**MEDBUDDY**

**Candidat: Cotulbea Adrian-Ionuț**

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

**Coordonator secundar: ing. Claudiu Groza**

Sesiunea: Iunie/Iulie 2023

**REZUMAT**

Aplicația mobilă MedBuddy reprezintă o platformă revoluționară 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 în acel moment, 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 doctori și să primești, ca răspuns, locul și numele medicului specialist care te va consulta, în momentul în care cererea ta a fost acceptată de acel medic.

**1. CUPRINS**

1. CUPRINS 3

2. INTRODUCERE 4

2.1 CONTEXT 4

2.2 DESCRIEREA PROIECTULUI 4

3. ANALIZA DOMENIULUI 6

3.1 APLICAȚII ASEMĂNĂTOARE 6

4. TEHNOLOGII FOLOSITE 7

4.1 KOTLIN 7

4.2 ANDROID STUDIO 7

4.3 C++ 7

4.4 VISUAL STUDIO CODE 7

4.5 DOCKER 7

4.6 MYSQL 7

4.7 VERSIONAREA CODULUI - GIT 7

4.8 [LIBRĂRII FOLOSITE](#_Toc105955609) 7

4.8.1 RETROFIT 7

4.8.2 ASIO 7

4.8.3 MySQL CONNECTOR/C++ 7

5. PROIECTAREA APLICAȚIEI 7

5.1 DIAGRAME UML A CAZURILOR DE UTILIZARE 7

6. IMPLEMENTARE 7

6.1 SERVER 7

6.2 APLICAȚIA MOBILE 7

6.2.1 UTILIZAREA DIN PUNCT DE VEDERE AL PACIENTULUI 7

6.2.2 UTILIZAREA DIN PUNCT DE VEDERE AL MEDICULUI 7

7. CONCLUZIE 7

8. BIBLIOGRAFIE 7

# 2. INTRODUCERE

Bine ați venit în aplicația Medbuddy, dezvoltată folosind Kotlin și C++.

Această aplicație este concepută pentru a oferi o platformă fără probleme și sigură pentru ca pacienții să se conecteze cu medicii lor și să-și administreze nevoile de îngrijire medicală. Cu această aplicație, pacienții pot adăuga și urmări ușor simptomele lor, pot alege un medic dintr-o listă de profesioniști calificați, pot accesa documentația medicală, pot primi un tratament, pot comunica prin chat în timp real cu medicul, pot primi feedback și pot seta memento-uri pentru medicamentele viitoare.

Motivația pentru alegerea acestei aplicații specifice este de a oferi o modalitate convenabilă și eficientă pentru pacienți de a comunica cu medicii lor și de a-și gestiona nevoile de îngrijire medicală. Medbuddy își propune să reducă decalajul dintre pacienți și medici prin furnizarea unei platforme în care pacienții pot adăuga și urmări ușor simptomele lor, pot alege un medic, pot accesa documentația medicală și pot seta mementouri pentru programări sau medicamente.

Aplicația Medbuddy este o combinație de funcționalități existente din diferite aplicații de îngrijire medicală și comunicare. Medbuddy are multe funcționalități pe care nu le avem într-o singură aplicație. Prin implementarea acestui proiect, aplicația își propune să ofere o modalitate mai convenabilă și eficientă pentru pacienți de a comunica cu medicii lor. Această aplicație poate ajuta pacienții să acceseze ușor informațiile lor medicale și să comunice cu medicii lor în mod prompt. În plus, Medbuddy poate contribui și la reducerea încărcării pe clinici și spitale, oferind pacienților o modalitate convenabilă și ușoară de a accesa informațiile de care au nevoie fără a fi necesară vizita la clinica sau spital. Ideea unui chat în timp real facilitează comunicarea între pacient și medic, după ce medicul a stabilit un tratament și pacientul are nevoie de mai multe informații.

