**Universitatea “Politehnica” Timişoara**

**Facultatea de Automatică şi Calculatoare**

**PROIECT DE DIPLOMĂ**

**Aplicatie android destinata procesului de inscriere la Facultatea de Automatica si Calculatoare**

**Coordonator diplomă: Student:**

As. Dr. Ing. **ANA-MARIA DAN BALINTONI DUMITRU-BENIAMIN**

**– TIMIŞOARA 2019 –**

**Universitatea “Politehnica” Timişoara**

**Facultatea de Automatică şi Calculatoare**

**Cuprins**

Abrevieri și termeni...................................................................................................................3

**Capitolul 1:Introducere**

* 1. Tema lucrarii. Context.............................................................................................5

Tema…....................................................................................................................5

Motivatie…..............................................................................................................5

Obiective…..............................................................................................................6

* 1. Solutii existente........................................................................................................7
  2. Structura lucrarii de licenta......................................................................................9

**Capitolul 2: Arhitectura aplicației**.........................................................................................10

2.1: Sistemul de Operare Android.................................................................................13

2.1.1: Prezentare Generală...............................................................................................13

2.1.2: Versiuni Android.....................................................................................14

2.1.3: Arhitectura Android.............................................................................................15

2.1.4: Funcționalități Android............................................................................19

2.2: Cloud Computing....................................................................................................20

2.2.1: Caracteristici generale..........................................................................................20

2.2.2: GoogleCloud........................................................................................................22

2.2.3: Google Firestore...................................................................................................23

2.2.4: Autentificarea Firebase ........................................................................................26

**Capitolul 3: Implementarea aplicatiei**

3.1: Prezentarea Implementării........................................................................................28

3.2: Informați referitoare la organizarea aplicației...........................................................35

**Capitolul 4:Utilizarea aplicației**

4.1: Prezentarea Interfețelor.............................................................................................38

**Capitolul 5:Concluzii**

5.1: Probleme Întampinate................................................................................................44

5.2: Propuneri....................................................................................................................44

Refetințe........................................................................................................................................45

**Abrevieri și termeni**

*Android*: O platformă open source concepută pentru dispozitive mobile.

*API*: Interfața de programare a aplicațiilor este un program software care facilitează

interacțiunea cu alte programe software.

*AVD:* Un dispozitiv virtual Android este o configurație a emulatorului care face posibilă

crearea un dispozitiv real definind hardware și software

*IDE*: Un mediu integrat de dezvoltare este o aplicație software care oferă

resurse cuprinzătoare pentru programatorii de calculatoare destinate dezvoltării de software rapid și eficient. Un IDE constă dintr-un editor de cod sursă, instrumente de construire și un program de depanare.

*IaaS*: Infrastructura ca serviciu este un model de serviciu care asigură stocarea, infrastructura și alte resurse hardware ca resurse pe care clienții le pot furniza.

*OS*: Sistemul de operare este o componentă software de infrastructură de bază a unui sistem computerizat. Controlează toate operațiile de bază ale computerului.

*PaaS*: Platforma ca serviciu este un concept care descrie o platformă de calcul care este

inchiriat sau livrat ca solutie integrata, stiva de solutii sau servicii printr-o conexiune internet.

*SaaS*: Software-ul ca serviciu este un model pentru distribuția de software unde clienții accesează software-ul prin Internet.

*Smartphone:* Un smartphone este un telefon mobil cu caracteristici avansate, putere de calcul, interfață, internet, gps, accelerometru, senzori.

*SDK*: Un kit de dezvoltare software, acesta este un pachet de programare care permite unui programator să dezvolte aplicații pentru o anumită platformă. Acesta include API-uri, instrumente de programare și documentație.

*NoSQL*: Cuprinde o gamă largă de tehnologii baze de date diferite care au fost dezvoltate ca răspuns la cerințele prezentate în construirea aplicațiilor moderne.

*URL*: Este adresa unei resurse pe Internet.

*Wi-Fi*: este o tehnologie de rețea fără fir care utilizează undele radio pentru a opține conexiuni wireless internet de mare viteză și conexiuni de rețea.

*XML*: Este un limbaj de markup care definește un set de reguli pentru

codificarea documentelor într-un format care este atât auto-descriptiv, cât și usor de citit.

*3G*: Prescurtare pentru a treia generație, este a treia generație de telecomunicații mobile .

*4G*: Prescurtare de la a patra generație de comunicații de telefonie mobilă.

*Open source software*: Dezvoltatorii de software care au pus la dispoziție codul sursă al programului oferă drepturile de a studia, modifica și distribui software-ul pentru oricine și pentru orice scop.

*APK*: Pachetul aplicației Android

*GUI*: Interfața grafică cu utilizatorul.

AOSP: Referință la codul sursă al sistemului de operare Android open source software.

*GMS*: Servici mobile Google.

*BSD*: Este numele unei distribuții UNIX

SSL: Vine de la niveluri de socket securizare, se folosește petru protejarea datelor care circula prin internet.

Snapshot: the state of a system at a particular point in time.

**Capitolul 1:Introducere**

* 1. **Tema lucrarii. Context**

***Tema*** acestui proiect este o aplicație mobilă android care să fluidizeze accesul candidaților în sali special amenajate puse la dispoziție de Univesritatea Politehnică Timișoara destinate înscrieri la facultate.

Aplicatia utilizeaza cloud computing pentru a îmbunătăți capacitatea de calcul a dispozitivelor mobile. Puterea de procesare, capacitatea de stocare, bateria durata de viață și dimensiunea afișării reprezintă o preocupare pentru dezvoltatori atunci când se creează aplicații pentru dispozitive mobile, în plus, diversitatea sistemelor de operare de pe piață astăzi este una mare și în continuă dezvoltare, fiecare producător îsi aduce proprile modificări și optimizări asupra sistemului de bază, astfel încât apare nevoia de a crea aplicații cross-platform și cloud computing oferă o soluție pentru această necesitate.

Cloud computing-ul devine o platformă dominantă de calcul pentru furnizare

servicii scalabile către o bază globală de clienți. Prin urmare, a fost folosită o platformă de cloud computing pentru a dezvoltă o aplicație mobilă care să accepte servicii bazate pe cloud.

***Motivația*** pentru dispozitivele mobile care au devenit o parte esențială a vieții noastre de zi cu zi. De la a fi un dispozitiv utilizat în principal pentru apeluri telefonice și mesaje text, telefonul mobil de astăzi a devenit un dispozitiv multifuncțional. Vânzările de smartphone-uri depășesc acum cu mult vânzările de sisteme desktop.

Îmbunătățirea rețelelor, planuri tarifare atractive, telefoane accesibile și disponibilitatea serviciilor îmbunătățite, servicii de date cum ar fi aplicații mobile, plăți mobile și internet mobil de mare viteză au influențat această creștere remarcabilă. Pe măsură ce popularitatea dispozitivului mobil crește, necesitatea utilizatorul final de a rula aplicații mai complexe sunt de asemenea în creștere.

Nevoia de a menține avantajele dispozitivelor mobile asupra sistemelor de tip desktop și laptop cum ar fi greutatea, dimensiunea și autonomia dispozitivului va dicta întotdeauna anumite limite de procesare, puterea, capacitatea de stocare, durata de viață a bateriei și dimensiunea afișajului.

Dezvoltatorii trebuie să reprogrameze aplicațiile desktop standard pentru a rula pe platforme hardware mobile, ceea ce ar putea necesita reducerea funcționalității, în timp ce aplicațiile mai exigente au nevoie de resurse hardware specifice care este puțin probabil să fie disponibile pe dispozitivele mobile. În ultimii ani, smartphone-urile au dovedit că sunt foarte capabile, acestea sunt acum aproape la fel de puternice ca un desktop sau laptop. Cu toate acestea, există aplicații care necesită mult mai multă putere de procesare și Cloud computing poate îmbunătăți capacitatea de calcul disponibilă pentru utilizatorii de telefonie mobilă prin susținerea aplicațiilor care necesită o capacitate mare de stocare, hardware grafic, cum ar fi mediile virtuale 3D sau alte servicii cloud indisponibile pe dispozitivele mobile.

Oferta de cloud computing mobile reprezinta o abordare care să răspundă cerințelor de creștere a funcționalității utilizatorilor. Cu toate acestea, durată de viață scurtă a bateriei, condiții diferite de retea wireless și latență de interacțiune sunt provocări pentru unele aplicații cloud pe dispozitive mobile.

Acest proiect face obiectul dezvoltării unei aplicații Android care sa faciliteze accesul candidaților la înscrierea pentru facultate. În zilele noastre, gadgeturile învart lumea, mulți oameni nu își pot imagina nici măcar o zi fără dispozitivul mobil preferat. Le folosim pentru orice: găsi informații, pentru a rămâne conectați cu noi prieteni și familia, pentru a găsi drumul cel mai scurt, pentru a decide ce să facem și multe alte lucruri.

Dar, foarte des, ajungem la punctul în care ne dorim să avem o aplicatie pentru o situație particulară sau pentru o anumită nevoie, dar nu există o astfel de aplicație.

Una dintre situațiile în care ne dorim acest lucru face obiectul aceste-i lucrări, și anume ne dorim să nu mai stăm la coadă, timpul fiecăruia este important, am urmărit pe parcursul mai multor ani procesul de înscriere la facultate care sa dovedit un proces foarte costisitor din punct de vedere al timpului, de accea avem nevoie de o soluție mobila inteligentă care folosește tehnologi de ultimă generatie pentru a facilita procesul de inscriere prin gestionarea cozilor.

***Obiectivul*** este acela ca procesul de înscriere să fie unul cât mai rapid și facil, cu ajutorul candidaților care vor primi un tichet virtual cu un numar, informații în timp real despre cozile pentru înscriere cât și un timp aproximativ, aceștia urmând să primeasca un mesaj de tip notificare atunci când se apropie rândul lor la coada pentru înscriere.

**1.2 Solutii existente**

Soluţia *ZeroCozi [1]*

Doi ingineri IT din Timişoara au dezvoltat o aplicaţie, care îşi propune să elimine  aglomerația și cozile de la ghișee.

Soluţia ZeroCozi poate fi folosită atât în mediul privat, cât şi în cel public, chiar de către administraţiile locale, dă asigurări unul dintre dezvoltatori, Radu Jakab.

Avem un server central care poate să gestioneze aceste cozi, avem în continuare acele aparate care ne dau bilet. Tot sistemul acesta poate să fie integrat și cu o interfață web cu telefoanele noastre mobile și atunci putem să ne luăm biletele respective de la birou, de acasă sau din mașină. El ne poate informa câți oameni sunt în coadă ca să avem o așteptare corectă vizavi de ceea ce vom găsi acolo”, explică Radu Jakab.

Dezvoltatori spun că mai multe instituţii publice din Timişoara, Reşiţa şi Hunedoara sunt interesate de noul sistem.

Soluţia *Skiplino [2]*

Promite că revoluționează gestionarea cozilor.

Skiplino este un sistem de gestionare a cozilor de așteptare care permite companiilor să gestioneze cozile de clienți inteligent și rapid.

