UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

APLICAȚIE MOBILE PENTRU PROGRAMĂRI MEDICALE

Absolvent
Popa Iulia-Andreea

Coordonator științific
Prof. Dr. Ștefănescu Alin

București, Septembrie 2022

Abstract

Trăind în secolul vitezei și dorind să realizeze cât mai multe dintre lucrurile pe care și le propun, oamenii pot neglija unele aspecte importante, precum sănătatea lor. Fie că uită sau că amână controlul medical din diverse motive, efectele acestor acțiuni pot fi în defavoarea pacientului pentru că, după cum știm cu toții, vindecarea este considerată a fi mai grea decât prevenția.

Lucrarea de licență cu titlul „Aplicație mobile pentru programări medicale" este o aplicație Android ce vine în ajutorul acestor oameni. Din considerentele menționate anterior, soluția prezentată are scopul de a oferi utilizatorilor optimizarea procesului de programare a unui control medical, un plus de siguranță împotriva efectelor cauzate de apariția evenimentelor neprevăzute precum și o utilizare facilă.

Acest tip de soluție necesită accesul lipsit de efort al publicului căruia i se adresează, motiv pentru care am ales dezvoltarea unei aplicații mobile, dată fiind utilizarea frecventă a unui smartphone cu acces la Internet în ziua de azi.

Sistemul de operare Android stă la baza a peste 2,5 miliarde de dispozitive active [1], motiv pentru care aplicația prezentată a fost dezvoltată pe baza lui, cu ajutorul limbajului de programare Kotlin. Am folosit ahitectura MVVM care asigură o bună structurare și o mentenanță facilă a codului, ușurință a testării și scalabilitate. Atât pentru autentificarea și notificarea utilizatorilor, cât și pentru stocarea datelor din aplicație și salvarea fișierelor în Cloud am folosit servicii Firebase. De asemenea, pentru încărcarea imaginilor mi-a fost de folos librăria Glide. Pentru a le oferi utilizatorilor cât mai multe informații, am folosit Google Maps API și Directions API pentru afișarea locației clinicii, calcularea distanței între aceasta și locația curentă a pacientului precum și afișarea traseului recomandat. Pe plan tehnic, pentru *request-urile* de tip HTTP către Directions API, am folosit librăria OkHttp dezvoltată de Square și librăria Gson pentru prelucrarea datelor între baza de date și aplicație.

Produsul final este o aplicație realizată printr-un proces bine structurat, ușor de folosit pentru publicul țintă și cu un potențial real.

Abstract

Living in the fast-paced century and wanting to achieve as many of the things they set out to do as possible, people can neglect some important aspects, such as their health. Whether they forget or put off medical check-ups for various reasons, the effects of these actions can be detrimental to the patient because, as we all know, healing is considered to be harder than prevention.

The thesis entitled "Mobile app for medical appointments" is an Android app that comes to the aid of these people. From the above mentioned considerations, the presented solution aims to provide users with optimization of the process of scheduling a medical check-up, added safety against the effects caused by the occurrence of unforeseen events as well as an easy use.

This type of solution requires effortless access to the target audience, which is why I chose to develop a mobile application, given the frequent use of a smartphone with Internet access nowadays.

The Android operating system underpins over 2.5 billion active devices [1], which is why the presented app was developed based on it, using the Kotlin programming language. We used the MVVM architecture which provides good structuring and easy code maintenance, ease of testing and scalability. Both for user authentication and notification, as well as for storing application data and saving files to the cloud, we used Firebase services. I also found the Glide library useful for uploading images. To provide users with as much information as possible, I used Google Maps API and Directions API for displaying the location of the clinic, calculating the distance between it and the patient's current location as well as displaying the recommended route. Technically, for HTTP requests to Directions API, we used the OkHttp library developed by Square and the Gson library for processing data between the database and the application.

The end product is an app that is built through a well-structured process, easy to use for the target audience and with real potential.

