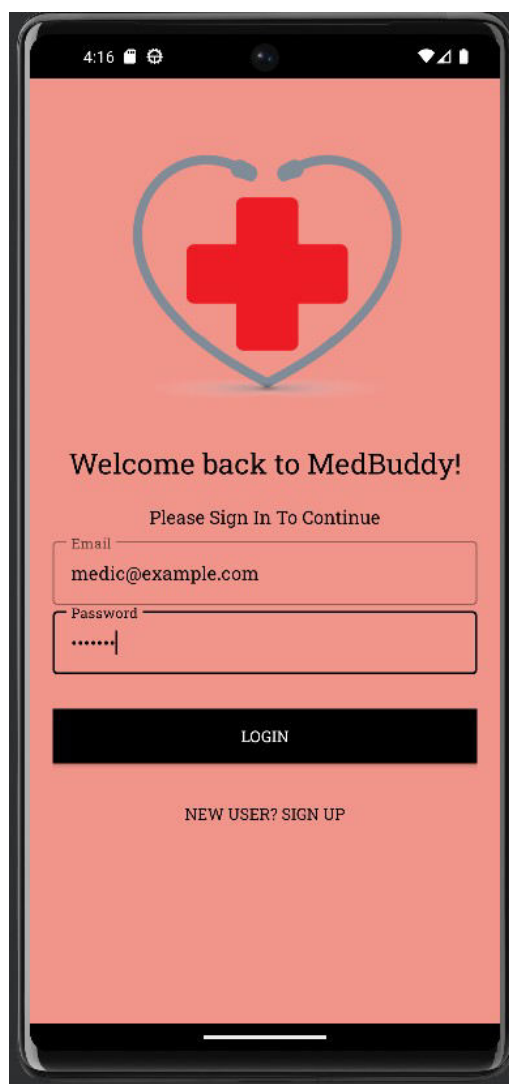


Figura 6. Fereastra de autentificare

În situația în care un nou utilizator dorește să se înregistreze, acestuia îi este prezentat scenariul de înregistrare, care este compus din 5 câmpuri de inserare de text, câte unul pentru numele complet, adresa de mail, numărul de telefon, parola și confirmarea parolei, și dintr-o listă de selecție ce conține două variante, medic și pacient, care vor defini tipul contului și interacțiunea utilizatorului cu aplicația.

Figura 7. Fereastra de înregistrare

Utilizatorul este obligat să completeze toate câmpurile înainte să apese pe butonul de înregistrare, altfel va fi atenționat să procedeze astfel și cererea de înscriere nu va fi trimisă către server.

De asemenea, anumite câmpuri au anumite limitări din punct de vedere al datelor inserate, cum ar fi:

- „Email” – adresa de mail introdusă trebuie să fie validă, aceasta se verifică cu o expresie regulată
- „Phone Number” – numărul introdus trebuie să conțină exact 10 cifre
- „Password” – parola introdusă trebuie să aibă o lungime de cel puțin 7 caractere
- „Confirm Password” – aceasta trebuie să coincidă cu ce a fost introdus în câmpul „Password”

După verificarea datelor introduse, acestea vor fi trimise către server cu ajutorul unui apel din API-ul construit pentru aplicație, și în funcție de răspunsul primit, aceasta se va întoarce la ecranul de autentificare, în cazul în care răspunsul este succes, altfel se va trata și loga răspunsul și de ce a eșuat.

```
    } else if (!sdEmail.matches(emailRegex.toRegex())) {
        email.error = "Email address must be valid!"
    } else if (sdPhoneNumber.length != 10) {
        phoneNumber.error = "Phone number must have 10 digits!"
    } else if (sdPassword.length < 7) {
        password.error = "Password must have least 7 characters!"
    } else if (sdPassword != sdConfirmPassword) {
        confirmPassword.error = "Passwords don't match!"
    } else {

        val apiService = ApiServiceBuilder.apiService
        // Make the register request
        val call =
            apiService.register(sdFullName, sdEmail, sdPhoneNumber,
sdPassword, sdRole)

        call.enqueue(object : Callback<String> {
            override fun onResponse(call: Call<String>, response:
Response<String>) {
                if (response.isSuccessful) {
                    Log.d("INFO", "Registration success.")

                    val intent = Intent(applicationContext,
Login::class.java)

                    startActivity(intent);
                    Toast.makeText(
                        applicationContext,
                        "Account created successfully",
                        Toast.LENGTH_SHORT
                    ).show()
                } else {
                    Log.d("ERROR", "Register failed. Response code: $
{response.code()}")

                    Toast.makeText(
                        applicationContext,
                        "Registration failed. Check the fields!",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
            override fun onFailure(call: Call<String>, t: Throwable) {
                Log.d("ERROR", "Register request failed. Error: $
{t.message}")

                Toast.makeText(
```

```
        applicationContext,  
        "Server error. Try again!",  
        Toast.LENGTH_SHORT  
    ).show()  
    }  
})  
}
```