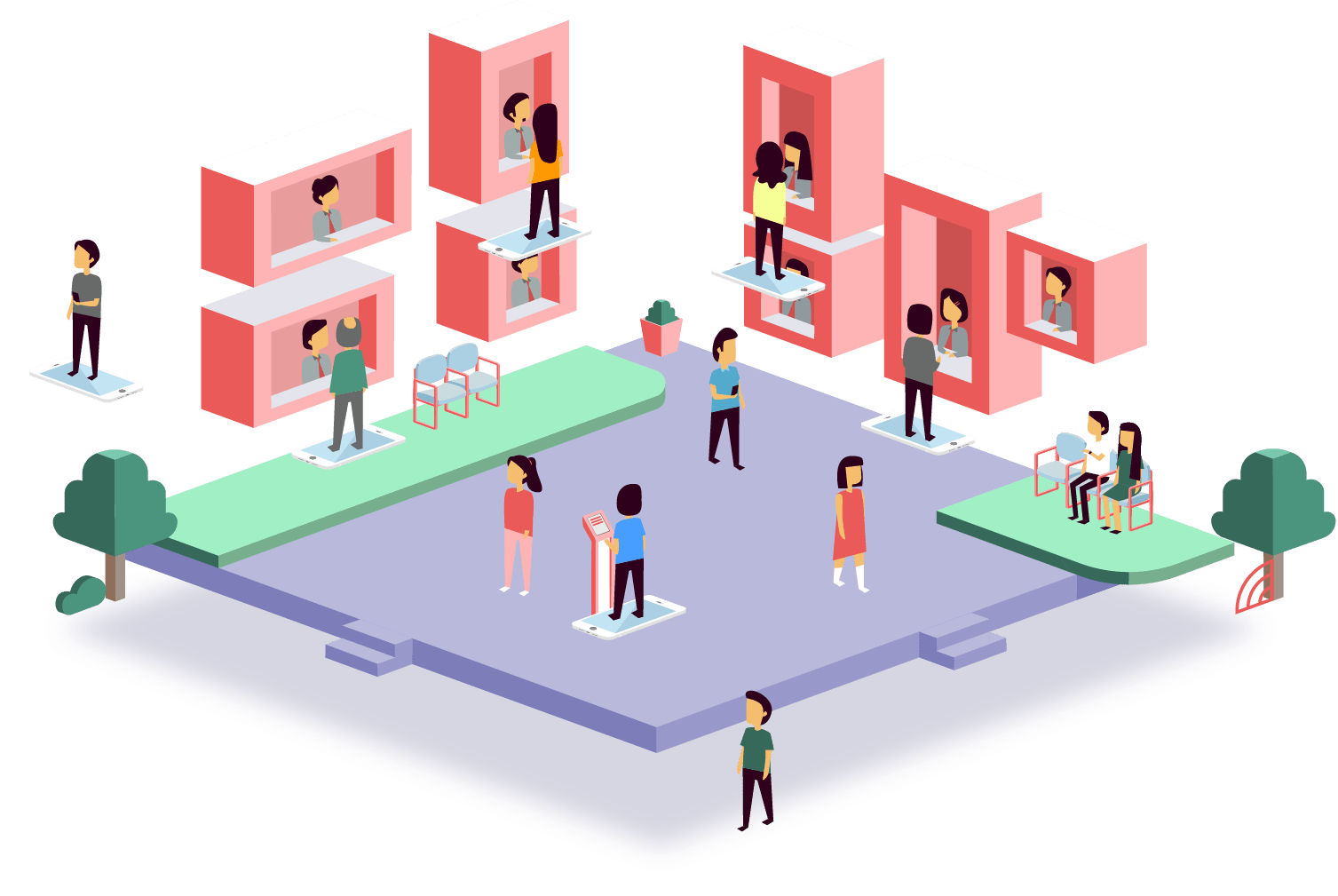


Fig. 1.1 Skiplino-[3]

Skiplino este un sistem inteligent și bazat pe cloud, care poate monitoriza datele legate de cozi în timp real și colectează feedback-ul clientului. Software-ul nostru bazat pe cloud poate apoi să evalueze aceste date pentru a accelera performanța agenților și a serviciilor dvs.

Cum funcționează: cunoaște cine vine, sistemul de management al cozilor Skiplino înregistrează clienții și serviciile pe care intenționează să le utilizeze.

Gestionează traficul, platforma analitică de gestionare a cozilor bazată pe cloud permite monitorizarea numărului de vizitatori care așteaptă în fiecare ramură, împreună cu informații anterioare despre aceștia și preferințele acestora.



Fig. 1.2 *Go-digital* -[4]

Soluţia *Go-digital [5]*

Sistemele sunt compuse în general din afișaje digitale pentru alegerea opțiunilor și eliberarea biletului de ordine, sisteme de îndrumare și gestionare complexe, software.

* **Kiosk Smart**

Acest tip de Kiosk este gândit pentru a face față oricărei cerințe cu privire la gestionarea cozilor de așteptare. Cu un design simplu și elegant acest sistem este ușor de folosit, accesibil oricărui tip de persoană (incluzând persoanele cu dizabilități). Kioskul poate fi adaptat în funcție de nevoile afacerii în cauză.

* **Kiosk Compact**

Acest tip de Kiosk este recomandat sediilor care au probleme de spațiu. Kioskul este unul compact, extrem de usor de configurat sau relocat. Ocupă un spațiu foarte mic și este extrem de eficient și fiabil.

* **On-line Booking**

Clientul poate să rezerve o ședință, audiență etc. în avans cu ajutorul modulului de booking oferit de softul pentru gestiunea cozilor de așteptare.

* **Alerta SMS**

Dacă timpul de așteptare depășește un anumit interval, clientul poate introduce numărul de telefon, iar aplicația îl va notifica în momentul în care îi vine rândul.

* **Monitoare tip Digital signage**

Toate soluțiile mai sus menționate pot fi conectate în mod automat către monitoarele dispuse în sălile de așteptare. Pe lângă ordinea de intrare, acestea pot fi personalizate astfel încât relaționarea cu clienții să fie mult mai dinamică prin reclame personalizate.

* **Sistem portabil**

Firma oferă un sistem dinamic și personalizat activităților ce se desfășoară în afară sediului (evenimente, expoziții, workshopuri).

**1.3 Structura lucrarii de licență**

Capitolul 2: Arhitectura aplicație.

În cadrul acestui capitol se vor prezenta diagrame atât de activitate cât și de stare, se va urmări prezentarea componentelor software folosite cât și a tehnologiilor, la nivel arhitectural, se va evidenția modul prin care aceste componente comunică una cu cealaltă.

Capitolul 3: Implementarea aplicației

În cadrul acestui capitol se va evidenția implementarea la nivel de cod sursă, se vor prezenta clasele, evenimentele, conexiunile și design-ul aplicației.

Capitolul 4: Utilizarea aplicatie

În cadrul acestui capitol se va prezenta interfața aplicației și felul în care aceasta se comportă în funcție de tipul de utilizator .

Capitolul 5: Concluzii

În cadrul acestui capitol se vor prezenta concluziile luate în urma realizării aplicației, probleme întâmpinate, modificări, constrângeri, decizii importante.

**Capitolul 2: Arhitectura aplicației**

***Diagrama de secvență*** pentru modelarea interacțiunii generale.

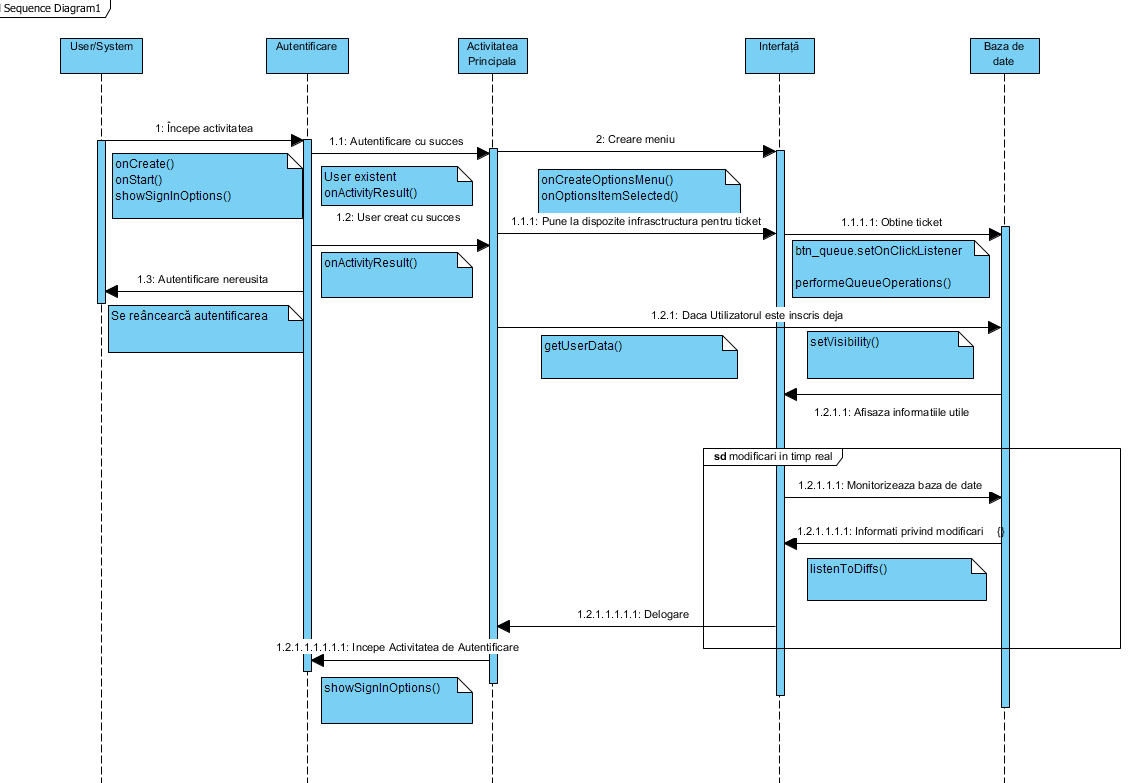
Reprezintă un model de interacțiune la nivel înalt între obiectele active dintr-un sistem reprezentativ pentru fiecare caz de utilizare în parte.

Fig 2.1 – Diagrama de secvență

Pentru aplicația mobilă se foloșeste sistemul Android prezentat în detaliu la *paragraful 2.1.* Aplicația a fost scrisă în mediul de dezvoltare *Android Studio,* în limbajul de programare *Kotlin.*

Android Studio este mediul oficial de dezvoltare integrat (IDE) pentru dezvoltarea aplicațiilor Android, bazat pe IntelliJ IDEA. Pe lângă editorul puternic de cod și instrumentele de dezvoltare IntelliJ, Android Studio oferă chiar și mai multe caracteristici care îmbunătățesc productivitatea atunci când construiți aplicații Android, cum ar fi:

* Un sistem flexibil de construcție bazat pe Gradle.
* Un emulator rapid și bogat în caracteristici.
* Un mediu unificat în care se pot dezvolta pentru toate dispozitivele Android.
* Instant Run pentru a împinge modificările în aplicația în desfășurare fără a construi un nou APK.
* Suport încorporat pentru Google Cloud Platform.

Cu ajutorul acestui IDE inteligent am dezvoltat o aplicație care poate rula pe marea majoritatea a dispozitivelor mobile android, datorită faptului că este concepută pentru a fi compatibilă cu dispozitive de la un nivel SDK nouăsprezece pâna la ultimul nivel de SDK existent. Acest lucru a fost posibil datorita tehnologiilor de ultimă generație puse la dispoziția dezvoltatorilor de aplicații de către serviciile de tip Cloud, acestea fiind prezentate la *paragraful 2.2.*

Autentificarea în cadrul aplicației a fost realizata tot cu o tehnologie pusă la dispoziție de servicile de tip Cloud ale Google și anume Autentificarea Firebase, care expune o interfață facilă și usor de folosit și care dispune de multe opțiuni de autentificare, tehnologia este prezentată în cadrul *paragrafului 2.2.3.* Diagrama deactivitatea a procesului de autentificare este prezentată în *figura* 2.2.

Aceste tehnologi fac posibilă dezvoltarea aplicației fără a fi nevoie de implemantare pe partea de server.

Aplicația comunică cu baza de date și o monitorizează în timp real. Pentru dezvoltarea aplicației a fost folosită o tehnologie de stocare de tip Cloud pusă la dispoziție de Google, Firestore este prezentat în paragraful 2.2.3.

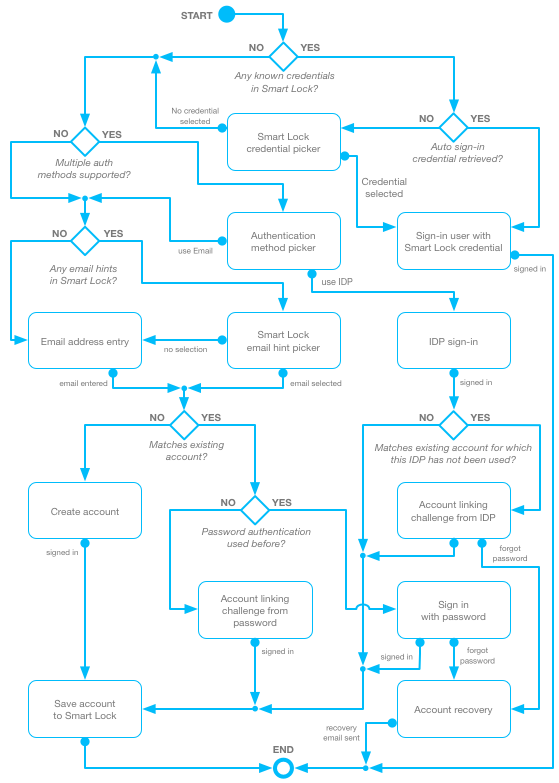


Fig. 2.2 -Diagrama de activitate pentru Autentificare -[6]

**2.1: Sistemul de Operare Android**

**2.1.1 Prezentare Generală**

Android este un OS mobil bazat pe o versiune modificată de Linux (pentru gestiunea componentelor hardware, a proceselor și a memoriei) și biblioteci Java (pentru telefonie (audio/video), conectivitate, grafică, programarea interfețelor cu utilizatorul).