Cuprins

Introducere 8
Context și motivație 8
Structura lucrării 9
I. Tehnologii folosite 11
I.1 Limbaje de programare 11
I.1.1 Kotlin 11
I.1.2 XML 12
I.2 Sistemul de operare – Android 12
I.2.1 Introducere 12
I.2.2 Structura fișierelor unei aplicații Android 13
I.2.3 Android Jetpack 14
I.3 Firebase 14
I.3.1 Firebase Authentication 14
I.3.2 Firebase Realtime Database 15
I.3.3 Firebase Cloud Messaging 16
I.3.4 Cloud Storage for Firebase 16
I.4 Maps SDK for Android 17
I.5 Directions API 18
I.6 Librării 18
I.6.1 Glide 18
I.6.2 OkHttp 19
I.6.3 Android Lifecycle (Jetpack) 19
I.6.4 Material Components for Android (MDC-Android) 20
II. Analiza aplicației 21
II.1 Grupul țintă și nevoile pe care aplicația le satisface 21
II.2 Prioritizarea funcționalităților 22
II.3 Puncte forte și puncte slabe 23
II.4 Aplicații similare 24
II.4.1 Evexia 24
II.4.2 DocBook – Programări la doctor 25
II.4.3 Regina Maria 25
Concluzii 25
II. 5 Ce aduce nou? 26
III. Arhitectura aplicației 27
III.1 Structura aplicației 27

III.2 Structura bazei de date....29
IV. Funcționalitățile aplicației....32
IV.1 Splash Screen....32
IV.2 Înregistrarea și logarea în aplicație....33
IV.3 Profilul utilizatorului....34
IV.4 Vizualizarea tuturor programărilor....35
IV.5 Crearea unei programări noi....36
IV.5.1 Completarea formularului....36
IV.5.2 Adăugarea programării în calendarul personal....37
IV.6 Vizualizarea detaliilor unei programări....37
IV.6.1 CardView cu detaliile programării....37
IV.6.2 Apelare din aplicație....37
IV.6.3 Share la detaliile programării....38
IV.6.4 Ștergerea unei programări....38
IV.6.5 Harta cu locația clinicii....38
IV.7 Vizualizarea detaliilor pacientului....39
IV.8 Schimbarea programării cu alt pacient....39
IV.8.1 Ecranul Emergency....40
IV.8.2 Pagina cu programul doctorului....40
IV.8.3 Cererea de schimb....41
IV.9 Bara de navigare....41
IV.10 Meniul din bara de acțiuni....41
IV.11 Notificări....42
IV.12 Mesaje informative, validări și *pop-ups*....43
Concluzii....44
Perspective....45
Bibliografie....46

Lista Figurilor

Figura 1.1 Structura aplicației 13
Figura 1.2 Stocarea informațiilor în Firebase Authentication 15
Figura 1.3 Stocarea datelor în Realtime Database.....15
Figura 1.4 Programarea campaniei de notificări 16
Figura 1.5 Stocarea imaginilor în Cloud Storage 17
Figura 1.6 Harta cu locația clinicii.....17
Figura 1.7 Interfața în timpul încărcării programărilor 19
Figura 2.1 Graficul priorității funcționalităților 22
Figura 3.1 Arhitectura MVVM [27] 27
Figura 3.2 Ecranele aplicației 28
Figura 3.3 Tabelul appointments 29
Figura 3.4 Tabelul doctors 29
Figura 3.5 Tabelul locations 30
Figura 3.6 Tabelul users 30
Figura 3.7 Tabelul requests 30
Figura 3.8 Regulile de securitate ale bazei de date 31
Figura 4.1 Splash Screen.....32
Figura 4.2 Înregistrarea în aplicație 33
Figura 4.3 Logarea în aplicație 33
Figura 4.4 Profilul utilizatorului 1 34
Figura 4.5 Profilul utilizatorului 2 34
Figura 4.6 Pagina programărilor pacientului 35
Figura 4.7 Pagina programărilor din clincă 35
Figura 4.8 Formularul pentru crearea programării 1 36
Figura 4.9 Formularul pentru crearea programării 2 36
Figura 4.10 Detaliile programării pentru pacient 37
Figura 4.11 Detaliile programării pentru administrator 37
Figura 4.12 Fereastra informativă din hartă.....38
Figura 4.13 Afișarea traseului recomandat 39
Figura 4.14 Detaliile pacientului 39
Figura 4.15 Pagina Emergency.....40

Figura 4.16 Programul doctorului 40
Figura 4.17 Cerere de schimbare a programării 41
Figura 4.18 Bara de navigare 41
Figura 4.19 Meniul din bara de acțiuni 41
Figura 4.20 Ecranul fără notificări 42
Figura 4.21 Ecranul cu notifcări 42
Figura 4.22 Mesaje informative și de confirmare 43

Introducere

Context și motivație

Cu toții ne dorim să ducem la bun sfârșit cât mai multe dintre lucrurile notate în agenda noastră încărcată. Într-o lume în care timpul zboară mai repede ca niciodată și în care oamenii încearcă să fie din ce în ce mai productivi, soluțiile care rezolvă sarcinile importante într-un timp scurt sunt cele care ne salvează.