Cod sursă 14. Tratarea erorilor de înregistrare

Toate cererile trimise din aplicația client către server se realizează cu ajutorul bibliotecii Retrofit, cu ajutorul căreia am definit un obiect pe care îl pot construi în orice clasă am avea nevoie să apelăm o funcție din API-ul construit.

Modificând parametrul din baseUrl, asigurăm accesul la serverul C++. Cererile sunt trimise în plain/text.

```
object ApiServiceBuilder {  
  
    private val retrofit: Retrofit = Retrofit.Builder()  
        .baseUrl("http://192.168.0.115:8080")  
        .addConverterFactory(ScalarsConverterFactory.create())  
        .build()  
    val apiService: ApiService = retrofit.create(ApiService::class.java)  
}
```

Cod sursă 15. Configurarea adresei pentru server

Aici este un exemplu despre cum am definit câteva dintre apelurile API-ului.

```
interface ApiService {  
    @FormUrlEncoded  
    @POST("/login")  
    fun login(  
        @Field("email") email: String,  
        @Field("password") password: String  
    ): Call<String>  
  
    @FormUrlEncoded  
    @POST("/register")  
    fun register(  
        @Field("fullName") fullName: String,  
        @Field("email") email: String,  
        @Field("phoneNumber") phoneNumber: String,  
        @Field("password") password: String,  
        @Field("role") role: String  
    ): Call<String>
```

Cod sursă. Exemplu de definire a apelurilor API

Conținutul cererii trimise către server este codificat URL.

6.2.1. Utilizarea din perspectiva pacientului

Odată autentificat, pacientului îi este prezentată pagina principală a aplicației din perspectiva unui pacient.

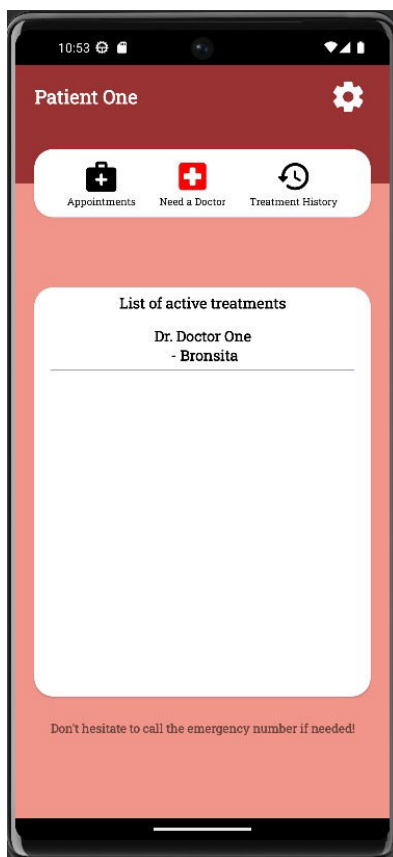


Figura 8. Meniul principal al pacientului

Acest meniu este compus din următoarele elemente:

- Butonul de navigare pentru a ajunge în ecranul de programări fizice
- Butonul pentru crearea unei cereri virtuale de ajutor medical
- Butonul de navigare pentru a ajunge la istoricul tratamentelor virtuale
- Lista pentru tratamentele virtuale active în acel moment
- Butonul de setări
- Numele pacientului

Astfel, interacțiunea cu aplicația devine sugestivă și execuția acțiunilor este bine delimitată.

Butonul de setări deschide o fereastră de tip „pop-up”, care prezintă două acțiuni, cea pentru delogare și cea de editare a profilului pacientului.

În momentul apăsării butonului pentru delogare, aplicația revine la fereastra de autentificare.