Este un produs open-source (putând fi dezvoltat de producătorii de dispozitive mobile cu extensii proprietare pentru a-și particulariza platforma), dezvoltat în prezent de compania Google, conceput pe ideea transformării dispozitivelor mobile în adevărate mașini de calcul.

Google încearcă totuși să realizeze tranziția de la AOSP către GMS peste care sunt construite cele mai multe aplicații, în încercarea de a-și apăra acest proiect în fața concurenței. În acest sens, a fost dezvoltat **proiectul Google One**, prin care este oferit un set de specificații (ecran de 4.5 inchi - 845×480 pixeli, procesor quad-core, memorie 1GB RAM, spațiu de stocare 4GB, suport pentru dual sim) pe care producătorii de dispozitive mai ieftine trebuie să le respecte astfel încât acestea să fie compatibile cu un sistem Android, fără a întâmpina probleme de performanță.

Acesta include toate aplicațiile și serviciile Google, la care se pot adăuga și altele, furnizate de producător sau operatorul de telefonie mobilă. În acest fel, se asigură calitatea (păstrând renumele Android) și controlul asupra veniturilor.

Comunitatea Android este în creștere, și are tot mai multă susținere deoarece cei mai importanți producători de telefonie mobilă aleg să folosească sistemul decât să investească în propriile lor sisteme.

În condițiile în care pe piața dispozitivelor mobile aplicațiile sunt cele care aduc avantajul competițional, beneficiul Android este reprezentat de abordarea unitară pentru dezvoltarea aplicațiilor. Cu alte cuvinte, o aplicație dezvoltată conform API-ului Android va putea rula pe mai multe dispozitive mobile pe care este instalat sistemul de operare respectiv.

**2.1.2 Versiuni Android**

| Nume de cod | Numărul versiunii | Data de lansare inițială | API | Patch-uri de securitate |
| --- | --- | --- | --- | --- |
| Pie | 9.0 | 6 august 2018 | 28 | Suportat |
| Oreo | 8.0 – 8.1 | 21 august 2017 | 26 – 27 | Suportat |
| Nougat | 7.0 – 7.1.2 | 22 august 2016 | 24 – 25 | Suportat |
| Marshmallow | 6.0 – 6.0.1 | 5 octombrie 2015 | 23 | Suportat |
| Lollipop | 5.0 – 5.1.1 | 12 noiembrie 2014 | 21 – 22 | Suportat |
| KitKat | 4.4 – 4.4.4 | 31 octombrie 2013 | 19 – 20 | Suportat |
| Jelly Bean | 4.1 – 4.3.1 | 9 iulie 2012 | 16 – 18 | Nesuportat |
| Ice Cream Sandwich | 4.0 – 4.0.4 | 18 octombrie 2011 | 14 – 15 | Nesuportat |
| Honeycomb | 3.0 – 3.2.6 | 22 februarie 2011 | 11 – 13 | Nesuportat |
| Gingerbread | 2.3 – 2.3.7 | 6 decembrie 2010 | 9 – 10 | Nesuportat |
| Froyo | 2.2 – 2.2.3 | 20 mai 2010 | 8 | Nesuportat |
| Eclair | 2.0 – 2.1 | 26 octombrie 2009 | 5 – 7 | Nesuportat |
| Donut | 1.6 | 15 septembrie 2009 | 4 | Nesuportat |
| Cupcake | 1.5 | 27 aprilie 2009 | 3 | Nesuportat |
| (Fără nume de cod) | 1.0 | 23 septembrie 2008 | 1 | Nesuportat |
| (Cunoscut intern ca "Petit Four") | 1.1 | 9 februarie 2009 | 2 | Nesuportat |

Fig. 2.3 – Versiuni Android

Pentru identificarea versiunilor se folosesc, de regulă, trei sisteme:

* un număr, ce respectă formatul major.minor[.build], desemnând dacă modificările aduse sunt substanțiale sau reprezintă ajustări ale unor probleme identificate anterior.
* un nivel de API (același putând grupa un număr de mai multe versiuni), prin care se indică funcționalitățile expuse către programatori.
* o denumire, având un nume de cod inspirat din lumea dulciurilor; termenii respectivi încep cu inițiale care respectă ordinea alfabetică.

În momentul în care se ia decizia cu privire la versiunea pentru care se dezvoltă o aplicație Android, trebuie avute în vedere și cotele de piață ale dispozitivelor mobile.

Dezvoltarea unei aplicații Android pentru cea mai nouă versiune are avantajul de a se putea utiliza cele mai noi funcționalități expuse prin API.

Dezvoltarea unei aplicații Android pentru cea mai veche versiune are avantajul unei adresabilități pe scară largă.

Un compromis în acest sens poate fi obținut prin intermediul **bibliotecilor de suport**, dezvoltate pentru fiecare versiune, prin intermediul cărora pot fi utilizate la niveluri de API mai mici funcționalități din niveluri de API mai mari (în limita capabilităților dispozitivului mobil respectiv). Utilizarea acestora reprezintă o practică recomandată în dezvoltarea aplicațiilor Android.

**2.1.3 Arhitectura Android**

Arhitectura sistemului de operare Android cuprinde cinci secțiuni grupate pe patru niveluri:

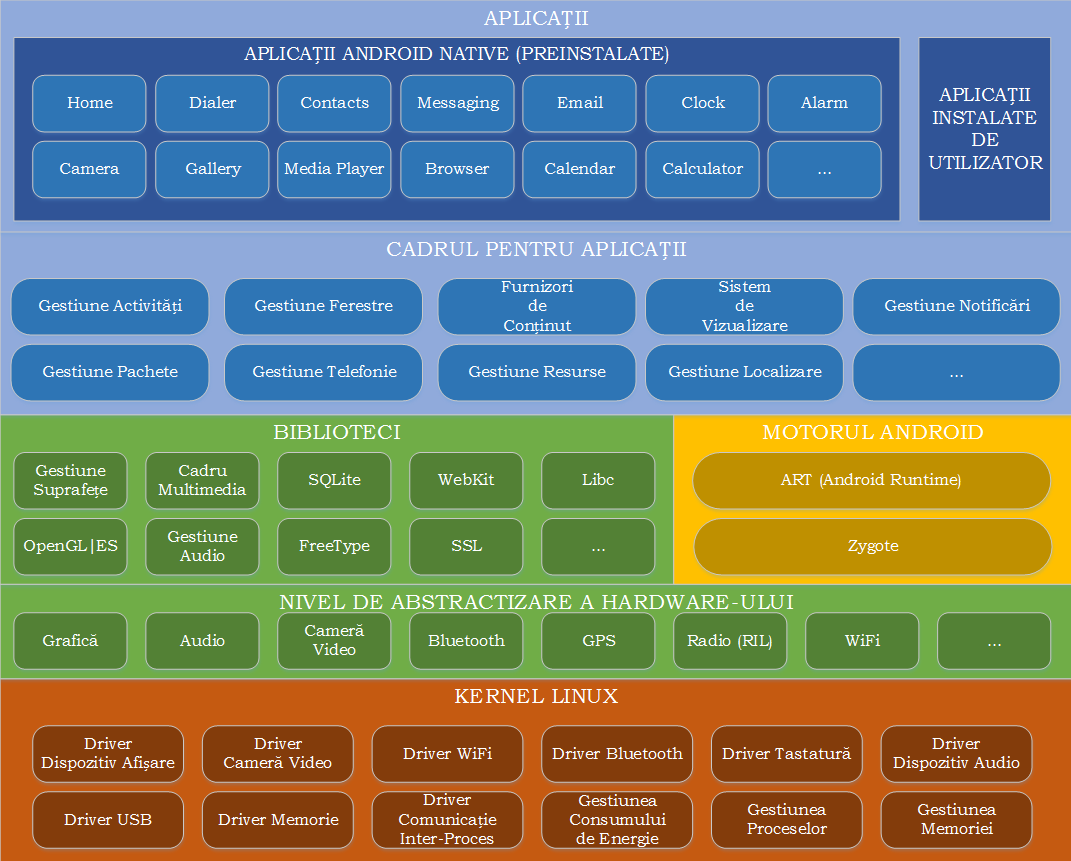
1. **Kernelul Linux** (cu unele modificări) conține driver-ele pentru diferitele componente hardware (ecran, cameră foto, tastatură, antenă WiFi, memorie flash, dispozitive audio), fiind responsabil cu gestiunea proceselor, memoriei, perifericelor (audio/video, GPS, WiFi), dispozitivelor de intrare/ieșire, rețelei și a consumului de energie; de asemenea, au fost implementate și unele îmbunătățiri:
   1. **Binder**, sistemul de comunicație inter-proces, a fost adaptat, întrucât reprezintă mediul de comunicație principal dintre aplicații și sistemul de operare, inclusiv funcțiile (serviciile) dispozitivului mobil; expunerea sa este realizată prin intermediul AIDL (Android Interface Definition Language) prin care pot fi manipulate obiecte transformate în primitive utilizate la comunicația propriu-zisă dintre aplicații și sistemul de operare;
   2. **Logger**, sistemul de jurnalizare, este esențial în cazul în care trebuie realizată depanarea aplicațiilor, în special pentru a detecta anumite situații particulare (informații cu privire la rețea, senzori); acesta este capabil să agrege datele provenite atât de la aplicația propriu-zisă cât și de la sistemul de operare, datele fiind disponibile prin intermediul unor utilitare specializate;
   3. sistemul prin intermediul căruia se previne transferul sistemului de operare într-o stare de latență (**wake locks**), în care consumul de energie este redus, întrucât se blochează execuția oricărei aplicații; utilizarea unui astfel de mecanism trebuie realizată cu precauție, întrucât poate determina epuizarea bateriei;
   4. sistemul de **alarme** oferă posibilitatea ca anumite sarcini să fie planificate la anumite momente de timp, putând fi executate, chiar dacă sistemul de operare se găsește într-o stare de latență;
   5. **Viking Killer** este un mecanism prin care sistemul de operare revendică memoria utilizată, atunci când nivelul acesteia atinge un anumit prag (aplicațiile Android care au fost rulate anterior sunt de regulă stocate în memorie pentru a se putea comuta rapid între ele, de vreme ce încărcarea în memorie este o operație costisitoare);
   6. **YAFFS2** (Yet Another Flash File System) este un sistem de fișiere adecvat pentru cipuri flash bazate pe porți NAND; platforma Android este stocată pe mai multe partiții, ceea ce îi conferă flexibilitate la actualizări, împiedicând modificarea sa în timpul rulării *(/boot* - conține secvența de pornire, /*system* - stochează fișierele de sistem și aplicațiile încorporate, /*recovery* - deține o imagine din care se poate restaura sistemul de operare, /*data* - include aplicațiile instalate și datele aferente acestora, /*cache* - utilizată pentru fișiere temporare, folosind memoria RAM, pentru acces rapid).
2. **Bibliotecile** (user-space) conțin codul care oferă principalele funcționalități a sistemului de operare Android, făcând legătura între kernel și aplicații. Sunt incluse aici motorul open-source pentru navigare WebKit, biblioteca FreeType pentru suportul seturilor de caractere, baza de date SQLite utilizată atât ca spațiu de stocare cât și pentru partajarea datelor specifice aplicațiilor, biblioteca libc (Bionic), biblioteca de sistem C bazată pe BSD și optimizată pentru dispozitive mobile bazate pe Linux, biblioteci pentru redarea și înregistrarea de conținut audio/video (bazate pe OpenCORE de la PacketVideo), biblioteci SSL pentru asigurarea securității pe Internet și Surface Manager, bibliotecă pentru controlul accesului la sistemul de afișare care suportă 2D și 3D. Aceste biblioteci nu sunt expuse prin API, reprezentând detalii de implementare Android.
3. **Motorul Android** rulează serviciile de platformă precum și aplicațiile care le utilizează, fiind reprezentat de:
   1. **ART (Android Runtime)** este mașina virtuală Java care a fost implementată începând cu versiunea 5.0, folosind un tip de compilare AOH (Ahead of Time), în care bytecode-ul este transpus în cod mașină la momentul instalării, astfel încât acesta este executat direct de mediul dispozitivului mobil; compatibilitatea cu versiuni anterioare (care folosesc mașina virtuală Dalvik, ce se bazează pe un compilator JIT - Just in Time) este asigurată prin transformarea pachetelor în format .dex (Dalvik Executable) la momentul compilării, urmând ca translatarea în format .oat să se realizeze la momentul instalării; fiecare aplicație Android rulează în procesul propriu, într-o instanță a mașinii virtuale ART, izolând astfel codul și datele sale prin intermediul unor permisiuni, care se aplică inclusiv la comunicația prin intermediul interfețelor de comunicare oferite de sistemul de operare Android;
   2. **Zygote** este procesul care gestionează toate aplicațiile, fiind lansat în execuție odată cu sistemul de operare:
      1. inițial, creează o instanță a mașinii virtuale Java pentru sistemul de operare Android, în contextul căreia plasează serviciile de bază: gestiunea energiei, telefonie, furnizori de conținut, gestiunea pachetelor, serviciul de localizare, serviciul de notificări;
      2. atunci când este necesar să lanseze în execuție o anumită aplicație, se clonează, partajând astfel componentele sistemului de operare Android, astfel încât să se asigure performanța (timp de execuție) și eficiența (memorie folosită), de vreme ce fiecare aplicație trebuie rulată în propria sa instanță a mașinii virtuale Java;
4. **Cadrul pentru Aplicații** expune diferitele funcționalități ale sistemului de operare Android către programatori, astfel încât aceștia să le poată utiliza în aplicațiile lor.
5. La nivelul de **aplicații** se regăsesc atât produsele împreună cu care este livrat dispozitivul mobil (Browser, Calculator, Camera, Contacts, Clock, FM Radio, Launcher, Music Player, Phone, S Note, S Planner, Video Player, Voice Recorder), cât și produsele instalate de pe Play Store sau cele dezvoltate de programatori.

