

Universitatea “Politehnica” din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

*Aplicație Android de organizare a activităților bazată pe localizare și  
servicii web*

## **Proiect de diplomă**

prezentat ca cerință parțială pentru obținerea titlului de  
*Inginer în domeniul Electronică și Telecomunicații*  
programul de studii de licență *Rețele și Software de Telecomunicații*  
(ETC – RST)

Conducător(i) științific(i)

*Conf. Dr. Ing. Eduard Cristian Popovici*

Absolvent

*Cristian Retea*

*Anul 2019*



Universitatea "Politehnica" din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației  
Departamentul Tc

**Anexa 1**

**TEMA PROIECTULUI DE DIPLOMĂ**  
a studentului **RETEA P.P. Cristian , 443D**

**1. Titlul temei:** Aplicatie Android de organizare a activitatilor bazata pe localizare si servicii web

**2. Descrierea contribuției originale a studentului (în afara părții de documentare) și specificații de proiectare:**

Proiectul consta intr-o aplicatie Android prin care utilizatorul va putea sa isi faca programare la anumite servicii (Stomatologie, Frizerie, etc).

Utilizatorul va putea selecta serviciul gasindu-l in apropierea sa folosind Google Maps sau cautandu-l dupa nume. El va trimite o cerere catre baza de date cu programarea dorita, iar detinatorul serviciului va putea accepta sau refuza respectiva cerere, de pe o alta aplicatie care ii va permite administrarea serviciului sau si modificarea datelor in baza de date a serviciului. Utilizatorului i se va actualiza baza de date locala cu programarea facuta si acesta va putea sa isi seteze notificari pentru a fi anuntat in prealabil.

Studentul va proiecta aceasta aplicatie si isi asuma ca va implementa urmatoarele optiuni, in plus fata de functionalitatea propriu-zisa a aplicatiei: autentificarea cu Google si Facebok folosind API-urile specifice fiecarei metoda, baza de date a utilizatorului (locala), conectarea la Google Maps pentru vizualizarea serviciilor care sunt in apropierea utilizatorului, posibilitatea administratorului de serviciu de a vedea lista programarilor pe care le are.

**3. Resurse folosite la dezvoltarea proiectului:**

Android Studio, Java, Google API, Facebook API, SQLite, Firebase, Picasso.

**4. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline:**

Programare obiect orientata, Inginerie software pentru comunicatii, Tehnici de programare in internet

**5. Proprietatea intelectuală asupra proiectului aparține:** studentului

**6. Data înregistrării temei:** 2019-01-14 21:36:08

**Conducător(i) lucrare,**

Conf. dr. ing. Eduard-Cristian POPOVICI

semnătura: .....

**Student,**

semnătura: .....

**Director departament,**

Conf. dr. ing. Eduard POPOVICI

semnătura: .....

**Decan,**

Prof. dr. ing. Cristian NEGRESCU

semnătura: .....

Cod Validare: **c4229c4893**



Copyright © 2019 , *Cristian Retea*

Toate drepturile rezervate

Autorul acordă UPB dreptul de a reproduce și de a distribui public copii pe hârtie sau electronice ale acestei lucrări, în formă integrală sau parțială.



## Declarație de onestitate academică

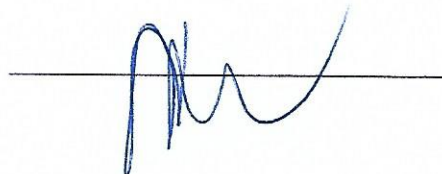
Prin prezenta declar că lucrarea cu titlul “*Aplicație Android de organizare a activităților bazată pe localizare și servicii web*”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică și Telecomunicații* programul de studii *Rețele și Software de Telecomunicații* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 06.09.2019

Absolvent *Cristian Retea*

A handwritten signature in blue ink, consisting of a series of loops and curves, positioned above a horizontal line.





# Cuprins

<b>Introducere.....</b>	<b>15</b>
Motivația .....	15
Recomandări și detalii .....	15
<b>Contextul actual.....</b>	<b>17</b>
I.1 Aspecte sociale.....	17
I.2 Specificarea cerinței .....	18
I.3 Aplicații asemănătoare și concurența.....	18
I.4 Concluzii .....	20
I.5 Obiective .....	21
<b>Analiza .....</b>	<b>23</b>
II.1 Analiza etapelor dezvoltării .....	23
II.1.1 Analiza pieței .....	23
II.1.2 Stabilirea conceptului inițial .....	24
II.1.3 Design-ul și flow-ul aplicației .....	24
II.1.4 Prototipare .....	25
II.1.5 Dezvoltarea aplicației .....	26
II.2 Analiza dezvoltării aplicației din punct de vedere tehnic .....	27
II.2.1 Java .....	27
II.2.2 XML .....	28
II.3 Android Studio .....	28
II.3.1 Activities.....	28
II.3.2 Layouts .....	30
II.3.3 Structura și fereastra proiectului .....	32
II.4 Firebase.....	34
II.4.1 Cloud Firestore .....	35
II.4.2 Authentication .....	35
II.4.3 Cloud Storage .....	36
II.4.4 Cloud Messaging .....	36
II.5 Github .....	36
<b>Proiectare .....</b>	<b>39</b>
III.1 Proiectare de nivel înalt.....	39

III.1.1 Biblioteci .....	40
III.1.2 Frameworks .....	40
III.2 Proiectarea aplicației pentru clienți .....	41
III.2.1 Vederea de ansamblu a claselor .....	41
III.2.3 Activitățile și schițele lor pentru aplicația destinată clienților .....	42
III.2.4 Activitățile și schițele lor pentru aplicația destinată administratorilor serviciilor .....	51
<b>Implementarea.....</b>	<b>55</b>
IV.1 Organizarea claselor în pachete .....	55
IV.2 Clasele modele .....	56
IV.3 Permițiuni .....	57
IV.4 Implementarea interfeței .....	59
IV.4.1 Adaptoarele .....	60
IV.4.2 Gestionarea locației .....	62
IV.5 Gestionarea bazei de date .....	65
IV.6 Baza de date locală .....	67
IV.7 Autentificare .....	68
IV.6 Notificări .....	68
IV.7 Transmiterea informației între activități .....	68
<b>Ghid de utilizare a aplicației .....</b>	<b>71</b>
<b>Concluzii.....</b>	<b>75</b>
Metode de monetizare .....	75
Probleme întâmpinate .....	75
Contribuțiile personale .....	76
Dezvoltări ulterioare .....	76
Rezumat .....	76
<b>Bibliografie.....</b>	<b>77</b>
Anexa 1 – Diagrama flux de activități destinată clienților .....	78
Anexa 2 – Diagrama flux de activități a aplicației destinate administratorilor serviciilor .....	79

## Listă de figuri

Figura I.1: Posesori de telefoane mobile inteligente .....	17
Figura I.2: Aplicații mobile descărcate anual.....	18
Figura I.3: Uber .....	19
Figura I.4: Glovo .....	20
Figura I.5: CalendariumApp .....	20
Figura I.6: KayusApp.....	20
Figura II.1: Diagrama de context .....	24
Figura II.2: Elemente de design ale unei aplicații mobile .....	25
Figura II.3: Caz de utilizare general .....	26
Figura II.4: Ciclul de viață al activităților .....	29
Figura II.5: ViewGroups and Views .....	30
Figura II.6: XML code, text .....	31
Figura II.7: ConstraintLayout în editorul XML .....	31
Figura II.8: Fereastra proiectului Android .....	32
Figura II.9: Gradle .....	33
Figura II.10: Servicii furnizate de Firebase.....	34
Figura II.11: Diferența dintre Firebase și modul de lucru tradițional.....	34
Figura II.12: Securitate în Firebase .....	35
Figura II.13: Cloud Storage.....	36
Figura II.14: Modul de lucru al VCS .....	37
Figura III.1: Diagrama de clase .....	41
Figura III. 2: Layout AuthenticationMethodsActivity – activity_authentication_methods.xml .....	42
Figura III.3: Layout EmailAndPasswordAuthenticationActivity – activity_email_and_passwordauthentication.xml .....	43
Figura III.4: Layout SettingsActivity – activity_settings.xml .....	44
Figura III.5: Layout DashboardActivity – activity_dashboard.xml .....	45
Figura III.6: Layout NearbyFragment + Toolbar – nearby_tab_fragment.xml.....	46
Figura III.7: Layout ProviderList Cardview – cardview_provider_list_item.xml .....	46
Figura III.8: Layout ProviderActivity – activity_provider.xml .....	47
Figura III.9: Layout ServiceList Cardview – cardview_service_list_item.xml .....	47
Figura III.10: Layout ServiceActivity activity_service.xml .....	48
Figura III.11: Layout MyProfileActivity activity_my_profile.xml.....	49
Figura III.12: Layout Cardview Appointment – cardview_appointment_list_item.xml.....	49
Figura III.13: Layout AppointmentActivity - activity_appointment.xml .....	50
Figura III.14: NavigationDrawer - navigation_menu.xml.....	51
Figura III.15: MainActivity - activity_main.xml .....	52
Figura III.16: CardView Appointment - cardview_appointment_list_item.xml.....	53
Figura III.17: Appointment - activity_appointment.xml.....	53
Figura IV.1: Organizarea pachetelor și claselor .....	55
Figura V.1: Metode de autentificare.....	72
Figura V.2: Ecranul principal.....	72

Figura V.3: Ecranul unui furnizor de servicii și ecranul unui serviciu .....	73
Figura V.4: Obținerea unei programări .....	73
Figura V.5: Ecranul profilului utilizatorului și ecranul programării .....	74
Figura V.6: Ecranele aplicației administratorului unui serviciu.....	74

## Lista acronimelor

SUA – Statele Unite ale Americii  
API – Application Programming Interface  
XML – Extensible Markup Language  
JVM – Java Virtual Machine  
POO – Programare Obiect Orientata  
GC – Garbage Collector  
IDE - Integrated development environment  
UI – User Interface  
UX – User Experience  
HTML - Hypertext Markup Language  
ID - Identificator  
FCM – Firebase Cloud Messaging  
SDK – Software Development Kit  
NOSQL – Not Only Structured Query Language  
VCS – Version Control System  
AVD – Android Virtual Device  
GPS – Global Positioning System  
GSON – Google library for JSON  
JSON - JavaScript Object Notation



# Introducere

## *Motivația*

În prezent, timpul reprezintă, poate, cea mai importantă resursă de care se bucură oamenii din lumea întreagă. Ca orice resursă, timpul este limitat, de aici venind și tendința oamenilor de a încerca să câștige timp prin simplificarea unor sarcini care, de multe ori, consumă mai mult timp decât am vrea. Datorită faptului că tot mai mulți oameni dispun de telefoane mobile inteligente, una din soluțiile prin care se vrea reducerea timpului pierdut este dezvoltarea de aplicații mobile.

Am realizat această aplicație, fiind motivat, atât de micșorarea timpului pe care îl pierdem atunci când căutăm anumite servicii precum și promovarea acestor servicii. Fiecare dintre noi a fost pus măcar o dată în situația de a fi nevoit să caute urgent un serviciu sau altul, moment în care singura soluție este să căutăm pe internet servicii disponibile, ca mai apoi să sunăm la respectivii pentru informații și pentru programări. Cu ajutorul acestei aplicații, acest proces lent va fi transformat într-un proces mult mai rapid și facil, atât din punctul de vedere al clientului, cât și din punctul de vedere al administratorului serviciului respectiv.

## *Recomandări și detalii*

Pentru simplificare, se recomandă ca cititorul să se gândească la următoarele servicii care au fost folosite ca modele în momentul dezvoltării aplicației:

- Stomatologie;
- Reparații de instalații sanitare;
- Servicii de curățare la domiciliu;
- Saloane de îngrijire personală;

Aceste servicii nu acoperă decât o mică parte din marea de servicii care există pe piață. În esență, această aplicație poate fi folosită pentru orice serviciu care presupune programări.

Lucrarea de față este structurată pe mai multe capitole care surprind elemente esențiale în proiectarea, implementarea și utilizarea aplicației.

**Capitolul I** va cuprinde o scurtă descriere a aplicațiilor asemănătoare care își propun, într-un fel sau altul, să rezolve probleme asemănătoare. În plus, acest capitol va conține aspectele sociale care au stat la baza ideii dezvoltării acestei aplicații.

**Capitolul II** va conține analiza etapelor necesare dezvoltării unei aplicații Android, tehnologiile utilizate în proiectarea și dezvoltarea aplicației, cea mai importantă fiind Android Studio și bibliotecile și interfețele aferente, precum cele de design, de baze de date sau de autentificare. În acest capitol voi prezenta cum am ajuns să folosesc aceste tehnologii, alternative la acestea și pași ce trebuie urmați pentru a le putea face să funcționeze fără conflicte.

**Capitolul III** are ca scop detalierea proiectării aplicației.

**Capitolul IV** va conține detalii despre dezvoltarea aplicației, în principal implementarea tuturor funcționalităților și metodele necesare funcționării lor.

**Capitolul V** conține modul în care aplicația trebuie folosită, cu exemplul unui caz de utilizare general.

În ultima parte voi prezenta concluzii, probleme generale întâmpinate și moduri și îmbunătățiri pentru dezvoltarea ulterioară a aplicației





# Capitolul I

## Contextul actual

### ***1.1 Aspecte sociale***

Trăim într-o perioadă în care devine din ce în ce mai greu să facem față cu evoluția tehnologică. Una din tendințele în creștere este utilizarea telefoanelor mobile inteligente. Conform unui studiu realizat recent, numărul posesorilor de telefoane inteligente a depășit, în anul 2018, granița de 3 miliarde, reprezentând aproximativ 39% din populația lumii. Se așteaptă ca până în anul 2021, acest număr să ajungă la 3.8 miliarde de utilizatori, ceea ce presupune o oportunitate extraordinară pentru dezvoltatorii de aplicații. La acest număr se poate adăuga și numărul de tablete achiziționate în acești ani.

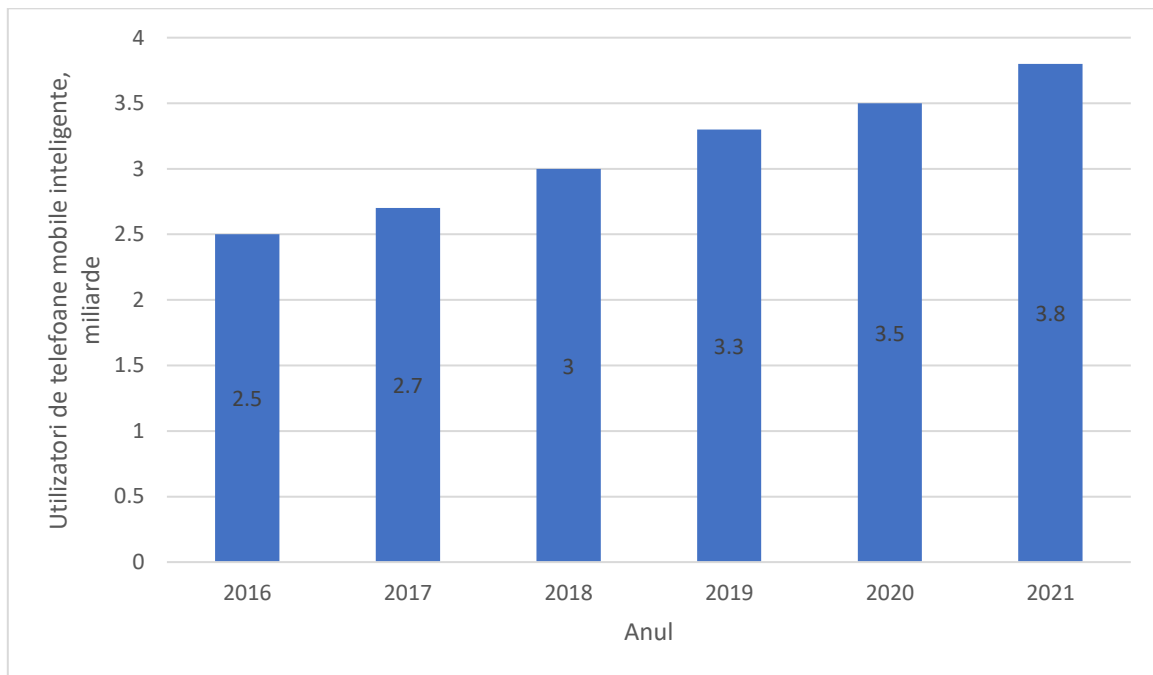


Figura I.1: Posesori de telefoane mobile inteligente

Un studiu realizat în SUA susține că, în medie, un om își verifică telefonul o dată la fiecare 12 minute. Mai mult, 10% din oameni își verifică telefonul chiar mai des, o dată la fiecare 4 minute. Studiul susține că 90% din timpul pe care utilizatorii îl petrec folosind telefoanele mobile inteligente este de fapt timp petrecut folosind aplicații mobile.

Pentru a avea o aplicație de succes trebuie îndeplinite două cerințe: utilizatorii să descarce aplicația, iar mai apoi, să o și folosească. Deși aceste cerințe par a fi ușor de realizat, multitudinea de aplicații prezente pe piață, fiecare dintre ele servind un anume scop, face ca aplicațiile noi să ajungă destul de greu descărcate.

Statistica spune că utilizatorii au descărcat 178.1 miliarde de aplicații în toată lumea pe telefoanele lor, până în 2017. Faptul că aceste telefoane devin din ce în ce mai disponibile tuturor

categoriilor sociale, cât și competiția acerbă între producătorii de telefoane mobile inteligente nu face altceva decât să asigure că această piață va crește în continuare. Deși numărul maxim de aplicații existente pe Google Play Store ajunsese la 3.6 miliarde, eforturile Google s-au îndreptat spre eliminarea aplicațiilor de calitate inferioară de pe piață. Din cauza faptului că majoritatea aplicațiilor de pe Google Play Store sunt gratuite, acestea necesită un model de business foarte sănătos.

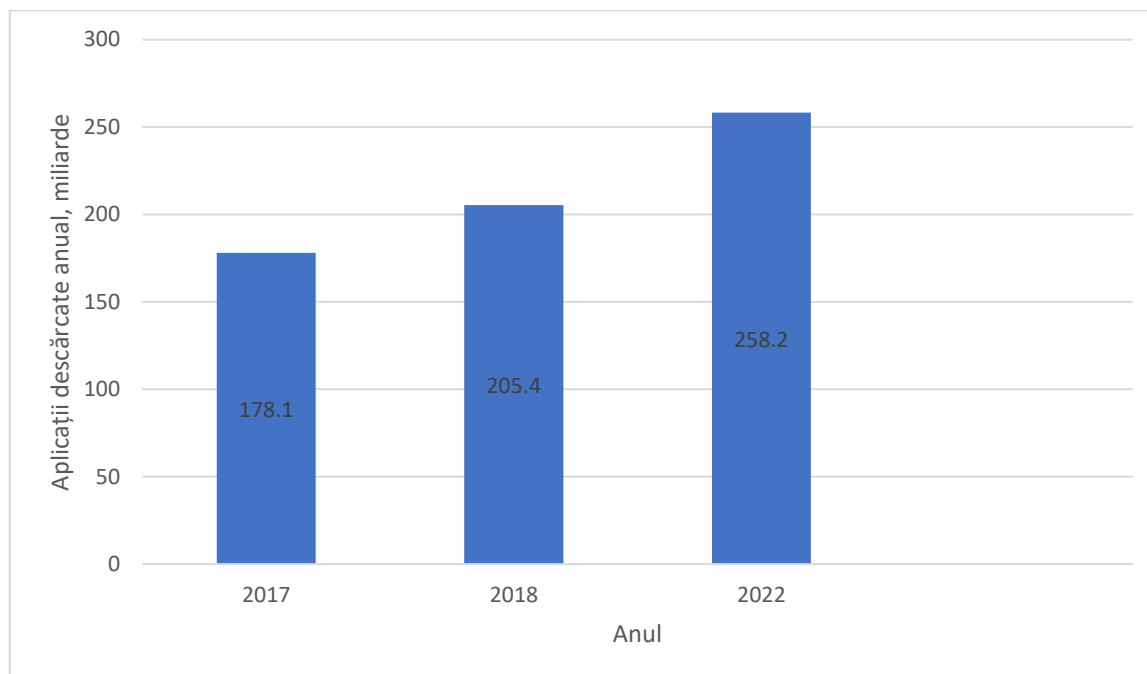


Figura I.2: Aplicații mobile descărcate anual

## ***1.2 Specificarea cerinței***

Proiectul constă într-o aplicație Android prin care utilizatorul va putea să își facă programare la anumite servicii (Stomatologie, Frizerie, etc).

Utilizatorul va putea selecta serviciul găsimdu-l în apropierea sa folosind Google Maps sau căutându-l după nume. El va trimite o cerere către baza de date cu programarea dorită, iar deținătorul serviciului va putea accepta sau refuza respectiva cerere, de pe o altă aplicație care îi va permite administrarea serviciului sau și modificarea datelor în baza de date a serviciului. Utilizatorului i se va actualiza baza de date locală cu programarea făcută și acesta va putea să își seteze notificări pentru a fi anunțat în prealabil.

Funcționalități suplimentare vor fi autentificarea cu Google și Facebook folosind API-urile specifice fiecărei metodă, baza de date a utilizatorului (locală), conectarea la Google Maps pentru vizualizarea serviciilor care sunt în apropierea utilizatorului, posibilitatea administratorului de serviciu de a vedea lista programărilor pe care le are.

## ***1.3 Aplicații asemănătoare și concurența***

În continuare vor fi descrise câteva aplicații care doresc să ușureze viața utilizatorului, prin simplificarea metodelor de obținere a unor servicii la un nivel superior. Aceste aplicații au servit ca inspirație la nivelul ideii și proiectării aplicației dezvoltate. La fel ca aplicația pe care încercăm să o dezvoltăm prin această lucrare, aceste aplicații au apărut datorită unor nevoi care au fost identificate

prin cercetarea pieței, prototiparea și încercarea mai multor soluții asemănătoare care ar fi putut, într-o anumită măsură să rezolve problemele. Un lucru cert este că pentru a ajunge la o soluție inovatoare, care să rezolve cu adevărat probleme oamenilor într-un mod ușor de înțeles și folosit este nevoie de mult timp și resurse.

### ***Uber/Bolt/Yango***

Acestea sunt unele dintre cele mai folosite aplicații în acest moment din categoria aplicațiilor care oferă un serviciu: *transportul de persoane*. Aplicațiile urmăresc să ofere clienților o calitate superioară curselor obișnuite de taxi, precum și facilitarea realizării comenzilor. Utilizatorul alege de unde este preluat și unde este dus, urmând apoi să plătească cu cardul, direct din aplicație sau cash.

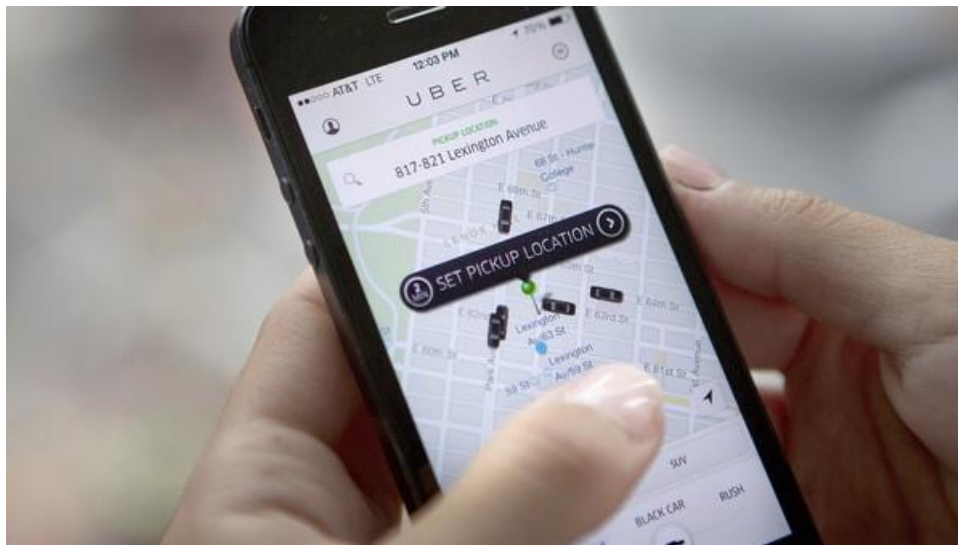


Figura I.3: Uber

### ***Uber Eats/Glovo/foodpanda***

Aceste aplicații fac parte și ele din categoria aplicațiilor care oferă un serviciu: *livrarea de mâncare*. Aplicațiile urmăresc să livreze mâncare rapid prin curieri care ajung la locațiile restaurantelor și clienților prin intermediul mașinilor, bicicletelor sau trotinetelor electrice. Utilizatorul stabilește unde trebuie livrată mâncarea, alege unul dintre restaurantele partenere, alege mâncarea din meniul disponibil direct în aplicație, urmând să plătească cu cardul, direct în aplicație, sau cash.

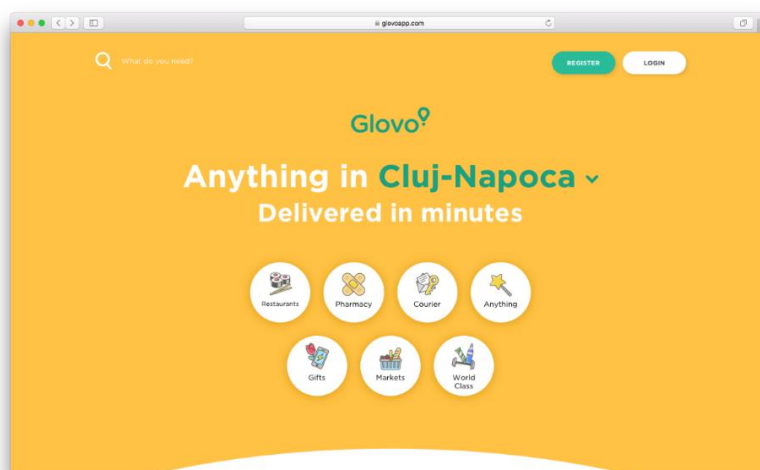


Figura I.4: Glovo

## ***Calendarium***

Această aplicație își propune să rezolve aceleași probleme pe care le rezolvă aplicația de față: management-ul programărilor la diferite servicii. *Calendarium* este o aplicație web, care oferă administratorilor posibilitatea de a adăuga și edita programări prin intermediul unui panou de control aferent serviciilor lor.

