**Documentatie Frontend**

**Autori:** Viju Tudor-Alexandru

Sandu Alexandru

Cuprins

[1. Scopul proiectului 3](#_Toc156389987)

[2. Tehnologii Utilizate 4](#_Toc156389988)

[2.1. Android Studio 4](#_Toc156389989)

[2.2. Biblioteci Utilizate 4](#_Toc156389990)

[3. Fluxul de lucru al aplicatiei 5](#_Toc156389991)

[4. Structura Interna a aplicatiei 7](#_Toc156389992)

[4.1. Structura Generala 7](#_Toc156389993)

[4.2. Conectarea cu Aplicatia Server 8](#_Toc156389994)

[4.3. Multi – Threading 8](#_Toc156389995)

# 1. Scopul proiectului

Scopul proiectului este de a crea o aplicatie android de comunicare in privat cu mai multi utilizatori.

Aplicatia Android va oferi o interfata utilizatorului care ii permite comunicarea cu alti utilizatori. Aceasta aplicatie va comunica cu un server conectat la o baza de date.

Scopul bazei de date este de a stoca toate datele necesare pentru buna functionare a aplicatiei. Datele acestea contin atat informatii despre utilizator, cat si diverse atribute ale camerelor de comunicare, inclusive mesajele trimise prin intermediul acestora.

Aplicatia server este programul care faciliteaza conexiunea dintre utilizatori cu aplicatia si manipularea bazei de date. Detalii despre baza de date si aplicatia server sunt gasite in **Documentatia Backend**.

# 2. Tehnologii Utilizate

## 2.1. Android Studio

Android Studio este un mediu de dezvoltare integrat (IDE) construit specific pentru dezvoltarea aplicatiilor Android. Acest IDE este dezvoltat de Google, in cooperare cu JetBrains. Cei din urma, au oferit platform IDE-ului IntelliJ IDEA pentru a dezvolta interfata Android Studio. Din acest motiv se poate observa interfata indentica dintre cele doua IDE-uri.

Android Studio pune la dispositie si o multime de biblioteci care permit constructia aplicatiilor Android. Folosind Maven sau Gradle, dezvoltatorii de aplicatii Android au libertatea de a folosi si alte biblioteci specifice Java.

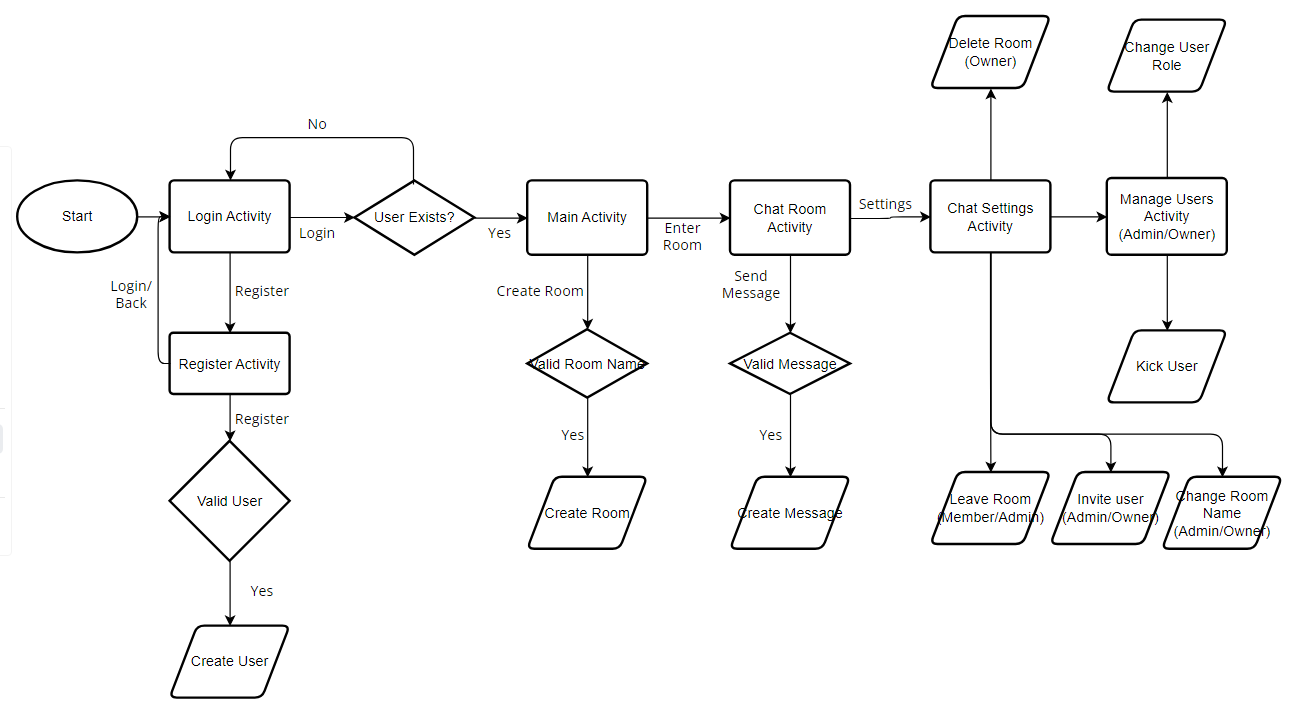
Cu toate ca aplicatiile Android Studio sunt dezvoltate prin limbajul de programare Java, IDE-ul permite si dezvoltarea prin limbajul Kotlin.