Fig. 2.4 – Arhitectura sistemului Android -[7]

**2.1.4 Funcții Android**

De vreme ce Android este un produs open-source, producătorii având posibilitatea de a-l modifica în mod gratuit, nu există configurații hardware sau software standard. Totuși, Android implementează următoarele funcționalități:

* **stocare** - folosește SQLite, o bază de date relațională ce utilizează resurse puține
* **conectivitate** - suportă GSM/CDMA, GPRS, EDGE, 3G, 4G, IDEN, EV-DO, UMTS, Bluetooth (inclusiv A2DP si AVRCP), WiFi, LTE, WiMAX
* **WiFi Direct** - tehnologie care permite aplicațiilor să se descopere și să se interconecteze peste o conexiune punct-la-punct având lățime de bandă mare
* **Android Beam** - o tehnologie bazată pe NFC (Near Field Communication) care permite utilizatorilor să partajeze conținut instant, prin apropierea dispozitivelor mobile respective
* **mesagerie** - atât SMS cât și MMS
* **navigare pe Internet** - bazat pe motorul open source pentru navigare WebKit impreună cu motorul JavaScript de la Chrome V8 suportând HTML5 și CSS3
* **multimedia** - suportă formatele H.263, H.264 (într-un container 3GP sau MP4), MPEG-4 SP, AMR, AMR-WB (într-un container 3GP), AAC, HE-AAC (într-un container MP4 sau 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF si BMP
* **grafică** - 2D optimizată, 3D (OpenGL ES)
* **senzori** - accelerometru, cameră foto, busolă digitală (magnetometru), senzor de proximitate, GPS / AGPS
* **multi-touch** - suportă ecrane cu posibilitate de contact în mai multe puncte concomitent
* **multi-tasking** - permite rularea de aplicații cu mai multe fire de execuție
* **GCM (Google Cloud Messaging)** - serviciu ce permite dezvoltatorilor să trimită date de dimensiuni mici către utilizatori pe dispozitive Android, fără a avea nevoie de o soluție de sincronizare proprietară
* **multi-Language** - suport pentru text unidirecțional și bidirecțional
* **suport pentru aplicații Flash** (până în versiunea 4.3)
* **legătură la Internet** - suportă partajarea conexiunilor la Internet ca punct de distribuție cu fir / fără fir

**2.2**: **Cloud computing**

În zilele noastre, este imposibil să citești despre tehnologie fără a trece peste termenul de cloud computing. În timp ce unii ar putea crede că cloud computing-ul este doar un nou cuvânt cheie, companiile îl folosesc pentru a vinde servicii, cloud computing transformă modul în care implementăm tehnologia.

**2.2.1**: **Caracteristici generale**

În general, cloudul încorporează următoarele caracteristici cheie:

* Punerea în comun a resurselor și elasticitatea
* Servicii self-service și automate
* Acces la Internet
* Facturarea se face pe un model plata-pe-utilizare

Fiecare dintre aceste caracteristici este descrisă mai detaliat în secțiunile următoare.

Reunirea și elasticitatea resurselor

Punerea în comun a resurselor este abilitatea de a scala în sus și în jos pentru a servi mai mulți clienți folosind un model multi-chiriași cu diferite resurse fizice și virtuale atribuite dinamic și reasignate în funcție de cerere.

De multe ori, furnizorul de servicii nu poate anticipa modul în care clienții vor utiliza serviciul. Unii clienți ar putea utiliza serviciul de câteva ori pe parcursul celor mai intense sesiuni, în timp ce alții ar putea să-l folosească ca o platformă de dezvoltare principala pentru restul aplicațiile sale.

Prin urmare, serviciul trebuie să fie disponibil tot timpul și trebuie să fie proiectat să scaleze în sus pentru perioade cand cererea este mare și în jos pentru cele mai ușoare. De asemenea, serviciul trebuie să scaleze când se adaugă utilizatori suplimentari și când se modifică cerințele aplicației.

Servicii self-service și automate

Cloud Computing permite clienților să obțină ușor servicii cloud fără a trece printr-un proces lung. Clientul cere pur și simplu puterea de calcul, stocare, software-ul, procesele sau alte resurse de la furnizorul de servicii.

Acesta este un avantaj pe care îl oferă cloud computing în comparație cu procesul pe care trebuie să îl parcurgi atunci când soliciți servicii noi de la un centru tipic de date. Înainte de a implementa o nouă aplicație, departamentul IT trebuie să transmită o solicitare centrului de date

pentru hardware, software, servicii sau resurse de procesare suplimentare.

Centrul de date evaluează toate solicitările de la diferite departamente și evaluează disponibilitatea resurselor existente față de nevoia de a achiziționa hardware nou.

După achiziționarea unui nou echipament, centrul de date este configurat pentru noua aplicație. Acest lucru se dovedește a fi un proces lung și complicat care poate fi facilitat prin utilizarea serviciilor cloud.

Accesul la Internet

Accesul serviciilor la Internet permite comoditatea. Accesul la Internet înseamnă că resursele găzduite în cloud sunt disponibile pentru accesul dintr-o gamă largă de dispozitive și din mai multe locații care oferă acces online. Utilizatorii pot accesa date și servicii oriunde și ori de câte ori au nevoie, de la computerul de acasă, tablete sau smartphone-uri. De obicei, acest lucru a fost făcut printr-un browser, pentru a evita necesitatea de a instala software local.

Cloud computing are diferite modele de implementare: nori privați, publici și hibrizi. Într-un cloud privat, datele securizate sunt accesate numai de către angajații companiei în firewall-ul companiei. Compania își desfășoară propria infrastructură, inclusiv un centru de date plin de servere.

Cloud computing public este atunci când clienții folosesc o companie externă pentru a găzdui servere sau alte servicii cloud pe care aceștia le accesează pentru angajații lor. Accesul pe Internet poate cauza probleme de securitate într-un cloud privat, dar, pe măsură ce mai mulți angajați utilizează smartphone-uri, tablete pentru a accesa resursele companiei sau au opțiunea de a lucra de acasa, apare necesitatea accesului la rețea prin Internet, ceea ce poate determina compania să adopte un nor hibrid care să combine atât un cloud privat, cât și un cloud public.

Facturarea se face pe unui model plata-pe-utilizare

Plătești numai pentru serviciile utilizate și nu mai mult. În loc să plătești 100% pentru serverele care se utilizează doar 20% din timp, se plătește doar pentru numărul exact de resurse utilizate.

Furnizorii ideali de cloud folosesc pentru a taxa utilizatorii termeni obisnuiti pentru ca nu doar administratorii de sisteme IT să înțeleagă. Un mediu cloud are nevoie de un serviciu încorporat care facturează clienții. Desigur, pentru a calcula factura, utilizarea trebuie urmărită.

De exemplu, măsurarea resurselor de stocare, lățime de bandă și de calcul consumate și încărcarea pe gigaocteți stocați, octeți transferați, ore de calcul folosite, număr de conturi active, de utilizatori pe lună sau tranzacții efectuate.

**2.2.2: GoogleCloud** -[8]

Dispozitivele mobile susținute de inteligența scalabilă a mașinilor în cloud reprezintă paradigma de calcul definitorie a timpului nostru. Instrumentele moderne necesită o atenție deosebită pentru provocările cu care se confruntă dezvoltatorii de pe mobil: capabilități fără server, un model de date de tip cloud capabil să mențină date chiar și atunci când dispozitivul este offline, acces la latență redusă la mass-media oriunde în lume și sincronizarea datelor în timp real pe toate platformele mobile. Platforma Google Cloud (GCP) oferă dezvoltatorilor soluții cuprinzătoare, cu accent pe ușurința utilizării și viteza - fără a fi nevoie să gestionați infrastructura.

Fără server și în timp real

Instrumentele moderne pot gestiona complexitatea aplicațiilor în timp real în numele dezvoltatorilor. Firebase este o platformă de aplicații unificată pentru iOS, Android și web, care îți permite să construieștii aplicații mobile mai bune și să îți dezvolți afacerea. Fără a necesita nicio gestionare a serverului, funcțiile Cloud permit scrierea de funcții unice simple care să răspundă evenimentelor din cloud. Soluțiile fără server, cum ar fi Firebase și Cloud Functions, lasă dezvoltatorii de pe mobil să se concentreze asupra a ceea ce este important: o experiență extraordinară pentru utilizatori și rapid de transport.

Cele mai bune aplicații îmbină clientul și cloud-ul într-o singură platformă, creând experiențe extrem de receptive pentru utilizatorii susținuți de resurse puternice de calcul de la distanță. GCP te ajută să atingi cu ușurință acest echilibru pentru backend-ul mobil, unde sarcini non-interactive sunt descărcate de GCP, ceea ce duce la o îmbunătățire a duratei de viață a bateriei pe telefonul mobil, la o utilizare mai redusă a lățimi de bandă și la o experiență cât mai optima a clientului pe telefonul mobil.

**2.2.3: Google Firestore** -[9]



Google Firestore este soluția utilizată în cadrul acestui proiect, aceasta are la bază o structura de date cloud NoSQL flexibilă și scalabilă pentru stocarea și sincronizarea datelor pentru dezvoltarea client-server.

Cloud Firestore este o bază de date flexibilă și scalabilă pentru dezvoltarea de dispozitive mobile, web și server de la Firebase și Google Cloud Platform. Ca baza de date Firebase Realtime, păstrează sincronizarea datelor în aplicațiile client prin intermediul ascultătorilor în timp real și oferă suport offline pentru mobil și web, astfel încât să poți crea aplicații receptive care funcționează indiferent de latența rețelei sau conectivitatea la Internet. Cloud Firestore oferă, de asemenea, integrare fără probleme cu alte produse Firebase și Google Cloud Platform, inclusiv funcțiile Cloud.

**Capabilități cheie:**

* Flexibilitate, modelul de date Cloud Firestore suportă structuri de date ierarhice flexibile. Ai posibilitatea de a stoca datele în documente, organizate în colecții. Documentele pot conține obiecte complexe imbricate în plus față de subcollecții.
* Interogări explicite, în Cloud Firestore, puteți utiliza interogări pentru a prelua documente individuale specifice sau pentru a prelua toate documentele dintr-o colecție care se potrivește cu parametrii specifici de interogare. Interogările pot include filtre multiple și înlănțuite și pot combina filtrarea și sortarea. De asemenea, acestea sunt indexate în mod prestabilit, astfel încât performanța interogării este proporțională cu dimensiunea setului de rezultate, nu cu setul de date.
* Actualizări în timp real, la fel ca și baza de date Realtime, Cloud Firestore utilizează sincronizarea datelor pentru a actualiza datele de pe orice dispozitiv conectat. Cu toate acestea, este, de asemenea, proiectat să facă în mod eficient interogări simple.
* Suport offline, Cloud Firestore memoreaza datele pe care aplicația le utilizează în mod activ, astfel încât aplicația să poată scrie, citi, asculta și interoga date, chiar dacă dispozitivul este offline. Când dispozitivul revine online, Cloud Firestore sincronizează orice schimbări locale în Cloud Firestore.
* Proiectat pentru a putea fi scalat, Cloud Firestore vă oferă cele mai bune infrastructuri puternice ale platformei Google Cloud: replicarea automată a mai multor regiuni de date, garanții de consecvență puternică, operațiuni cu loturi atomice și suport real pentru tranzacții. Cloud Firestore poate rezolva cele mai dificile sarcini de lucru din bazele de date din cele mai mari aplicații din lume



Fig 2.5 – Organizarea datelor pe colecții

Urmând modelul de date NoSQL al Cloud Firestore, datele sunt stocate în documente care conțin câmpuri care corespund valorilor. Aceste documente sunt stocate în colecții, care sunt containere pentru documentele pe care le poți utiliza pentru a organiza datele și pentru a crea interogări.