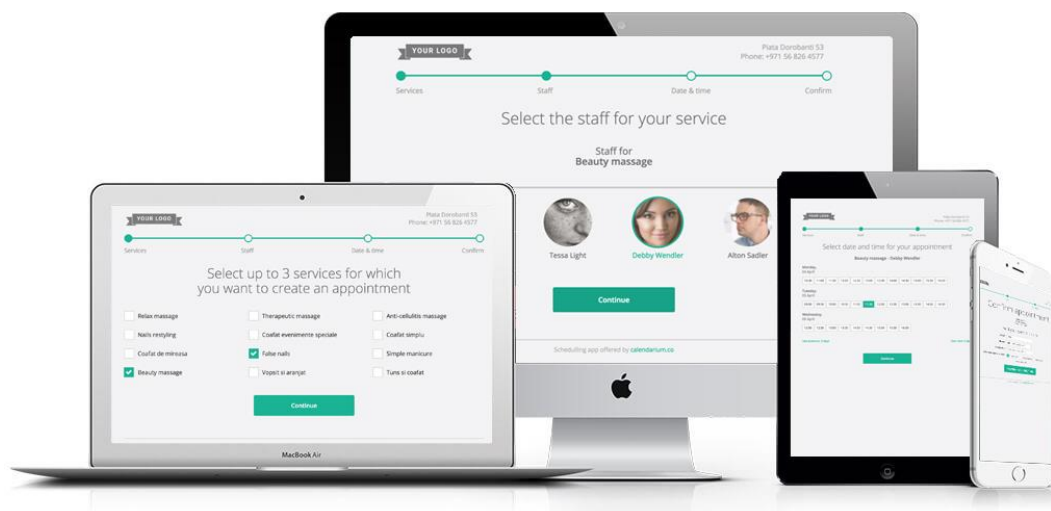


Figura I.5: CalendariumApp

## ***KayusApp***

Această aplicație oferă posibilitatea utilizatorului de a își face programare la diferite saloane de înfrumusețare. În continuare, nici această aplicație nu dispune de o varietate mare de servicii de care clienții pot beneficia.

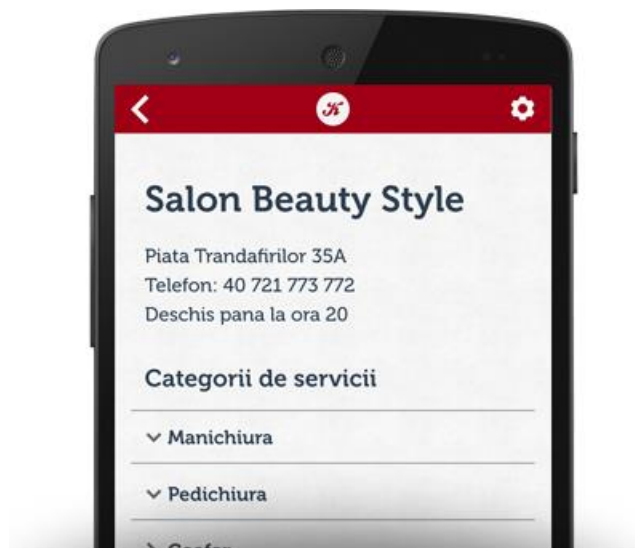


Figura I.6: KayusApp

## ***I.4 Concluzii***

Dezvoltarea aplicațiilor mobile este o piață care mai are mult până a ajunge la saturare. Această perioadă în care încercăm să ne bazăm din ce în ce mai mult pe tehnologie face ca aplicații care par rudimentare să fie folosite în întreaga lume.

Această perioadă a consumatorismului face ca oamenii să vrea să încerce din ce în ce mai multe aplicații, tocmai din dorința de a achiziționa ceva nou. Ușurința prin care un utilizator poate să descarce și, mai apoi, să se descotorosească de anumite aplicații oferă multor dezvoltatori oportunitatea ca aplicațiile lor să fie încercate.

Deși există foarte multe aplicații pe piață care își propun să ofere utilizatorilor diferite tipuri de servicii, nu există o aplicație consacrată care să ofere utilizatorului posibilitatea de a căuta mai multe tipuri de servicii, pentru ca apoi să își poată face programări. Faptul, însă, că există o multitudine de aplicații care oferă servicii mult mai nișate validează ideea că utilizatorul obișnuit caută în continuare metode de a-și facilita sarcinile prin intermediul unor aplicații specializate.

O altă concluzie este faptul că există aplicații de tip calendar în care administratorii pot să modifice programul manual, însă prin aplicația dezvoltată urmărim automatizarea acestui proces, reducând factorul uman în acest aspect la un nivel mult mai scăzut decât este în acest moment.

## ***1.5 Obiective***

Pornind de la conceptul aplicațiilor mobile care oferă servicii, această lucrare își propune abordarea dezvoltării unei aplicații care să automatizeze pe cât posibil procesul de obținere a programărilor la diferite servicii, precum și oferirea unui mediu structurat în care se pot găsi cu ușurință mai mulți furnizori ale acestor servicii.

De asemenea, aplicația își propune integrarea serviciilor de localizare, astfel încât utilizatorul să poată vedea unde se află serviciile respective.



## Capitolul II

### Analiza

#### ***II.1 Analiza etapelor dezvoltării***

Crearea aplicațiilor este un proces complex care poate dura de la câteva zile la câțiva ani, putând implica de la un dezvoltator până la o companie întreagă. Dezvoltarea aplicațiilor se face de obicei în echipe. Deși nu există un set clar de reguli și pași ce trebuie urmați pentru a realiza o aplicație de succes, există totuși etape care conferă liniaritate procesului de creare. În condițiile în care aceste etape sunt efectuate, putem spune că ne asigurăm că am făcut toate eforturile pentru realizarea unei aplicații necesare, utilă și de succes. Vom considera că etapele succed etapele de idee (cele în care descoperim ce fel de aplicație dorim să dezvoltăm), iar ele vor fi următoarele:

- analiza pieței și stabilirea publicului țintă;
- stabilirea conceptului inițial împreună cu funcționalitățile (feature-urile) principale ale aplicației;
- schițarea design-ului și flow-ului aplicației;
- producerea unui prototip care să fie testat și acceptat de utilizatori;
- începerea dezvoltării aplicației pe baza informațiilor aflate în pașii precedenți;
- finisarea aplicației;
- securitate, ofuscare și publicare;

##### ***II.1.1 Analiza pieței***

Această etapă incipientă presupune studiul pieței actuale. În această etapă încercăm să validăm ideea de la care pornim. Pentru validarea ideii trebuie să identificăm dacă există cu adevărat o problemă în piață, și anume problema pe care încercăm să o rezolvăm. Eșuarea identificării acestei probleme sugerează că, de fapt, aplicația nu ar rezolva o problemă reală, deci ar fi foarte greu să găsim utilizatori dornici de a o încerca.

În momentul în care identificăm problema în rândul utilizatorilor trebuie să ne putem întreba: *Problema va fi sau nu rezolvată de aplicație?*. În cazul în care răspunsul este *nu*, acest lucru înseamnă că premisa de la care am plecat este greșită. Dacă răspunsul este *da*, putem trece la cercetarea pieței, căutând alte aplicații care să rezolve această problemă, încercând să descoperim funcționalitățile principale ale acestora și cum rezolvă aplicațiile concurente problema respectivă. Pentru ca aplicația dezvoltată să aibă succes, aceasta trebuie să difere de cele deja existente, având cât mai multe elemente inovatoare.

În urma acestor pași, trebuie luate decizii în privința publicului pe care aplicația îl va avea ca țintă. Dezvoltarea unei aplicații fără a avea un public țintă bine stabilit poate duce la idei care s-ar bate cap în cap, în momentul proiectării acestuia. Aspecte importante în stabilirea acestuia sunt:

- vârsta;
- categoria socială din care fac parte;
- mediul în care locuiesc;

În cazul de față, lucrarea își propune ca publicul țintă să fie persoanele tinere cu venit mediu-mare, care locuiesc în zona urbană a țării. Pentru moment, aplicația va fi proiectată pentru a fi folosită în România.

### II.1.2 Stabilirea conceptului inițial

Această etapă reprezintă trecerea de la stadiul de *idee* la stadiul de *concept*. Pentru creionarea conceptului trebuie luate în considerare entitățile participante și operațiunile prin care acestea interacționează cu aplicația.

În următoarea figură va fi prezentat conceptul inițial al aplicației. Acest concept prezintă cele două entități participante, *clientul* și *furnizorul*, și modul în care acestea interacționează între ele prin intermediul aplicației.

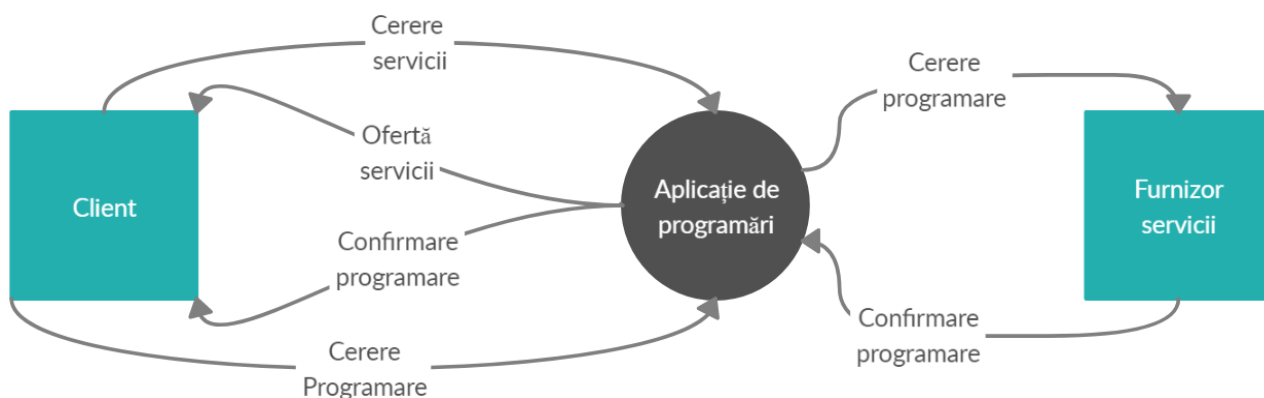


Figura II.1: Diagrama de context

Stabilirea funcționalităților principale în acest stadiu ne permite să ne gândim atât la design-ul aplicației cât și la resursele necesare pentru realizarea acestora.

### II.1.3 Design-ul și flow-ul aplicației

În această etapă trebuie alese, în mare, elementele, atât vizuale cât și funcționale, ce vor ghida utilizatorul în cadrul aplicației. Aceste elemente trebuie alese și poziționate strategic astfel încât ele să confere o utilizare intuitivă a aplicației, lipsită de ambiguitate.

Deși majoritatea deciziilor par ușor de luat, utilizatorii care vor testa aplicația nu au de unde să știe detalii pe care dezvoltatorul le cunoștea în momentul implementării funcționalităților. Un exemplu simplu este faptul că, deși în multe aplicații imaginile pot fi accesate prin apăsare, pentru majoritatea utilizatorilor un buton ar fi mult mai intuitiv. Aceste probleme apar în momentul în care prototipul este dat la testat.





Figura II.2: Elemente de design ale unei aplicații mobile

O traducere a cuvântului *flow*, din engleză, ar fi curent sau curs. Pentru o aplicație este foarte important ca utilizatorul să fie ghidat spre funcționalitățile principale, iar *drumul* pe care acesta îl parcurge spre diferitele funcționalități să fie ușor de urmat. Ambiguitatea cursului aplicației ar face ca utilizatorul să devină confuz, să nu știe cum ajunge la o funcționalitate importantă a acesteia.

Flow-ul aplicației constă proiectarea aplicației sub formă de schiță sau desen, fie în varianta clasică, pe hârtie, fie în variantele mai noi, folosind aplicații precum Adobe XD oferă o varietate mare de elemente deja consacrate. Prima iterație a flow-ului aplicației poate fi văzută, atât cea din perspectiva clientului cât și cea din perspectiva administratorului serviciului în anexele **1** și **2**.

#### **II.1.4 Prototipare**

Prototiparea reprezintă procesul prin care majoritatea ideilor, funcționalităților și design-urilor sunt puse cap la cap, astfel încât prototipul să poată fi oferit spre testare unor utilizatori. În această etapă de testare se urmărește validarea celor mai sus menționate.

Un prototip poate fi o fractură de aplicație sau niște design-uri inteligente care să conțină una sau mai multe funcționalități. În această etapă nu se urmărește modul de implementare al acestor funcționalități sau corectitudinea codului sursă, important fiind ca utilizatorul să înțeleagă despre ce este vorba pentru a putea da feedback pe această temă.

Pentru realizarea prototipului putem considera un caz general de utilizare care să conțină majoritatea funcționalităților pe care vrem să le încercăm, și, mai apoi, validăm. Dezvoltând conceptul inițial, ne putem imagina un scenariu de folosire a aplicației de către utilizator.

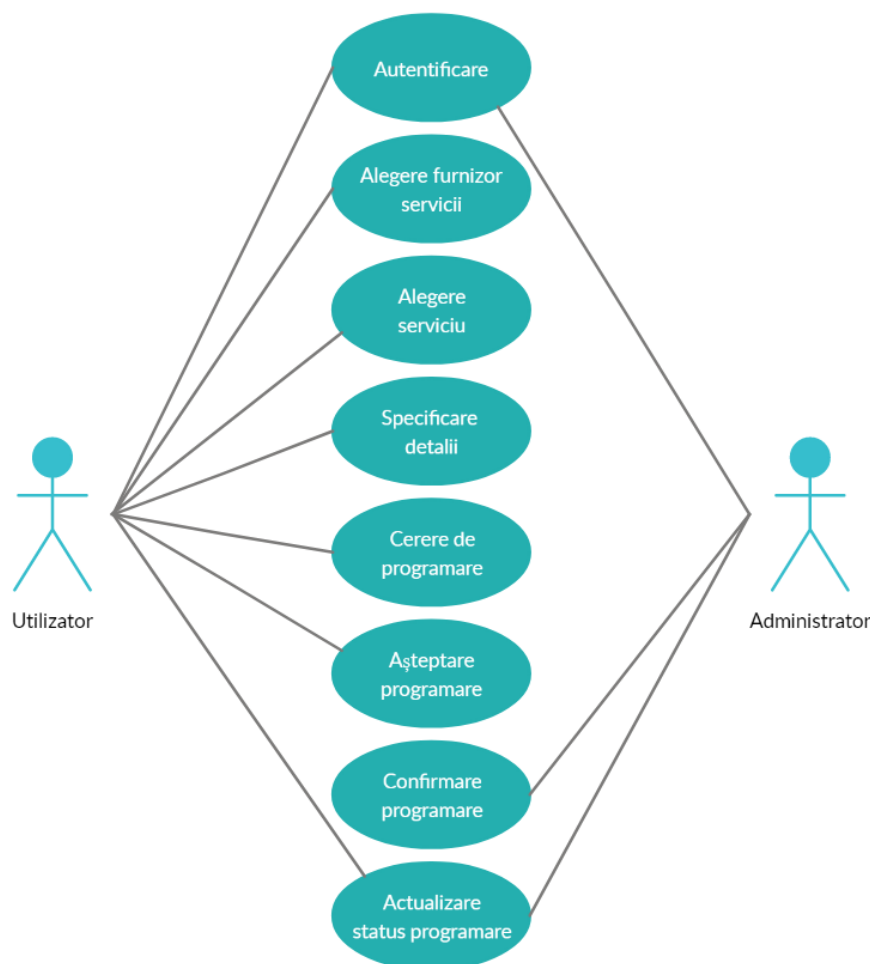


Figura II.3: Caz de utilizare general

### ***Caz de utilizare general:***

Aplicația va necesita ca ambii participanți ai tranzacției să fie autentificați înainte să poată să folosească aplicația. Ulterior, clientului i se vor prezenta furnizorii de servicii, în principal cei din apropiere, sub formă de listă. După selectarea furnizorului de servicii dorit, beneficiarul va putea alege unul dintre serviciile prezentate. În urma acestei alegeri utilizatorul va putea vedea toate detaliile acestui serviciu, putând apoi să solicite o programare setând ora și data dorită.

Aplicația va trimite o notificare administratorului serviciului, iar acesta va putea accepta sau refuza programarea respectivă. După alegerea acestuia, aplicația va actualiza statusul programării și va trimite utilizatorului corespunzător notificare cu actualizarea respectivă.

În urma acestor operații, dacă programarea a fost acceptată, atât clientului cât și furnizorului li se va sugera să își seteze o notificare pentru a nu uita de programul făcut. Această notificare va fi făcută sub formă de Reminder.

În această lucrare nu s-a efectuat procesul de prototipare.

### ***II.1.5 Dezvoltarea aplicației***

Presupunând că toate etapele precedente au fost efectuate cu succes, însemnând că majoritatea ideilor referitoare la concept, funcționalități și design au fost validate, putem trece mai departe la etapa de dezvoltare aplicației. Înainte, însă, ca această etapă să înceapă, trebuie alese toate resursele de care va fi nevoie în realizarea aplicației. Fie că este vorba de mediul de programare, de interfețe, de

bibliotecile oferite, de API-uri externe sau de elemente de design planificarea în perspectivă ajută ca, pe parcurs, să nu avem grija obținerii resurselor necesare.

În urma analizei contextului și cazului de utilizare general putem lua deja o parte din deciziile privind macro-arhitectura software folosită.

## ***II.2 Analiza dezvoltării aplicației din punct de vedere tehnic***

Dezvoltarea și implementarea tehnică a unei aplicații mobile presupune folosirea unei serii de resurse de mai multe tipuri. În procesul de creare al aplicației am folosit următoarele resurse care au fost disponibile gratuit, fiecare dintre acestea având importanță în ducerea la bun sfârșit a proiectului de față:

- Java;
- XML;
- Android Studio;
- Firebase;
- Github.

În cele ce urmează le voi prezenta, pe scurt, încercând să surprind caracteristicile principale ale acestora, utilitatea lor de ce au fost alese și cu ce m-au ajutat să rezolv anumite probleme întâlnite pe parcurs.

### ***II.2.1 Java***

Java este un limbaj de programare proiectat pentru a fi folosit în mediul distribuit al internetului. Este cel mai popular limbaj de programare în dezvoltarea aplicațiilor Android pentru telefoanele mobile inteligente, dar este printre limbajele de programare favorite și pentru dezvoltarea dispozitivelor edge sau pentru cele IoT (Internet of things).

E dificil să furnizezi un singur motiv pentru care programarea în Java a devenit ubicuă, dar o mare parte din succesul acestui limbaj este atribuită caracteristicilor sale majore, printre care se numără următoarele:

*Programele create în Java oferă portabilitate în rețea.* Codul sursă este compilat în ceea ce Java numește *BYTECODE*, care poate fi rulat oriunde în rețea, fie pe un server sau pe o aplicație de tip client care deține o *mașină virtuală Java(JVM)*. JVM-ul interpretează bytecode-ul ca fiind cod ce va fi rulat pe hardware-ul unui computer. În contrast, majoritatea limbajelor de programare, precum C++, va compila codul într-un fișier binar. Fișierele binare sunt specifice anumitor platforme, deci un program scris pentru sistemul Windows nu va fi putut rulat pe un dispozitiv bazat pe Linux sau Mac.

*Java folosește paradigma programării orientate pe obiect(POO).* Un obiect este conceput din informație precum câmpuri sau attribute, iar codul din proceduri sau metode. Un obiect poate face parte dintr-o clasă de obiecte sau poate moșteni codul unei clase. Obiectele pot fi percepute ca *substantive* la care utilizatorul poate apela prin *verbe*. O metodă reprezintă capacitățile sau comportamentele unui obiect. Pentru că design-ul acestui limbaj a fost influențat de C++, Java a fost creat în principiu ca fiind un limbaj de programare orientat pe obiecte. De asemenea, Java folosește ceea ce se numește *colector de gunoi(garbage collector)* pentru managementul ciclului de viață al obiectelor. Programatorul se ocupă de crearea obiectele, dar acest colector se ocupă de recuperarea memoriei odată ce obiectul nu mai este folosit.

*Informația este securizată.* Spre deosebire de C++, Java nu folosește pointeri, care de multe ori pot aduce nesiguranță din punct de vedere al securității. Informația convertită de către Java în bytecode nu poate fi citită de oameni. În plus, Java va rula programele într-un *sandbox* pentru a preveni schimbări venite din partea unor surse necunoscute.

*Dezvoltatorii pot învăța Java repede.* Cu o sintaxă similară cu C++, acest limbaj este relativ ușor de învățat, mai ales pentru cei care provin dintr-un background bazat pe limbajul C.

O concepție greșită este asociere dintre Java și Javascript. Deși cele două limbaje împart anumite similitudini din punctul de vedere al sintaxei, cele două sunt foarte diferite.

Ultima versiune de Java lansată este Java 12, dar o versiune mai veche, Java 8, încă se folosește și primește în continuare suport tehnic din partea Oracle.

## **II.2.2 XML**

Extensible Markup Language, pe scurt XML, este un limbaj de marcare a textului care definește un set de reguli pentru codificarea documentelor într-un format care poate fi citit atât de om cât și de mașină. Țelul de proiectare al XML pune accent pe simplitate, generalitate și pe faptul că poate fi folosit în tot internetul. Este un format de informație textuală puternic susținut prin Unicode. Deși design-ul XML se focusează pe documente, acest limbaj este larg folosit pentru reprezentarea arbitrară a structurilor de date, cum ar fi cele utilizate de serviciile web.

Mai multe sisteme de scheme există pentru a ajuta în definirea limbajelor bazate pe XML, în timp ce programatorii au dezvoltat multe interfețe pentru a procesa informațiile de tip XML.

Acest limbaj va fi folosit pentru definirea elementelor grafice care vor fi ulterior folosite în implementarea funcționalităților în interiorul aplicației dezvoltate.

## **II.3 Android Studio**

Android Studio este IDE-ul(Integrated Development Environment) oficial pentru dezvoltarea aplicațiilor Android. Acesta se bazează pe IntelliJ IDEA, iar pe lângă funcționalitățile IntelliJ, oferă în plus:

- asistarea codului;
- un sistem de construire flexibil bazat pe gradle(Gradle-based build system);
- coduri șablon pentru a putea ajuta la crearea unor funcționalități obișnuite ale aplicațiilor;
- un editor de cod bogat în resurse;
- etc.

Android Studio este un mediu de dezvoltare al aplicațiilor mobile care oferă un suport total pentru realizarea de la zero a unui proiect. Pentru o mai bună înțelegere a acestui mediu vom introduce doi noi termeni: *activități* și *scheme(layout-uri)*.

### **II.3.1 Activities**

Clasa *Activity* este o componentă crucială a aplicațiilor Android, iar modul în care activitățile sunt pornite și asamblate reprezintă o parte fundamentală a modelului de aplicație. Spre deosebire de alte paradigme de programare în care aplicațiile sunt pornite cu metoda *main()*, sistemul Android

inițiază codul din activități prin invocarea unor metode specifice de *callback* care corespund unor stadii specifice din ciclul de viață al aplicației.

Experiența aplicațiilor mobile diferă față de cea din aplicațiile de tip desktop prin faptul că, în timp ce o aplicație desktop pornește mereu în același loc, o aplicație mobilă de multe ori începe într-un mod non-determinist. Spre exemplu, dacă pornești o aplicație de email din ecranul principal, poți vedea o listă de emailuri, dar dacă folosești o aplicație de social media care îți pornește aplicația de emailuri cel mai probabil vei fi direcționat spre ecranul specific scrierii unui nou mail.

Clasa *Activity* este proiectată pentru facilitarea acestei paradigme. Când o aplicație o invocă pe cealaltă, aplicația chemătoare invocă o anumită activitate din cealaltă aplicație, în loc să cheme aplicația ca un tot. În acest mod, activitățile servesc ca puncte de intrare pentru interacțiunea aplicației cu utilizatorul. O activitate este implementată ca fiind o subclasă a clasei *Activity*.

O activitate furnizează fereastra în care aplicație desenează UI-ul. Această fereastră umple, de obicei, ecranul, dar poate fi și mai mică decât tot ecranul, plutind pe alte ferestre. În general, o activitate implementează la un moment dat un ecran în cadrul aplicației. Spre exemplu, o activitate poate implementa *ecranul de autentificare*, iar alta poate implementa *ecranul de selectare al unei poze*.

Majoritatea aplicațiilor conțin mai multe ecrane, deci sunt compuse din mai multe activități. De obicei, o activitate este specificată ca fiind *activitatea principală* (*Main Activity*), care este primul ecran care apare atunci când aplicația este pornită. Fiecare activitate poate începe o alta pentru a efectua alte acțiuni. Spre exemplu, activitatea principală a unei aplicații de mail poate afișa inbox-ul unui utilizator. De aici, aceasta poate porni alte activități care efectuează alte sarcini cum ar fi scrierea de emailuri sau deschiderea unor emailuri individuale.

Deși activitățile funcționează împreună pentru a oferi o experiență coezivă a utilizatorului, acestea sunt conectate între ele astfel încât să dependințele dintre ele să fie minime.

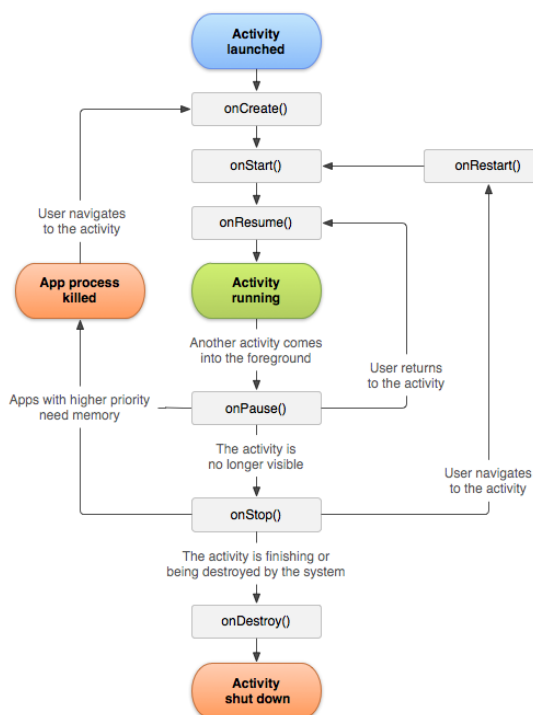


Figura II.4: Ciclul de viață al activităților

Pentru a fi folosite, activitățile trebuie să fie înregistrate *manifestul* aplicației, iar ciclul de viață al acestora trebuie să fie manageriat corespunzător. Această manageriere presupune suprascrierea metodelor corespunzătoare, precum: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onRestart()` și `onDestroy()` în funcție de ce avem nevoie să facă aplicația sau de flow-ul acesteia. Metodele acestea dau comportamentul activității în momentul în care acțiunea respectivă este efectuată, de aceea sunt numite astfel.