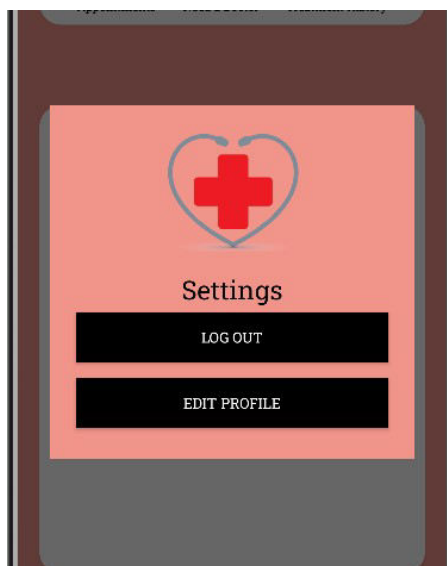


Figura 9. Setările pacientului

Apăsarea butonului de editare profil rezultă în prezentarea unui formular în care toate câmpurile prezentate necesită a fi completate pentru a putea executa operația de actualizare, iar aceasta are loc doar dacă butonul „Save” este apăsat.

Este necesar să actualizăm profilul, deoarece aici se specifică proprietățile fizice ale pacientului.

A mobile application interface for editing a patient's profile. At the top, it says "Patient One" next to a gear icon. Below this is a red heart icon with a white cross. The title "Edit your account" is centered. There are four input fields: "Full Name", "Age", "Weight", and "Phone Number". Below these is a dropdown menu currently showing "Male". At the bottom, there is a black button with white text that says "SAVE". The background is a light pink color.

Figura 10. Actualizarea profilului ca pacient

Această operație este executată trimițând o cerere prin apelarea API-ului.

Apăsând butonul de navigare „Need a doctor”, pacientului îi este prezentată fereastra de creare a unei noi cereri de asistență medicală.

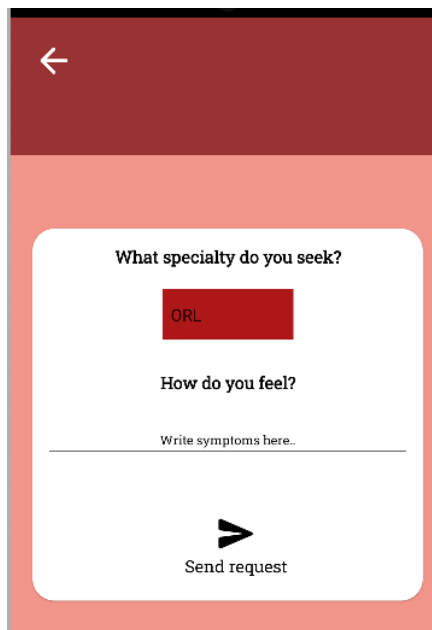


Figura 11. Crearea unei cereri de asistență virtuală

Fereastra include un buton de întoarcere la meniul principal și un formular în care pacientul este întrebat ce tip de medic specialist ar avea nevoie apoi trebuie să completeze un câmp în care descrie ce simptome manifestă și cum se simte, cererea fiind trimisă după apăsarea butonului „Send request”.

Lista de tratamente virtuale active este actualizată interogând baza de date printr-un apel API, iar aceasta este populată cu cereri care au primit deja răspuns de la unul dintre medici. Fiecare dintre elementele disponibile în listă duc la fereastra care servește scopul acestei aplicații, și anume interacțiunea cu mediul specialist.

În fereastra de interacțiune cu medicul specialist, pacientului îi este prezentat ca titlu numele complet al medicului și poate observa, într-un card prezentat pe ecran, diagnosticul și medicația prescrisă.

Pacientul poate comunica liber cu medicul, exprimându-și evoluția stării de sănătate și mulțumirea eficacității medicației prin cardul de chat prezentat pe ecranul interacțiunii, astfel reducând din formalitate și din timpul pierdut prin a scrie mail-uri ce necesită un format mai elevat.

Acesta își poate seta o notificare pentru aducere aminte cu ajutorul căreia aplicația îi va trimite o notificare odată la câteva ore, acest parametru fiind setat de pacient, pentru a nu rata luarea de tratament.



Figura 12. Interacțiunea cu medicul

Odată de tratamentul a luat sfârșit, acesta este adăugat în istoricul medical al pacientului, accesat din meniul principal.

Acesta prezintă un buton de întoarcere la meniul principal și o listă cu toate tratamentele din trecut, iar prin apăsarea uneia dintre intrările din aceasta, pacientul poate revedea diagnosticul, simptomele și medicamentația.