**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 facilitate 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 avantaj potențial 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 | [Practo](https://play.google.com/store/apps/details?id=com.practo.fabric) | [Doctor On Demand](https://play.google.com/store/apps/details?id=com.doctorondemand.android.patient) | [Talkspace](https://play.google.com/store/apps/details?id=com.talkspace.talkspaceapp) | MedBuddy |
| *Adresa site-ului web* | [Practo.com](https://www.practo.com/) | [Doctorondemand.com](https://doctorondemand.com/) | [Talkspace.com](https://www.talkspace.com/) | - |
| *Link magazin* | [Practo](https://play.google.com/store/apps/details?id=com.practo.fabric) | [Doctor On Demand](https://play.google.com/store/apps/details?id=com.doctorondemand.android.patient) | [Talkspace](https://play.google.com/store/apps/details?id=com.talkspace.talkspaceapp) | - |
| *Rating Magazin* | 4.4 / 5 | 4 / 5 | 2 / 5 | - |
| *Număr de instalări* | 10M+ | 1M+ | 500k+ | - |
| *Număr de instalări* | 300k | 40k | 6k | - |
| *Anunțuri/achiziții în aplicație* | x | x | x | - |
| Login/user | x | x | x | x |
| *Chat* | - | - | - | x |
| *Acces la înregistrările medicale* | x | x | x | x |
| *Funcții de feedback* | - | x | - | x |
| *Editare profil* | - | - | - | x |
| *Programare de consultații* | x | x | - | x |
| *Gestionarea medicamentelor* | x | - | x | x |
| *Notificări* | - | x | x | x |
| *Cerere virtuală* | - | - | x | x |

**4. TEHNOLOGII FOLOSITE**

**4.1. Kotlin**

MedBuddy fiind o aplicație mobile a trebuit sa aleg 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. <https://kotlinlang.org/>

**4.2. Android Studio**

Mediul de dezvoltare ales pentru această aplicație este Android studio, deoarece acesta ofera foarte multe utilități incorporate în interfața lui, cum ar fi dezvoltarea fisierelor 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.

<https://developer.android.com/studio>  
 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.

**4.3. C++**

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

C++ este un limbaj de programare general, puternic și versatil, bazat pe limbajul C. A fost dezvoltat în anii 1980 și este utilizat într-o varietate largă de aplicații, de la dezvoltarea de software de sistem până la jocuri și aplicații de înaltă performanță. C++ combină caracteristicile limbajului C cu adăugiri semnificative, cum ar fi programarea orientată pe obiecte, gestionarea dinamică a memoriei și suport pentru șabloane. Acesta oferă un control fin asupra resurselor și performanță înaltă, fiind preferat în domenii care necesită eficiență și control detaliat asupra hardware-ului.

C++ este un limbaj puternic de programare de nivel scăzut și oferă acces la operațiuni de nivel jos, 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.

<https://en.cppreference.com/w/cpp>

**4.4. 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 sursă deschis, 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.

<https://code.visualstudio.com/>

**4.5. Docker**

Am ales sa folosesc Docker pentru a definii un container(un Dockerfile) pentru serverul C++, deoarece acesta este inclus alături de o imagine docker mysql server într-un fisier yml (docker-compose), cu ajutorul caruia pot sa pornesc toată functionalitatea server

cu o singura comanda.

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.

<https://www.docker.com/what-docker>

**4.6. MySQL**

Server-ul C++ se conecteaza la o baza 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.

<https://www.mysql.com/>

**4.7. Versionarea codului – GIT**

Codul aplicației și al serverului au fost versionate cu ajutorul platformei GitHub, unde am făcut issue tracking și am folost ca mod de dezvoltare GitHub Flow, fiecare issue se rezolva/implementeaza pe un branch separat fata 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.

<https://github.com/about>

**4.8. Librării folosite**[Error: Reference source not found](#_Toc105955609)

**4.8.1 Retrofit**

Retrofit a fost folosit pentru a consuma API-ul construit pentru a face schimbul de date dintre server și aplicație, apeland 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.

https://square.github.io/retrofit/

**4.8.2 Asio**

Librăria Asio standalone (non-boost) a fost folosită pentru a rezolva request handle-urile pe partea de server, astfel implementând logica server-ului.

Asio Standalone este o bibliotecă C++ open-source, 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.

<https://think-async.com/Asio/asio-1.28.0/doc/>

**4.8.3 MySQL Connector/C++**

Librăria 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.

https://dev.mysql.com/doc/connector-cpp/

**5. PROIECTAREA APLICAȚIEI**

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



Fig 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 doua roluri, medic sau pacient.