### II.3.2 Layouts

Un *layout*(o schemă) definește structura necesară pentru interfața utilizatorului în aplicație, cum ar fi, în activități. Toate elementele din layout sunt construite folosind o ierarhie a obiectelor *View* și *ViewGroup*. De obicei, un *view* reprezintă ceva ce utilizatorul poate vedea și cu care poate interacționa, în timp ce un *viewgroup* este un *container*(recipient) invizibil care definește structura pentru alte obiecte de tip *View* și *ViewGroup*.

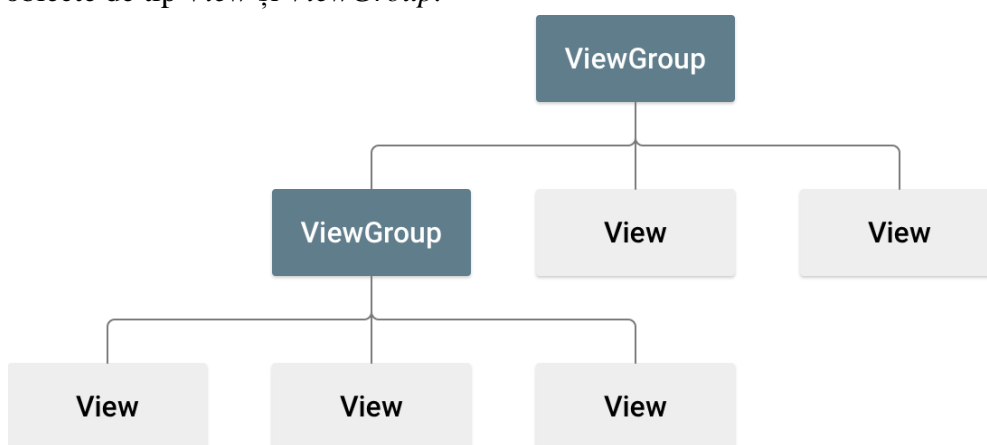


Figura II.5: ViewGroups and Views

Obiectele de tip *View* sunt de obicei numite widget-uri și pot fi de mai multe subclase, cum ar fi *Butoane*, *TextView*, etc. Obiectele de tip *ViewGroup* sunt de obicei numite layout și pot furniza mai multe tipuri de structuri, cum ar fi *LinearLayout*, *ConstraintLayout*, *RelativeLayout*, etc.

Un layout poate fi declarat în două moduri:

- prin **declararea elementelor de UI în XML**. Android furnizează un vocabular direct XML care corespunde cu clasele și subclasele de tip *View*, cum ar fi widget-urile și layout-urile;
- prin **instanțierea elementelor la momentul rulării**. Aplicația poate crea obiecte de tip *View* sau *ViewGroup* (și să manipuleze proprietățile) prin programare;