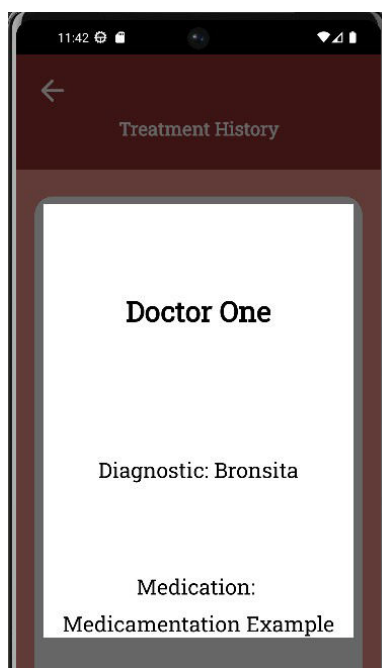


Figura 13. Istoricul tratamentelor

Ultima dintre utilizările propuse pentru un pacient de către MedBuddy este cererea rapidă pentru o programare la consult cu un medic specialist.

Pacientul poate vedea dacă cererea lui a fost acceptată de un medic, cât și eventuale alte programări în fereastra de programări accesată din meniul principal prin butonul „Appointments”.

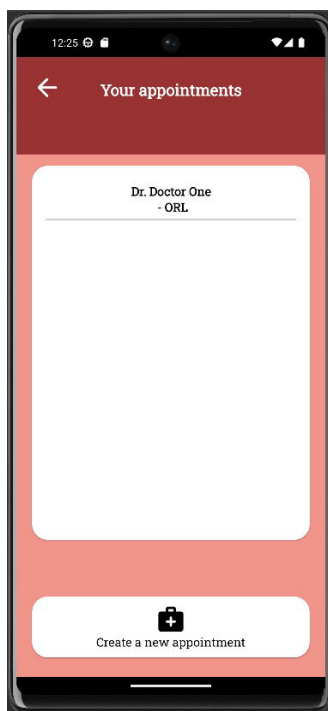


Figura 14. Consultațiile viitoare ale pacientului

Pacientul poate vedea datele aferente consultului apăsând pe intrarea din lista de consulturi, iar apoi i se va prezenta o fereastră de tip „Pop-up” cu detaliile și numele medicului.

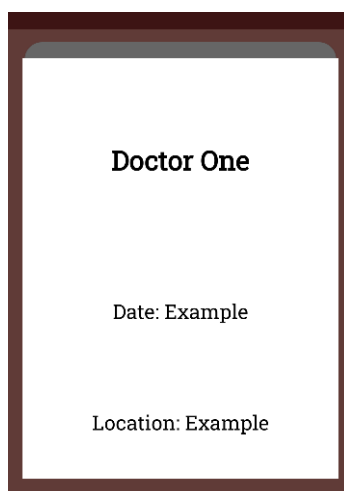


Figura 15. Detaliile consultului

Apăsând pe butonul „Create a new appointment”, pacientului îi este prezentat un formular în care acesta trebuie să aleaga doar specialitatea medicului care îl va consulta.

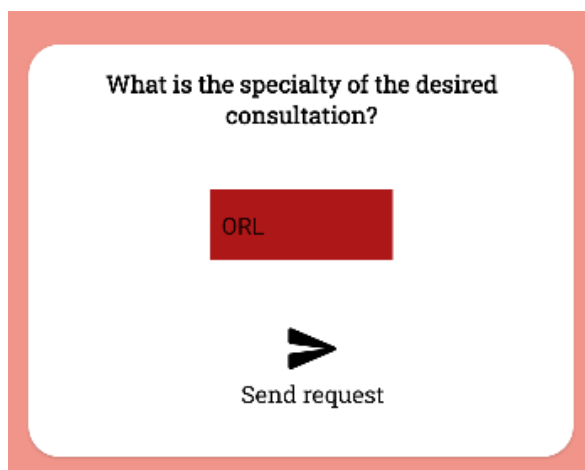
A screenshot of a mobile application interface for creating a consultation request. The screen has a light pink background. At the top, a white rounded rectangle contains the text "What is the specialty of the desired consultation?". Below this text is a red rectangular button with the white text "ORL". Underneath the button is a large black right-pointing arrow. At the bottom of the white rectangle is the text "Send request".

Figura 16. Crearea unei cereri de consultație

6.2.2. Utilizarea din perspectiva medicului

După ce s-a conectat cu succes și a trecut de fereastra de autentificare, medicului îi este prezentat meniul principal din perspectiva aferentă rolului său.