In cadrul acestui proiect am optat pentru Gradle si limbajul de programare Java.

## 2.2. Biblioteci Utilizate

In mare parte am utilizate biblotecile native Java cu cele oferite de Android Studio. Singura bibloteca speciala este cea care faciliteaza conexiunea cu aplicatia server prin SignalR.

# 3. Fluxul de lucru al aplicatiei



Aplicatia Android va introduce utilizatorul intr-o pagina de log in. Acesta are posibilitatea de as crea un cont, daca acesta nu are inca unul creat.

Un utilizator este definit prin nume de utilizator, email si parola. Numele de utilizator este un sir de caractere care va fi utilizat pentru a diferentia utilizatorii in interiorul aplicatiei Android. Emailul este tot un sir de caractere care este folosit pentru autentificare utilizatorului. Din acest motiv, doi utilizatori nu pot exista sub acelasi E-mail, insa pot avea acelasi nume de utilizator. De asemenea, este obligatoriu ca E-Mailul sa fie valid din punct de vedere sintactic (exemplu: [app.user231@domain.org](mailto:app.user231@domain.org)). Parola este tot un sir de caractere, utilizat strict pentru autentificarea unui utilizator.

Dupa autentificare valida, utilizatorul va intra in fereastra de baza. Aceasta fereastra permite intrarea intr-o camera de comunicare, crearea unei asemenea camera, deconectarea de la contul curent si reactualizarea camerolor disponibile.

Intrarea intr-o camera de comunicare va schimba din nou interfata. Aici se vor putea vizualiza toate mesajele trimise te toti mebrii ale acestei camere. Tot aici se pot trimite mesaje noi in camera. De asemenea exista si un buton de setari.

La apasarea acestui buton de setari se va deschide o noua interfata. Depinzand de rolul utilizatorului in camera, acesta va avea un numar variat de optiuni. Utilizatorii pot fi Membri, Admini sau Owner. Ownerul este unic pe camera si este cel care creaza camera. Acesta nu isi poate modifica rolul sau iesi din camera. De asemenea, membrii pot deveni admini si vice versa, dar nu owner.

# 4. Structura Interna a aplicatiei

## 4.1. Structura Generala

Clasele din intermediul aplicatiei sunt impartite pe categorii: activitati, modele, adaptoare si clase utilitare.

Clasele activitate sunt cele mai importante clase in proiectul Andoid de oarece ele definesc logica interfetei grafice si a conexiunii cu alte parti din aplicatie. Aceste clase extrag datele de design grafic salvate in fisiere xml si le conecteaza la obiecte definite in interiorul clasei pentru a putea fi manipulate.

Clasele model definesc structura obiectelor din clasa de date. Aceste modele sunt folosite pentru a putea comunica mai usor cu aplicatie sarver, dar si pentru a putea afisa mai usor cantitati mari de informatii, cum ar fi mesajele din camerele de comunicare.