Declararea elementelor de UI în XML poate fi efectuată în două moduri, fie prin metoda drag-and-drop a widget-urilor într-un preview disponibil în editorul de interfață oferit de Android studio, fie prin declararea scrisă a acestor widget-uri, în fișierul text XML. Editorul care va fi folosit în prima variantă nu face altceva decât să scrie singur în fișierul text ceea ce noi *desenăm* în fereastra de preview. Această fereastră este vizibilă pe tot parcursul construirii interfeței vizuale, astfel încât poate fi urmărit progresul design-ului unui layout la fiecare pas, fie că decidem să îl modificăm folosind o metodă sau alta.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>

```

Figura II.6: XML code, text

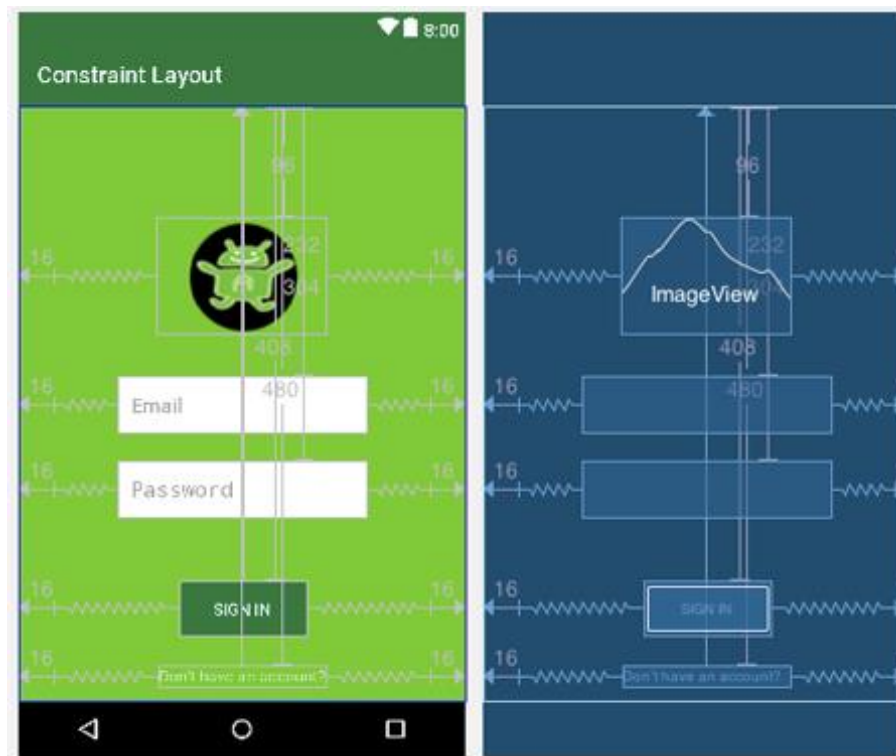


Figura II.7: ConstraintLayout în editorul XML

Declararea interfeței în XML permite separarea prezentării aplicației de codul care controlează comportamentul acesteia. Folosind fișiere de tip XML facilitează furnizarea diferitelor tipuri de layout pentru diferite orientări și dimensiuni de ecrane.

Framework-ul Android oferă flexibilitate pentru a folosi fie una sau ambele metode de construire a UI-ului aplicației. Spre exemplu, interfața poate fi declarată în XML și apoi layout-ul poate fi modificat la rulare.

Folosind vocabularul XML al Android, designul layout-urilor și elementelor de UI pe care acestea le conțin pot fi făcute rapid, în același mod în care sunt create paginile web în HTML, cu o serie de elemente, unul în celălalt.

Fiecare fișier layout trebuie să conțină exact un element rădăcină(sau părinte), care trebuie să fie un obiect de tip View sau ViewGroup. Odată ce elementul rădăcină a fost definit, pot fi adăugate obiecte de layout sau widget-uri ca fiind copii ai aceluia element părinte, construind gradual o ierarhie care va defini layout-ul.

Fiecare obiect de tip View sau ViewGroup are o varietate de attribute XML. O parte din aceste attribute sunt specifice obiectului respectiv(cum ar fi atributul `textSize`, care definește mărimea textului dintr-un `TextView`), dar aceste marea parte a acestor attribute moștenite de orice subclase ale obiectului View care extind această clasă. Unele sunt comune fiecărui obiect de tip View, pentru ca sunt moștenite din clasa View, precum atributul `id`. Fiecare dintre aceste obiecte poate avea un identificator unic de tip Integer cu care este asociat. Când o aplicație este compilată, acest ID este referențiat ca fiind Integer, dar ID-ul este alocat în interiorul layout-ului XML ca fiind String. Acest atribut este folosit în interiorul activităților pentru legarea dintr-o instanță la widget-ul din layout-ul corespunzător.

### II.3.3 Structura și fereastra proiectului

Din start, Android Studio afișează fișierele profilului în fereastra de proiect. Această fereastră arată o versiune condensată a structurii proiectului, oferind acces rapid la fișierele sursă cheie pentru respectivul proiect. Această fereastră oferă posibilitatea de a lucra ușor cu sistemul de construire bazat pe Gradle.

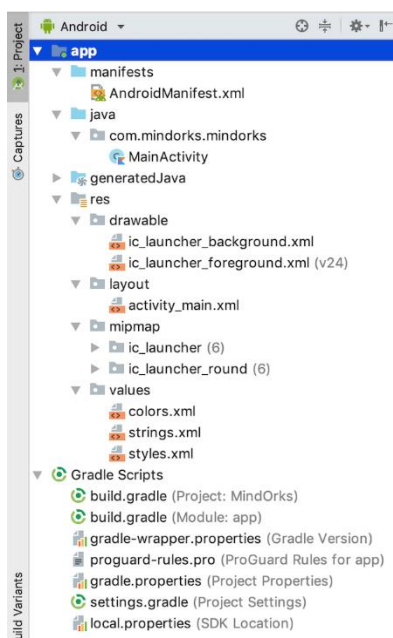


Figura II.8: Fereastra proiectului Android

Figura de mai sus reprezintă locul unde sunt toate fișierele asociate proiectului deschis. Aici, putem observa că există un singur modul de Aplicație Android care conține *manifestul*, *fișierele Java* și *fișierele res*. Fiecare aplicație trebuie să conțină un `AndroidManifest.xml`(fix cu acest nume) în fișierul părinte al proiectului.

Manifestul este un fel de registru în care trebuie descrise informațiile esențiale legate de uneltele de construire ale aplicației, despre sistemul de operare și despre Google Play. Printre altele în manifest trebuie declarate următoarele:

- numele pachetului aplicației, care este de obicei același cu cel din codul sursă;
- componentele aplicației, adică toate activitățile, serviciile, boardcast receiverele și content providerile. Fiecare componentă trebuie să descrie proprietățile de bază cum ar fi numele clasei pe care o reprezintă. În plus, pot fi definite și capacități precum configurări ale dispozitivului sau moduri în care componentele sunt pornite;



- permisiunile de care aplicația are nevoie pentru a accesa părți protejate ale sistemului și permisiuni pe care trebuie să le aibe alte aplicații pentru a avea acces la conținutul din această aplicație;
- caracteristicile de tip hardware și software necesare rulării aplicației, care afectează dispozitivele care pot instala aplicația de pe Google Play;

În fișierul de resurse, pe scurt `res`, se găsesc elementele de design ale aplicației, acesta fiind fișierul care acoperă modul de prezentare al aplicației:

- fișierul *drawable* conține imaginile de sine stătătoare ale aplicației, cele ce nu trebuie să fie descărcate;
- fișierul *layout* conține toate layout-urile necesare construirii aplicației, inclusiv elemente personalizate cum ar fi meniurile, Toolbar-urile, ProgressBar-urile, etc;
- fișierul *values* conține stiluri, stringuri și culori, care trebuiesc definite în prealabil. Considerând că, în general, obiectele de tip View trebuie să aibă unul din stilurile prestabilite ale aplicației, atribuirea unui stil la fiecare instanțiere a unui astfel de obiect este o practică greșită și anevoioasă. Stilul se referă la atribute cum ar fi: culori, dimensiune text, dimensiune margini, culoare text, aliniere, etc.;

În partea de jos a figurii se pot observa scripturile, atât cele la nivelul proiectului, cât și cele la nivelul aplicației, care ajută la construirea acesteia. Android Studio folosește Gradle, un toolkit avansat, pentru automatizarea și managementul procesului de construire, oferind posibilitatea definirii unor construcții și configurații personalizate. Fiecare configurație poate defini setul propriu de resurse și cod, refolosind părțile comune ale tuturor versiunilor aplicației. Acest plugin funcționează cu toolkitul de construcție pentru a oferi procese și setări care sunt specifice construirii și testării aplicațiilor Android.

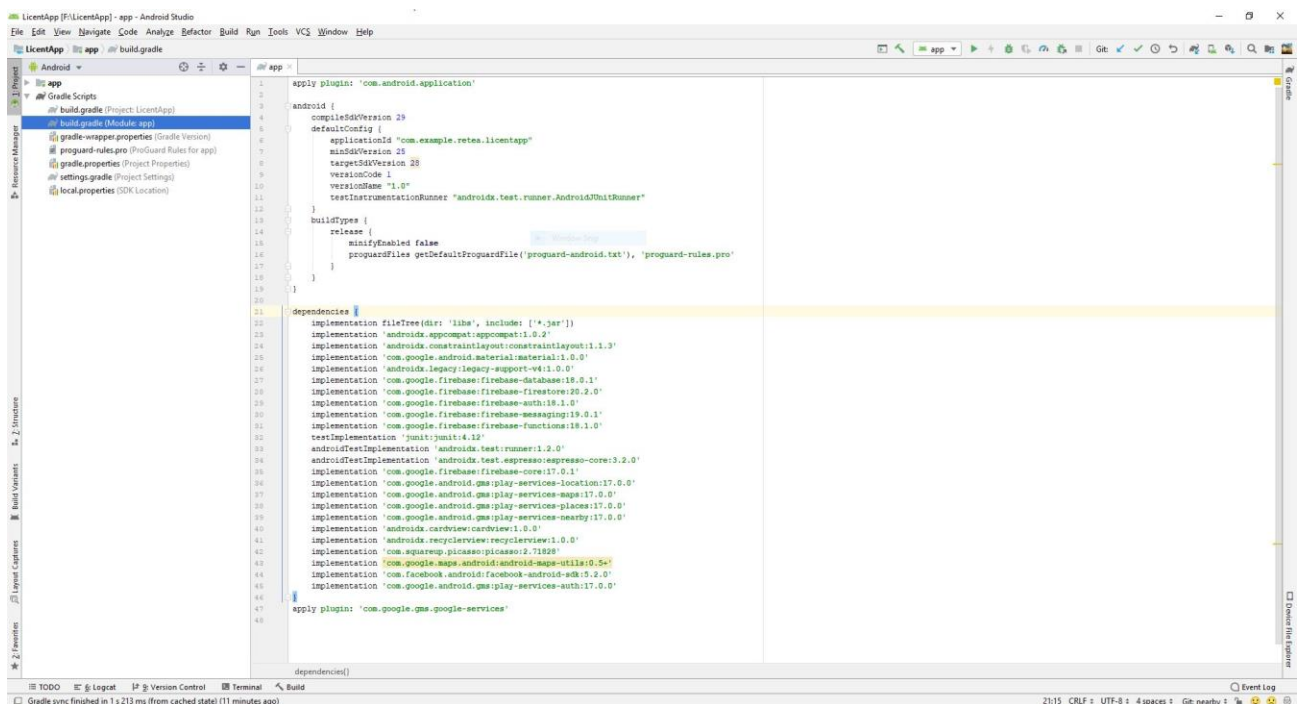


Figura II.9: Gradle

## II.4 Firebase

Firebase este o platformă, deținută de Google, care ajută la dezvoltarea, îmbunătățirea și creșterea aplicațiilor mobile, oferind o multitudine de unelte care acoperă o mare porțiune din serviciile pe care un dezvoltator de aplicații ar trebui să le construiască singur, permițându-i acestuia să se focalizeze pe experiența utilizatorului în aplicație. O parte din aceste servicii au fost utilizate și în construirea aplicației noastre.

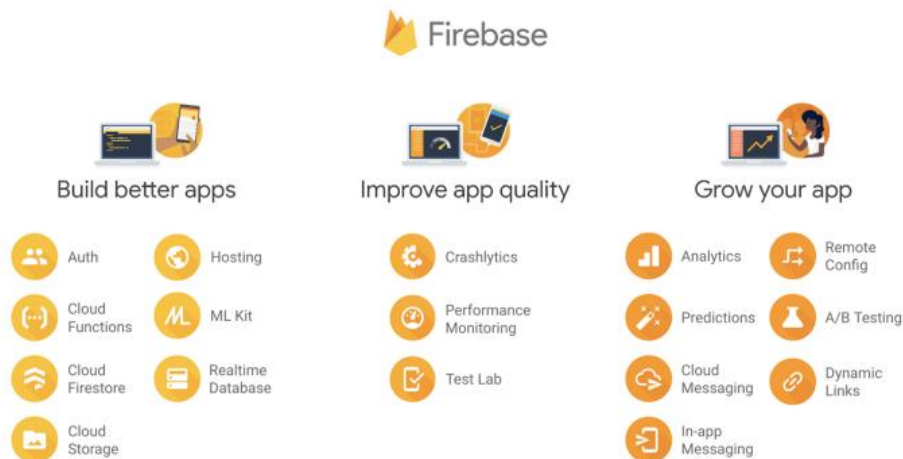


Figura II.10: Servicii furnizate de Firebase

Toate aceste servicii oferite de Firebase sunt hostate în cloud și scalează aproape fără niciun efort cu nevoile fiecărui dezvoltator de aplicații.

Prin hostarea în cloud înțelegem că toate aceste produse au componente backend care sunt păstrate, menținute și operate de Google. Clientul aplicației interacționează direct cu aceste servicii, fără să fie nevoie ca dezvoltatorul să construiască un mijloc specific de comunicare între acestea. Acest lucru este diferit față de dezvoltarea tradițională a aplicațiilor, care impune dezvoltatorului să scrie atât partea de backend cât și partea de frontend a software-ului. Cu produsele Firebase, tradiționalul backend este ocolit, punând la muncă clientul. Accesul administrativ la toate acestea este furnizat de consola Firebase.

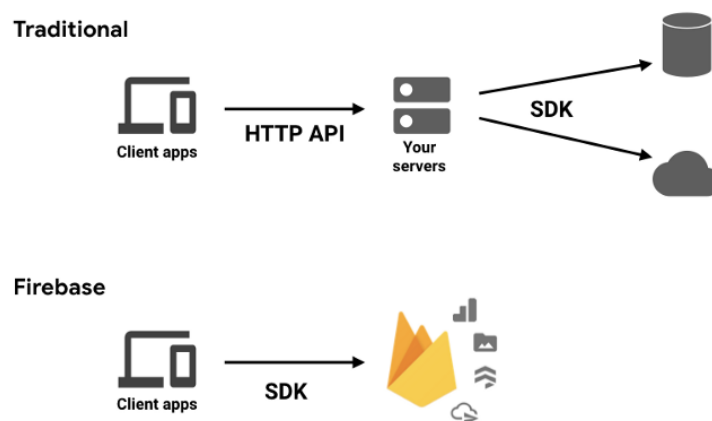


Figura II.11: Diferența dintre Firebase și modul de lucru tradițional

Pentru dezvoltarea aplicației curente am folosit o parte din serviciile oferite de Firebase printre care:

- Cloud Firestore;
- Authentication;
- Cloud Storage;
- Cloud Messaging.

### **II.4.1 Cloud Firestore**

Cloud Firestore este un serviciu de baze de date oferit de Firebase, de tip NoSQL bazat pe documente care oferă administratorului posibilitatea de a stoca, sincroniza și cere informații pentru aplicațiile mobile, la scală globală.

Acest serviciu permite structurarea datelor sub formă de colecții și documente, permite construirea ierarhiilor pentru stocare informațiilor dependente între ele. Firestore folosește query-uri expresive, facilitând obținerea informației necesare. Acest serviciu scalează la orice dimensiune de aplicație.

Firestore funcționează împreună cu SDK-urile mobile și cu un set de reguli de securitate complet, astfel încât nu este nevoie de crearea unui server de sine stătător. Folosind acest serviciu se pot sincroniza automat informațiile între toate dispozitivele datorită notificărilor livrate atunci când informația este schimbată, astfel facilitând experiența aplicațiilor în timp real.

### **II.4.2 Authentication**

Autentificarea Firebase țintește să faciliteze construirea sistemelor de autentificare, totodată îmbunătățind procesul de înregistrare și de conectare a utilizatorilor. Acesta are suport software pentru metode de autentificare prin: email și parolă, Google, Facebook și altele.

Construit de aceeași echipă care au dezvoltat metodele de autentificare Google, Smart Lock sau Chrome Password Manager, securitatea Firebase aplică expertiza Google în ceea ce privește cea mai mare bază de date din lume.

Deși implementarea unui sistem de autentificare propriu poate dura foarte mult, necesitând o echipă întreagă pentru a menține sistemul scalabil și în viitor, folosirea acestui serviciu face mult mai ușor de rezolvat această problemă.

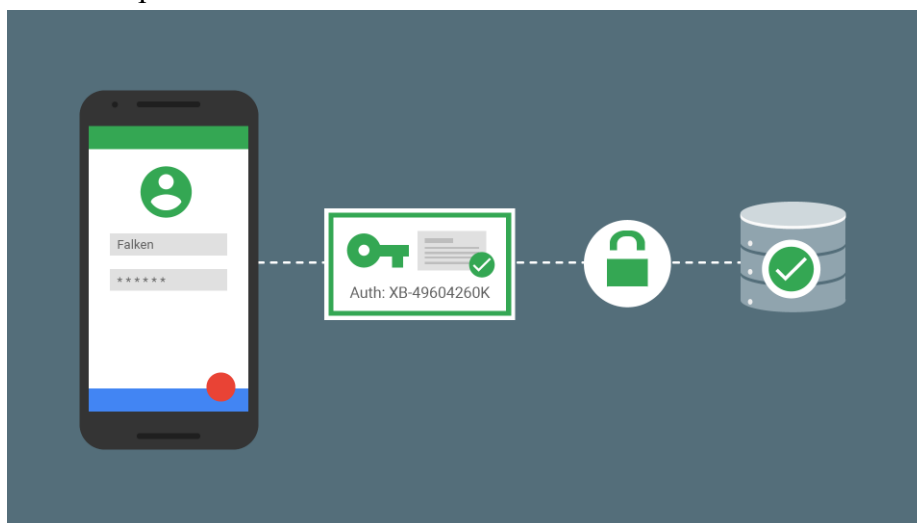


Figura II.12: Securitate în Firebase

### II.4.3 Cloud Storage

Serviciul de Cloud Storage este proiectat pentru a ajuta la administrarea aplicațiilor să stocheze și să genereze conținut cum ar fi pozele sau videoclipurile mult mai rapid. Infrastructura este construită să scaleze, indiferent dacă vorbim despre o aplicație prototip sau una de dimensiune imense precum Spotify.

Acest serviciu ține cont de conectivitatea la internet a utilizatorilor, prioritizând descărcările de date în momentul în care aceștia sunt pe rețele de tip WiFi și gestionând aceste descărcări în cazul în care conexiunea este pierdută, salvând astfel utilizatorilor timp și resurse.

Cloud Storage poate fi integrat cu metodele de Autentificare Firebase pentru a asigura un control al accesului simplu și intuitiv. Acest acces poate fi permis în funcție de identitatea utilizatorului, proprietățile fișierului(cum ar fi numele, mărimea), tipul de conținut sau alte date.

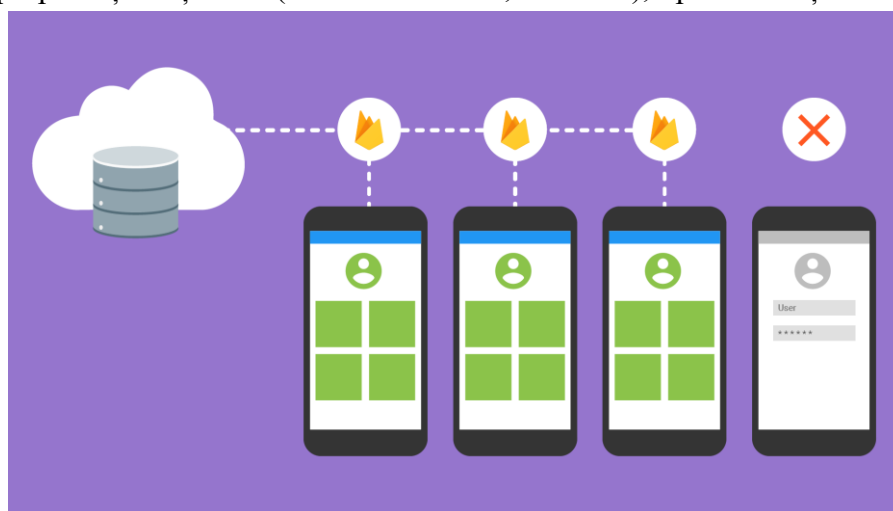


Figura II.13: Cloud Storage

### II.4.4 Cloud Messaging

Serviciul Firebase Cloud Messaging(FCM) asigură o conexiune între server și dispozitive. Acesta este un serviciu de încredere și eficient din punctul de vedere al bateriei folosite, asigurând trimiterea și primirea mesajelor și notificărilor.

FCM facilitează trimiterea mesajelor oferind posibilitatea de a ținti anumiți utilizatori sau anumite segmente de utilizatori. Serviciul este scalabil, putând trimite mesaje atât la câte un singur utilizator, până la segmente foarte mari de utilizatori.

## II.5 Github

Pentru a înțelege GitHub, trebuie înțeles mai întâi conceptul de Git. Git este un sistem open-source de control al versiunii(Version Control System) care a fost început de Linux Trovalds, aceeași persoană care a creat Linux. Git este similar altor VCS precum Subversion, CVS sau Mercurial, sau altele.

Când dezvoltatorii de aplicații creează ceva(aplicații, website-uri, etc), ei modifică codul în mod constant, aducând versiuni îmbunătățite ale lansării inițiale. Sistemele de control al versiunii păstrează aceste revizuiți ale codului, păstrând modificările într-un *repository*(loc de stocare). Acest

serviciu permite dezvoltatorilor să colaboreze ușor, permițându-le să descarce noi versiuni ale software-ului, să facă schimbări și să încarce mai apoi versiunea cu schimbările făcute înapoi. Fiecare dezvoltator poate vedea schimbările, să le descarce și să contribuie la rândul lui.

În mod similar, oameni care nu au nicio legătură cu dezvoltarea unui proiect pot descărca la rândul lor fișierele pentru a le folosi. Majoritatea utilizatorilor Linux ar trebui să fie deja familiarizați cu acest proces, considerând că aceste sisteme se folosesc pentru descărcarea anumitor fișiere necesare, mai ales în prepararea pentru compilare a unui program din codul sursă.

Git este sistemul de control al versiunii preferat de majoritatea dezvoltatorilor pentru că oferă mai multe avantaje decât celelalte sisteme disponibile. Stocază fișiere mult mai eficient și asigură integritatea acestora.

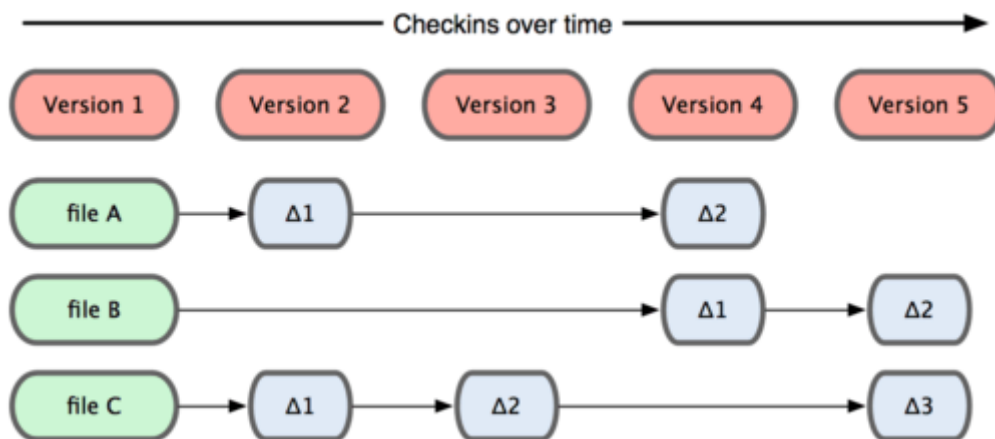


Figura II.14: Modul de lucru al VCS



## Capitolul III

### Proiectare

#### III.1 Proiectare de nivel înalt

Proiectul va fi compus din două aplicații Android diferite care vor comunica între ele, una fiind destinată clienților, celor ce vor să găsească un anumit serviciu, iar cea de-a doua fiind destinată furnizorilor de servicii, cei ce vor să pună la dispoziție servicii pentru primul tip de utilizatori. Pentru evitarea unor confuzii, în continuare, ne vom referi la utilizatorii primei aplicații sub denumirea de *clienți*, iar pe utilizatorii celei de-a doua aplicații îi vom numi *administratori*.

În continuare voi prezenta modurile în care acest proiect poate fi construit, aducând argumente pro și contra pentru fiecare dintre ele:

*Modul 1:* Se va construi o singură aplicație care va conține funcționalitățile pentru ambele tipuri de utilizatori. Această aplicație va decide în momentul autentificării utilizatorului ce conținut trebuie arătat acestuia.

Argumente pro:

- nu trebuie realizarea a două aplicații;
- pentru ambele tipuri de utilizatori va fi nevoie de aceleași clase model, aceleași elemente de design și aceleași metode de obținere a informației din baza de date;

Argumente contra:

- anumite activități construite pentru un tip de utilizator nu vor putea fi accesate de celălalt tip, deci vor fi inutile după momentul autentificării;
- va trebui să avem mare grijă pentru separarea conținutului, astfel încât un utilizator de un tip să nu poată vedea sub nicio formă conținutul celuilalt tip;
- aplicația va avea dimensiuni mai mare, conținând ambele tipuri de funcționalități, schițe și activități;

*Modul 2:* Se vor construi două aplicații, una pentru clienți și una pentru administratori. Fiecare aplicație va fi construită să îndeplinească funcționalitățile necesare tipului de utilizator pentru care este proiectată.

Argumente pro:

- separarea conținutului;
- securitate mai bună;
- dimensiuni mai mici;
- fiecare utilizator va descărca doar aplicația de care are nevoie;
- deși aplicațiile sunt separate, putem recicla unele clase din prima aplicație, astfel încât implementarea nu va fi mult mai complicată;

Argumente contra:

- necesitatea găsirii unei soluții prin care cele două aplicații să comunice între ele;

În urma analizei celor două metode am decis să merg mai departe alegând al doilea mod de proiectare deoarece argumentele găsite sugerau că proiectarea a două aplicații este mai intuitivă și eficientă.

### **III.1.1 Biblioteci**

Aplicația pe care o voi dezvolta se va folosi de resursele specificate în capitolul anterior, modul de prezentare al acestora va fi realizat în limbaj XML. Pentru elementele grafice voi folosi biblioteca Google care respectă standarde de design general valabile pentru aplicații mobile și alte câteva biblioteci necesare pentru alte elemente grafice. Menționez că aceste biblioteci sunt imperios necesare pentru că dorim ca aplicația să aibă widget-uri comune, astfel încât interfața să fie ușor de citit și experiența utilizatorului să fie intuitivă.

Bibliotecile pe care le-am folosit în proiectarea schițelor:

- 'androidx.constraintlayout:constraintlayout:1.1.3'
- 'androidx.cardview:cardview:1.0.0'
- 'androidx.recyclerview:recyclerview:1.0.0'
- 'com.google.android.material:material:1.0.0'

Pentru funcționalitățile legate de servicii de localizare și hărți vor fi folosite următoarele biblioteci oferite tot de Google:

- 'com.google.android.gms:play-services-location:17.0.0'
- 'com.google.android.gms:play-services-maps:17.0.0'
- 'com.google.android.gms:play-services-places:17.0.0'
- 'com.google.android.gms:play-services-nearby:17.0.0'
- 'com.google.maps.android:android-maps-utils:0.5+'

Din multitudinea de servicii puse la dispoziție de Firebase, am folosit serviciile următoare, urmate de bibliotecile necesare pentru implementarea lor:

- Bază de date în timp real - 'com.google.firebase:firebase-firestore:21.0.0'
- Autentificare - 'com.google.firebase:firebase-auth:19.0.0'
- Notificări și mesaje - 'com.google.firebase:firebase-messaging:20.0.0'
- Stocare fișiere - 'com.google.firebase:firebase-storage:19.0.0'
- Core - 'com.google.firebase:firebase-core:17.1.0'

Alte biblioteci necesare:

- Picasso, pentru încărcarea eficientă a pozelor - 'com.squareup.picasso:picasso:2.71828'
- autentificare prin Facebook - 'com.facebook.android:facebook-android-sdk:5.2.0'
- autentificare prin Google - 'com.google.android.gms:play-services-auth:17.0.0'
- bibliotecă pentru compatibilitatea cu dispozitive cu versiuni mai vechi ale sistemului de operare - 'androidx.appcompat:appcompat:1.0.2' și 'androidx.legacy:legacy-support-v4:1.0.0'

Menționez că toate aceste biblioteci sunt scrise sub forma unor dependențe de tip Gradle, acest format fiind necesar pentru ca aplicația să descarce informația necesară înainte de construirea ei.

### **III.1.2 Frameworks**

Pentru dezvoltarea aplicațiilor voi folosi framework-uri java și Android puse la dispoziție de IDE-ul Android Studio. Voi folosi framework-uri pentru a realiza funcționalitățile aplicațiilor. Printre acestea se numără Framework urile corespunzătoare: autocompletării codului, colecțiilor(List, ArrayList, etc.), ActivityManager, LocationManager, StorageManager. Unul dintre cele mai importante framework-uri pe care îl voi folosi este cel ce permite testarea aplicației pe un Android



Virtual Device(AVD). În momentul detalierii implementării aplicațiilor voi prezenta cum am folosit aceste framework-uri pentru realizarea acestora.

### III.2 Proiectarea aplicației pentru clienți

Am ales să încep cu proiectarea aplicației pentru clienți deoarece aceasta dispune de mai multe funcționalități, automat însemnând că ar dura mai mult proiectarea și implementarea acesteia. În plus, principala funcționalitate a celei de-a doua aplicații este confirmarea sau refuzarea programărilor, deci mai întâi aveam nevoie de funcționalitatea care permite înregistrarea acestor programări.

#### III.2.1 Vederea de ansamblu a claselor

După ce am analizat diagrama de context și diagrama cazului general de utilizare am dedus principalele entități care participă la realizarea funcționalităților și modelele de obiecte necesare interacțiunii dintre acestea.

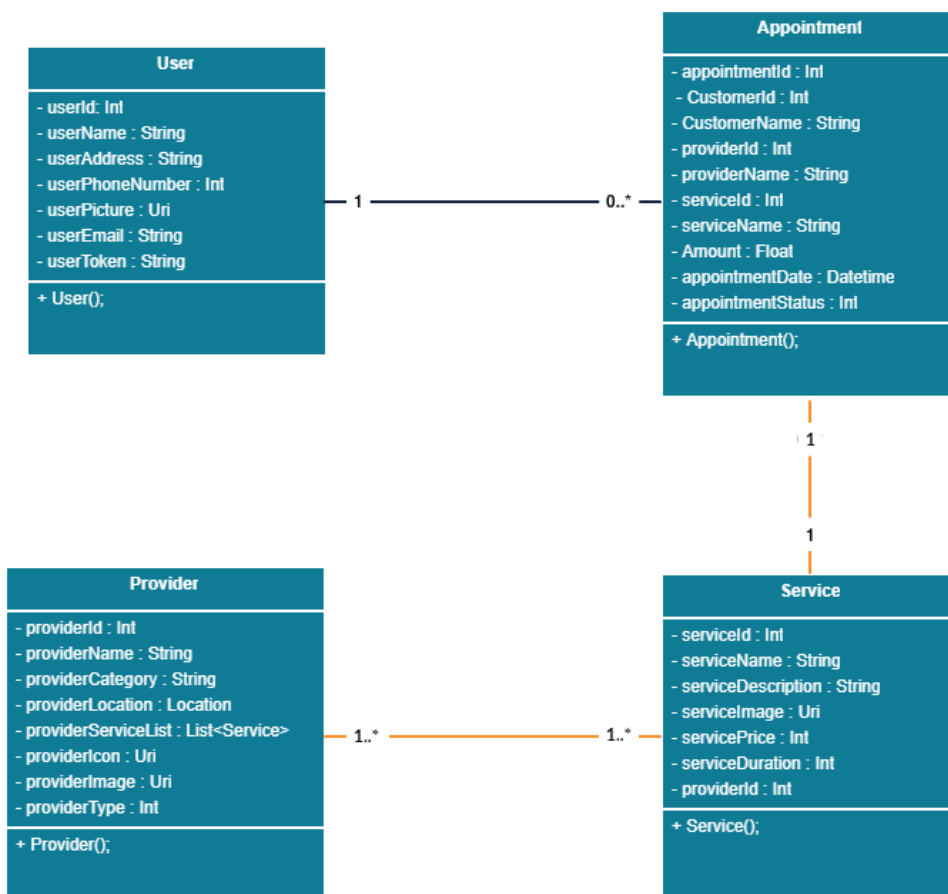


Figura III.1: Diagrama de clase

Aceste patru clase reprezintă modelele inițiale pentru realizarea aplicațiilor. Toate câmpurile acestor clase sunt private, iar metodele care apar în schemă reprezintă constructorii pe care îi vom folosi pentru instanțierea acestor clase. Cel mai probabil, clasele vor avea doi constructori, unul abstract și altul propriu-zis, care va necesita adăugarea tuturor câmpurilor pentru crearea unei noi instanțe. Voi încerca să folosesc cât mai puțin constructorii abstractizați deoarece instanțele ce rezultă

din aceștia au valori **null** în toate câmpurile, astfel încât posibilitatea apariției erorilor crește, iar chiar dacă evitarea erorilor de tip NPE nu necesită decât verificarea obiectelor înainte de folosirea lor, acest lucru este anevoios și consumă resursele aplicației.

### III.2.3 Activitățile și schițele lor pentru aplicația destinată clienților

Pachetul *activities* este cel mai stufos pachet pe care îl avem, deoarece avem nevoie de mai multe activități pentru realizarea tuturor funcționalităților pe care ni le propunem.

Am ales ca activitatea de tip LAUNCHER, activitatea de pornire a aplicației să fie *StartActivity*, o aplicație fără interfață vizuală în care nu facem altceva decât să solicităm utilizatorului să ne permită să folosim resursele necesare, printre care: permisiunea de folosire a internetului, permisiunea de folosire a locației, permisiunea de folosire a GPS-ului. Tot în această clasă dorim obținerea locației dispozitivului în momentul în care utilizatorul oferă permisiunile necesare.

#### Atenție!

Aici utilizatorul este înștiințat de faptul că aplicația nu funcționează dacă acesta nu furnizează permisiunile necesare. În acest caz utilizatorul poate decide să ne ofere permisiunile necesare sau să părăsească aplicația. Tot în această aplicație se verifică dacă utilizatorul care tocmai a accesat aplicația este sau nu autentificat. În cazul în care acesta nu este autentificat, va fi redirecționat spre activitatea de alegere a metodei de autentificare.

*AuthenticationMethodsActivity* nu este altceva decât o activitate de tranziție, în care utilizatorul poate decide metoda prin care dorește să se autentifice alegând din cele trei tehnici pe care le vom implementa: Email și parolă, autentificare prin Google și autentificare prin Facebook.

Layout-ul aceste aplicații nu va conține decât metodele de a ajunge spre activitatea de autentificare dorită: *EmailAndPasswordAuthenticationActivity*, *FacebookAuthenticationActivity* sau *GoogleAuthenticationActivity*.

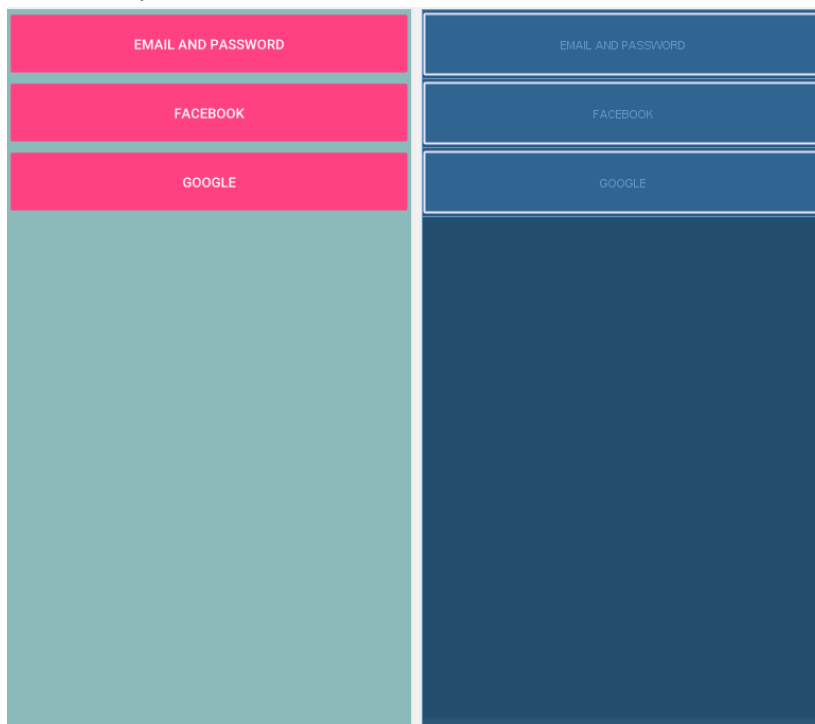


Figura III. 2: Layout *AuthenticationMethodsActivity* – *activity\_authentication\_methods.xml*

În cele ce urmează voi prezenta metoda de autentificare clasică, prin email și parolă. Layout-urile celorlalte metode de autentificare seamănă foarte mult cu acesta, diferă doar partea de jos a schiței care conține butoanele.

În clasa *EmailAndPasswordAuthenticationActivity* vom avea două obiecte de tip View și anume EditText, unul pentru scrierea adresei de email și celălalt pentru parolă. Importante mai sunt cele două butoane, cel pentru autentificare și cel pentru crearea contului. În primul rând, ambele butoane vor necesita ca cele două câmpuri să fie completate în concordanță. În al doilea rând, cele două butoane vor implementa metode care să verifice în baza de date dacă există sau nu un utilizator cu credențialele adăugate. Celelalte elemente precum, imaginea sau textele de deasupra acestei bucăți se vor schimba în funcție de operațiunile butoanelor.

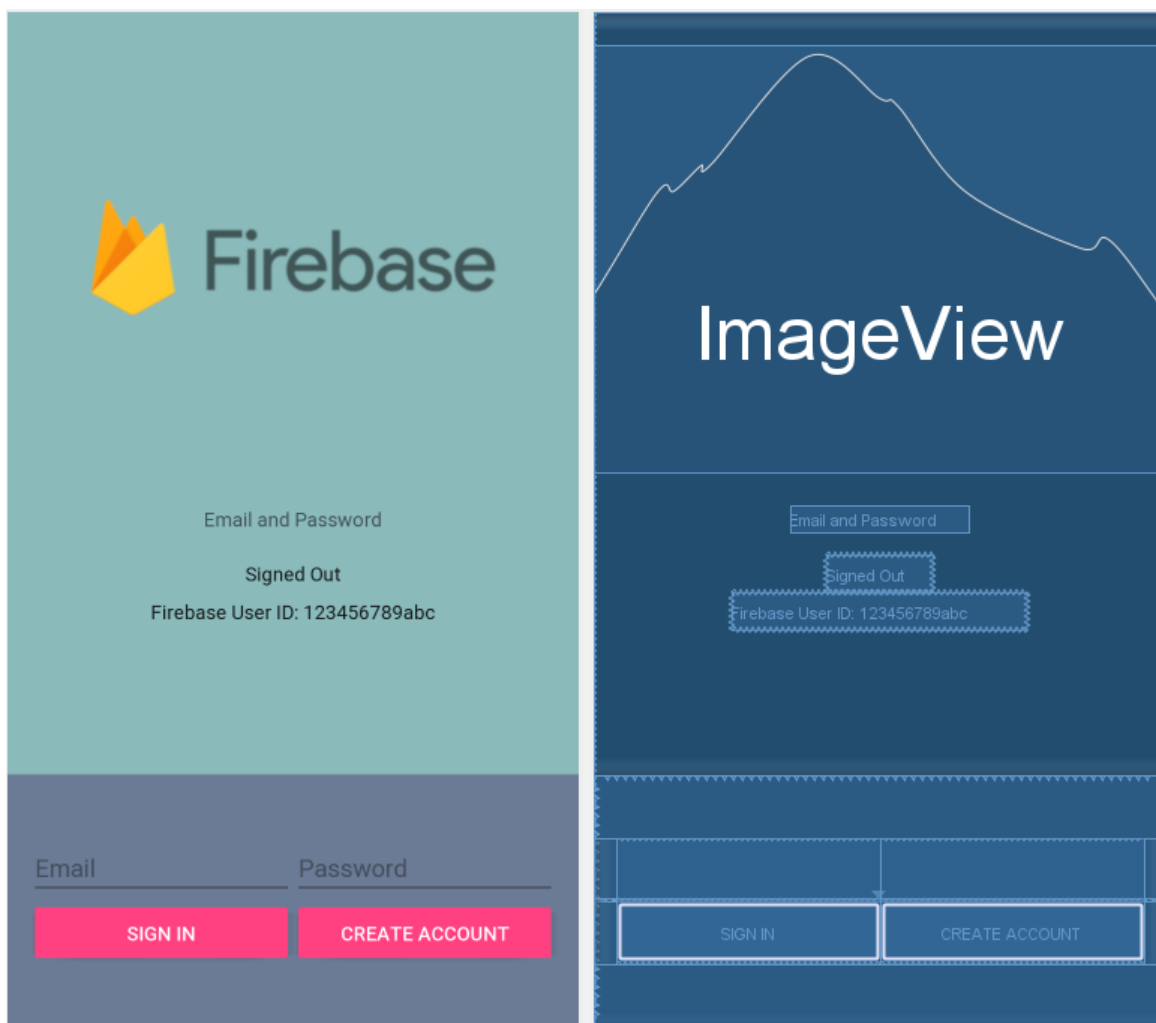


Figura III.3: Layout EmailAndPasswordAuthenticationActivity –  
activity\_email\_and\_passwordauthentication.xml

Dacă utilizatorul își creează un cont nou, acesta va trebui să confirme operația prin validarea unui email trimis pe adresa de email inserată. După această validare, el va fi îndrumat spre *SettingsActivity*, o activitate în care i se va cere completarea unor detalii suplimentare pentru realizarea ulterioară a programărilor. Această activitate mai poate fi accesată și din profilul propriu al clientului, din activitatea *MyProfileActivity*.

În activitatea de setări, utilizatorul poate să își schimbe adresa sau să adauge numărul de telefon, numele sau avatarul. Adăugarea acestora se va prin completarea obiectelor de tip EditText, apoi setarea lor prin butoanele corespunzătoare. În ceea ce privește adăugarea pozei, utilizatorului i se vor cere permisiuni pentru accesarea galeriei sau camerei foto. Dacă acesta oferă permisiunile respective, are posibilitatea de a alege o poză pentru avatarul său.

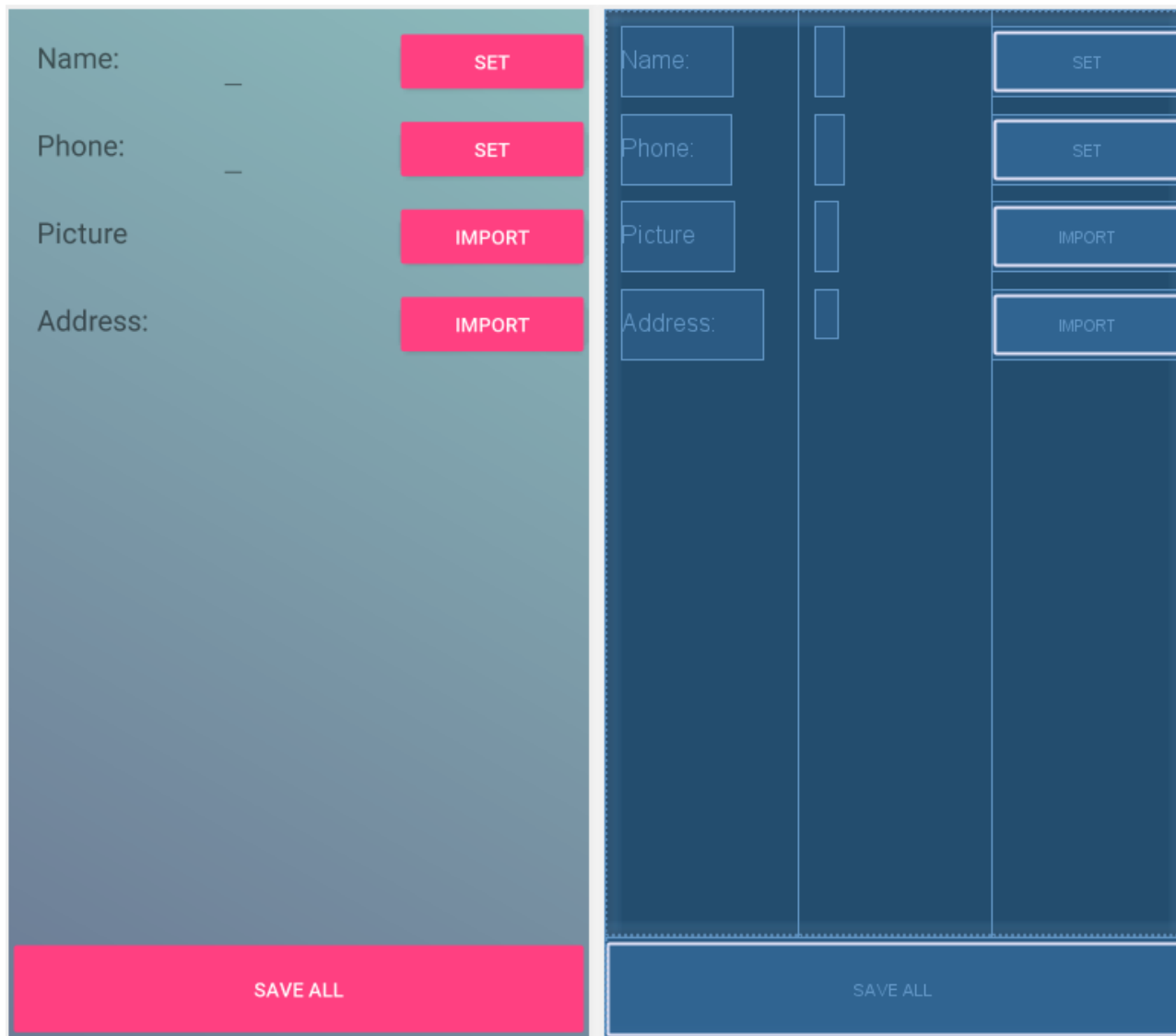


Figura III.4: Layout SettingsActivity – activity\_settings.xml

Pentru schimbarea locației, în urma apăsării butonului Import, un dialog în formă de hartă va solicita adăugarea unei alte locații. După ce au fost setate aceste detalii, utilizatorul poate folosi în mod normal aplicația împreună cu toate funcționalitățile sale.

În momentul în care toate aceste date au fost setate prin folosirea metodelor descrise mai sus, utilizatorul poate apăsa pe butonul *Save All*, prin care se va trimite o cerere de modificare a utilizatorului curent cu detaliile inserate în câmpurile de mai sus. În momentul apăsării butonului, se va verifica dacă câmpurile respective sunt completate, pentru a nu trimite mai departe valori de tip null.

Dacă autentificarea sau crearea contului au avut succes, utilizatorul va fi direcționat către activitatea *DashboardActivity*, de unde acesta poate alege tipul de servicii de care are nevoie, fie un serviciu care necesită deplasarea sa la sediul furnizorului, cum ar fi servicii de tip stomatologie sau

salon de înfrumusețare sau un serviciu care necesită deplasarea unui echipaj la adresa clientului, cum ar fi servicii de instalații sanitare, dezinsecție sau altele. Pentru navigarea mai departe spre serviciile dorite trebuie apăsată una din elementele care pot fi observate mai jos.



Figura III.5: Layout DashboardActivity – activity\_dashboard.xml

Cele două elemente sunt alcătuite dintr-un părinte de tip `CardView` care conține, mai întâi, un layout liniar cu orientare orizontală, în care sunt puse o imagine și un alt layout liniar cu orientare verticală. În layout-ul liniar cel din urmă observăm cele două widget-uri `TextView` și anume un titlu și o descriere. Am folosit această înșiruire de layout-uri liniare deoarece desenarea acestora consumă mult mai puține resurse decât alt tip de layout-uri părinte tocmai datorită faptului că elementele sunt adăugate consecutiv în funcție de orientare.

În plus, aici afișăm, pentru prima dată în aplicație, locația dispozitivului, obținută în `StartActivity`.

După alegerea tipului de serviciu dorit, utilizatorul va fi redirectionat spre activitatea care afișează lista furnizorilor cerută. În esență indiferent ce variantă este aleasă, clientul va ajunge tot la activitatea principală a aplicației, `MainActivity`. Această activitate va conține doar o bară de tip

Toolbar de unde vom solicita unul dintre cele trei fragmente disponibile: NearbyFragment, FavouritesFragment sau SearchFragment.

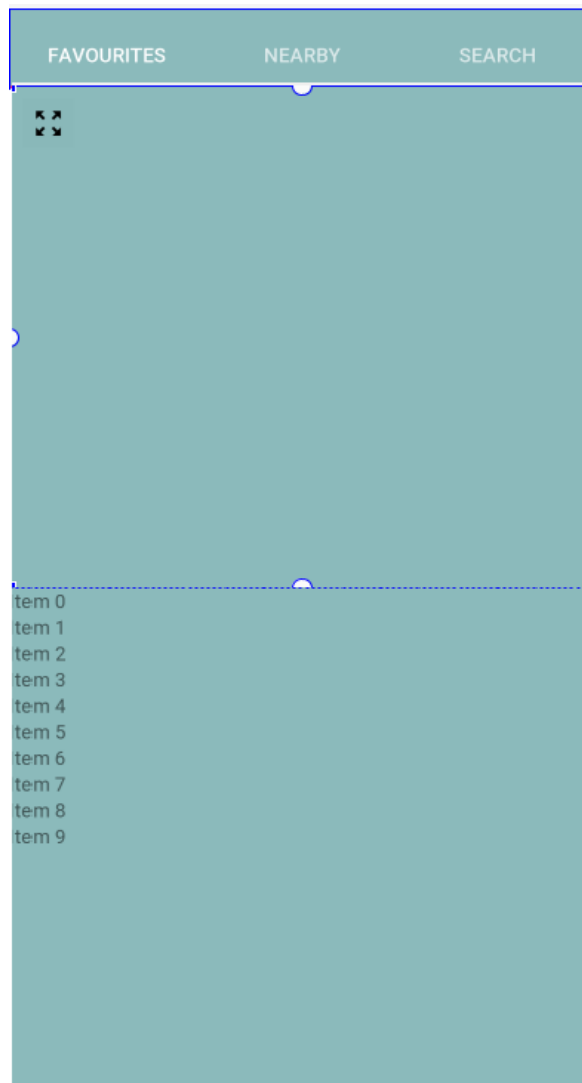


Figura III.6: Layout NearbyFragment + Toolbar – nearby\_tab\_fragment.xml

Layout-ul acestui fragment va fi împărțit în două bucăți. În partea de sus va fi un obiect de tip MapView, unde se va putea vizualiza harta și unde vor fi marcate locațiile furnizorilor de servicii cu markere personalizate cu iconițele lor. Se poate observa în colțul din stânga sus un buton creat și pus în acel loc cu ideea de a implementa opțiunea de a mări harta, astfel încât acest MapView să acopere tot Fragmentul.

În partea de jos a layout-ului va fi un obiect de tip RecyclerView, un obiect care ne permite afișarea obiectelor dintr-o listă, sub o anumită formă, ori un design specific Android, ori unul creat de noi. Acest RecyclerView este scrollable. Fiecare obiect din această listă va urma pattern-ul următorului design:

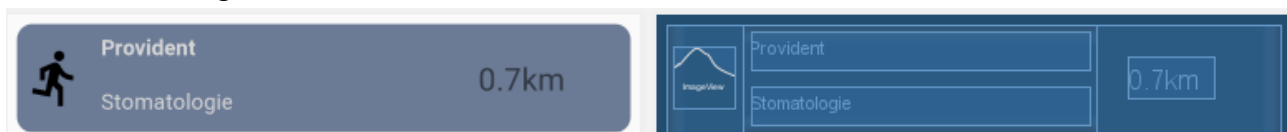


Figura III.7: Layout ProviderList Cardview – cardview\_provider\_list\_item.xml

În urma apăsării unui element din listă sau a unui marker de pe hartă, va fi accesată activitatea *ProviderActivity*, care va reprezenta pagina furnizorului de servicii din cadrul aplicației. Layout-ul părinte va fi un *ScrollView*, deci această activitate va fi scrollable.

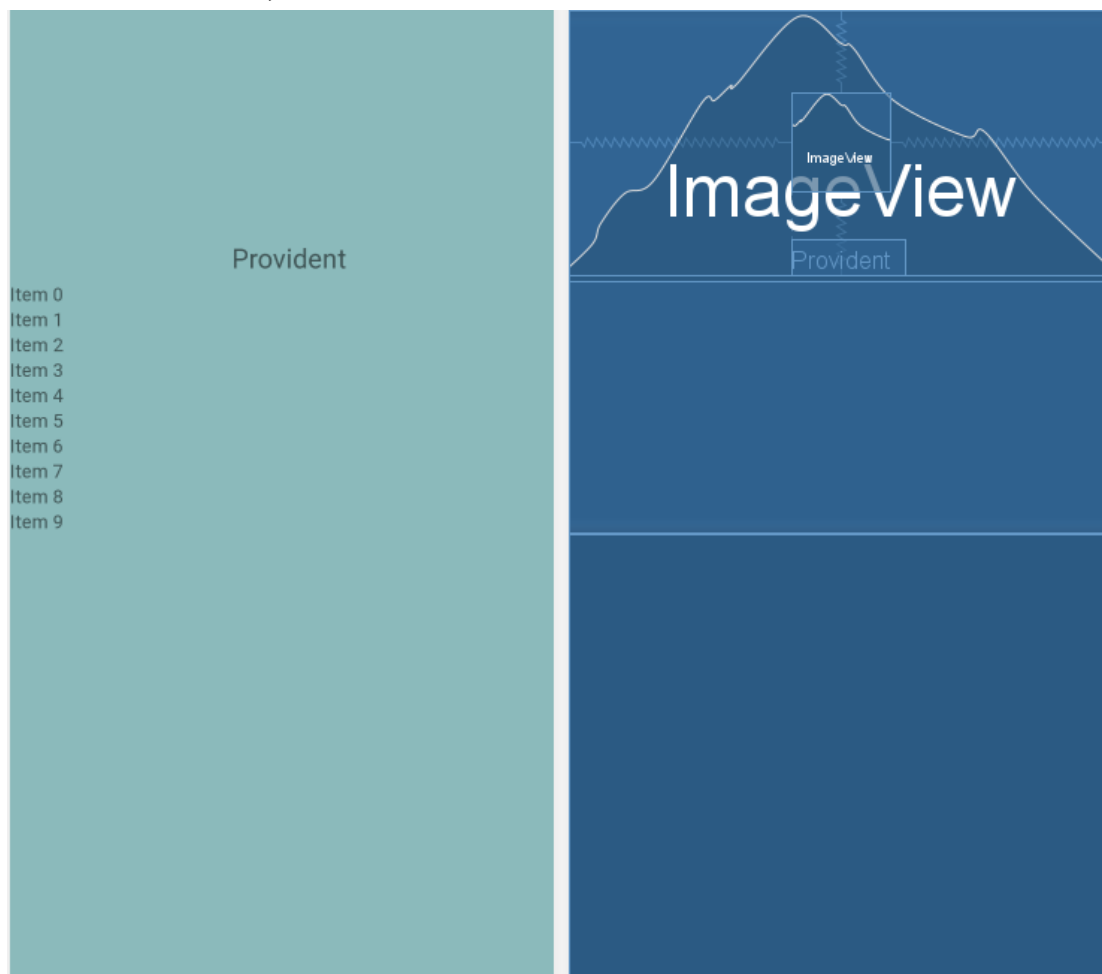


Figura III.8: Layout ProviderActivity – *activity\_provider.xml*

.Partea de sus a schiței conține imaginea de fundal a furnizorului, iconița sau logo-ul furnizorului și numele său, iar partea de jos, ca în layout-ul precedent, va fi un *RecyclerView*, unde va fi prezentată o listă cu toate serviciile furnizorului respectiv.

Fiecare obiect din această listă va urma pattern-ul următorului design:

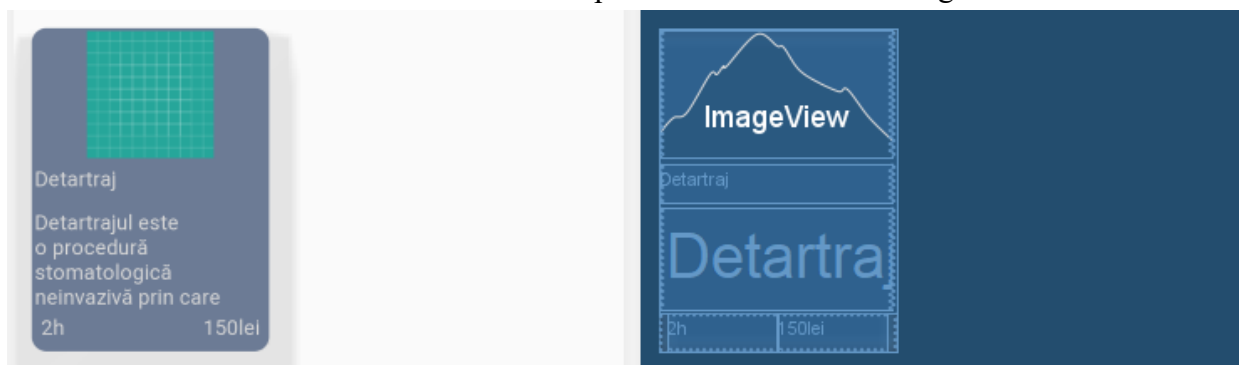


Figura III.9: Layout ServiceList Cardview – *cardview\_service\_list\_item.xml*

În urma selectării unui serviciu din această listă, va fi pornită activitatea *ServiceActivity*, activitatea care va conține mai multe detalii despre serviciul respectiv.

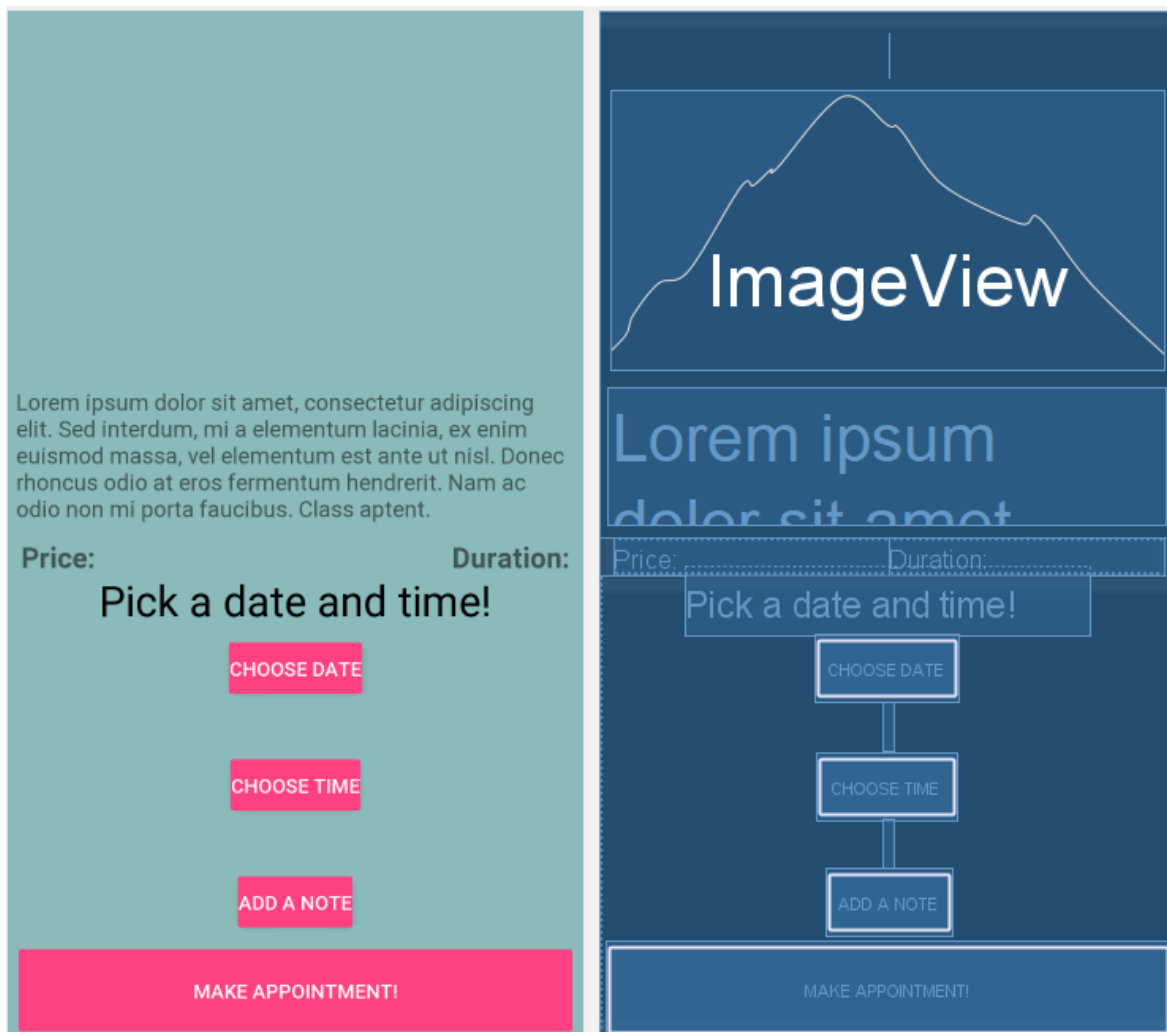


Figura III.10: Layout ServiceActivity activity\_service.xml

Această activitate deține toate detaliile serviciului, inclusiv toată descrierea, care nu se vede pe obiectul din listă. Aici poate fi pusă o altă poză a serviciului, dacă acesta are o poză specifică sau va fi pusă poza de fundal a furnizorului de servicii.

În această activitate se va afla metoda de obținere a unei programări. În urma apăsării butoanelor respective, vor fi deschise dialoguri separate în care utilizatorul va putea alege data și ora. Adăugarea unei note este opțională, însă am considerat că acest câmp poate fi util în caz că clientul vrea să transmită ceva furnizorului în momentul cererii unei programări.

Desigur, butonul cu nume sugestiv, *Make Appointment*, va verifica înainte să trimită informația către baza de date dacă au fost setate în prealabil data, ora și/sau nota. În cazul în care acestea nu sunt setate, butonul nu va funcționa, iar utilizatorului i se va atrage atenția că în cazul în care dorește să facă o programare este necesară setarea acestor parametrii.

De aici, dacă a fost cerută cu succes o nouă programare, clientul va fi redirecționat către activitatea numită *MyProfileActivity*, unde poate să urmărească statusul programărilor făcute și să acceseze direct activitatea pentru schimbarea setărilor, *SettingsActivity*.

*MyProfileActivity* va fi împărțită în trei părți. Prima parte va conține poza, sub forma unui avatar circular fixată sub această formă folosind o formă circulară personalizată în XML și API-ul Picasso, folosit special pentru încărcarea mai eficientă a pozelor. Tot aici apar numele utilizatorului și email-ul acestuia. A doua parte va conține lista de programări pe care le are deja clientul, listă



afișată tot într-un obiect de tip RecyclerView, iar a treia parte va conține programările care așteaptă confirmare din partea administratorului serviciilor respective, din nou într-un obiect de tip RecyclerView.

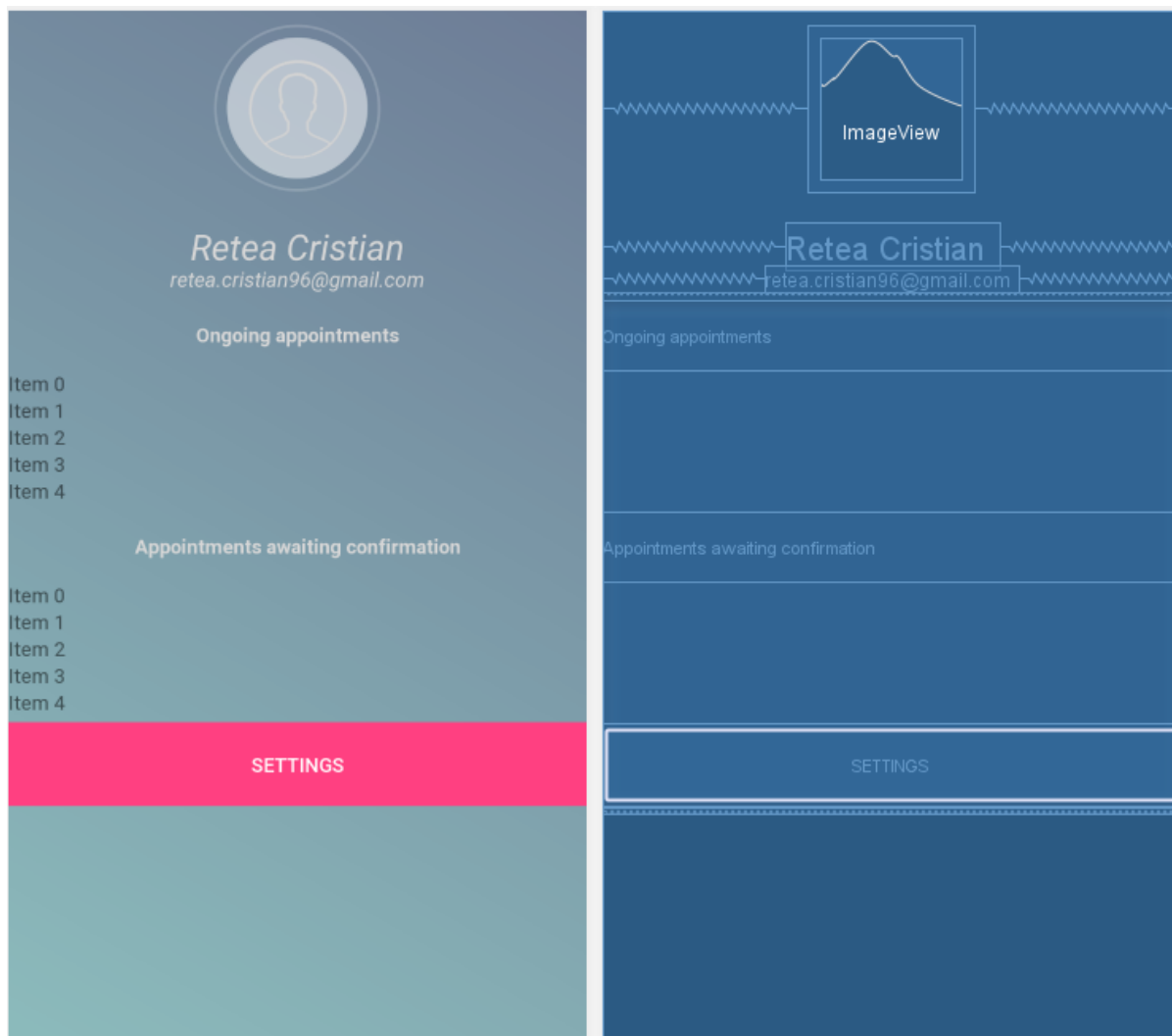


Figura III.11: Layout MyProfileActivity activity\_my\_profile.xml

Fiecare programare din cele două liste va putea fi accesată pentru a porni activitatea numită *AppointmentActivity*, aceasta deținând toate detaliile corespunzătoare respectivei programări. La fel ca în cazurile precedente, aceste obiecte din listă vor fi construite după pattern-ul design-ului următor.



Figura III.12: Layout Cardview Appointment – cardview\_appointment\_list\_item.xml

Imaginea din dreapta cardului se va modifica în funcție de statusul programării, fiind în așteptare, acceptată sau refuzată.

Activitatea responsabilă pentru prezentarea detaliată a programării va conține detalii precum titlul și descriere serviciului, numele, email-ul și numărul de telefon al furnizorului serviciului, data și ora programării, precum și durata și prețul serviciului. Mai jos va putea fi văzută nota, dacă aceasta a fost scrisă.

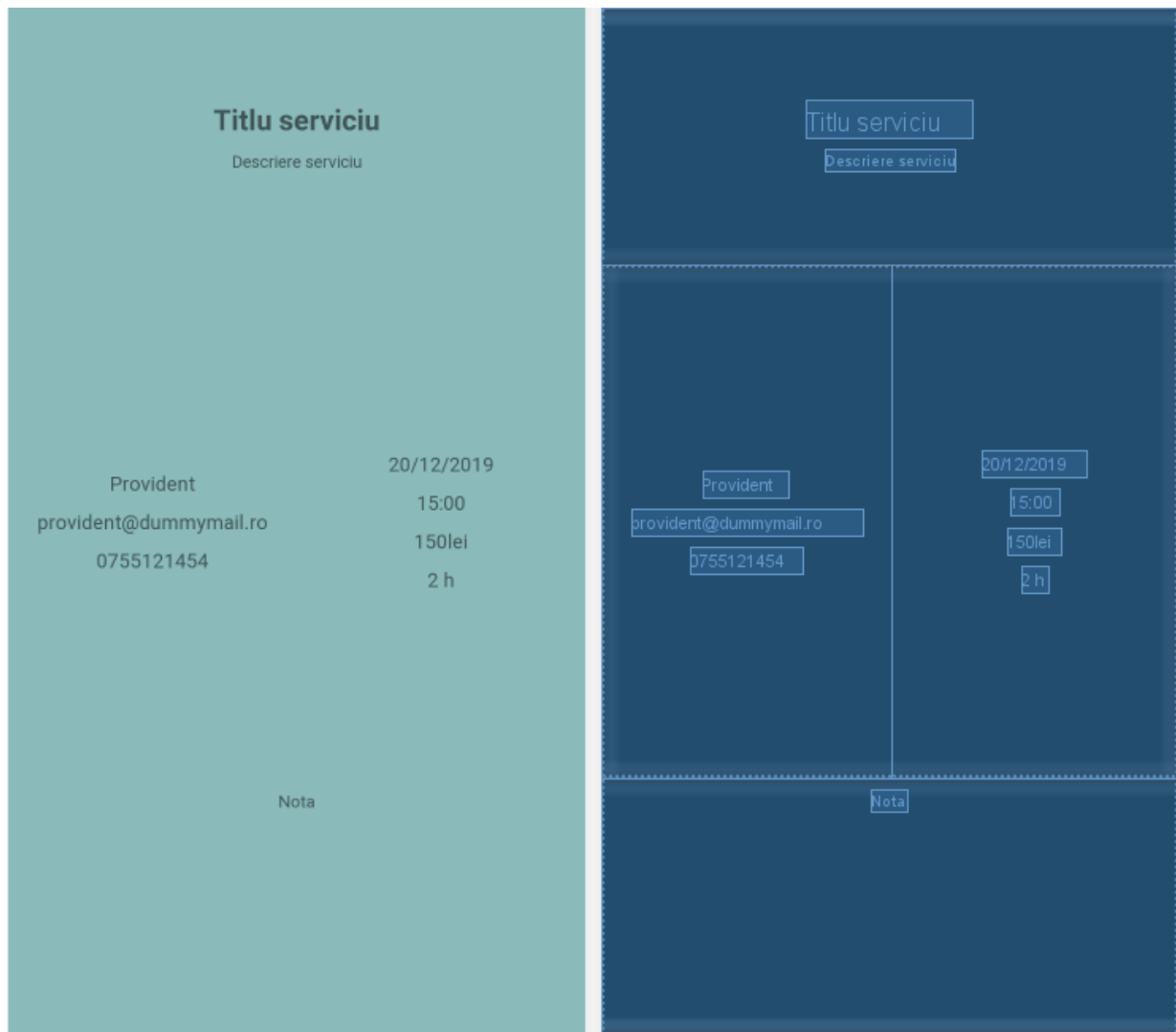


Figura III.13: Layout AppointmentActivity - activity\_appointment.xml

Una dintre cele mai importante funcționalități care ajută foarte mult utilizatorul pentru orientarea în aplicație o reprezintă activitatea *BaseNavigationDrawer*, care extinde clasa *NavigationDrawer*. Această clasă va permite utilizatorului să treacă în unele dintre activitățile prezentate mai sus prin butoanele prezente în meniu.

Astfel, din acest meniu se poate trece din activitatea curentă în una dintre:

- *MyProfileActivity*;
- *DashboardActivity*;
- *MainActivity*, fragmentul *NearbyFragment*, cu scopul vederii listei de furnizori de servicii la domiciliu;
- *MainActivity*, fragmentul *NearbyFragment*, cu scopul vederii listei de furnizori de servicii la sediul furnizorului;
- *MainActivity*, fragmentul *SearchFragment*, cu scopul căutării unui anume furnizor;
- *SettingsActivity*;

În plus, din acest meniu utilizatorul se poate deconecta, urmând să fie redirecționat spre activitatea din care să aleagă metoda prin care vrea să se autentifice, dat fiind faptul că buna funcționare a aplicației necesită ca utilizatorul să fie conectat la contul său.

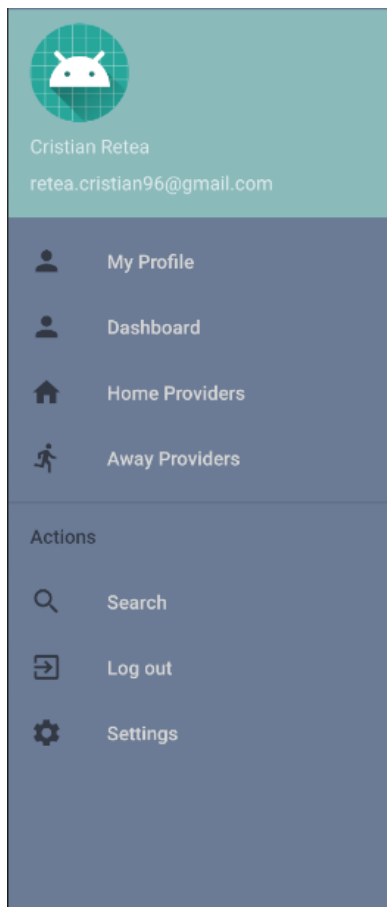


Figura III.14: NagivationDrawer - *navigation\_menu.xml*

Aceasta va fi o activitate abstractă, urmând ca celelalte activități din care dorim să lăsăm utilizatorul să plece direct folosind acest meniu să extindă această clasă.

Celelalte pachete și clase despre care nu s-a vorbit în acest capitol vor fi descrise în capitolul de implementare deoarece au mai mare legătură cu crearea funcționalităților decât cu proiectarea aplicației în sine.

Diagrama fluxului de activități poate fi văzută în Anexa 1.

#### ***III.2.4 Activitățile și schițele lor pentru aplicația destinată administratorilor serviciilor***

Această aplicație nu va fi la fel de stufoasă precum prima aplicație descrisă deoarece principalele funcționalități constau în primirea noilor programări și posibilitatea de a le accepta sau refuza, deci nu va avea decât strictul necesar funcționării și completării ciclului de realizare a programărilor.

Pentru a avea acces pe platforma aceasta, administratorul de serviciu va primi credențiale cu care se va autentifica, contul lui fiind făcut odată cu adăugarea serviciilor lui în baza de date. Cel puțin deocamdată acesta nu va putea să își adauge sau modifice serviciile pe care le pune la dispoziție pe platformă.

În plus față de aplicația precedentă, mai avem nevoie de o clasă model de User, considerând că va trebui descărcarea utilizatorului care a făcut programarea pentru obținerea numelui, email-ului și numărului de telefon pentru afișarea detaliată a programării. În rest se păstrează modelele folosite și în aplicația precedentă: Service, Provider, Appointment.

La fel ca în prima aplicație, activitatea de tip LAUNCHER se va numi *StartActivity*. Această activitate nu are interfață vizuală, fiind folosită doar pentru obținerea permisiunilor necesare funcționării aplicației. Tot aici se verifică dacă utilizatorul este sau nu autentificat. În cazul în care acesta nu este autentificat el va fi redirecționat spre activitatea *AuthenticationActivity*. Această activitate este foarte asemănătoare cu activitatea de conectare din prima aplicație, inclusiv la nivelul layout-ului, dar nu se pot crea noi conturi folosind-o. Conturile noi vor fi adăugate direct în baza de date în momentul în care se adaugă furnizorul de servicii pe platformă.

În urma autentificării, utilizatorul va fi redirecționat către activitatea unde va putea să își urmărească programările, *MainActivity*.

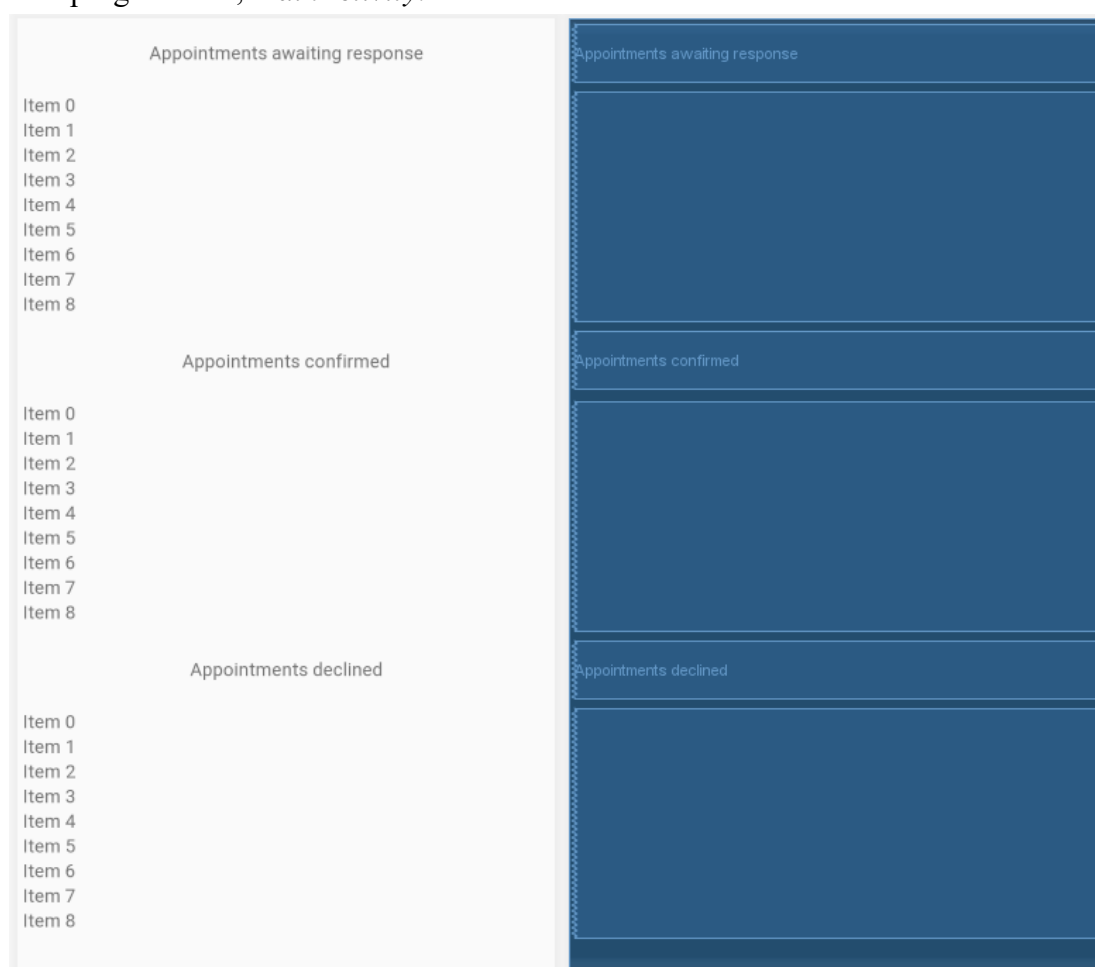


Figura III.15: MainActivity - *activity\_main.xml*

Această activitate va conține TextView-uri care vor indica lista care urmează, și trei liste care vor conține programările de trei tipuri: care necesită actualizarea statusului, confirmate sau refuzate. Listele vor fi conținute în trei RecyclerView-uri și pe lângă textul de deasupra listei, obiectele în sine vor conține o iconiță care va da statusul programării, un ceas galben pentru cele ce trebuie actualizate, un bifat verde pentru cele confirmate sau un cerc roșu de interzis pentru cele refuzate.

Obiectele de tip programare din listă vor urma același pattern, în stânga fiind numele clientului și serviciul solicitat, iar în partea dreaptă fiind data, ora și iconița aferentă statusului programării.



Figura III.16: CardView Appointment - cardview\_appointment\_list\_item.xml

În urma apăsării pe unul din aceste obiecte din listă, administratorul va fi redirecționat către activitatea care va oferi detalii mai multe despre programarea respectivă, *AppointmentActivity*.

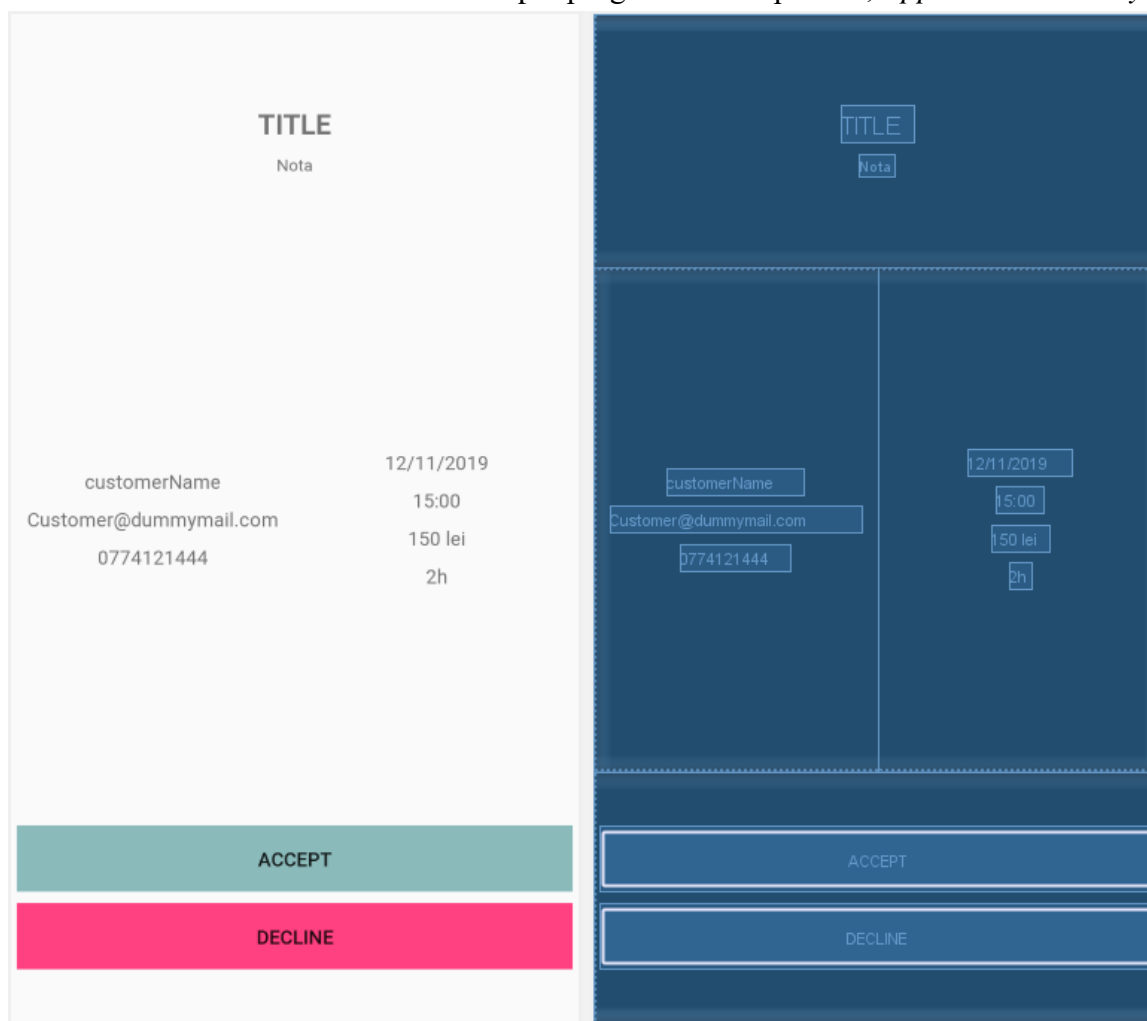


Figura III.17: Appointment - activity\_appointment.xml

În această aplicație administratorul va putea vedea numele serviciului, pus în locul titlului, nota clientului, dacă va avea una, data și ora programării și datele de contact ale clientului.

Mai jos sunt două butoane prin care administratorul poate accepta sau refuza programarea. În momentul actualizării programării, utilizatorul va fi redirecționat înapoi la activitatea în care poate să-și urmărească toate programările, inclusiv cea pe care tocmai a actualizat-o, aceasta mutându-se în lista corespunzătoare. Doar programările care sunt în stadiul de așteptare a actualizării pot fi acceptate sau refuzate, restul înseamnă că au trecut deja prin acest proces.

Diagrama fluxului de activități poate fi văzută în anexa 2.



# Capitolul IV

## Implementarea

### IV.1 Organizarea claselor în pachete

Am organizat clasele încercând să le grupez după tipul lor și am ajuns la o organizare în șapte pachete, lăsând în mod special o singură clasă în afara pachetelor, și anume clasa care extinde clasa Application.

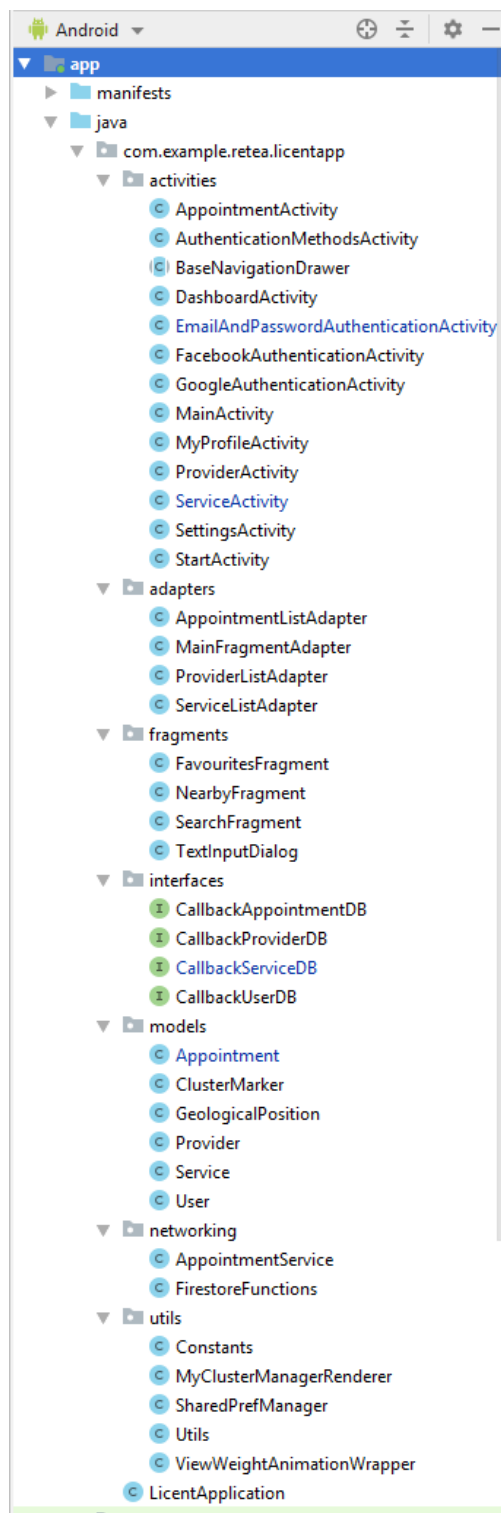


Figura IV.1: Organizarea pachetelor și claselor

Pachetul *activities* conține toate clasele care moștenesc superclasa Activity, fie că au sau nu interfață vizuală. Despre activități s-a vorbit mai mult în capitolul doi, acolo unde s-a vorbit despre resursele necesare implementării.

Pachetul *adapters* conține toate clasele care moștenesc superclasele de tip Adapter. În această aplicație vom extinde două tipuri de adaptoare, RecyclerViewAdapter<ViewHolder> pentru a popula RecyclerView-urile din aplicația această cu liste și FragmentAdapter, pentru a popula layout-urile activității MainActivity.

Pachetul *fragments* conține cele trei fragmente care fac parte din activitatea MainActivity, și anume FavouritesFragment, NearbyFragment și SearchFragment.

Pachetul *models* conține clasele care servesc ca model pentru comportamentele necesare în realizarea aplicației.

Pachetul *networking* conține o clasă care moștenește superclasa Service care va fi necesară la manevrarea notificărilor și o clasă în care vom avea metodele de obținere a informației din baza de date Firestore.

Pachetul *utils* conține mai multe clase utilitare care facilitează anumite operațiuni la nivelul aplicației.

În partea de jos se poate observa clasa LicentApplication care extinde superclasa Application, o clasă de tip Singleton care va rula permanent în background, aici având diferite metode de care este nevoie în buna funcționare a aplicației.

## IV.2 Clasele modele

Am încercat să prezic încă din momentul proiectării toate modelele necesare și toate câmpurile necesare pentru fiecare model, însă fără reușită. De multe ori am realizat că a mai era nevoie de câmpuri suplimentare pentru realizarea tuturor funcționalităților sau pentru ca obiectele să aibă toate detaliile necesare.

Sunt câteva reguli, totuși, pe care le-am respectat în momentul în care am creat aceste clase, și anume:

- toate câmpurile obiectelor sunt câmpuri private, din motive de securitate;
- fiecare obiect are cel puțin un constructor, acela prin care se construiește un obiect instanțiând toate câmpurile obiectului respectiv(dacă un obiect are 7 câmpuri, va fi nevoie de completarea tuturor acestor câmpuri pentru crearea unei noi instanțe a obiectului);
- fiecare obiect are metode de tip getter și setter pentru obținerea și setarea câmpurilor respective;
- fiecare obiect are implementată metoda toString() care returnează o parte din câmpurile obiectului respectiv sub formă de String;

Menționez că inițial am încercat o formulă cu identificatori de tip Integer, dar era mult mai dificilă operarea cu baza de date, dat fiind faptul că Firestore folosește identificatori alfanumerici, astfel că am trecut la id-uri de tip String pentru toate modelele. De altfel, am încercat să evit folosirea câmpurilor de tip Integer unde s-a putut.

Chiar dacă compararea obiectelor de tip Integer era foarte ușoară, la fel este și compararea obiectelor de tip String folosind sintaxa: **if(String1.equals(String2))**, această metodă returnând un boolean: true dacă cele două sunt egale și fals dacă diferă.



Am creat modelul `GeologicalPosition`, care are două câmpuri de tip `double`: *longitude* și *latitude*, deoarece obiectul oferit de Android care a fost folosit inițial a complicat de multe ori lucrurile. Astfel obiectul creat de mine are doar cele două câmpuri necesare folosirii serviciilor de localizare.

Am creat modelul `ClusterMarker`, care va constitui obiectul de tip `marker` care va apărea pe hartă la locațiile furnizorilor de servicii. Acest obiect are câmpuri precum poziția, titlul, descrierea, logo-ul și provider-ul respectiv.

### IV.3 Permiuni

Permiuniile sunt foarte importante, considerând că aplicațiile mobile nu pot face anumite lucruri fără să ca utilizatorul să știe. În general, aplicațiile cer permiuni ori în momentul în care utilizatorul deschide aplicația, ori în momentul în care utilizatorul încearcă să facă o operațiune care necesită permiuni. În această aplicație se întâmplă ambele lucruri.

Pentru început, permiuniile de care vom avea nevoie trebuie menționate în manifestul aplicației, astfel:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA"/>
```