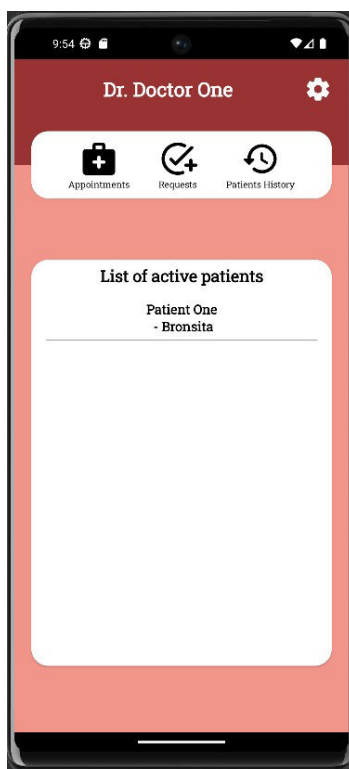


Figura 17. Meniul principal al medicului

Acest meniu este compus din următoarele elemente:

- Butonul de navigare pentru a ajunge în ecranul de programări fizice
- Butonul de navigare pentru a ajunge în ecranul de programări virtuale
- Butonul de navigare pentru a ajunge la istoricul pacienților
- Lista pentru tratamentele virtuale active în acel moment
- Butonul de setări
- Numele medicului

Butonul de setări se comportă la fel ca și în perspectiva pacientului, dar formularul de editare de profil diferă, astfel medicul fiind nevoit să specifice specialitatea lui.

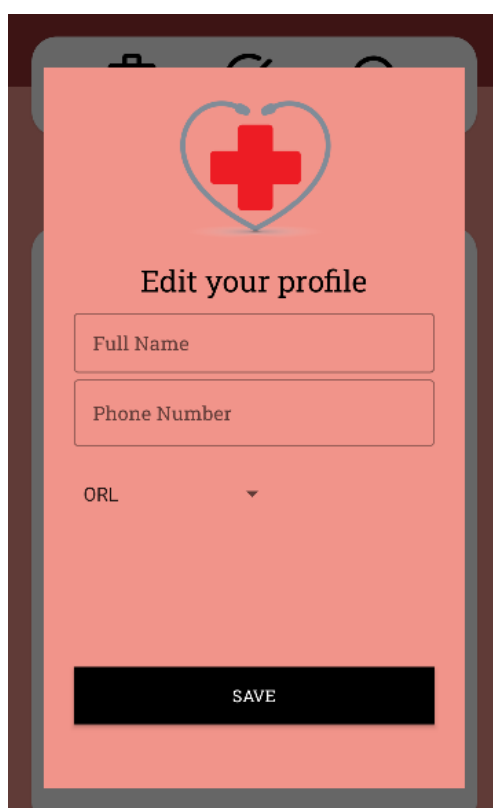


Figura 18. Actualizarea profilului ca medic

Navigând cu ajutorul butonului „Requests”, medicului îi este prezentată o fereastră ce include un buton de întoarcere la meniul principal și o listă de cereri de asistență virtuală din partea pacienților pentru specialitatea aleasă de medic.

Acesta poate accepta sau refuza una dintre cererile respective accesându-le, moment în care medicului îi este prezentată o nouă fereastră care conține detaliile oferite de pacient, numele, proprietățile lui fizice, simptomele pe care le manifestă și starea acestuia. Astfel medicul poate diagnostica pacientul după detaliile respective și poate oferi o medicație aferentă.