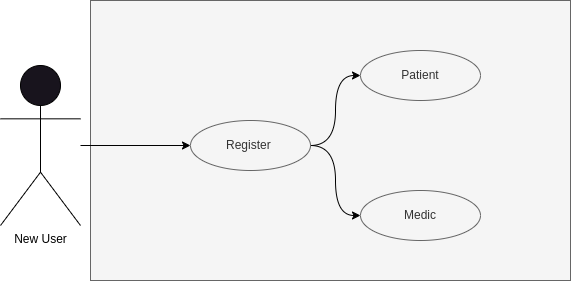
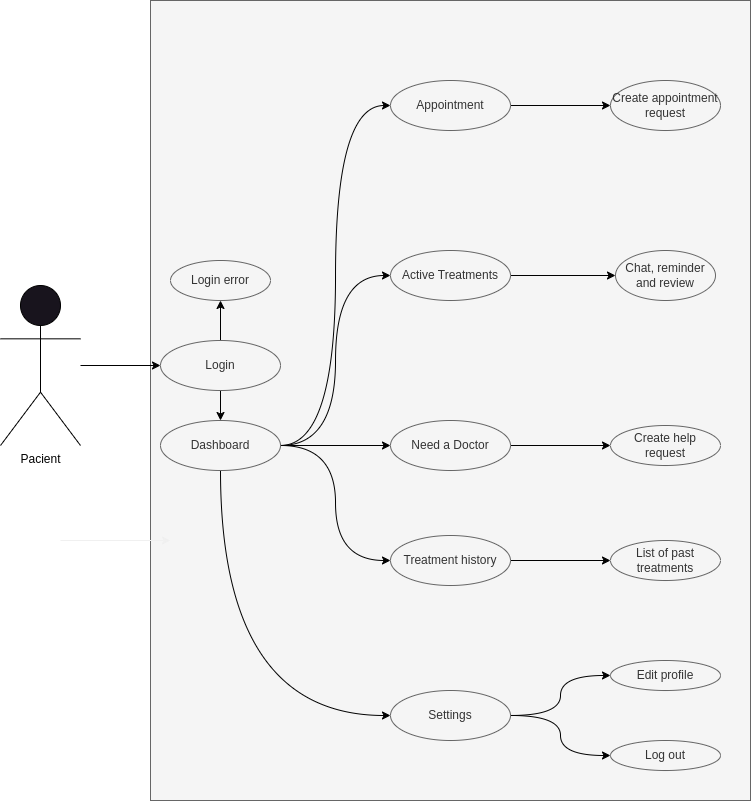
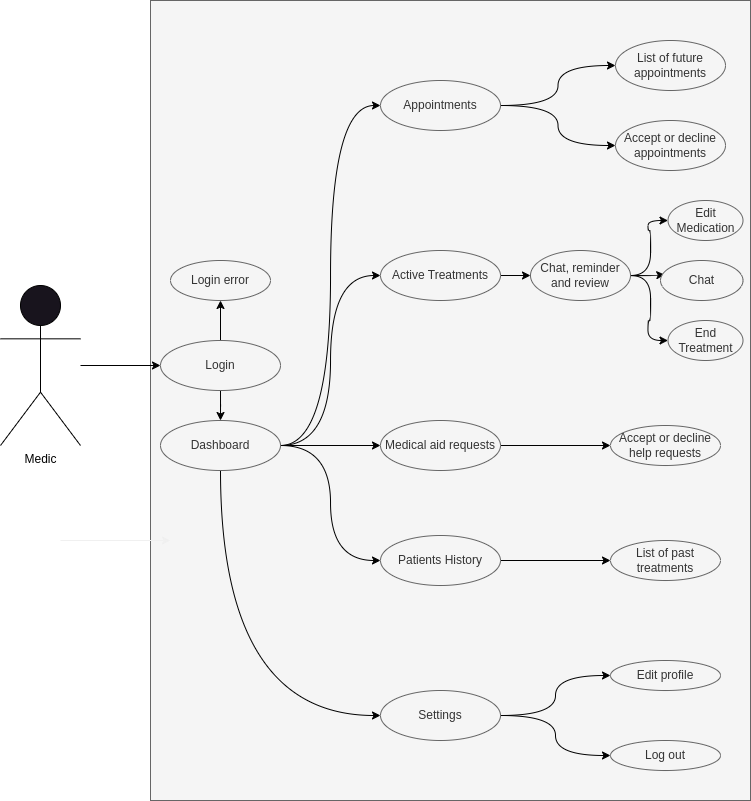