Pentru început, în activitatea `StartActivity` verificăm dacă utilizatorul a oferit permiuniile legate de locație. În cazul în care acesta nu a oferit deja aceste permiuni vom construi un mesaj de tip `AlertDialog` care va apărea pe ecran, cerând utilizatorului să ofere aceste permiuni. Acest dialog pornește de fapt o activitate, cea de obținere a permiunii și așteaptă rezultatul ca răspuns, dacă utilizatorul a oferit sau nu permiunea. În continuare am suprascris metodele care primesc rezultatul activității chemate, `onRequestPermissionsResult()` și `onActivityResult`, urmărind rezultatul cererii permiunilor. Dacă utilizatorul oferă permiunea, vom schimba o variabilă locală boolean `mLocationPermissionGranted` din `false` în `true`, în prima metodă suprascrisă, iar în a doua metodă vom chema metoda `getDeviceLocation()` care va obține locația dispozitivului utilizatorului conectat. După obținerea acestei locații, o vom seta ca parametru static în clasa de tip `Application`, `LicentApplication`.

```
private void getLocationPermission() {
    if (ContextCompat.checkSelfPermission(this,
        android.Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
        mLocationPermissionGranted = true;
        getDeviceLocation();
        if (mAuth.getCurrentUser() != null) {
            startActivity(new Intent( packageContext: this, DashboardActivity.class));
        } else startActivity(new Intent( packageContext: this, AuthenticationMethodsActivity.class));
    } else {
        ActivityCompat.requestPermissions( activity: this,
            new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
            PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
    }
}
```

```

private void getDeviceLocation() {
    Log.d(TAG, msg: "getDeviceLocation: called ");
    FusedLocationProviderClient mFusedlocationProviderClient = LocationServices
        .getFusedLocationProviderClient( activity: this);

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) !=
        PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission( context: this,
        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return;
    }
    mFusedlocationProviderClient.getLastLocation().addOnCompleteListener((task) → {
        if (task.isSuccessful()) {
            Location location = task.getResult();
            GeologicalPosition geoPos = new GeologicalPosition(location.getLatitude(),
                location.getLongitude());
            Log.d(TAG, msg: "onComplete: latitude:" + geoPos.getLatitude());
            Log.d(TAG, msg: "onComplete: longitude" + geoPos.getLongitude());
            LicentApplication.getInstance().setDeviceGeologicalPosition(geoPos);
        }
    });
}
}