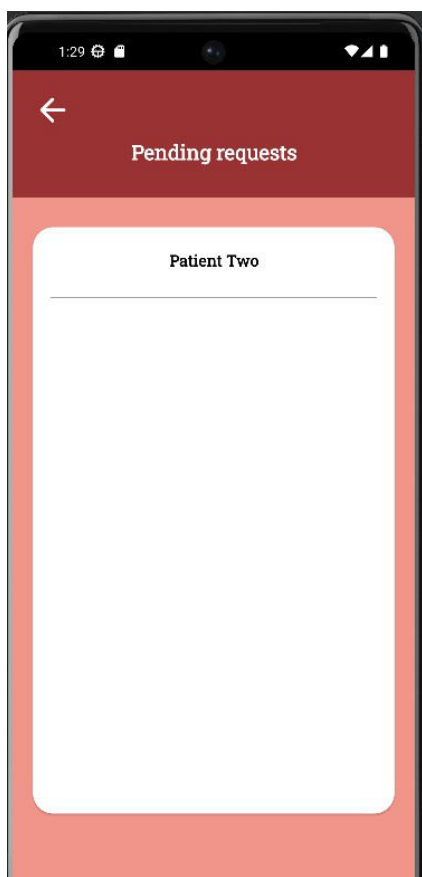


Figura 19. Lista cererilor

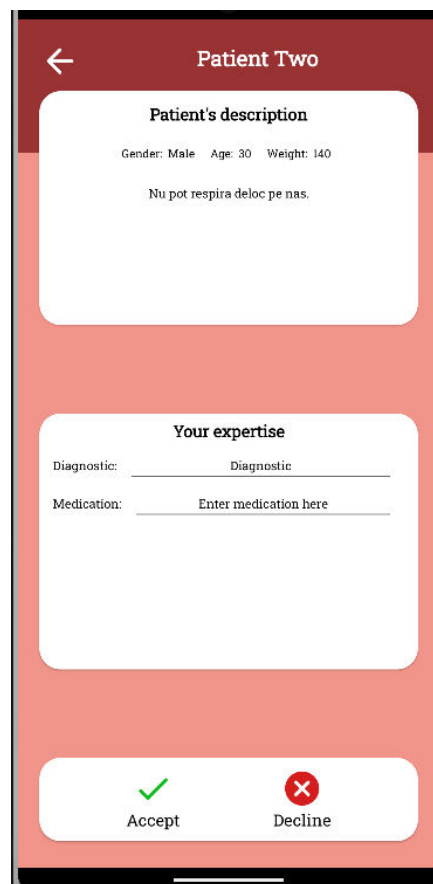


Figura 20. Răspunsul medicului

Lista afișată în meniul principal reprezintă tratamentele active pentru care medicul este responsabil și astfel acesta poate accesa interacțiunea cu pacientul, ceea ce servește scopul aplicației.

În fereastra de interacțiune cu pacientul, medicului îi este prezentat un card cu numele pacientului, alături de diagnostic, simptomele manifestate și medicamentația stabilită.

Cel de al doilea card conține conversația dintre medic și pacient, unde cei doi pot păstra legătura despre evoluția stării de sănătate a pacientului.

Dacă medicul decide că medicamentația nu are efectul dorit, acesta poate edita acest câmp pentru a spori tratarea pacientului cu ajutorul butonului „Edit”.

În momentul în care medicul ia concluzia, din conversația cu pacientul, că acesta este vindecat, acesta poate încheia tratamentul cu ajutorul butonului „End”.

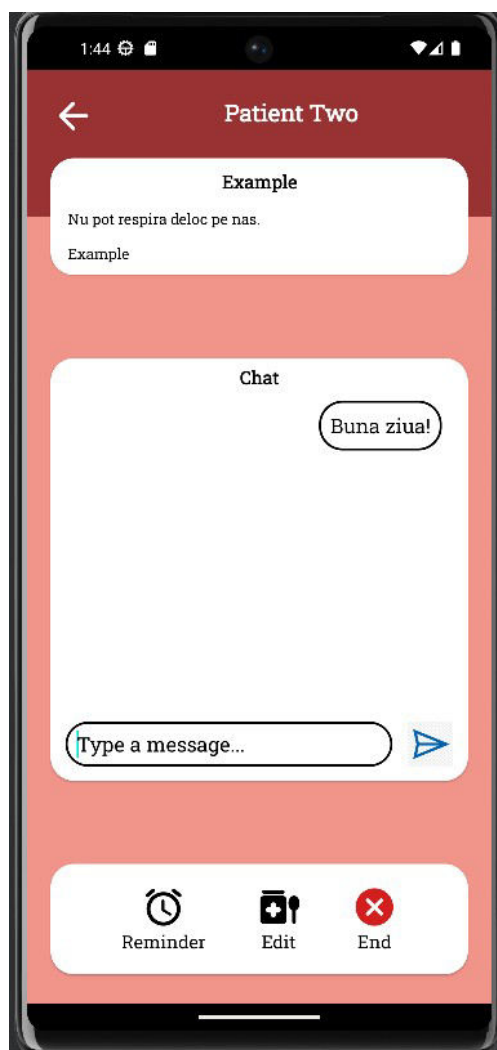


Figura 21. Interacțiunea cu pacientul

Medicul își poate verifica tratamentele accesând butonul „Patients History” pentru a vedea tratamente din trecut și pentru a observa ce medicație a avut efect într-un caz similar.

Această fereastră conține o listă care este populată cu tratamentele precedente și butonul de întoarcere în meniul principal.

În momentul în care medicul apasă pe o intrare din acea listă, i se va afișa o fereastră de tip „pop-up” cu detaliile perioadei de tratament, ce includ numele, diagnosticul, medicația și simptomele.