Clasele adaptor sunt clase speciale care mostenesc un obiect de timp ArrayAdapter. Aceste clase sunt folosite pentru a defini functionalitatea unei liste complexe de obiecte salvate intr-o variabila ListView. In mod implicit, un obiect ListView nu are nevoie de un adaptor definit, ci poate utiliza un sablon predefinit de android studio. In cele mai multe aceste sabloane sunt mult prea simple. Pentru a facilita functionarea unor liste care contin elemente complexe, se utilizeaza aceste clase Adapter.

Clasele utilitare sunt clase ajutatoare care pot fi folosite in diverse parti ale proiectului. Acestea sunt in general simple si sunt folosite pentru a apela o functie des folosite in mai multe parti diferite ale codului.

## 4.2. Conectarea cu Aplicatia Server

Comunicarea intre aplicatia Android si Server se face prin SignalR. Din acest motiv trebuie mai intai creat un obiect HubConnection care permite aceasta comunicare.

In aplicatie se poate gasi clasa utilitara HubConnector care gestioneaza obiectul HubConnection. Aceasta clasa permite pornirea, respectiv, oprirea conexiunii si utilizarea HubConnection-ului pnetru a cere si primi informatii de la server. Pentru a face cereri de la server se apeleaza functiaa invoke pe obiectul HubConnection.

De notat este contructorul clasei HubConnector in care se seteaza calea URL de baza catre endpointurile programate pe aplicatie server. Aceasta cale este esentiala pentru a putea avea succes in comunicarea dintre cele doua aplicatii.

## 4.3. Multi – Threading

Comunicarea dintre aplicatia Android si splicatie Server este in general rapida, dar nici o data nu instanta. De asemenea, daca aplicatia server se intampla sa fie inchisa, sau conexiunea de internet de pe telefonul mobil este pierduta, atunci incercarea de comunicare fa esua dupa cateva secunde de incercare. Pe parcursul acestor asteptari, aplicatia Android ar avea un efect de „inghet” al procesului. Utilizatori ar presupune ca aplicatia a esuat, cand aceasta doar asteapta un raspuns de la server.

Pentru a evita efectul enuntat anterior, am optat pentru utilizarea firelor de executie multiple (multi - threading). Aceste fire de executie vor fi utilizate de fiecare data cand se cer date de la aplicatia server.

In acest caz am optat pentru utilizarea Executorilor din Java. Unde este nevoie de un fir de executie separat, se poate crea un obiect de tip ExecutorService pe un sigur fir, care executa condul care face o cerere de la server. La final, serviciul executor poate fi inchis. Astfel, toate invocarile la aplicatia server pot fi facute fara ca utilizatorul aplicatiei web sa intalneasca efectul de „inghet”.

O alta utilizare a firelor de excutie multiple, pote fi gasita exclusiv in clasa MainActivity. Amintesc ca am oferit utilizatorului posibilitatea de a reactualiza lista de camere valibile acestuia manual. Cu toate acestea am indrodus si o apelare automata a acestei funtii de reactualizare, fara ca utilizatorul sa intervina. Am dobandit acest efect prin utilizarea unui ofiect Runnable si a unui Handler. Obiectul runnable este o simpla functie care ruleaza cod (mai nou inlocuit de expresii lambda), in cazul acesta, reactualizarea listei de camere. Obiectul handler va rula un obiect runnable la un interval de timp pe un fir de executie diferit. Astfel am obtinut efectul de reactualizare automata la un interval de 30 de secunde.

Este de mentionat ca Android Studio functioneaza pe o logica care utilizeaza implicit pe mai multe fire de executie. Unul din aceste fire de executie este dedicat schimbarii elementelor grafice ale aplicatiei. Pentru face schimbari folosind acest fir de executie, se foloseste functia runOnUIThread care accespta o functie lambda. Aceasta functie este necesara in special cand se doresc modificari facute pe interfata grafica, atunci cand informatiile necesare modificarilor se afla intr-un fir de executie diferit. Un caz frecvent in care s-a folosit acest fir de executie este in interiorul unui Executor.