```

Alt moment în care solicităm permisiuni este în *SettingsActivity*, în momentul în care cerem utilizatorului să încarce o poză, oferindu-i posibilitatea de a încărca-o din galerie sau din cameră.

```

private void verifyPermissions() {
    Log.d(TAG, msg: "verifyPermissions: ");
    String[] permissions = {Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.WRITE_EXTERNAL_STORAGE,
        Manifest.permission.CAMERA};

    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
        permissions[0]) == PackageManager.PERMISSION_GRANTED
        && ContextCompat.checkSelfPermission(this.getApplicationContext(),
        permissions[1]) == PackageManager.PERMISSION_GRANTED
        && ContextCompat.checkSelfPermission(this.getApplicationContext(),
        permissions[2]) == PackageManager.PERMISSION_GRANTED) {

        startPickImageIntent();

    } else {
        ActivityCompat.requestPermissions( activity: SettingsActivity.this,
            permissions, PERMISSION_REQUEST);
    }
}
}

```

La fel ca mai devreme vom cere permisiunile, dar de această dată vom crea un vector de String-uri care să conțină toate numele permisiunile cerute, astfel încât se vor cere pe rând toate trei, iar dacă aceste permisiuni sunt oferite, vom cere utilizatorului să încarce o imagine prin metoda `startPickImageIntent()`, care nu face altceva decât să apeleze metoda `startActivityResult()`, ținta fiind galerie, iar rezultatul fiind imaginea încărcată.

## IV.4 Implementarea interfeței

Pentru implementarea design-ului din fișierele de layout în format XML am urmat, în mare, aceeași pași.

*Pasul 1.* M-am asigurat că fiecare container sau widget din fișierele de layout care necesită aplicarea codului din activități să aibă câte un identificator, astfel încât ele să poată fi referite prin instanțiere prin sintaxa următoare:

```
<Button
    android:id="@+id/EmailAndPasswordLoginButton"
    android:layout_width="match_parent"
    android:layout_height="70dp"
    android:padding="10dp"
    android:elevation="8dp"
    style="@style/Widget.AppCompat.Button.Colored"
    android:theme="@style/ThemeOverlay.MyDarkButton"
    android:text="Email and password" />
```

*Pasul 2.* În fiecare activitate am creat variabile locale private cu widget-urile necesare implementării schițelor prin sintaxa următoare:

```
private Button emailAndPassButton;
```

*Pasul 3.* Am setat conținutul fiecărei activități ca fiind schița special creată pentru respectiva activitate prin metoda următoare:

```
setContentView(R.layout.activity_authentication_methods);
```

*Pasul 4.* Am legat widget-urile din fișierele layout de variabilele locale create în clasele java corespunzătoare activităților prin metoda următoare:

```
emailAndPassButton = findViewById(R.id.EmailAndPasswordLoginButton);
```

*Pasul 5.* Am implementat codul necesar tuturor activităților pentru realizarea tuturor funcționalităților care au fost concepute în etapa de proiectare.

```
emailAndPassButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        startActivity(new Intent( packageContext: AuthenticationMethodsActivity.this,
                                EmailAndPasswordAuthenticationActivity.class));
    }
});
```

Am luat un caz foarte simplu pentru a exemplifica acești pași, fiind vorba despre un buton care nu va face altceva decât să ducă utilizatorul dintr-o activitate în alta. Pentru asta a fost setat un obiect de tip `OnClickListener` creat local care va aștepta ca butonul să fie apăsător, iar la apăsarea acestuia va porni metoda `onClick()`, care va efectua codul de mai jos. Metoda `startActivity()` are doi parametri necesari, primul este un obiect de tip `Context`, care se referă la locul din care a fost apelată această metodă (activitatea de unde se pleacă) și ținta (activitatea în care se ajunge).

În general, am urmat aceeași pași pentru toate widget-urile din aplicație, marea diferență fiind la nivelul implementării codului. În ceea ce privește butoanele, toate acestea sunt implementate exact în același mod până la conținutul metodei `onClick()`, deoarece fiecare buton poate să aibă un rol diferit.

Fiecare activitate respectă ciclul de viață al acestora, cel prezentat în figura II.4, astfel că pentru implementarea codului se va face suprascriind una sau mai multe din metodele respective. În marea majoritatea a cazurilor, codul implementate, fie în mod direct, fie sub formă de funcții private corespunzătoare clasei respective.

#### IV.4.1 Adaptoarele

Unele widget-uri folosite sunt ceva mai pretențioase din punctul de vedere al implementării. Unul dintre ele este obiectul RecyclerView, pentru care am folosit o bibliotecă specială adăugată în fișierul de tip Gradle. Am decis să folosesc pentru afișarea listelor doar obiecte de tip RecyclerView din motive ce țin clar de resursele pe care telefonul le folosește în construirea interfeței acestor liste. Alternativa acestui obiect constă în folosirea unor obiecte de tip GridView sau ListView. Avantajul imens pe care RecyclerView le are în fața acestor alte doua moduri de afișare a listelor este faptul că telefonul nu încarcă toată lista odată.

Pentru modul de funcționare al acestora vom considera o listă de zece obiecte. Spațiu disponibil widget-ului de afișare a listei va fi doar o treime de ecran. În această suprafață, widget-ul poate afișa doar 3 elemente ale listei. Un obiect de tip ListView sau GridView va încărca toate cele zece obiecte, în cazul în care utilizatorul decide să gliseze până la finalul listei. Un obiect de tip RecyclerView va încărca mereu doar câte elemente se văd, cele trei vizibile plus încă unul în partea de jos, iar dacă utilizatorul începe să gliseze, un element din partea de sus a listei va fi șters și unul din partea de jos, spre care se glisează, va fi adăugat și tot așa. Acest obiect face să nu irosim resursele telefonului, de care dacă am abuza, am îngreuna funcționarea lui.

În această aplicație folosim trei adaptoare pentru containere de tip RecyclerView, *AppointmentListAdapter*, *ProviderListAdapter* și *ServiceListAdapter*. În continuare voi explica doar implementarea clasei *ProviderListAdapter* și folosirea ei în activitatea *MainActivity* deoarece celelalte două adaptoare sunt implementate și funcționează foarte similar.

#### *ProviderListAdapter*

Menționez că acest adaptor implementează design-ul din figura 28.