Fig 2. Diagrama UML Roluri

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

**5.1 DIAGRAME UML A CAZURILOR DE UTILIZARE**

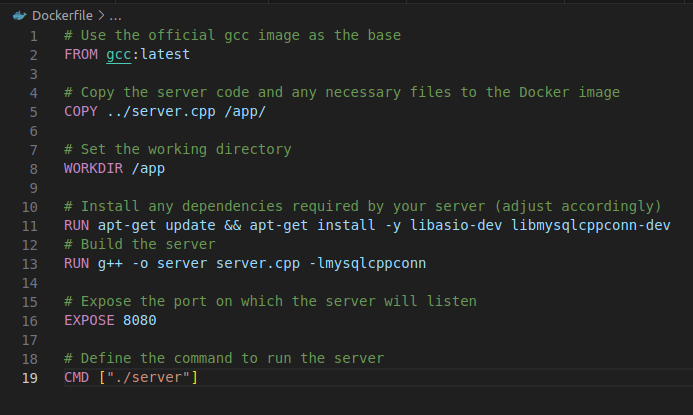
Fig 3. Diagrama UML a cazului de utilizare al pacientului

Fig 4. Diagrama UML a cazului de utilizare al medicului

**6. IMPLEMENTARE**

**6.1 Server**

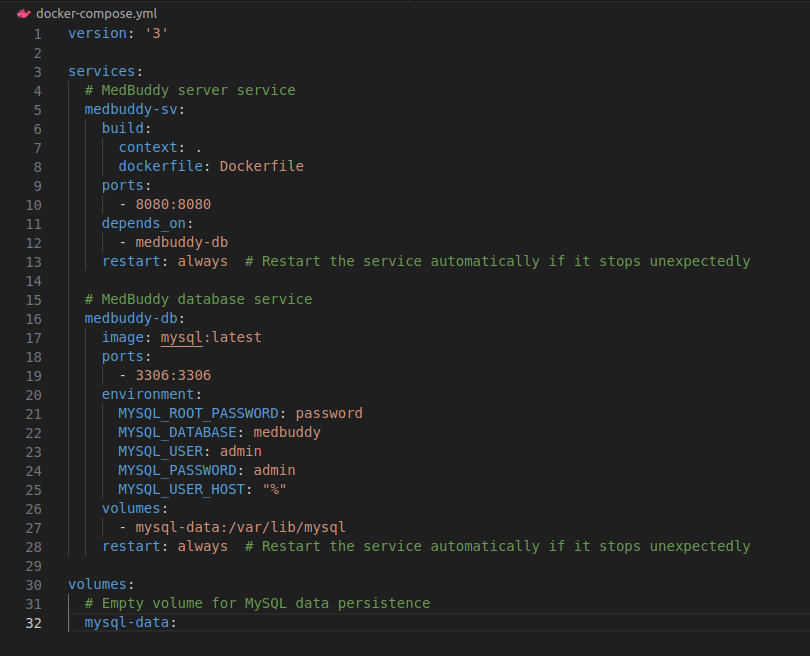
Partea de backend a acestei lucrări constă într-un server dezvoltat în C++ configurat într-un Dockerfile pentru a crea o imagine de docker personalizată.

Fig 5. Dockerfile C++ Server

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 libmysqlcppcon-dev’ - cu ajutorul comenzii ‘RUN’ executăm două instrucțiuni consecutive prin care se actualizeaza lista de pachete și se instalează ‘libasio-dev’ și ‘libmysqlcppcon-dev’, pachete necesare pentru librăriile 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 accesata în sincron cu o imagine mysql în următorul fisier:

Fig 6. Fișierul docker-compose

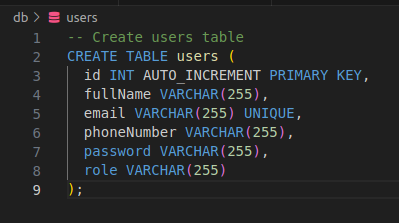
Acest fișier descrie o configurație Docker Compose pentru cele doua servicii necesare: ‘medbuddy-sv’, imaginea personalizată de mai sus, și ‘medbuddy-db’, o imagine mysql.