Documentele suportă mai multe tipuri diferite de date, de la șiruri simple și numere, la obiecte complexe, imbricate. De asemenea, se pot crea subcollecții în interiorul documentelor și se pot construi structuri de date ierarhice care se extind pe măsură ce crește baza de date. Modelul de date Cloud Firestore acceptă orice structură de date funcționează cel mai bine pentru aplicația respectivă.

În plus, interogarea în Cloud Firestore este expresivă, eficientă și flexibilă. Poți face interogări superficiale pentru a prelua date la nivel de document fără a fi nevoie să preluați întreaga colecție sau subcolecții imbricate.

Adaugi sortarea, filtrarea și limitele interogărilor sau cursorilor pentru a obține rezultatele dorite. Pentru a păstra actualitatea datelor în aplicație, fără a recupera întreaga bază de date de fiecare dată când se produce o actualizare, se poate adăuga un eveniment care monitorizeaza baza de date în timp real.

Adăugarea în aplicație a evenimentelor de monitorizare în timp real vă anunță cu un *Snapshot* de date ori de câte ori datele pe care aplicația le monitorizează sufera modificări, preluând numai noile modificări.

**2.2.4: Autentificarea Firebase** -[9]

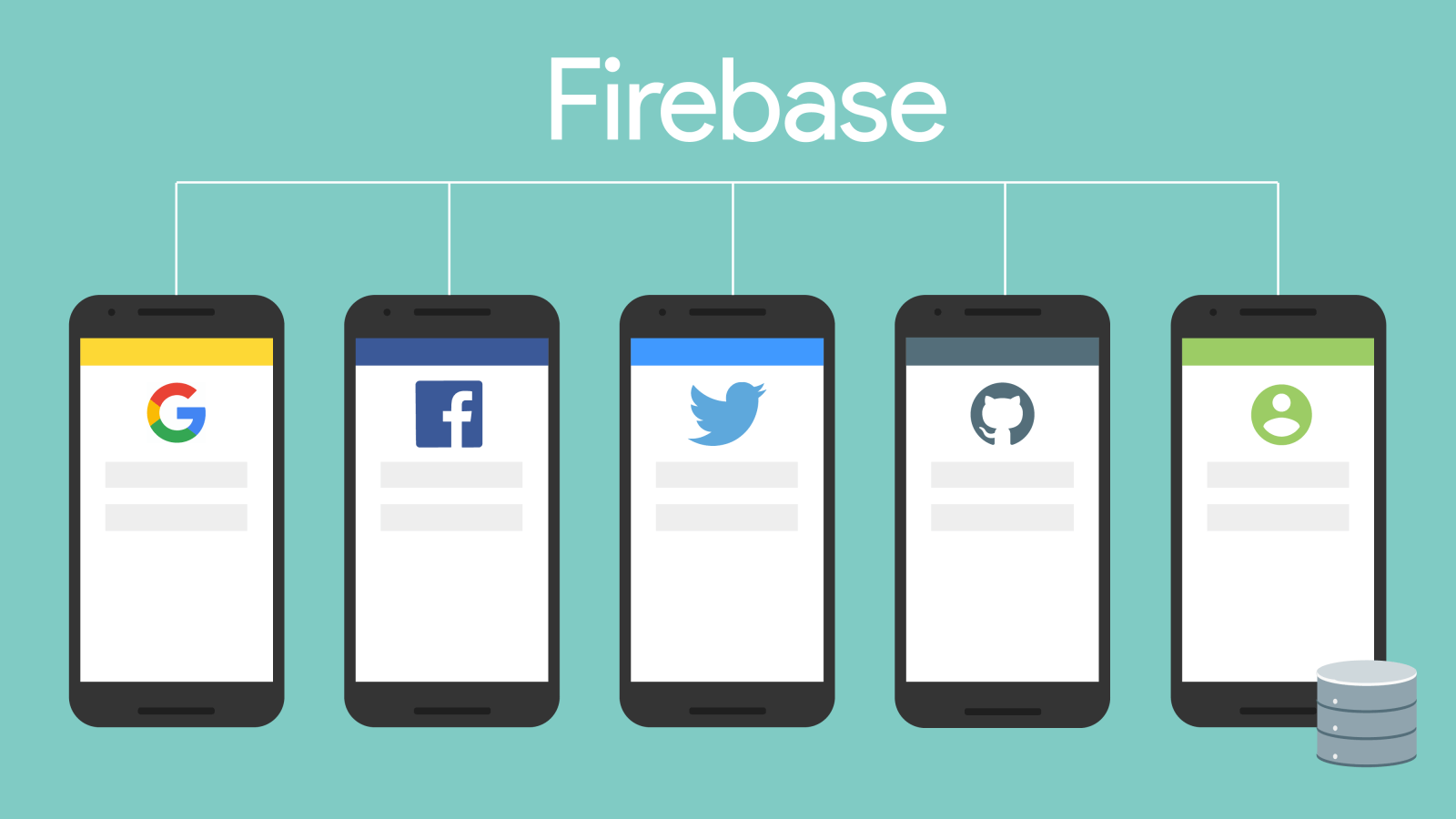
Cele mai multe aplicații trebuie să cunoască identitatea unui utilizator. Cunoașterea identității unui utilizator permite unei aplicații să salveze în siguranță datele utilizatorului în cloud și să ofere aceeași experiență personalizată în toate dispozitivele utilizatorului.

Firebase Authentication oferă servicii backend, SDK-uri ușor de utilizat și biblioteci UI gata de utilizare pentru a autentifica utilizatorii în aplicație.

Acesta susține autentificarea utilizând parole, numere de telefon, furnizori de identitate federalizați, cum ar fi Google, Facebook și Twitter și multe altele.

Firebase Authentication se integrează în strânsă legătură cu alte servicii Firebase și utilizează standardele industriale precum OAuth 2.0 și OpenID Connect, astfel încât acestea să poată fi ușor integrate cu backend-ul personalizat.

Poți să conectezi utilizatorii în aplicația Firebase fie prin utilizarea FirebaseUI ca soluție completă de autori drop-in, fie prin utilizarea pachetului SDK de autentificare Firebase pentru a integra manual una sau mai multe metode de conectare în aplicație.



**Capabilități cheie:**

Soluție de autentificare drop-in**:**

* Modul recomandat de a adăuga un sistem complet de conectare la aplicație.
* Componenta FirebaseUI Auth implementează cele mai bune practici pentru autentificarea pe dispozitive mobile și pe site-uri web, ceea ce poate maximiza conectarea și conversia de înscriere pentru aplicația respectiva. De asemenea, se ocupă și de cazuri speciale, cum ar fi recuperarea contului și crearea unui cont nou, care poate fi sensibilă la securitate și predispusă la erori.
* FirebaseUI poate fi personalizat cu ușurință pentru a se potrivi cu restul stilului vizual al aplicației și este open source, astfel încât să nu fi constrâns în realizarea experienței utilizatorului.

**Autentificare cu ajutorul unui email:**

* Utilizatorii se autentifica cu adresele lor de e-mail și parolă. Firebase Authentication SDK oferă metode de creare și gestionare a utilizatorilor care utilizează adresele lor de e-mail și parolele pentru a se conecta. Firebase Authentication se ocupă, de asemenea, trimiterea e-mailurilor de resetare a parolei.

Autentificarea cu ajutorul numerelor de telefon**:**

* Utilizatorii pot fi autentificați prin primirea de mesaje SMS pe telefoanele lor.

**Capitolul 3: Implementarea aplicației**

**3.1 Prezentarea Implementării**

Aplicația a fost scrisă intr-un limbaj de programare recent, actualizat, stabil și de notorietate în randul dezvoltatorilor de aplicații Android.

***Kotlin*** este princilapa alegere a programatorilor pentru dezvoltarea aplicațiilor Android, aducând toate avantajele unui limbaj de programere modern platformei Android fără a introduce noi restricții:

Compatibilitate: Kotlin este pe deplin compatibil cu JDK 6, asigurându-se că aplicațiile Kotlin pot funcționa pe dispozitive Android mai vechi, fără probleme. Instrumentul Kotlin este pe deplin acceptat în Android Studio și compatibil cu sistemul Android.

Performanță: o aplicație Kotlin rulează la fel de repede ca o versiune Java echivalentă, datorită unei structuri de bytecode foarte asemănătoare. Cu ajutorul lui Kotlin pentru funcții inline, codul care utilizează lambda rulează adesea mai rapid decât același cod scris în Java.

Interoperabilitate: Kotlin este 100% interoperabil cu Java, permițând utilizarea tuturor bibliotecilor existente Android într-o aplicație Kotlin.

Amprenta: Kotlin are o bibliotecă de execuție foarte compactă.

Clasa Activity este o componentă crucială a unei aplicații Android, iar modul în care sunt lansate și realizate activitățile este o parte fundamentală a aplicației. Spre deosebire de programele în care aplicațiile sunt lansate cu ajutorul metodei principale main(), sistemul Android inițiază codul într-o instanță de activitate prin invocarea unor metode de apel invers specific care corespund anumitor etape ale ciclului său de viață.

Atunci când o aplicație invocă o altă aplicație, apelul invocă o activitate în cealaltă aplicație, mai degrabă decât aplicația ca un întreg atomic. În acest fel, activitatea servește ca punct de intrare pentru interacțiunea unei aplicații cu utilizatorul. O activitate reprezintă o subclasă a clasei Activity.

O activitate oferă fereastra în care aplicația își desenează interfața utilizator. Această fereastră în mod obișnuit umple ecranul, dar poate fi mai mică decât ecranul și plutește deasupra altor ferestre. În general, o activitate implementează un ecran într-o aplicație. In cadrul aplicației avem două activități, reprezentante pentru cele două tipuri de utilizatori. Activitatea principală, care este primul ecran expus atunci când utilizatorul lansează aplicația. Fiecare activitate poate apoi să înceapă o altă activitate pentru a efectua diferite acțiuni.

*Activitatea princilala* a aplicației lansează prin intermediul metodei onCreate() UI-ul menit să faciliteze autentificarea utilizatorilor prin apelul funcției showSingInOptions().

Fiecare metodă de autentificare expusă prin interfața de autentificare firebase trebuie inițializată conform următorului exemplu (cod):

providers = Arrays.asList<AuthUI.IdpConfig>(

AuthUI.IdpConfig.EmailBuilder().build(), //Email builder

AuthUI.IdpConfig.FacebookBuilder().build(), //Facebook builder

AuthUI.IdpConfig.GoogleBuilder().build(), //Google builder

AuthUI.IdpConfig.PhoneBuilder().build() //Phone builder

)

Pentru a creea și a lansa intenția de conectare se folosește urmatoarea secventă de cod:

startActivityForResult(  
        AuthUI.getInstance()  
                .createSignInIntentBuilder()  
                .setAvailableProviders(providers)  
                .build(), RC\_SIGN\_IN)

Când întregul proces de conectare este complet, utilizând una din metodele de autentificare puse la dispoziție, se primește rezultatul prin funcția ***onActivityResult.***

Modul recomandat de a obține utilizatorul curent este prin apelarea metodei getCurrentUser. Dacă niciun utilizator nu este conectat, getCurrentUser returnează null:

val user = FirebaseAuth.getInstance().currentUser  
if (user != null) {  
    // User is signed in  
} else {  
    // No user is signed in  
}

Există câteva cazuri în care getCurrentUser va returna un FirebaseUser non-null dar tokenul de bază nu este valabil. Acest lucru se poate întâmpla, de exemplu, în cazul în care utilizatorul a fost șters pe un alt dispozitiv și tokenul local nu s-a refăcut. În acest caz, este posibil ca un utilizator valid să fie returnat de metora getCurrentUser dar apelurile ulterioare la resursele de autentificate nu vor reuși.