Pentru început, va fi necesară crearea unei clase interioare care extinde superclasa *RecyclerView.ViewHolder*. Pentru înțelegerea acestei clase este utilă traducerea ei, *ViewHolder* s-ar traduce prin *ținător de vedere*. Fiecare obiect din listă va avea aceleași widget-uri, dar ele vor depinde de obiectul respectiv. În această clasă vom respecta pașii doi și patru din cei prezentați mai sus, astfel:

```
static class MyViewHolder extends RecyclerView.ViewHolder {
    LinearLayout providerItemLinearLaoyout;
    ImageView providerItemIcon;
    TextView providerItemName;
    TextView providerItemCategory;
    TextView providerItemDistanceDifference;

    MyViewHolder(@NonNull View itemView) {
        super(itemView);
        providerItemLinearLaoyout = itemView.findViewById(R.id.ProviderItemLinearLayout);
        providerItemIcon = itemView.findViewById(R.id.ProviderItemIconId);
        providerItemName = itemView.findViewById(R.id.ProviderItemNameId);
        providerItemCategory = itemView.findViewById(R.id.ProviderItemCategoryId);
        providerItemDistanceDifference = itemView.findViewById(R.id.ProviderItemDistanceDifferenceId);
    }
}
```

Funcționarea acestui adaptor depinde de suprascrierea metodelor obligatorii, impuse de mediul de programare, și anume:

- `onCreateViewHolder()` – metoda care va folosi un `LayoutInflater` pentru a lega fiecare articol în parte de obiecte de tipul clasei pe care tocmai am creat-o. În această metodă menționăm modul în care arată fiecare articol, legând layout-ul din figura 28.

```
@NonNull
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int i) {

    LayoutInflater mInflater = LayoutInflater.from(parent.getContext());
    View view = mInflater.inflate(R.layout.cardview_provider_list_item, parent, attachToRoot: false);
    return new MyViewHolder(view);
}
```

- `onBindViewHolder()` – metoda care va folosi clasa interioară creată de noi pentru a modifica fiecare articola din listă corespunzător;

```
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {

    final Provider provider = mList.get(position);

    holder.providerItemName.setText(provider.getName());
    holder.providerItemCategory.setText(provider.getCategory());
    holder.providerItemImage.setImageResource(provider.getIcon());
    holder.providerItemDistanceDifference.setText((distanceTo(provider.getProviderGeologicalPosition())) + " km");
    holder.providerItemLinearLaoyout.setOnClickListener((v) -> {
        listener.onItemClick(provider);
    });
}
```

- `getItemCount()` – metodă care va returna mărimea listei folosite pentru crearea acestui adaptor.

Pentru crearea unui adaptor avem nevoie, în general, doar de lista de articole pe care vrem să le afișăm, dar în adaptoarele pe care le-am construit am adăugat în constructor o interfață de tipul `OnItemClickListener()` pentru a obliga activitățile care folosesc aceste adaptoare să implementeze metodele de manevrare a apăsărilor pe articole, dat fiind faptul că dorim ca apăsarea pe aceste articole să deschidă alte activități în funcție de articolul ales.

În plus, acest adaptor conține și o metodă numită `distanceTo()`, care ia ca parametru o locație(locația unui furnizor de servicii) și calculează distanța de la locația dispozitivul utilizatorului până la locația furnizorului, astfel:

```
private String distanceTo(GeologicalPosition geoPos) {
    Location deviceLocation = new Location( provider: "");
    deviceLocation.setLatitude(LicentApplication.getDeviceGeologicalPosition().getLatitude());
    deviceLocation.setLongitude(LicentApplication.getDeviceGeologicalPosition().getLongitude());
    Location desiredLocation = new Location( provider: "");
    desiredLocation.setLatitude(geoPos.getLatitude());
    desiredLocation.setLongitude(geoPos.getLongitude());
    Log.d(TAG, msg: "distanceTo: " + deviceLocation.distanceTo(desiredLocation) / 1000);
    float distance = deviceLocation.distanceTo(desiredLocation) / 1000;
    return String.format(Locale.getDefault(), format: "%.2f", distance);
}
```

După finalizarea construirii acestui adaptor, acesta trebuie implementat în cadrul activității unde este necesară afișarea articolelor din lista de furnizori de servicii(providers). Se va crea un obiect



de tip `ProviderListAdapter` și i se va atribui obiectului de tip `RecyclerView`. Această funcționalitate necesită un animator pentru scrollul articolelor din listă și un obiect de tip `LinearLayoutManager`.

```
providerListRecyclerView = view.findViewById(R.id.ProviderListRecyclerView);
providerListRecyclerView.setHasFixedSize(true);

ProviderListAdapter providerListAdapter = new ProviderListAdapter(mProviderList, listener: this);
layoutManager = new LinearLayoutManager(this.getContext());
providerListRecyclerView.setLayoutManager(layoutManager);

providerListRecyclerView.setAdapter(providerListAdapter);
providerListRecyclerView.setItemAnimator(new DefaultItemAnimator());
```

A fost creată și interfața obligatorie care va manevra apăsările articolelor din listă. În momentul apăsării unui articol utilizatorul va fi redirectionat către o altă activitate, unde va vedea pagina principală a furnizorului de servicii accesat. Pentru acest lucru, vom transmite informație prin intermediul obiectului cu care deschidem altă activitate, `Intent`-ul. Astfel:

```
@Override
public void onItemClick(Provider provider) {
    Intent intent = new Intent(this.getContext(), ProviderActivity.class);
    intent.putExtra( name: "providerId", provider.getId());
    intent.putExtra( name: "providersType", provider.getType());
    startActivity(intent);
    Log.d(TAG, msg: "onItemClick: " + provider.toString());
}
```

Informația transmisă va fi folosită pentru încărcarea conținutului corespunzător articolului accesat.

Atât `ServiceListAdapter` cât și `AppointmentListAdapter` funcționează absolut similar, singura diferență fiind că la `ServiceListAdapter` articolele listei sunt așezate sub forma unei matrice, pe două coloane, folosind un `GridLayoutManager`.

În afară de aceste trei adaptoare, mai avem un singur adaptor pentru fragmentele ce vor oferi conținutul clasei `MainActivity`, în funcție de tab-ul selectat. Această clasă este destul de rudimentară, fără să fi fost nevoie de multe modificări de la `Adapter`-ul oferit de Android.

## IV.4.2 Gestionarea locației

Am vorbit despre cum obținem permisiunile necesare pentru localizarea dispozitivului, dar nu am vorbit despre ce facem după ce obținem aceste permisiuni.

Pentru început, tot în `StartActivity`, am creat metoda `getDeviceLocation()` care va returna locația dispozitivului. Pentru acest lucru am folosit serviciile de locație ale telefonului. Am instanțiat un obiect de tip `FusedLocationProviderClient`. Folosind acest obiect avem posibilitatea obținerii *ultimei locații înregistrate* de către telefon, care, dacă are GPS-ul pornit, va coincide cu locația curentă. Vom atașa un `OnCompleteListener` și vom verifica dacă sarcina a avut succes. Dacă aceasta a avut succes vom primi ca rezultat o locație din care vom extrage latitudinea și longitudinea, creând un obiect de tip `GeologicalPosition` și setându-l în clasa `LicentApplication` ca fiind locația dispozitivului.

Suntem siguri de faptul că această operațiune va avea succes pentru că înainte de a apela metoda de obținere a locației am verificat și serviciile Google (necesare pentru hărți) și permisiunile referitoare la internet și localizare.

```

private void getDeviceLocation() {
    Log.d(TAG, msg: "getDeviceLocation: called ");
    FusedLocationProviderClient mFusedLocationProviderClient = LocationServices
        .getFusedLocationProviderClient( activity: this);

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) !=
        PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission( context: this,
        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return;
    }
    mFusedLocationProviderClient.getLastLocation().addOnCompleteListener((task) -> {
        if (task.isSuccessful()) {
            Location location = task.getResult();
            GeologicalPosition geoPos = new GeologicalPosition(location.getLatitude(),
                location.getLongitude());
            Log.d(TAG, msg: "onComplete: latitude:" + geoPos.getLatitude());
            Log.d(TAG, msg: "onComplete: longitude" + geoPos.getLongitude());
            LicentApplication.setDeviceGeologicalPosition(geoPos);
            LicentApplication.setDeviceLocation(location);
        }
    });
}

```

În urma finalizării acestei metode vom ajunge în activitatea DashboardActivity, unde vom transforma locația din coordonate geografice în adresă propriu-zisă sub formă de text. Pentru acest lucru am creat metoda getAddressFromLocation(), care va folosi locația obținută mai devreme pentru a transforma și afișa această locație sub formă de adresă în formatul text: numele străzii, numărul străzii, orașul, țara.

```

private void getAddressFromLocation(){
    Location currentLocation = LicentApplication.getDeviceLocation();
    Geocoder geocoder = new Geocoder( context: DashboardActivity.this, Locale.getDefault());
    try {
        List<Address> addressList = geocoder.getFromLocation(currentLocation.getLatitude(),
            currentLocation.getLongitude(), maxResults: 1);
        Log.d(TAG, msg: "getDeviceAddress: list size is " + addressList.size());
        String address = addressList.get(0).getAddressLine( index: 0);
        Log.d(TAG, msg: "getDeviceAddress: " + address);
        AddressTextView.setText(address);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

În momentul în care utilizatorul accesează activitatea MainActivity va vedea mai întâi fragmentul NearbyFragment, unde sunt metodele de afișare a hărții. Pentru folosirea unui obiect de tip MapView avem nevoie să suprascriem metodele de ciclu de viață ale fragmentului pentru a instrui comportamentul acestui obiect în diferite momente și stadii ale acestuia. În plus, au fost suprascrise și metodele de salvare a statusului hărții și metoda care obligă harta la un comportament aparte în momentul în care telefonul nu are suficientă memorie.

Pentru buna funcționare a hărții au fost modificate sau adăugate următoarele metode:

- onMapReady() – această metodă verifică încă o dată că permisiunile sunt obținute și setează locația dispozitivului pe hartă. Tot aici va fi apelată metoda de adăugare a markerelor pe hartă;
- setCamerView() – această metodă ia ca parametru locația dispozitivului și creează un cadru potrivit pentru vederea hărții, astfel încât să nu fie nici prea aproape, nici prea departe de locația sa;
- expandMapAnimation() – această metodă va fi folosită pentru mărirea hărții, la apăsarea unui buton pus în partea de stânga sus a hărții;

- `contractMapAnimation()` – această metodă va fi folosită pentru aducerea hărții la dimensiunile inițiale, la apăsarea aceluiași buton, dacă harta este deja mărită. Aceste două metode sunt realizate prin modificarea atributului XML `weight`.
- `addMapMarkers()` – această metodă transformă articolele listei folosite în acest fragment în locații pe hartă;

```
private void addMapMarkers() {
    if (mGoogleMap != null) {
        if (mClusterManager == null) {
            mClusterManager = new ClusterManager<>(Objects.requireNonNull(getActivity()),
                .getApplicationContext(), mGoogleMap);
        }
        if (mClusterManagerRenderer == null) {
            mClusterManagerRenderer = new MyClusterManagerRenderer(
                getActivity(),
                mGoogleMap,
                mClusterManager
            );
            mClusterManager.setRenderer(mClusterManagerRenderer);
        }
        for (Provider provider : mProviderList) {
            Log.d(TAG, msg: "addMapMarkers: location: " + provider.getProviderGeologicalPosition().toString());
            try {
                String snippet;
                snippet = getProviderAddress(provider.getProviderGeologicalPosition());
                int providerIcon = R.drawable.powered_by_google_light; // set the default avatar
                try {
                    providerIcon = provider.getIcon();
                } catch (NumberFormatException e) {
                    Log.d(TAG, msg: "addMapMarkers: no avatar for " + provider.getName() +
                        ", setting default.");
                }
                ClusterMarker newClusterMarker = new ClusterMarker(
                    new LatLng(provider.getProviderGeologicalPosition().getLatitude(),
                        provider.getProviderGeologicalPosition().getLongitude()),
                    provider.getName(),
                    snippet,
                    providerIcon,
                    provider
                );
                mClusterManager.addItem(newClusterMarker);
                mClusterMarkers.add(newClusterMarker);
            } catch (NullPointerException e) {
                Log.e(TAG, msg: "addMapMarkers: NullPointerException: " + e.getMessage());
            }
        }
        mClusterManager.cluster();
        setCameraView(LicentApplication.getDeviceGeologicalPosition());
    }
}
```

Această metodă va pune pe hartă obiecte de tip `ClusterMarker` care au ca parametrii:

- locația, dată de locația furnizorului de serviciu;
- numele furnizorului;
- snippet, care se referă la un scurt text ce apare sub numele obiectului în momentul în care utilizatorul apasă pe el. Acesta va fi adresa obținută din coordonatele serviciului prin metoda `getProviderAddress()`, o metodă asemănătoare cu cea pe care am descris-o mai devreme, `getAddressFromLocation()`;
- `providerIcon`, logo-ul furnizorului de servicii;
- un obiect de tip `Provider`, pentru a putea trimite mai departe utilizatorul spre activitatea corespunzătoare serviciului selectat;



## IV.5 Gestionarea bazei de date

După cum am menționat în etapele precedente, voi folosi o bază de date NOSQL, oferită de Firebase. Pentru accesarea acestei baze de date avem nevoie de adresa referință pentru colecțiile corespunzătoare, astfel:

```
private static FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
private static CollectionReference providerRef = firebaseFirestore.collection( collectionPath: "providers");
private static CollectionReference serviceRef = firebaseFirestore.collection( collectionPath: "services");
private static CollectionReference appointmentRef = firebaseFirestore.collection( collectionPath: "appointments");
private static CollectionReference userRef = firebaseFirestore.collection( collectionPath: "users");
```

Firestore oferă posibilitatea descărcării a unui singur element prin referința la document sau descărcarea întregii colecții. Vom folosi metoda de descărcare a întregii colecții pentru că avem nevoie de toți furnizorii de servicii. Pentru acest lucru vom înlănțui două metode: mai întâi vom descărca toate serviciile, apoi vom descărca toți furnizorii de servicii și le vom atribui serviciile corespunzătoare. Vom folosi interfețe de tip Callback pentru a semnaliza finalizarea descărcării tuturor datelor necesare într-o metodă. Suntem nevoiți să folosim aceste interfețe deoarece descărcările se fac în mod asincron, deci nu putem asigura liniaritatea lor.

```
public static void downloadServices(final CallbackServiceDB callbackServiceDB) {

    serviceRef.get().addOnCompleteListener((task) -> {
        if (task.isSuccessful()) {
            List<Service> services = new ArrayList<>();
            for (QueryDocumentSnapshot documentSnapshot : task.getResult()) {
                String id = (String) documentSnapshot.get("id");
                String name = (String) documentSnapshot.get("name");
                String price = (String) documentSnapshot.get("price");
                String duration = (String) documentSnapshot.get("duration");
                String shortDescription = (String) documentSnapshot.get("shortDescription");
                String longDescription = (String) documentSnapshot.get("longDescription");
                String imageUri = (String) documentSnapshot.get("imageUri");
                String providerId = (String) documentSnapshot.get("providerId");
                Service service = new Service(id, name, price, duration, shortDescription,
                    longDescription, Uri.parse(imageUri), providerId);
                services.add(service);
                Log.d(TAG, "onSuccess: " + service.toString());
            }
            callbackServiceDB.onSuccess(services);
        }
    });
}
```

Colecția se descarcă element cu element. Voi folosi variabile locale pentru stocarea pe moment a fiecărui câmp descărcat din baza de date, iar după obținerea tuturor datelor necesare pentru un obiect voi folosi constructorul pentru a instanția un obiect și îl voi adăuga în lista corespunzătoare. La terminarea colecției voi semnaliza că a fost terminată operația chemând callback-ul necesar.

Această metodă va fi apelată în interiorul clasei LicentApplication, urmând să adăugăm aceste servicii în lista locală, pentru ca, ulterior, să nu mai trebuiască să descărcăm tot din baza de date aceste date.

Cea de-a doua metodă va avea ca parametrii Callback-ul de tip Providers, dar și lista de servicii pe care tocmai am descărcat-o. Astfel după crearea fiecărui obiect de tip Provider, vom verifica elementele din lista de servicii, căutând serviciile care au providerId egal cu Id-ul Provider-ului pe care tocmai l-am instanțiat. Serviciile care verifică acest criteriu vor fi adăugate într-o listă

locală, iar la final vor fi atribuite Provider-ului. Lucrul acesta se întâmplă pentru toate obiectele de tip Provider descărcate.

```
public static void downloadProviders(final List<Service> serviceList, final CallbackProviderDB callbackProviderDB) {

    providerRef.get().addOnCompleteListener((task) -> {
        if (task.isSuccessful()) {
            List<Provider> providers = new ArrayList<>();
            List<Service> currentServices = new ArrayList<>();
            for (QueryDocumentSnapshot documentSnapshot : task.getResult()) {
                currentServices.clear();
                String id = (String) documentSnapshot.get("id");
                String name = (String) documentSnapshot.get("name");
                String category = (String) documentSnapshot.get("category");
                GeoPoint geoPoint = (GeoPoint) documentSnapshot.get("geoPoint");
                int icon = Integer.parseInt(String.valueOf(documentSnapshot.get("icon")));
                String imageUri = (String) documentSnapshot.get("imageUri");
                int type = Integer.parseInt(String.valueOf(documentSnapshot.get("type")));
                Provider provider = new Provider(id, name, category, new GeologicalPosition(geoPoint.getLatitude(),
                    geoPoint.getLongitude()), icon, Uri.parse(imageUri), type);
                for (Service service : serviceList) {
                    if (service.getProviderId().equals(provider.getId())) {
                        currentServices.add(service);
                    }
                }
                provider.setServiceList(currentServices);
                providers.add(provider);
                Log.d(TAG, msg: "onSuccess: " + provider.toString());
            }
            callbackProviderDB.onSuccess(providers);
        }
    });
}
```

Înlănțuirea celor două metode se face în LicentAplication, iar acestea vor fi adăugate în una din cele două liste, AwayProviders sau HomeProviders în funcție de categoria de care aparțin.

```
FirestoreFunctions.downloadServices((serviceList) -> {
    Log.d(TAG, msg: "onSuccess: download services: Got here");
    mServiceList.clear();
    mServiceList.addAll(serviceList);
    FirestoreFunctions.downloadProviders(serviceList, (providerList) -> {
        Log.d(TAG, msg: "onSuccess: download providers: Got here");

        for (Provider provider : providerList) {
            if (provider.getType() == PROVIDER_TYPE_HOME) {
                mHomeProviderList.add(provider);
                Log.d(TAG, msg: "onSuccess: found home provider");
            } else if (provider.getType() == PROVIDER_TYPE_AWAY) {
                mAwayProviderList.add(provider);
                Log.d(TAG, msg: "onSuccess: found away provider");
            }
        }
    });
});
```

Metodele care descarcă listele de programări și descărcarea utilizatorului sunt asemănătoare, ambele folosind tot referința colecției corespunzătoare și semnalarea finalizării descărcării prin Callback, în care se adaugă lista sau obiectul rezultat.

Altă metodă folosită în gestionarea bazei de date este metoda de a adăuga programări, folosită în activitatea ServiceActivity. Această metodă folosește tot o referință la colecție, de data aceasta fiind vorba de colecția de programări. Adăugarea informației în baza de date este mai facilă decât descărcarea ei pentru că putem adăuga chiar un obiect făcut de noi, singura condiție fiind ca atributele

acestui obiect să nu fie de un tip pe care baza de date să nu-l cunoască. Astfel, adăugarea unui nou document în baza de date se va face folosind sintaxa următoare:

```
CollectionReference appointmentRef = FirebaseFirestore.getInstance().collection( collectionPath: "appointments");
appointmentRef.add(new Appointment(FirebaseAuth.getInstance().getCurrentUser().getUid(),
    LicentApplication.getCurrentUser().getName(), mCurrentProvider.getId(), mCurrentProvider.getName(),
    mCurrentService.getId(), mCurrentService.getName(), mAppointmentDay, mAppointmentMonth,
    mAppointmentYear, mAppointmentHour, mAppointmentMinute, mNote, confirmed: 0));
```

Singura metodă care mai presupune utilizarea bazei de date este metoda de actualizare a statusului unei programări, metodă care este folosită în aplicația de administratori. Această metodă caută în baza de date programarea care să aibă același identificator cu programarea actualizată și în momentul în care o găsește, schimbă statusul acesteia în funcție de ce a ales administratorul.

```
private void updateAppointment(final Appointment appointment, final int confirmationId){
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    CollectionReference usersReference = db.collection( collectionPath: "appointments");
    usersReference.get().addOnCompleteListener((task) -> {
        if (task.isSuccessful()) {
            for (QueryDocumentSnapshot documentSnapshot : task.getResult()) {
                if(documentSnapshot.getId().equals(appointment.getId())){
                    Log.d(TAG, msg: "onComplete: FOUND USER MATCHING ID WITH DEVICE TO BE UPDATED");
                    DocumentReference userReference = documentSnapshot.getReference();
                    userReference.update(
                        field: "confirmed", confirmationId).addOnCompleteListener((task) -> {
                            if(task.isSuccessful()){
                                Log.d(TAG, msg: "onComplete: updated user: " + appointment.getId());
                            }else{
                                Log.d(TAG, msg: "onComplete: couldn't update user, check log");
                            }
                        });
                }
            }
        }
    });
}
```

## IV.6 Baza de date locală

Pentru baza de date locală am implementat clasa SharedPreferencesManager pentru a salva informația într-un fișier al telefonului. Pentru acest lucru am folosit o bibliotecă externă GSON care facilitează parsarea obiectelor în formatul necesar salvării lor.

```
public SharedPreferencesManager(Context context) {
    sharedPreferences = context.getSharedPreferences(PREF_NAME, PRIVATE_MODE);
}

public void saveProviderListInSharedPrefs(List<Provider> providers){
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    String json = gson.toJson(providers);
    editor.putString( s: "providerList",json);
    editor.commit();
    Log.d(TAG, msg: "saveProviderListInSharedPrefs: saved providerlist in shared prefs");
}