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’ creata în interiorul instanței conține următoarele tabele pentru organizarea datelor:

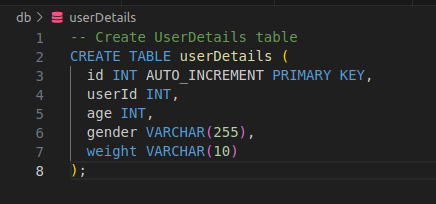
* users

Fig 7. Tabela ‘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:

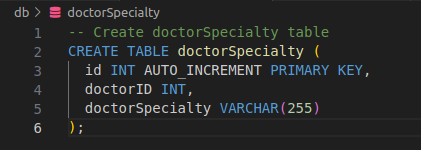
* ‘id’ = cheia primară a acestei tabele și numărul unic de identificare a fiecărui utilizator
* ‘fullName’ = numele complet al utilizatorului
* ‘email’ = adresa de email al utilizatorului
* ‘phoneNumber’ = numărul de telefon al utilizatorului
* ‘password’ = parola cu care acesta se autentifică în aplicație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

Fig 8. Tabela ‘userDetails ‘

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

Câmpurile reprezintă următoarele lucruri:

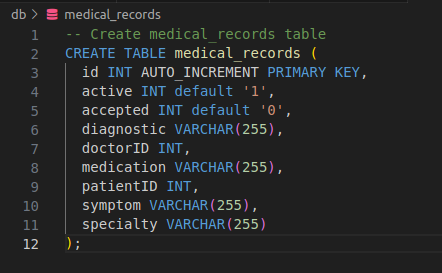
* ‘id’ = cheia primară a acestei tabele
* ‘userId’ = ID-ul prin care este identificat pacientul în tabela ‘users’
* ‘age’ = vârsta utilizatorului
* ‘gender’ = sexul utilizatorului
* ‘weight’ = greutatea utilizatorului
* doctorSpecialty

Fig 9. Tabela ‘doctorSpecialty’

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

Câmpurile reprezintă următoarele lucruri:

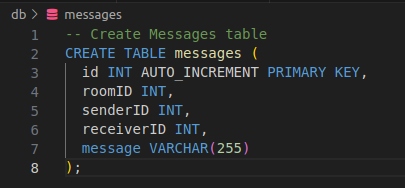
* ‘id’ = cheia primară a acestei tabele
* ‘doctorID’ = ID-ul prin care este identificat medicul în tabela ‘users’
* ‘doctorSpecialty’ = specialitatea medicului
* medicalRecords

Fig 10. Tabela ‘medicalRecords’

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

Câmpurile reprezintă următoarele lucruri:

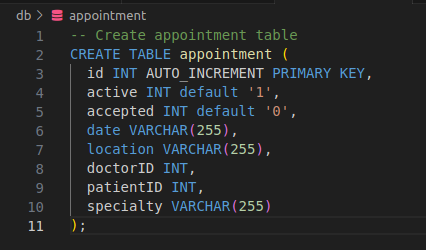
* ‘id’ = cheia primară a acestei tabele
* ‘active’ = statusul cererii din punct de vedere al activității, valoarea inițiala fiind 1, iar aceasta devine 0 în momentul în care medicul încheie perioada de tratament
* ‘accepted’ = statusul cererii din punct de vedere al validării, valoarea în momentul în 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 medicul specialist responsabil din momentul acceptării cererii
* ‘patientID’ = ID-ul pacientului care a creat cererea
* messages