Pentru a obține informați cu privire la utilizatorul curent se utilizează metode ale unei instanței a FirebaseUser. De exemplu (cod):

val user = FirebaseAuth.getInstance().currentUser  
user?.let   
    val name = user.displayName  
    val email = user.email  
    val photoUrl = user.photoUrl

val emailVerified = user.isEmailVerified

val uid = user.uid  
}

Informațiile referitoare la utilizatorul curent sunt în funcție de caz și metoda de autentificare: nume, adresă de email, URL către poză, dacă adresa de email este verificată sau nu și număr de identificare unic.

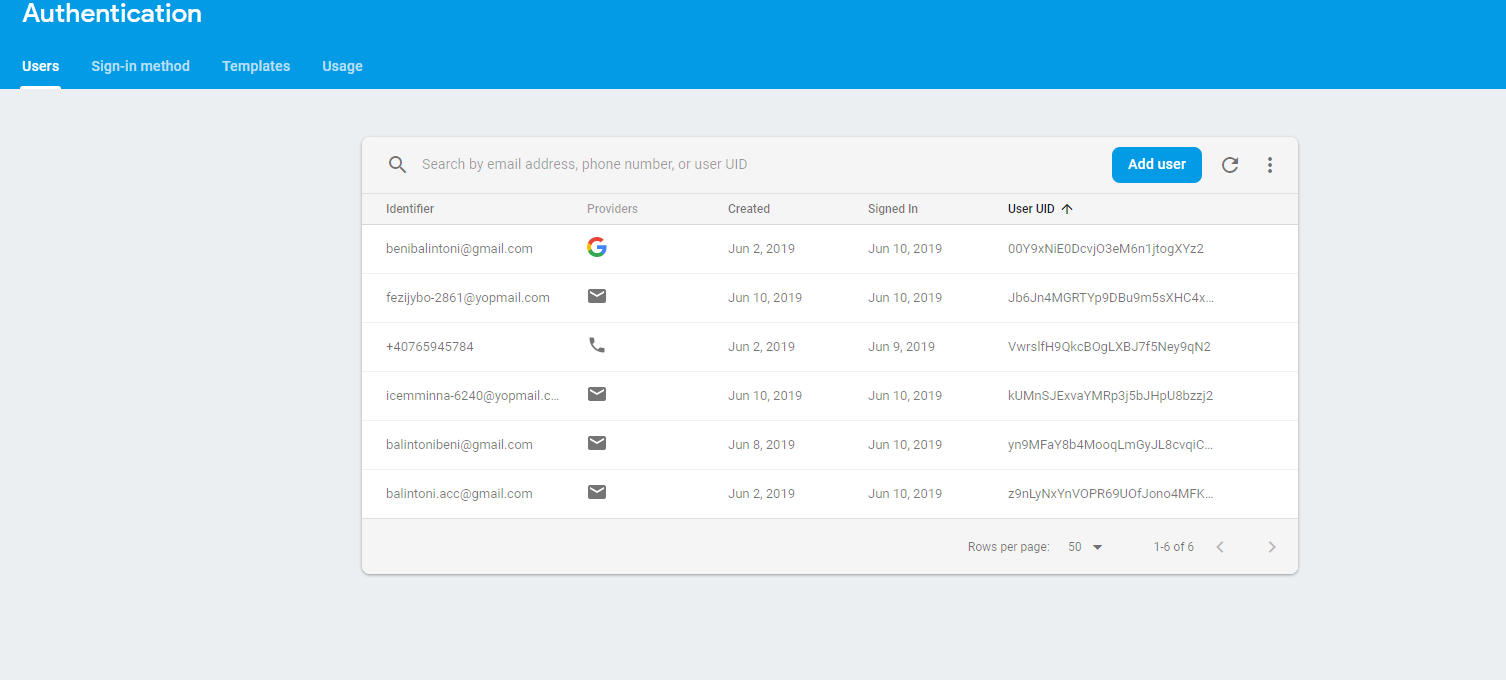
De asemenea, se pot crea noi utilizatori utilizând secțiunea Authentication a consolei Firebase, în pagina Users asa cum se poate vedea in Fig 3.1

Fig 3.1 – Consola unde sunt introduși utilizatorii

Fiecare utilizator are un identificator care poate fi email, număr de telefon sau cont de FaceBook, are informați privind data creeri contului și ultima dată când a fost activ, precum și un numar UID. Deasemenea sunt operați care pot fi executate direct în consolă: adăugarea, dezactivarea, ștergerea sau resetarea parolei.

După ce autentificarea utilizatorului sa efectuat cu succes se verifică cu ajutorul metodei

getUserData() dacă acesta a mai fost autentificat și a fost repartizat la o coadă de așteptare cu un anumit număr de ordine urmând să fie dispuse prin interfață informațiile utile acestuia.

Există două modalități de a prelua datele stocate în Cloud Firestore. Oricare dintre aceste metode poate fi utilizată cu documente, colecții de documente sau cu rezultatele interogărilor: Se poate apela o metodă pentru a obține datele sau se seta un ascultător pentru a primi evenimente de schimbare a datelor.

Când se setează un ascultător, Cloud Firestore trimite ascultătorului un snapshot inițial al datelor și apoi un alt instantaneu de fiecare dată când documentul se schimbă.

Următorul exemplu arată modul de preluare a conținutului unui singur document utilizând metoda get ():

val docRef = db.collection("users").document(FirebaseAuth.getInstance().*currentUser*!!.*uid*)  
docRef.get()  
        .addOnSuccessListener { document ->  
            if (document != null) {  
                //Datele au fost preluate cu succes   
            } else {  
                //Utilizatorul nu a fost găsit în baza de date  
            }  
        }  
        .addOnFailureListener { exception ->  
            //Blocul unde se pot pride și trata excepțiile  
        }

Dacă operatiunea a fost executată cu succes informațiile sunt puse la dispoziția utilizatorului cu ajutorul metodei setVisibility().

Pentru meniu a fost creeata o resursa menu.xml care conține 4 articole puse la dispoziția utilizatorului cu ajutorul funcției *onCreateOptionsMenu*(). Implementarea evenimentelor referitoare meniului se regăsește în funcția *onOptionsItemSelected*().

Una din opțiunile pe care utilizatorul le are atunci când opereaza meniul este aceea de Sign Out, pentru aceasta se va folosi o metodă pusă la dispoziție de UI-ul Autentificării Firebase AuthUI.getInstance().signOut(this@MainActivity).

În momentul în care este autentificat și nu a mai fost înscris utilizatorul poate opta pentru obtinerea unui ticket virtual prin apăsarea unui buton, acesta va conține informații depre numarul laboratorului la care a fost repartizat, numarul de ordine în coada de așteptare cât și numarul curent de persoane în așteptare. Utilizatorul va fi introdus în baza de date, colecția *users* ,numele documentului fiind dat de UID-ul său, documentul va conține toate informatiile puse la dispoziție de metoda de autentificare aleasă dar și informații privind poziția în coada de asteptare.

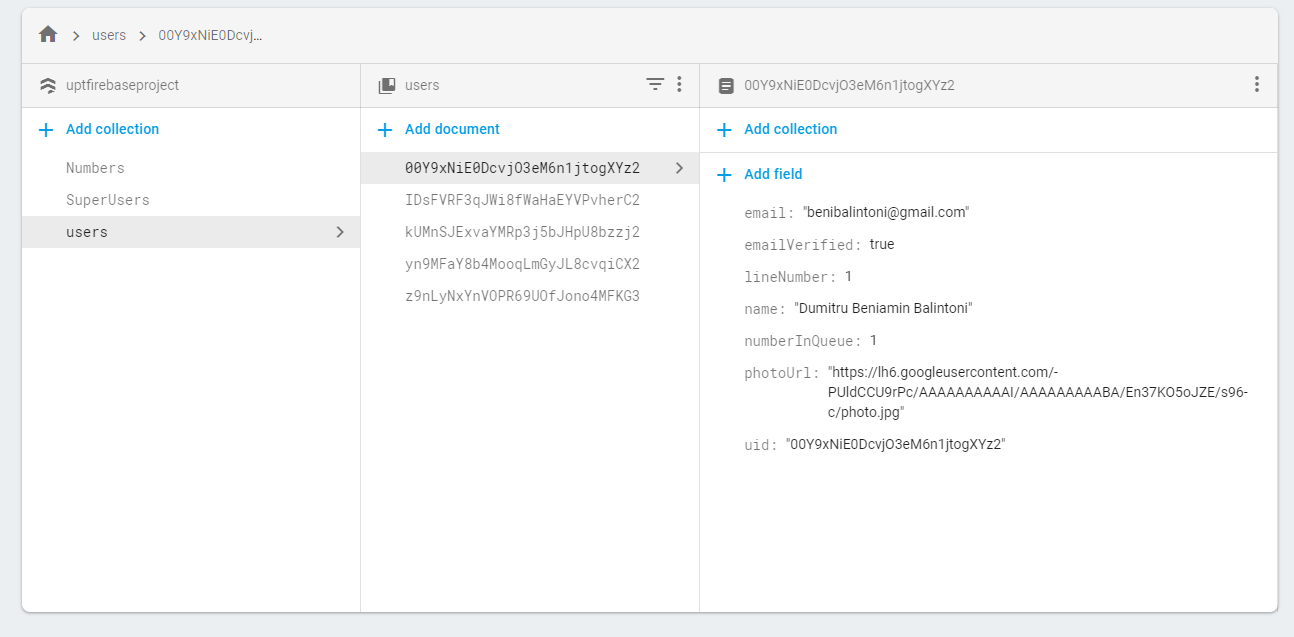
Exemplu de utilizator:

Fig 3.2 – Structura bază de date

Există mai multe moduri de a scrie date în Cloud Firestore:

* Prin a seta datele unui document într-o colecție, specificând în mod explicit un identificator de document.
* Prin adăugarea unui document nou într-o colecție. În acest caz, Cloud Firestore generează automat identificatorul documentului.
* Prin a creea un document gol cu un identificator generat automat și datele acestuia vor fi atribuite mai târziu.

În cadrul aplicației se va folosi prima metodă. Exempe cod:

val userData = FirebaseAuth.getInstance().*currentUser*userData?.*let* **{** val user = HashMap<String, Any>()  
 user["name"] = userData.*displayName*.*toString*()  
 user["email"] = userData.*email*.*toString*()  
 user["photoUrl"] = userData.*photoUrl*.*toString*()  
 user["emailVerified"] = userData.*isEmailVerified* user["uid"] = userData.*uid*

*db*.collection("users").document(userData.*uid*)  
 .set(user)  
 .addOnSuccessListener **{** Log.d(*TAG*, "User adăugat cu succes !")**}** .addOnFailureListener **{** e **->** Log.w(*TAG*, "Eroare la scriere!", e) **}**

Din momentul în care utilizatorul este intregistrat cu succes se începe o monitorizare continuă a bazei de date pentru a putea oferi informații în timp real cu privire la numărul de persoane în așteptare, pentru a putea alerta și notifica utilizatorii atunci când le vine rândul, baza de date se monitorizează cu ajutorului metodei *addSnapshotListener* și pot fi de trei feluri:

* DocumentChange.Type.ADDED
* DocumentChange.Type.MODIFIED
* DocumentChange.Type.REMOVED

Aplicația se folosește de evenimentele de tip Removed pentru a recalcula în timp real numărul persoanelor aflate în coada de așteptare respectivă.

În momentul în care unui utilizator îi vine randul, acesta va fi notificat prin aplicație și va fi rugat să se prezinte în laboratorul unde a fost repartizat pentru a se înscrie la facultate.

**3.1 Informați referitoare la organizarea aplicației**

Pentru ca aplicația să poată utiliza activități, acestea și anumite atribute ale acestora trebuie declarate intr-un fisier de tip xml numit AndroidManifest.xml

Exemplu pentru declararea unei activități:

<manifest ... >  
  <application ... >  
      <activity android:name=".MainActivity" />  
      ...  
  </application ... >  
  ...  
</manifest >

Acest xml este utilizat pentru a defini toate componentele aplicației. Acesta include numele pachetelor de aplicații, activitățile, receptoarele, serviciile și permisiunile pe care le cere aplicația.

XML este un limbaj de markup folosit pentru a descrie datele, folosit predominant deoarece este ușor inteligibil atât de om cât și de mașină. De asemenea, este scalabil și ușor de dezvoltat. În Android, se folosește xml pentru proiectarea layout-urilor.

Fișierele xml de configurare sunt utilizate pentru a defini interfața utilizator actuală a aplicației. Acesta conține toate elementele (vizuale) sau instrumentele pe care dorim să le folosim în aplicație. Asemenea elementelor TextView, Button și alte elemente de interfață.