public List<Provider> loadProviderListFromSharedPrefs(){
    Gson gson = new Gson();
    Type type = new TypeToken<List<Provider>>(){}
    .getType();
    List<Provider> list = new ArrayList<>();
    String json = sharedPreferences.getString( s: "providerList", s1: null);
    Provider[] mArray = gson.fromJson(json,Provider[].class);
    Collections.addAll(list, mArray);
    Log.d(TAG, msg: "loadProviderListFromSharedPrefs: loaded provider list form shared prefs");
    return list;
}
```

Biblioteca externă GSON are metode prin care transformă un obiect cu diferite câmpuri în obiecte String cu format JSON, dat fiind faptul că în fișierul SharedPreferences al telefonului nu pot fi adăugate decât String-uri.

La momentul sustragerii informației din SharedPreferences, transformăm String-urile formate sub formă de JSON în obiecte, menționând tipul obiectului în care dorim să se facă transformarea. Desigur, dacă tipul nu coincide cu JSON-ul obținut, această operațiune nu va fi posibilă.

## ***IV.7 Autentificare***

Considerând faptul că autentificarea reprezintă o funcționalitate întâlnită la majoritatea aplicațiilor mobile, indiferent de tipul acestora, implementarea acestei acesteia a fost făcută fără prea mare implicare din partea mea deoarece metodele corespunzătoare acestui serviciu, Firebase Authentication, facilitează foarte mult tot accesul la baza de date. Acest serviciu oferă metode specifice atât pentru crearea conturilor cât și pentru verificarea credențialelor în momentul în care se face autentificarea pentru fiecare din cele trei metode implementate, fie că vorbim despre Facebook, Google sau Email&Password. În plus, exemplele de cod pe care le-am găsit în documentația Firebase Authentication au fost mai mult decât suficiente pentru realizarea a ceea ce mi-am propus, lăsându-mi mai mult timp pentru realizarea celorlalte funcționalități.

După folosirea acestor metode din biblioteca oferită de Firebase, am ales să creez o clasă model User pentru a memora datele importante ale utilizatorului curent pentru a nu fi necesară obținerea lor dinspre baza de date de fiecare dată când utilizatorul folosește aplicația. Această clasă va fi instanțiată, cu detalii pe care le va introduce utilizatorul în activitatea SettingsActivity și cu parametrii obținuți din obiectul de tip FirebaseAuthentication User, și anume Uid și Token.

## ***IV.6 Notificări***

Pentru realizarea sistemului prin care utilizatorii primesc notificări în momentul realizării sau actualizării unei programări am decis să folosesc un sistem de mesagerie Firebase, și anume Firebase Cloud Messaging. Din păcate, la 1 ianuarie, Firebase a schimbat acest serviciu, însemnând că nu mai pot fi țintite dispozitive singure, pot fi țintite doar categorii de utilizatori. Planul inițial folosea un parametru numit Token generat la crearea contului fiecărui utilizator. Folosind acest parametru am fi putut trimite aceste notificări direct la dispozitivul corespunzător acestui Token. Firebase permite în continuare trimiterea notificărilor prin FCM, dar acest mod ne lasă să trimitem folosind colecțiile, deci aceeași notificare la toți utilizatorii și nu asta ne dorim.

Pentru realizarea acestei funcționalități de trimitere a notificărilor trebuie folosit alt serviciu Firebase, și anume Firebase Functions, dar pentru acest lucru trebuie să creăm un server http prin care să trimitem aceste notificări, ceea ce am încercat să evit folosind restul funcționalităților. Am încercat să pun la punct acest serviciu, dar am întâmpinat probleme la instalarea acestuia, fiind necesar de mai multe cunoștințe legate de scripturi linux.

## ***IV.7 Transmiterea informației între activități***

La momentul descărcării utilizatorului, furnizorilor de servicii sau utilizatorului am copiat rezultatul în clasa LicentApplication pentru a putea obține ușor informația din orice loc(activitate) al aplicației, tocmai în ideea în care vom avea nevoie să facem acest lucru.

Există două metode de a transmite informație, o soluție este trimiterea întregului obiect sub formă JSON, însemnând că descompunem atributele obiectului într-un script, iar în clasa țintă va fi construit din nou obiectul respectiv. Această operațiune de a transforma obiectul se numește Parse. A doua soluție este să transmitem spre activitatea țintă un singur parametru esențial pentru identificarea obiectului de care avem nevoie, cum ar fi Id-ul obiectului. Această operațiune se poate face mai ușor pentru că este mai simplă transmiterea informației sub formă de obiecte primare(Int, String, Float, etc).

Indiferent de soluția pe care o alegem, informația este trimisă prin intermediul unui obiect numit Intent, cel care se folosește pentru deschiderea unei noi activități. Am ales să folosesc a doua metodă, trimițând doar informația esențială, urmând ca în activitățile în care vom avea nevoie de această informație să implementez metoda prin care verificăm acest obiect de tip Intent pentru a obține parametrii necesari. Informația se transmite sub forma de mapare, adică o valoare pentru o cheie stabilită.

În activitățile în care am trimis informație prin acest obiect am implementat metoda privată, mereu cu același nume: `checkIntentForInfo(Intent intent)`. Am verificat mai întâi ca intent-ul să nu fie null. Dacă acesta nu este null, am chemat metoda de mai sus. Folosind această metodă și cunoscând cheile necesare obținerii informației necesare, în cazul de față id-ul, am preluat lista de furnizori salvată local în `LicentApplication` și prin intermediul unui for și un if am obținut furnizorul de care este nevoie în activitatea de față, `ProviderActivity`. După ce obținem furnizorul accesat putem popula toate câmpurile din interfața activității care depind de acest obiect `Provider`.



## Capitolul V

### Ghid de utilizare a aplicației

În continuare voi prezenta un scenariu general de utilizare a aplicației care va servi ca ghid pentru potențialii utilizatori ai acesteia.

Vom presupune că utilizatorul a oferit deja permisiunile necesare pentru folosirea aplicației. Astfel, primul lucru pe care îl vede utilizatorul este ecranul de alegere a metodei preferate de autentificare sau de creare a contului. În urma alegerii uneia dintre metode, acesta va fi redirecționat către ecranul corespunzător.

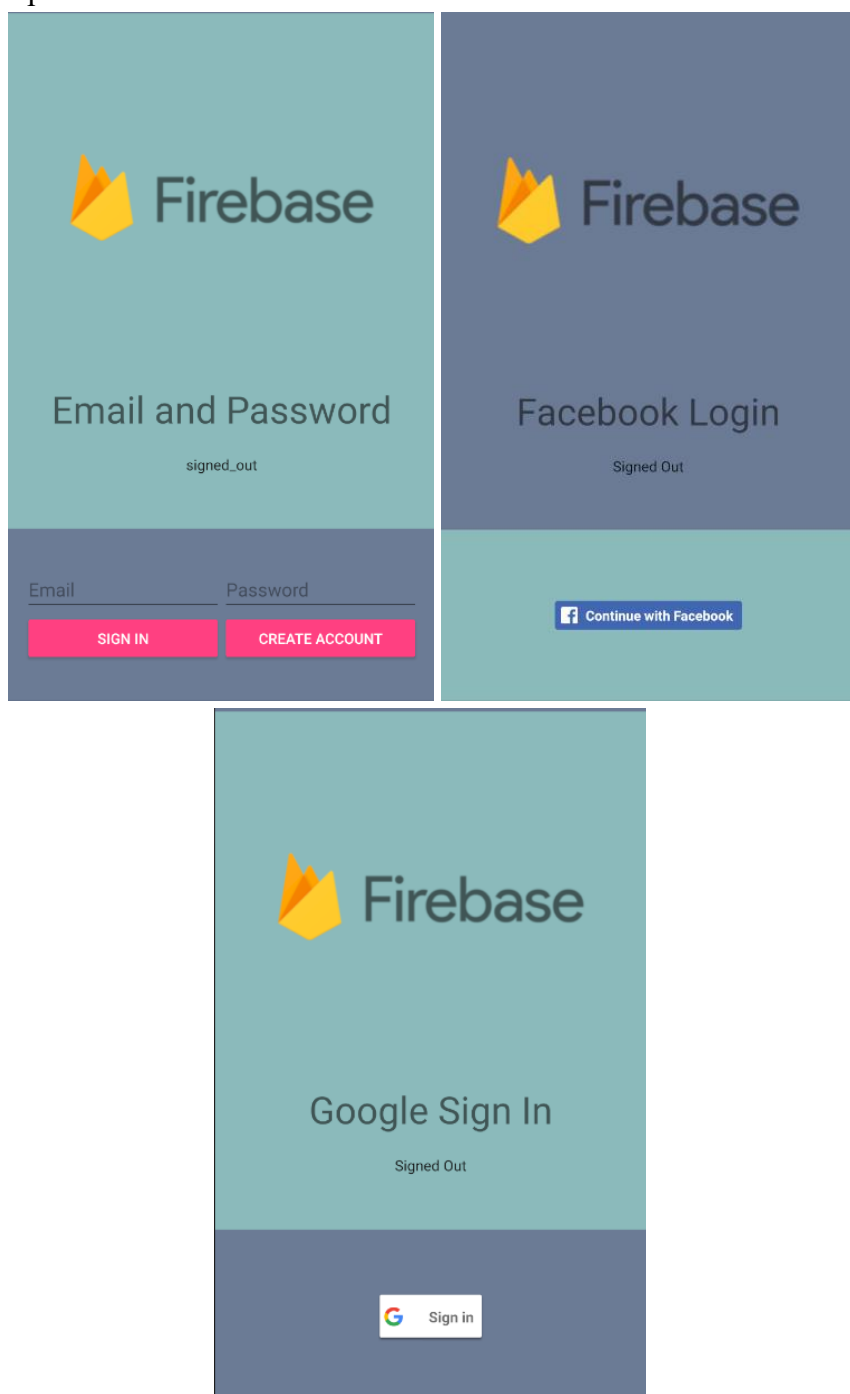


Figura V.1: Metode de autentificare

În urma autentificării, utilizatorul va fi redirecționat în ecranul principal al aplicației, oferit de activitatea DashboardActivity. Aici utilizatorului îi va fi afișată adresa la care se află, urmând ca acesta să poată alege ce fel de servicii dorește:

- Home – servicii la domiciliu;
- Away – servicii la sediul furnizorului.

În acest moment aplicația deține infrastructura necesară susținerii unei multitudini de furnizori de diferite servicii, dar pentru simplificare, am introdus în baza de date doar câțiva furnizori, iar cel mai bine structurat furnizor este cel de Stomatologie, unde am adăugat o multitudine de servicii și poze. Deocamdată baza de date conține doar informații false, adăugate doar pentru exemplificarea funcționării aplicației.

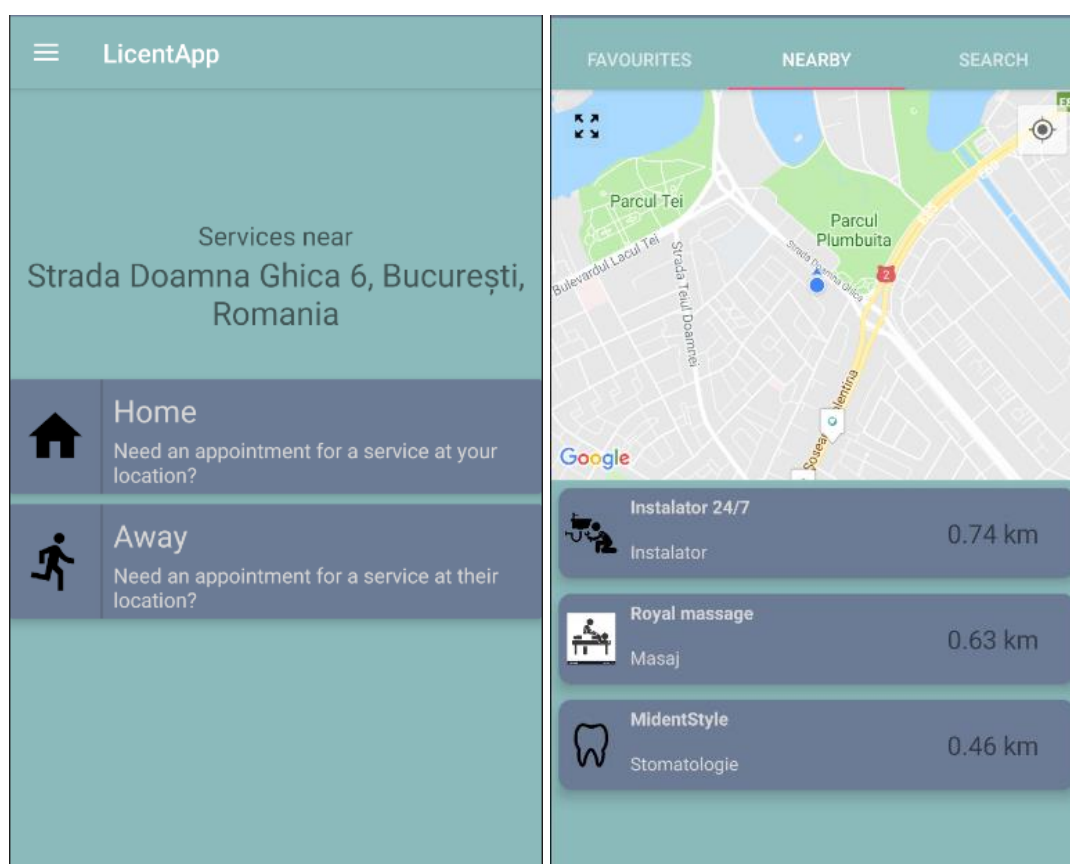


Figura V.2: Ecranul principal

În urma selectării tipului de servicii, va fi afișată lista furnizorilor de servicii cerute împreună cu harta unde se vor putea vedea aceste servicii și locația utilizatorului.

Din această listă se poate selecta oricare dintre furnizorii de servicii afișați. Utilizatorul va fi redirecționat pe pagina corespunzătoare furnizorului accesat, urmând a fi afișată lista serviciilor pe care acesta le pune la dispoziție. În această activitate fiecare serviciu va avea o scurtă descriere, un preț și o anumită durată setate de furnizorul serviciilor respective.





Figura V.3: Ecranul unui furnizor de servicii și ecranul unui serviciu

Ecranul unui serviciu conține informații despre acesta, fie că vorbim despre o poză, o descriere, preț sau durata acestuia. Pentru obținerea unei programări, utilizatorului îi este sugerat să aleagă data, ora și să adauge anumite informații suplimentare sub forma de Notă.

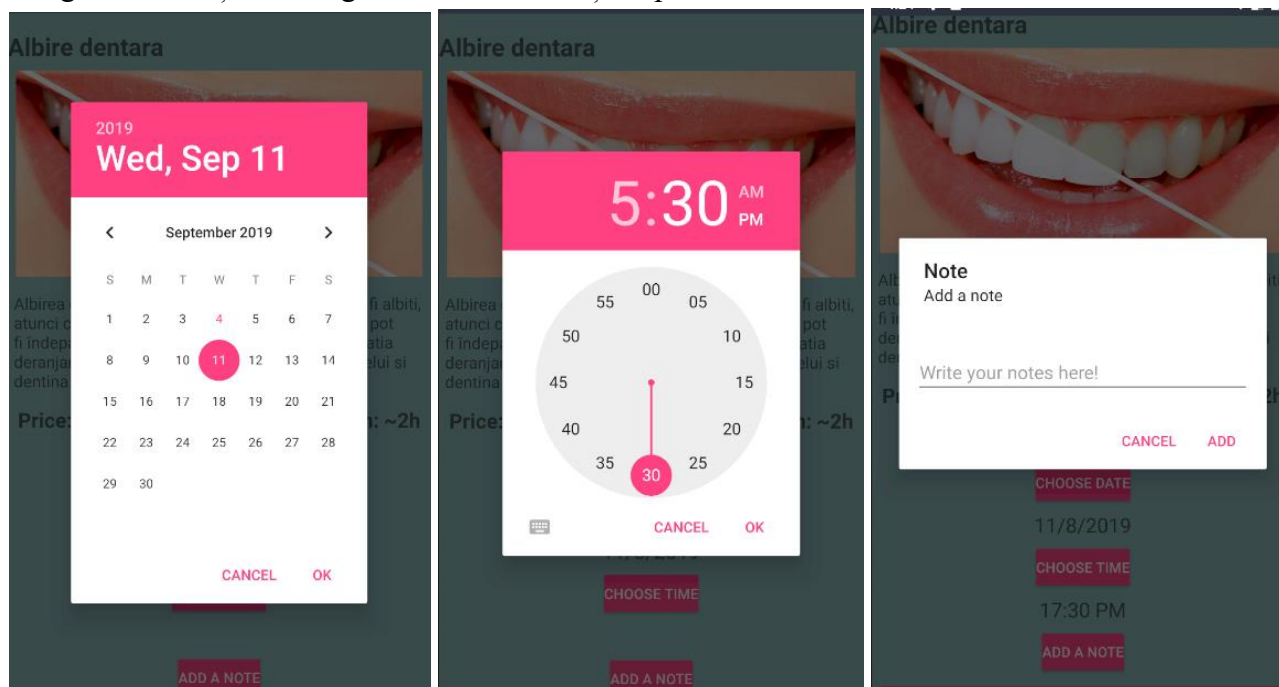


Figura V.4: Obținerea unei programări

După alegerea datelor necesare creării unei programări(data și ora) va apărea și butonul pentru crearea acestuia. Acesta este ascuns pentru că operațiunea nu este posibilă fără o dată și o oră.

Utilizatorul va putea să verifice stadiul programărilor pe care le-a solicitat în cadrul ecranului profilului personal, MyProfileActivity.

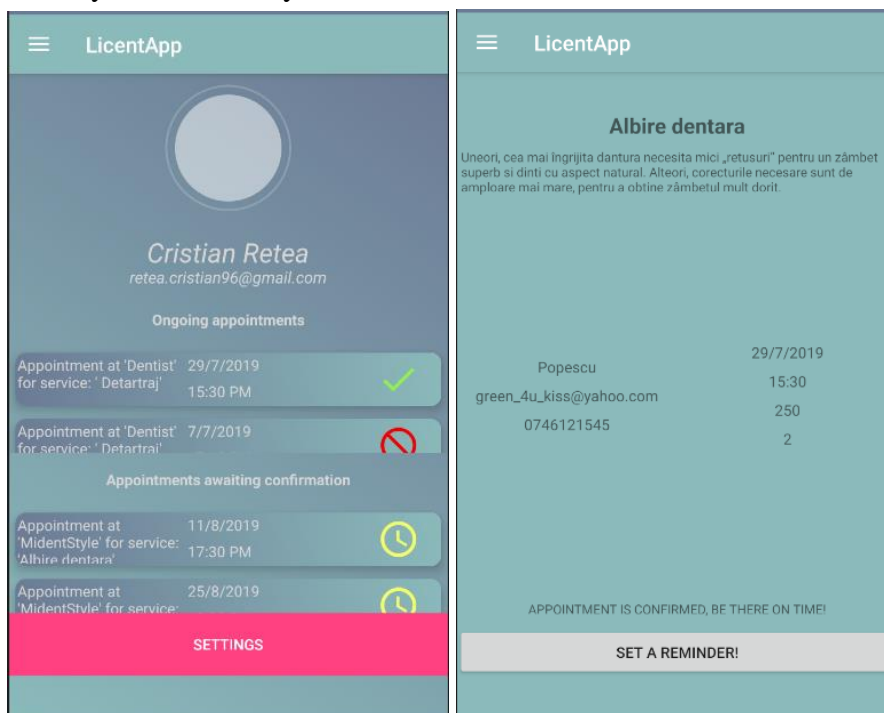


Figura V.5: Ecranul profilului utilizatorului și ecranul programării

Se poate vedea stadiul fiecărei programare prin imaginea atașată acestuia în liste, dar utilizatorul poate de asemenea să acceseze una dintre programări pentru a verifica detalii despre programarea pe care o are.

Din punctul de vedere al administratorului serviciilor cerute nu vor fi decât două ecrane, cel în care acesta să vadă toate programările în listele sale și un ecran în care să poată vedea detaliile programării accesate.

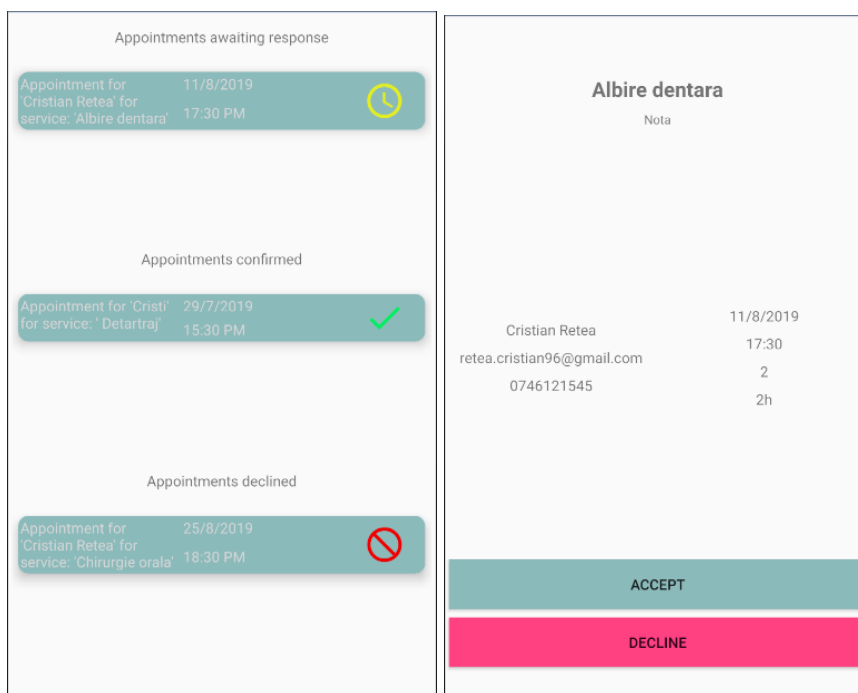


Figura V.6: Ecranele aplicației administratorului unui serviciu

# Concluzii

## *Metode de monetizare*

Din punct de vedere al monetizării aplicației se pot considera câteva tipuri de interacțiuni. Se pot considera următoarele metode:

- metodă de monetizare prin care furnizorii de servicii plătesc pentru a avea serviciile prezente în cadrul aplicației, aceasta fiind, eventual, o taxă lunară;
- metodă de monetizare prin care furnizorii au serviciile publicate gratuit în cadrul aplicației, dar plătesc o anumită taxă pentru fiecare client pe care îl au;
- metodă mixtă, o combinație dintre cele două de mai sus;
- reclame în cadrul aplicației.

Gândul cu care s-a realizat aplicația a fost semnarea unui contract între furnizorul de servicii și administratorul aplicației astfel încât serviciile să fie verificate și calitatea acestora să fie ridicată. În acest mod se poate modera foarte ușor conținutul aplicației.

## *Probleme întâmpinate*

Am întâmpinat o serie de probleme, în principal în momentul implementării funcționalităților aplicației. O problemă mare a fost faptul că în momentul proiectării aplicației am decis să folosesc anumite funcționalități ale IDE-ului folosit, Android Studio, funcționalități ce au fost scoase din cadrul acestuia până la momentul implementării lor. Amintesc aici:

- Android a decis ca ProgressDialog să nu mai fie folosit pentru că blochează interfața vizuală a aplicației. Acest dialog era folosit pentru a fi arătat utilizatorului în momentul în care aplicația era solicitată prin faptul că interfața vizuală depindea de obținerea anumitor informații. Astfel a trebuit să obțin informația în prealabil pentru a putea-o arăta fără a bloca aplicația.
- În timpul implementării Android a decis ca anumite modele de design să fie clasificate ca fiind legacy(vintage/vechi) și trecute într-o bibliotecă suplimentară, iar tot proiectul a trebuit să fie transformat într-unul care să suporte o astfel de schimbare.
- Android a decis ca aplicațiile să nu mai poată fi abonate la procese din cadrul sistemului, astfel încât notificarea administratorului și a clientului în momentul actualizării unei programări a fost foarte dificilă.

De asemenea, Firebase a decis că pentru a notifica câte un singur utilizator la un moment dat este neapărat nevoie de un server separat al utilizatorului și un serviciu de tip pagină web pentru trimiterea respectivelor notificări. Până la urmă nu a mai fost implementată funcționalitatea de notificare a utilizatorilor din cauza faptului că pe lângă cunoștințele necesare implementarea paginii web și server-ului, erau necesare cunoștințe pentru implementarea unui serviciu Firebase Functions, ceea ce la momentul actual nu posed.

Am reușit implementarea metodele de obținere a fotografiilor de profil pentru actualizarea pozelor utilizatorilor, dar nu am reușit stocarea acestora. Pentru stocarea acestora se poate folosi Firebase Storage.

## ***Contribuțiile personale***

Contribuția personală constă în realizarea unei aplicații Android care permite folosirea unei serii de funcționalități. Dintre toate, amintesc realizarea unor funcționalități precum:

- UX/UI design;
- Autentificare prin trei metode: Email și parolă, Facebook sau Google;
- Obținerea, stocarea și afișarea locației dispozitivului;
- Transmiterea informației dintre activități pentru încărcarea conținutului în interfața vizuală;
- Crearea bazei de date; descărcarea, stocarea și afișarea informației din baza de date.
- Stocarea informației într-un fișier al telefonului;
- Posibilitatea utilizatorilor să își urmărească programările, statutul acestora și istoricul lor.

## ***Dezvoltări ulterioare***

Aplicația poate fi dezvoltată mai departe prin eficientizarea unor funcționalități deja prezente în aplicație, precum și prin adăugarea altor funcționalități necesare. Printre acestea se numără:

- Notificările;
- Adăugarea fragmentelor pentru care este deja creată infrastructura: fragmentul de căutare a serviciilor și fragmente de servicii preferate;
- Adăugarea unui suplinitor ProgressBar-ului pentru a putea eficientiza încărcarea anumitor elemente de design, evitând astfel momente în care anumite elemente se încarcă mai repede decât altele;

## ***Rezumat***

Din analiza realizată inițial am aflat faptul că piața aplicațiilor Android este într-o continuă creștere și faptul că aplicațiile care ajută utilizatorii când vine vorba despre ușurarea vieții acestora prin economisirea timpului lor printr-o metoda sau alta sunt în vogă.

Dezvoltarea de aplicații Android nu necesită achiziționarea unei licențe, dezvoltarea fiind gratuită și la îndemâna oricărei persoane care dorește să creeze ceva.

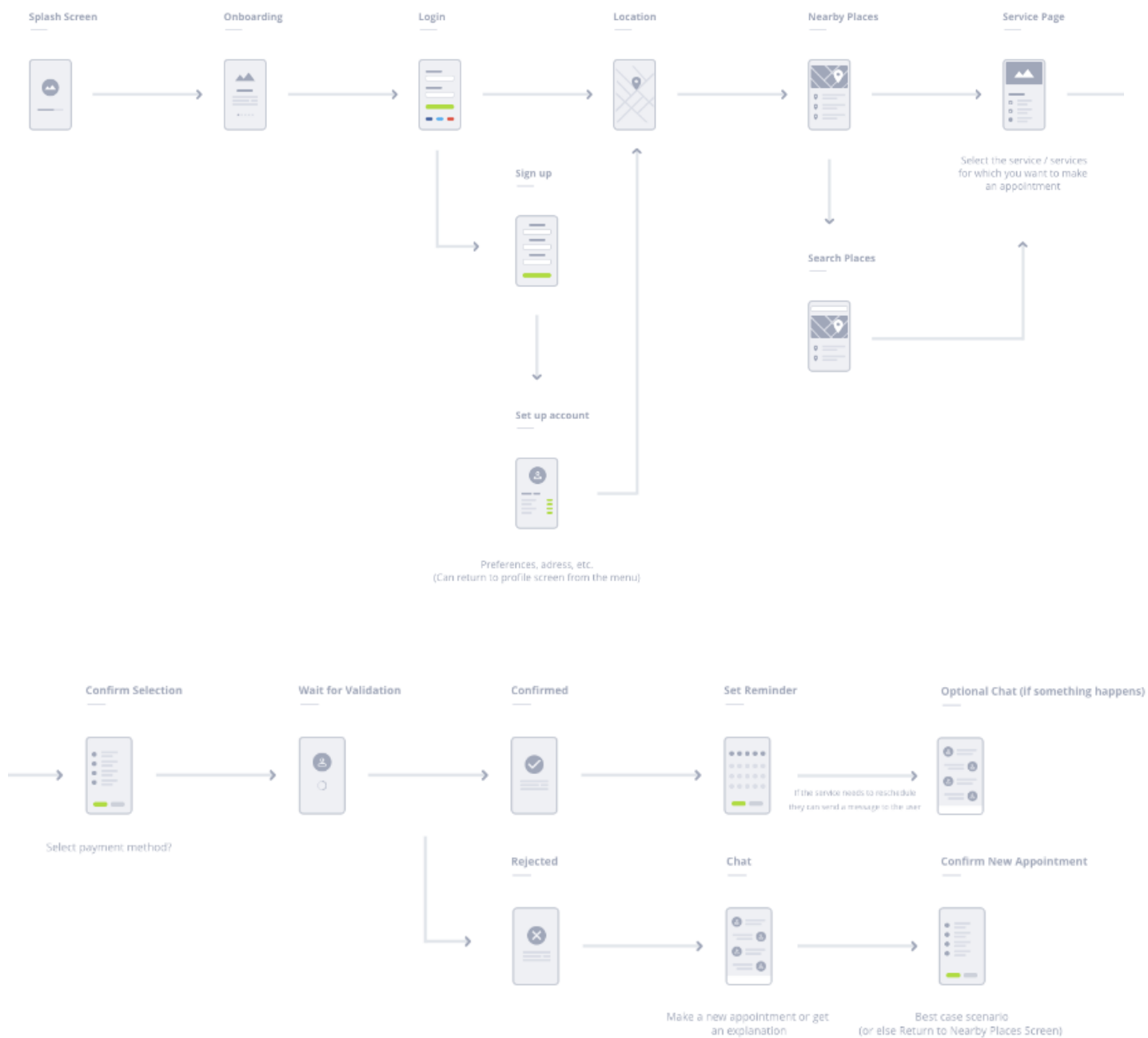
În urma analizei conceptului inițial am proiectat o aplicație care nu părea a fi foarte complicată sau greu de realizat, dar la momentul implementării tuturor funcționalităților s-a dovedit necesitatea multor altor lucruri pentru ca aplicația să funcționeze cum mi-am dorit.

Una dintre provocările cele mai mari a fost realizarea design-ului aplicației, fiind necesar să iau în considerare cum ar fi pentru utilizatori mai ușor și mai intuitiv să folosească funcționalitățile oferite.

## Bibliografie

- [1] Ghiduri Android din documentația oficială, <https://developer.android.com/guide>
- [2] Documentație Android, <https://developer.android.com/docs>
- [3] Documentație Firebase, <https://firebase.google.com/docs>
- [4] <https://techjury.net/stats-about/app-usage/>, accesat la data: 13/8/2019
- [5] Americans check their phones 80 times a day: study, <https://nypost.com/2017/11/08/americans-check-their-phones-80-times-a-day-study/>
- [6] Mobile App Download and Usage Statistics (2019), <https://buildfire.com/app-statistics/>
- [7] Java, <https://www.theserverside.com/definition/Java>, accesat la data: 15/8/2019
- [8] Android, [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)), accesat la data: 15/8/2019
- [9] What is Firebase? <https://howtofirebase.com/what-is-firebase-fcb8614ba442>, accesat la data: 15/8/2019

## Anexa 1 – Diagrama flux de activități destinată clienților



## ***Anexa 2 – Diagrama flux de activități a aplicației destinate administratorilor serviciilor***