Fig 11. Tabela ‘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

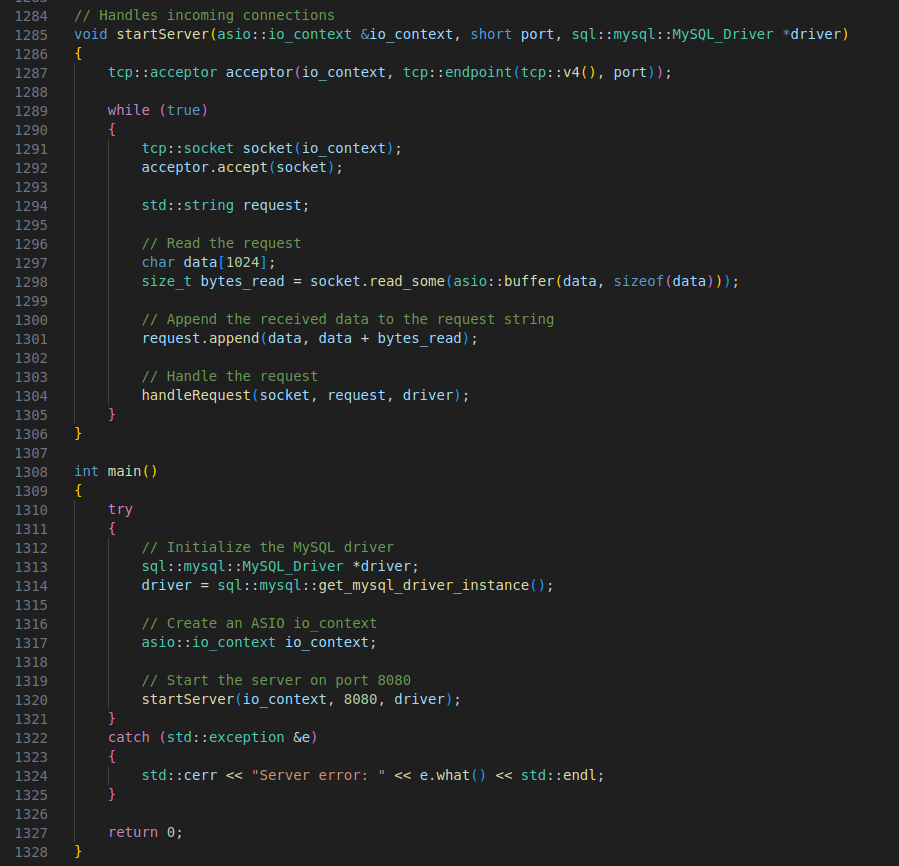
Fig 12. Tabela ‘appointment’

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 inițiala 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 medicul specialist responsabil din momentul acceptării cererii
* ‘patientID’ = ID-ul pacientului care a creat cererea

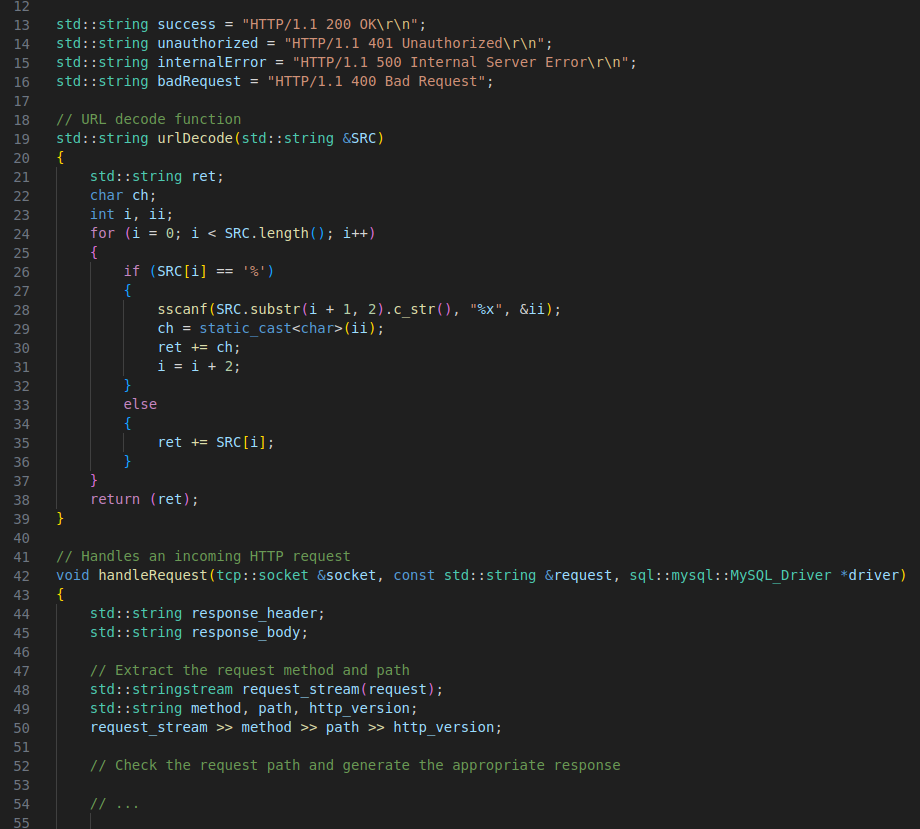
Fișierul C++ (server.cpp) în care a fost dezvoltat serverul pornește și este configurat astfel:

Fig 13. Pornirea fișierului C++

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 functia ‘startServer’ cu parametrii corespunzatori, 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.