Locația în Android Studio: Acest fișier se află în interiorul folderului res și în interiorul acestuia există un alt folder numit layout unde sunt toate fișierele de tip aspect pentru activitățile lor respective sau fragmente.

Fisiere xml (strings.xml): Acest fisier xml este folosit pentru a înlocui șirurile de caractere cu un singur element de tip sir. Acest fișier este utilizat pentru o organizare cât mai buna a aplicației și sporește reutilizarea codului.

Întregul concept al interfeței utilizator Android este definit folosind ierarhia obiectelor View și ViewGroup. Un ViewGroup este un container invizibil care organizează alte vizualizări. Acestea din urmă sunt alte widget-uri care fac parte din UI. Un ViewGroup poate avea un alt ViewGroup ca element intermediar, după cum se arată în figura :

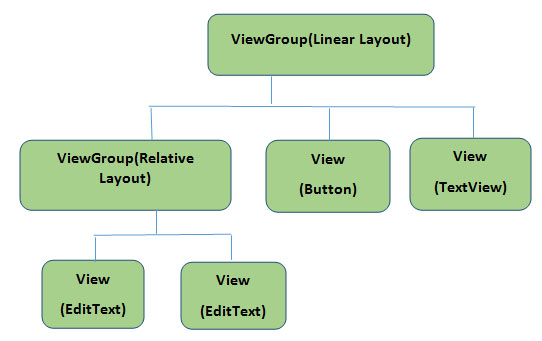


Fig 3.3 – Model de structură interfețe

Sistemul Android compilează resursele aplicației și codul sursă și le ambalează într-un fișier APK. Android Studio folosește un set de instrumente avansate pentru a automatiza și gestiona procesul de construire, în timp ce permite definirea de configurații personalizate flexibile.

Fiecare configurație poate defini propriul set de cod și resurse, în timp ce reutilizează componente comune tuturor versiunilor aplicației. Pluginul Android pentru Gradle funcționează împreună cu setul de instrumente pentru a furniza procese și setări configurabile care sunt specifice pentru construirea și testarea aplicațiilor Android.

Procesul de construire implică multe instrumente și procese care convertesc proiectul într-un APK. Procesul de construire este foarte flexibil, și de o importanță foarte mare in cadrul proiectului.

Crearea configurațiilor personalizate necesită modificări ale unuia sau mai multor fișiere de configurare. Aceste fișiere text simplu utilizează Domain Specific Language (DSL) pentru a descrie și manipula logica construirii folosind Groovy, un limbaj dinamic pentru Java Virtual Machine (JVM).

Plugin-ul Android pentru Gradle introduce majoritatea elementelor DSL de care aplicația are nevoie printr-o structură conform cu Fig 3.4

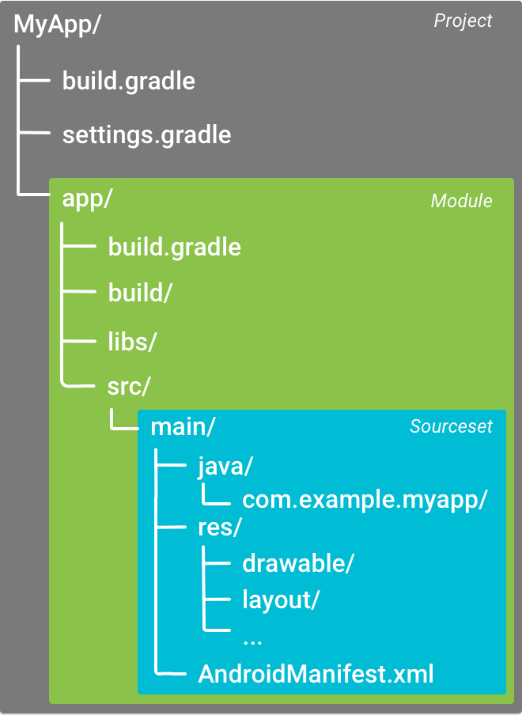


Fig 3.4 – Structura unui proiect Andorid

Procesul de construire al aplicației este prezentat in Fig

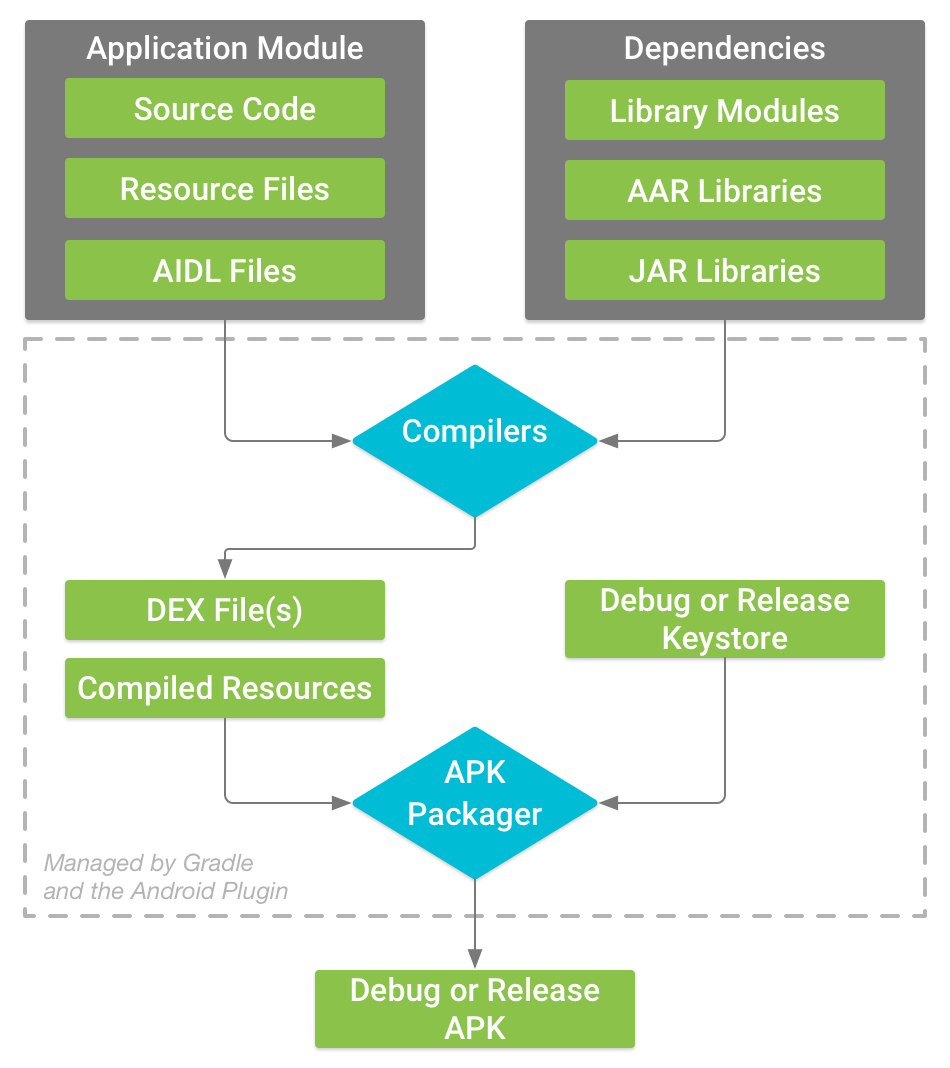


Fig 3.5 – Alcatuirea aplicației

**Capitolul 4:Utilizarea aplicatie**

Sistemele de gestionare a cozilor sau managementul cozilor se referă la manipularea fluxului de clienți începând cu prealuarea clientului, informarea sau direcționarea către serviciul solicitat. Sistemele de gestionare cozi ajută la creșterea productivității, reducerea costurilor și creșterea satisfacției clientului.

**4.1: Prentarea Interfețelor**

Interfața de utilizare a aplicației este tot ceea ce utilizatorul poate vedea și interacționa cu acesta. Android oferă o varietate de componente UI pre-construite, cum ar fi obiecte structurate și comenzi UI care permit realizarea de interfețe grafice pentru aplicația respectivă. Android oferă, de asemenea, alte module UI pentru interfețe speciale, cum ar fi dialoguri, notificări și meniuri.

Lansarea în executie a aplicației pune la dispoziția utilizatorului interfața destinată autentificări acetuia, utilizator poate să se autentifice în aplicație prin una din metodele puse la dispoziție de Firebase precum: email, cont de google, cont de facebook sau prin confirmarea numărului de telefon conform Fig 4.1

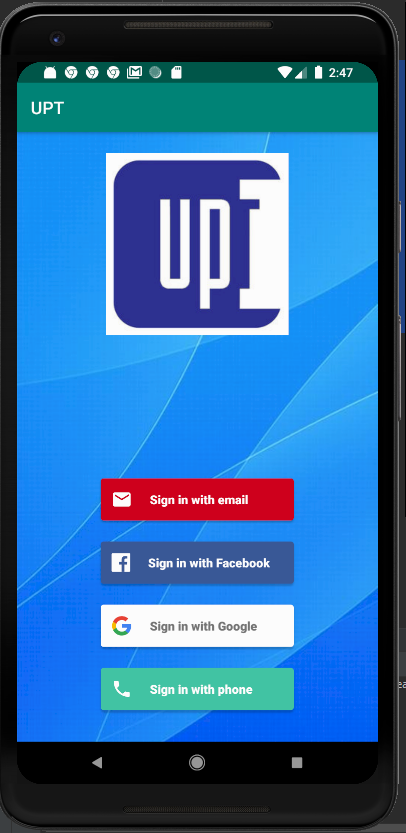


Fig 4.1 – Interfața utilizator destinată autentificări

Utilizatorul alege cea mai potrivită metoda de autentificare în funcție proprile preferințe.

Cele mai utilizare și preferate de utilizatori sunt metodele care folosesc platforme pe care aceștia deja dețin un cont și nu mai trebuie să țină minte un alt cont și o parola în plus.

Dacă utilizatorul optează pentru a se autentifica cu un cont de email acesta trebuie să completeze trei câmpuri: email, nume, prenume și parola. Toate câmpurile sunt obligatori și parola trebuie să conțină minim șase caractere și cel puțin un număr. Interfața cu utilizatorul este prezentată în Fig 4.2

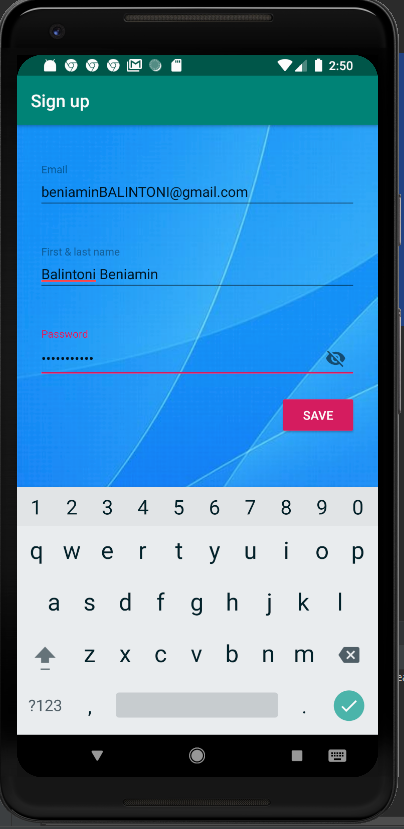


Fig 4.2 – Autentificarea prin email

Este absolut sigur ca utilizatorii să se conecteze la aplicație folosind contul de Facebook sau Google. Companiile de tehnologie imensă utilizează un standard numit OAuth, care permite aplicațiilor terțe să acceseze și să recupereze anumite informații din aceste site-uri mari pentru a autentifica utilizatorii.

Mecanismul tehnologic în cauză, OAuth este un protocol care ajută site-urile "mari" (site-uri care au un număr foarte mare de utilizatori, cum ar fi Google, Facebook) să permită accesul la informațiile utilizatorilor lor în aplicații terțe, fără a partaja parolele utilizatorilor sau alte detalii private, delicate.

OAuth este un standard deschis pentru delegarea accesului securizat, ceea ce înseamnă că este un serviciu care permite giganților web precum Google sau Facebook să ofere utilizatorilor obțiunea de a împărtăși propriile informații selectate cu aplicații terțe dar în același timp menține nivelul ridicat de securitate privind protecția informațiilor confidențiale ale utilizatorilor.

Interfața cu utilizatorul pentru autentificarea cu ajutorul Facebook este prezentată în Fig 4.3.