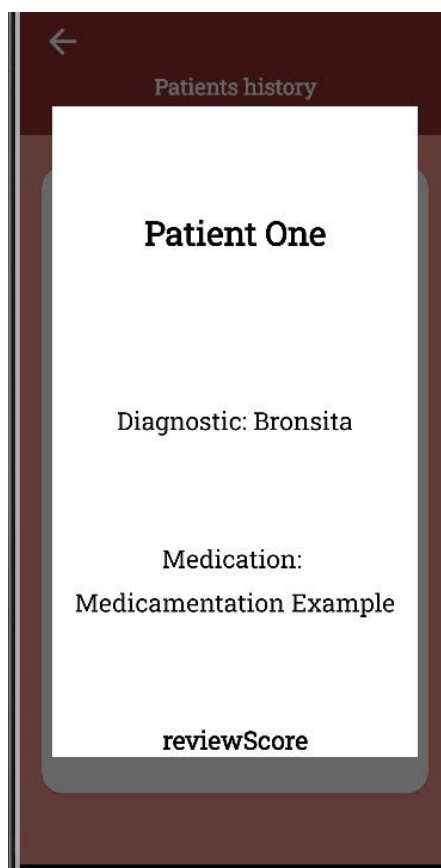


Figura 22. Istoricul pacientilor

Ultima dintre funcționalitățile pe care le poate folosi medicul sunt programările la consult, care pot fi accesate din meniul principal apăsând pe butonul „Appointments”, rezultând în afișarea unei ferestre care conține o listă cu programările deja stabilite, butonul de întoarcere la meniul principal și butonul pentru accesarea listei cu cereri de consult care așteaptă răspuns de la unul dintre medicii specialiști din respectivul domeniu.

Medicul poate vedea detaliile unei consultații deja stabilite apăsând pe intrarea din listă, apoi i se va prezenta o fereastră de tip „pop-up” cu detaliile, care include și un buton „Delete”, în cazul în care medicul dorește să anuleze consultația.

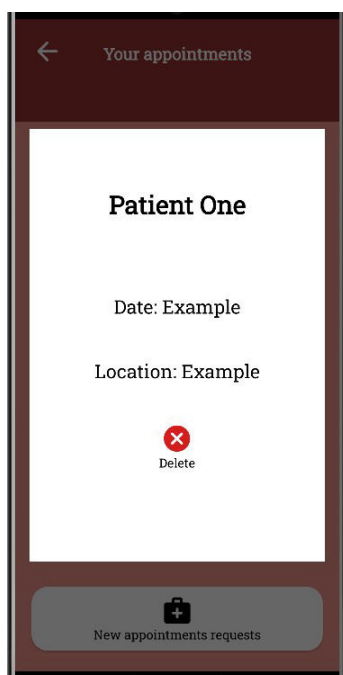


Figura 23. Detaliile consultației

După ce medicul apasă pe butonul „New Appointments requests”, acesta navighează către lista cu cererile de consult nestabilite, iar ca să răspundă trebuie doar să apese pe intrarea din listă ca să i se afișeze fereastra unde trebuie să completeze formularul cu data și locația consultului.

Figura 24. Stabilirea unui consult

7. Concluzie și direcții viitoare de dezvoltare

7.1 Concluzie

În concluzie, MedBuddy reușește să gestioneze cu succes cererile de ajutor medical virtual și cererile de programări la consult față în față, eficientizând comunicarea dintre medic și pacient cu ajutorul ferestrei de dialog pentru a ajuta la oferirea unei medicamentației potrivite pentru fiecare caz.

Atât medicii și pacienții au la dispoziție un istoric al tratamentelor, unde pot observa cu cine au interacționat și ce medicamentație a fost stabilită pentru diagnosticul respectiv.

7.2 Direcții viitoare de dezvoltare

Versiunea actuală a aplicației mobile MedBuddy nu prezintă versiunea finală, ci doar o primă versiune a acesteia, asemenea și în cazul serverului. Aplicația poate fi îmbunătățită, astfel am putea adăuga noi implementări. Aceste câteva lucruri noi pe care am putea să le includem în dezvoltarea acestora ar fi:

- Trecerea serverului de la HTTP la HTTPS și adaptarea aplicației mobile în consecință.
- Implementarea funcționalității de notificare în toate cazurile necesare, în momentul primirii unui mesaj nou sau primirea unei noutăți legate de cererile de ajutor și consultații.
- Criptarea datelor pentru un transfer mai sigur de date între client și server.
- Implementarea sistemelor de testare automată a tuturor funcționalităților aplicației mobile și a apelurilor API pentru o funcționare corectă a serverului, deoarece aplicația nu a ajuns la etapa de testare.
- Îmbunătățirea aspectului aplicației.
- Dezvoltarea serverului pentru ca acesta să ruleze pe mai multe fire de execuție.