Fig 14. Începutul funcției handleRequest

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.

Funcția urlDecode primește ca parametru un string pe care apoi îl returnează decodat. Avem nevoie de aceasta, deoarece API-ul creat în aplicația mobilă codifică parametrii trimiși în request-uri.

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 ușurința 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.

Fig 15. Procesarea request-ului „getName”

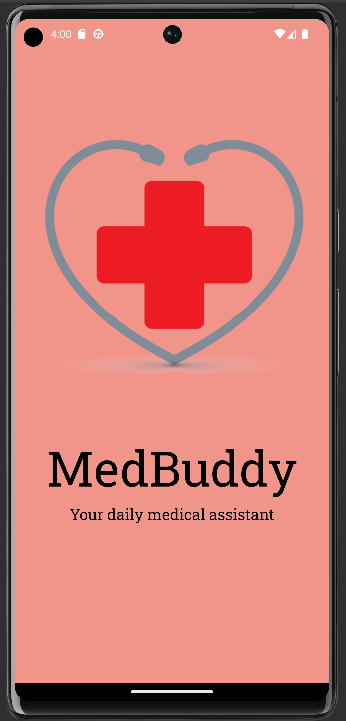
La început se iterează stream-ul ‘request\_stream’ pentru a ajunge la șirul care conține datele pentru procesare, în acest caz fiind doar id-ul utilizatorului, apoi se decodează acel șir și se extrage valoare id-ului.

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.

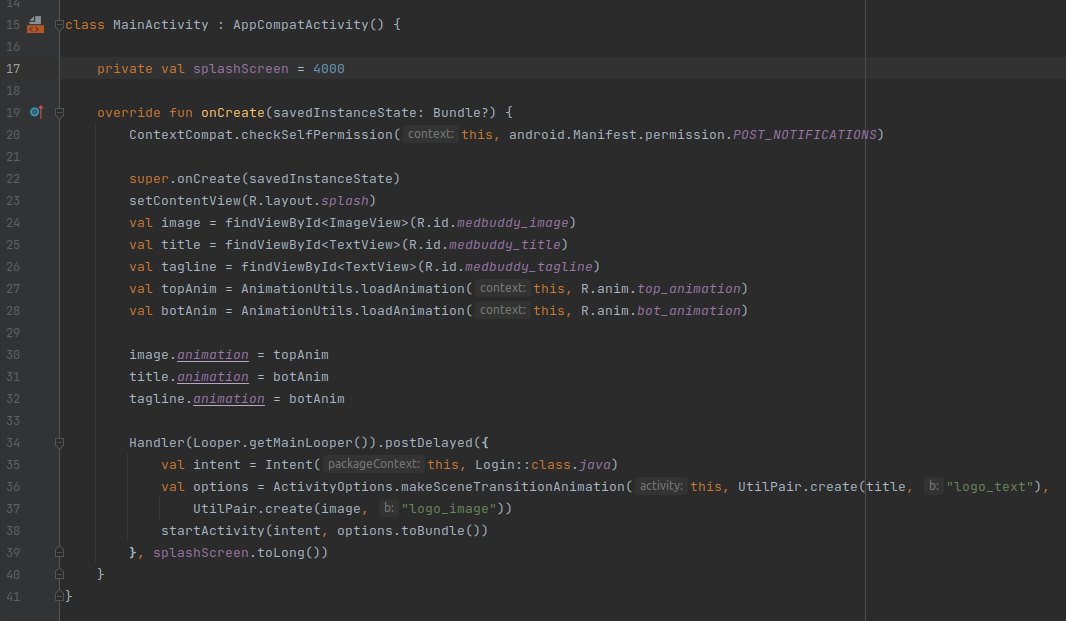
În această manieră se procesează fiecare tip de request trimis de către client, dar diferă numărul de interogări executate și de parametrii.