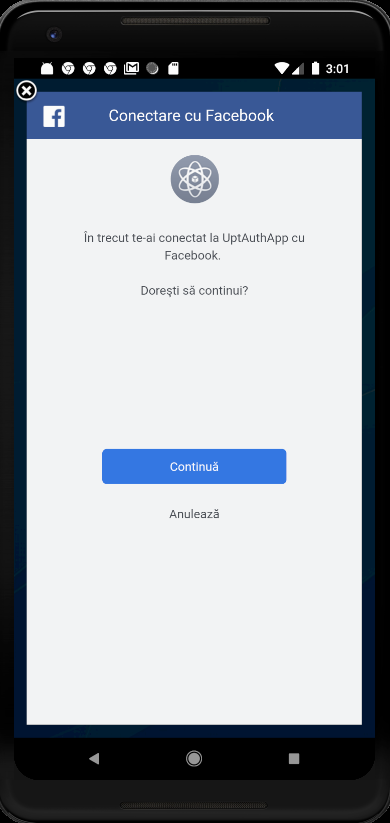
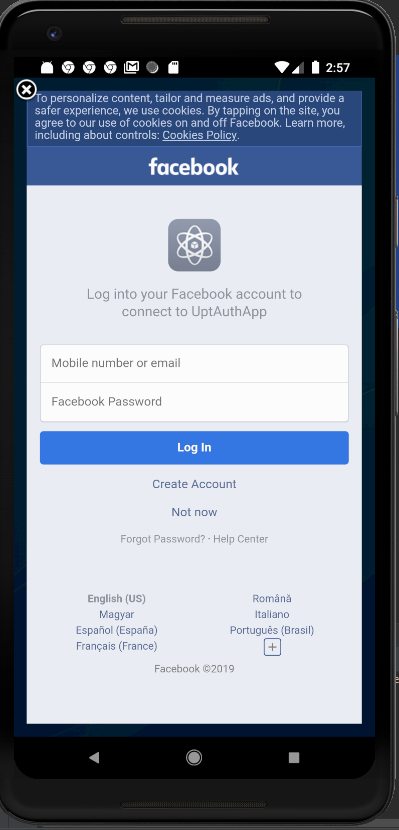


Fig 4.3 – Autentificarea cu ajutorul facebook.

Autentificarea pe baza contului de google poate fi cea mai bună opțiune deoarece folosind un sistem de operare android ai nevoie să fi autentificat cu un cont google, prin urmare ai posibilitatea să folosești același cont, lucru sugerat de interfață, sau poți să îți creezi un alt cont google.

Interfața cu utilizatorul pentru autentificarea cu ajutorul unui cont google este prezentată în Fig 4.4.

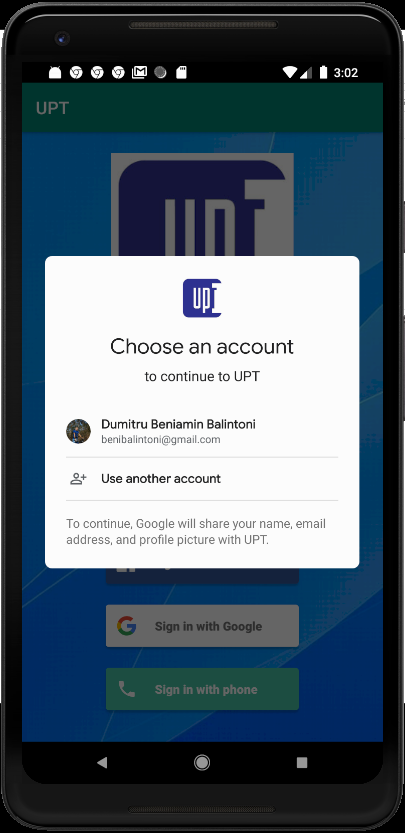
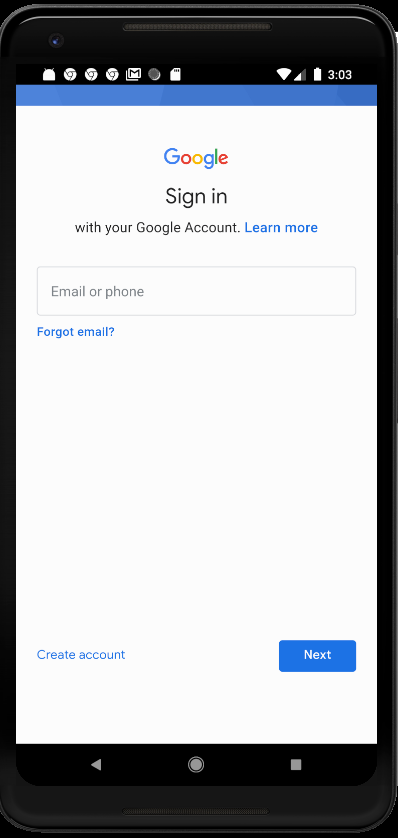
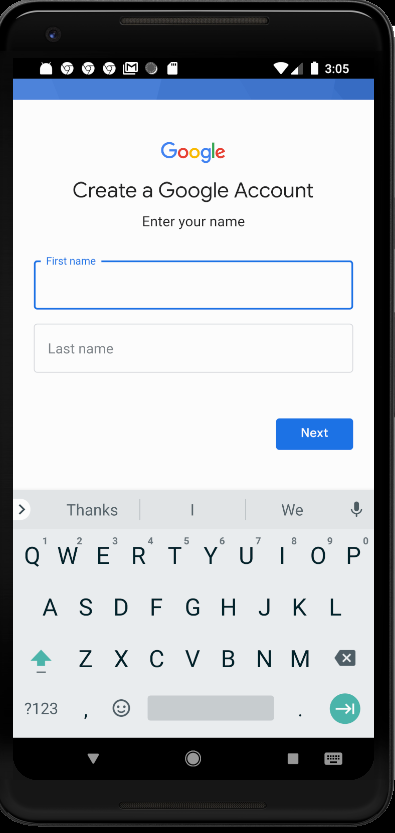


Fig 4.3 – Autentificarea cu ajutorul google.

Ultima metodă de autentificare pusă la dispoziția utilizatorilor le conferă acestora posibilitatea de a creea un cont pe baza numarului de telefon, aceștia primesc un mesaj cu un pin alcătuit din 6 digiți pentru confirmarea numarului, dacă pinul nu este introdus corect sau trec doisprezece secunde un alt pin va fi trimis prin mesaj.

Interfața cu utilizatorul pentru autentificarea cu ajutorul numărului de telefon este prezentată în Fig 4.4.

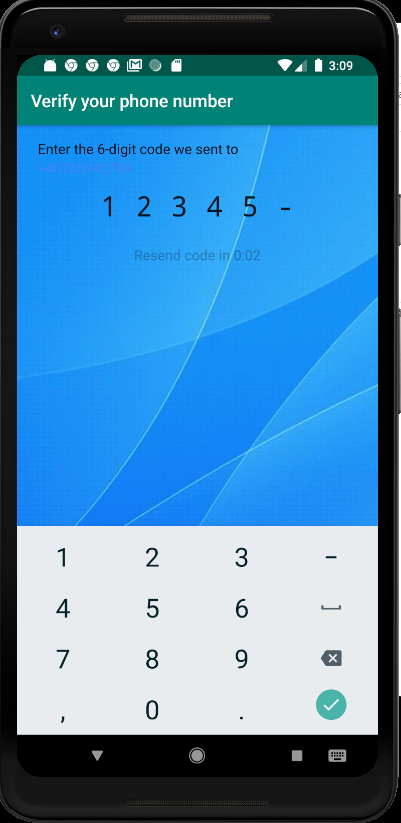
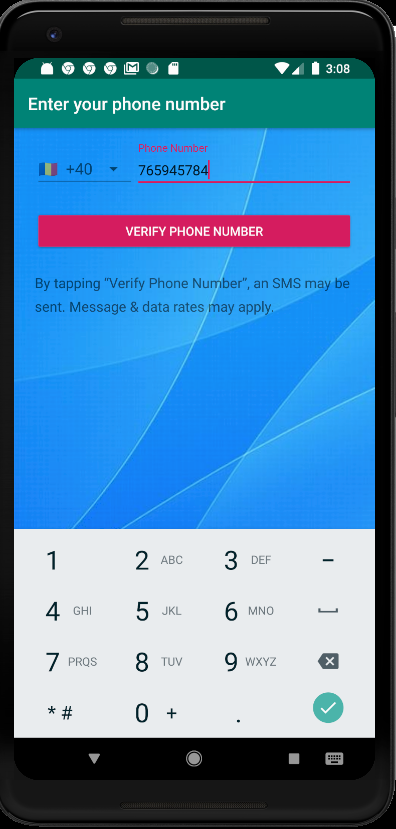


Fig 4.3 – Autentificarea cu ajutorul numărului de telefon.

Daca utilizatorul a mai fost autentificat dar aplicația a ieșit din execuție, în momentul în care utilizatorul revine este notificat cu privire la faptul că a mai fost autentificat în aplicație cu ajutorul *google smart lock* și are posibilitatea să reia activitatea direct din meniul respectiv.

Aplicatți are doua tipuri de utilizatori, candidații respectiv studenții voluntari. Odata ce utilizatorul este autentificat cu succes în funcție de tipul acestuia are posibilitatea:

-Dacă este student voluntar este redirecționat sper o activitate specială de unde acesta poate scoata candidații care au terminat din coada.

-Dacă este candidat acesta are posibilitatea de a obține un ticket virtual care conține informații despre coada la care acesta a fost repartizat, numarul de ordine și câte persoane sunt în aștepare. Moment in care utilizatorul este înscris în coada de așteptare, în funcție de informațiile puse la dispoziție de către aplicație acesta poate să îsi planifice restul activităților.(Fig4.5)

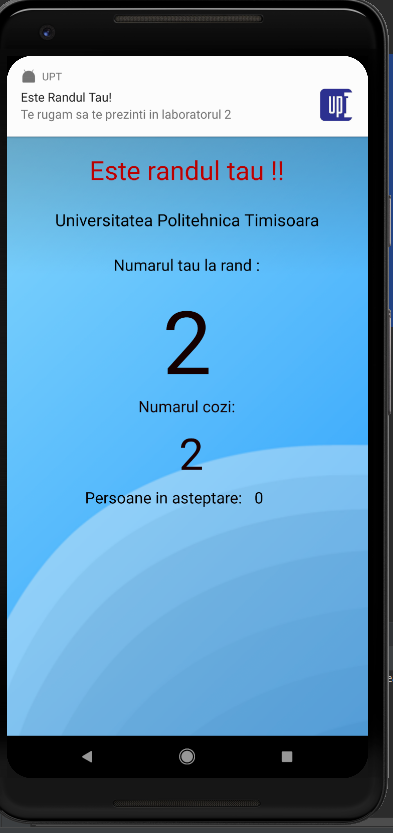


Fig 4.5 Fig 4.6

Utilizatorul este notificat în momentul în care este rândul său, în cadrul notificării se specifică laboratorul unde acesta a fost repartizat, daca aceata are aplicația pornită va putea observa si un mesaj cu privire la faptul că este rândul său conform Fig 4.6.

În cadrul meniului exista opțiuni de informare care sunt surse oficiale U.P.T. dar si opțiunea de delogare.

**Capitolul 5:Concluzii**

În concluzie

**5.1: Probleme Întampinate**

Principala problemă întâmpinată a fost comunicarea asincronă cu baza de date, comunicarea se executa pe un alt fir de executie al programului fapt ce determina o funcționare incorectă.

Probleme privind compatibilitatea resurselor folosite și integrarea lor în program.

**5.1: Propuneri**

Pentru o mai bună experiență a candidaților, organizarea de workshop-uri interactive, prezentări, activități chiar și cursuri.

Referințe:

*[1] -* Soluţia *ZeroCozi*: http://www.radiotimisoara.ro/2018/10/17/aplicatia-mobila-care-ne-poate-scapa-de-cozi-dezvoltata-de-doi-it-isti-din-timisoara/

*[2] [3] - Skiplino*: *https://skiplino.com/?gclid=Cj0KCQjwrdjnBRDXARIsAEcE5YnxcPzxTBceA5tbMRG1rJdMGkdCfLQ6n4dCfSkQVtoK8vkzzDm3\_OgaAkpsEALw\_wcB/*

*[4] [5] -* go-digital: <https://go-digital.ro/sisteme-de-management/>

[6] -

[7] - FirebaseUI <https://github.com/firebase/firebaseui-web>

[8] - <https://cloud.google.com/solutions/mobile/>

[9] – Documentația Google Firestore: <https://firebase.google.com/docs>