8. Listă figuri

- 1) Diagrama UML Relație Server-Client
- 2) Diagrama UML Roluri
- 3) Diagrama UML a cazului de utilizare al pacientului
- 4) Diagrama UML a cazului de utilizare al medicului
- 5) Fereastra Splash
- 6) Fereastra de autentificare
- 7) Fereastra de înregistrare
- 8) Meniul principal al pacientului
- 9) Setările pacientului
- 10) Actualizarea profilului ca pacient
- 11) Crearea unei cereri de asistență virtuală
- 12) Interacțiunea cu medicul
- 13) Istoricul tratamentelor
- 14) Consultațiile viitoare ale pacientului
- 15) Detaliile consultului
- 16) Crearea unei cereri de consult
- 17) Meniul principal al medicului
- 18) Actualizarea profilului ca medic
- 19) Lista cererilor
- 20) Răspunsul medicului
- 21) Interacțiunea cu pacientul
- 22) Istoricul pacienților
- 23) Detaliile consultației
- 24) Stabilirea unui consult

8. Listă cod sursă

- 1) Dockerfile pentru fișierul C++
- 2) Fișierul docker-compose pentru pornirea serverului
- 3) Definiția tabelii „users”
- 4) Definiția tabelii „userDetails”
- 5) Definiția tabelii „doctorSpecialty”
- 6) Definiția tabelii „medical_records”
- 7) Definiția tabelii „messages”
- 8) Definiția tabelii „appointment”
- 9) Pornirea serverului
- 10) Șirurile pentru răspunsuri
- 11) Declararea variabilelor necesare pentru procesarea cererii
- 12) Exemplu de procesare al unei cereri
- 13) Implementarea ferestrei Splash și trecerea la autentificare
- 14) Tratarea erorilor de înregistrare
- 15) Configurarea adresei pentru server
- 16) Exemplu de definire a apelurilor API

10. Bibliografie

- [1] Kotlin. (n.d.). Accesat la adresa <https://kotlinlang.org/docs/home.html>
- [2] Hüsken, S., & Döring, M. (2018). Kotlin for Android Developers. Packt Publishing.
- [3] C++. (n.d.). Accesat la adresa <https://en.cppreference.com/w/cpp>
- [4] Stroustrup, B. (2013). Limbajul de programare C++. Addison-Wesley Professional.
- [5] Android Studio. (n.d.). Accesat la adresa <https://developer.android.com/docs>
- [6] Visual Studio Code. (n.d.). Accesat la adresa <https://code.visualstudio.com/docs>
- [7] Docker. (n.d.). Accesat la adresa <https://www.docker.com/resources/what-docker>
- [8] MySQL. (n.d.). Accesat la adresa <https://dev.mysql.com/doc/refman/8.0/en>
- [9] GitHub. (n.d.). Accesat la adresa <https://github.com/about>
- [10] Retrofit. (n.d.). Accesat la adresa <https://square.github.io/retrofit/>
- [11] Biblioteca Asio C++. (n.d.). Accesat la adresa <https://think-async.com/Asio/asio-1.28.0/doc/>
- [12] MySQL Connector/C++. (n.d.). Accesat la adresa <https://dev.mysql.com/doc/refman/8.0/en/>
- [13] Celko, J. (2011). SQL for Smarties. Editura TEORA.

Declarație de autenticitate a lucrării
de finalizare a studiilor

Subsemnatul Cotulbea Adrian-Ionuț, legitimat cu CI, seria AR, nr. 859497, CNP 5000410020071, autorul lucrării MedBuddy, elaborată în vederea susținerii examenului de finalizare a studiilor de Calculatoare și Tehnologia Informației organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara, sesiunea iunie/iulie a anului universitar 2022-2023, coordonator principal prof. dr. habilit. ing. Marius Marcu și coordonator secundar ing. Claudiu Grosa, luând în considerare art. 34 din Regulamentul privind organizarea și desfășurarea examenelor de licență/diplomă și disertație, aprobat prin HS nr. 103/14.05.2020 și cunoscând faptul că în cazul constatări ulterioare a unor declarații false, voi suporta sancțiunea administrativă prevăzută de art. 146 din Legea nr. 1/2011 - legea educației naționale și anume anularea diplomei de studii, declar pe proprie răspundere că:

- această lucrare este rezultatul propriei activități intelectuale
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să mi fi citate, inclusiv situația în care sursa reprezintă o altă lucrare/altă lucrări ale subsemnatului.
- sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație.

Timișoara
Data
25.06.2023

Semnătura
Cotulbea