**6.2 Aplicația mobile**

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

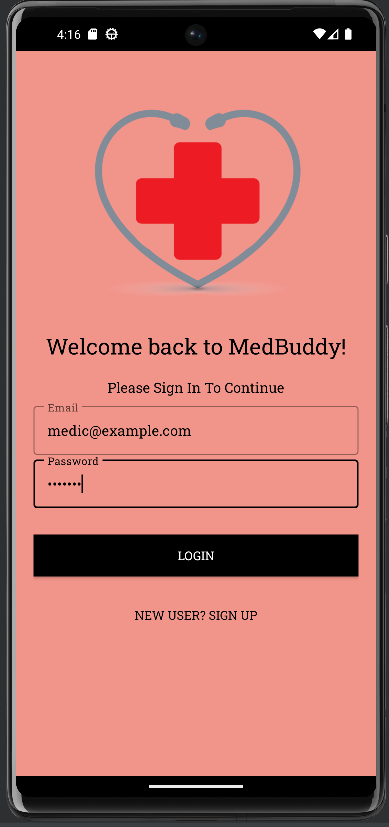
Fig 16. Splash Screen

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:

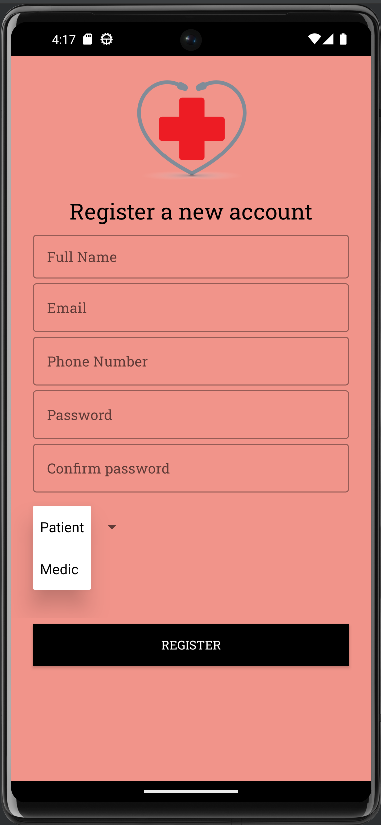
Fig 17. MainActivity

Pentru a îmbunătății design-ul de pornire a aplicației, am adaugat 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 inregistrare a unui cont nou, în cazul în care este vorba despre un utilizator nou.

Fig 18. Login Screen

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 doua variante, medic și pacient, care vor defini tipul contului și interacțiunea utilizatorului cu aplicația.

Fig 19. Register Screen

Utilizatorul este obligat să completeze toate câmpurile înainte sa 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:

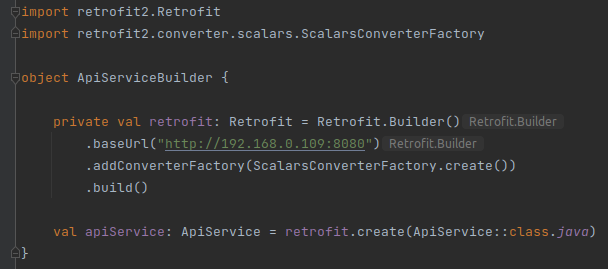
* „Email” – adresa de mail introdusă trebuie să fie valida, aceasta se verifica cu o expresie regulată
* „Phone Number” – numărul introdus trebuie sa conțină exact 10 cifre
* „Password” – parola introdusă trebuie să aibă o lungime de cel puțin 7 caractere
* „Confirm Password” – aceasta trebuie să coincidă 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.

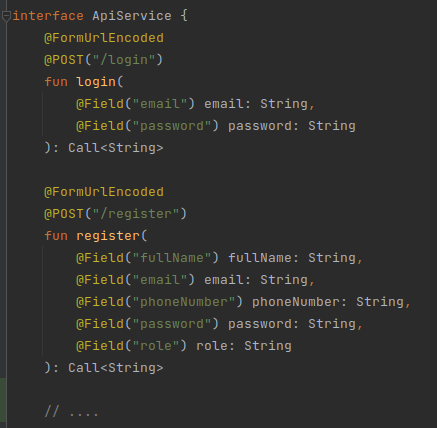
Fig 20. Logica înregistrării

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 sa apelăm o funcție din API-ul construit.

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

Fig 21. Retrofit builder

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

Fig 22. Exemplu apel API

Conținutul cererii trimise către server este codificat URL.