Acest lucru și-l propune și aplicația „Smart Planner", venind cu o soluție pentru optimizarea timpului petrecut atât de pacienți, cât și de personalul medical în procesul de programare a vizitelor la medic.

Cu doar câteva clickuri, pacientul poate creea o programare la clinica și specializarea dorită, evitând astfel așteptarea îndelungată pe linia telefonică, eventuala căutare în baza de date a clinicii pentru un pacient deja existent, sau crearea unui profil nou pentru un pacient la prima programare. Astfel, este economisit timpul atât al pacientului, cât și al personalului medical responsabil pentru gestionarea programărilor.

Inițial, pacientul își creează un cont în aplicație fie cu ajutorul unul email personal, fie cu contul său de Google sau de Facebook. Acesta își va putea adăuga apoi datele personale necesare în pagina de profil. Un lucru care trebuie menționat este faptul ca utilizatorii nu pot folosi aplicația fără crearea unui cont.

În cadrul aplicației, pacientul are acces în orice moment la detalii despre programările sale, despre clinicile alese sau disponibilitatea doctorilor, pe propriul *smartphone*, atâta timp cât acesta are acces la Internet. El poate vedea inclusiv istoricul programărilor.

Mai mult de atât, într-un scenariu nefavorabil în care apariția unor evenimente neprevăzute cauzează imposibilitatea pacientului de a onora o programare, acesta poate solicita unui alt pacient care are la rândul său o programare la o dată convenabilă pentru primul, să facă schimb, la acordul acestuia din urmă. Astfel, în cazul unei astfel de potriviri, el nu ar mai avea nevoie de o nouă programare pentru care ar putea aștepta chiar câteva săptamâni și nu va mai fi nevoit să anuleze programarea inițială cu scurt timp înainte de data stabilită. În felul acesta se poate reduce numărul programărilor neonorate sau anulate din scurt.

Pacienții primesc notificări în aplicație pentru a li se reaminti că urmează o programare în următoarele zile sau notificări de încurajare pentru programarea unei vitize medicale după ce aceștia nu au folosit aplicația o perioadă mare de timp.

De asemenea, aplicația este destinată personalului medical. Membrii se vor conecta în aplicație cu rolul de administrator și pot ține evidența programărilor clinicii, le pot gestiona și au acces la informațiile necesare despre pacienți. Aceștia primesc notificări atunci când pacienții adaugă sau modifică date personale în profilul lor.

Motivația principală care a stat la baza dezvoltării aplicației în discuție a fost soluționarea unei probleme reale, cu care se confruntă un anumit public țintă, în cazul de față unul larg și fără prea multe criterii restrictive. Intenția a fost de a obține un produs finit cu un scop clar, care să aibă un impact pozitiv asupra publicului căruia i se adresează.

Astfel, alegerea temei a pornit pe baza unor experiențe proprii, în urma cărora am realizat cât timp, energie sau alte resurse pot fi pierdute din lipsa eficientizării unor sarcini repetitive de care ne lovim cu toții la un moment dat. Așteptând în medie 15 minute pe linia telefonică în multe dintre dățile când am vrut să îmi fac o programare la medic, și apoi încă 10 în apel pentru oferirea informațiilor personale și găsirea de comun acord a unui interval potrivit, am realizat că acest proces nu are nevoie exclusiv de interacțiune umană și poate fi mult simplificat în mediul digital.

Mai mult decât atât, atunci când, din cauza unor probleme neprevăzute ce nu au depins de mine, am realizat cu trei zile înainte de un control medical programat cu 4 săptămâni în urmă că acesta trebuie anulat și reprogramat, iar următorul loc disponibil fiind la 3 săptămâni distanță, am decis că trebuie să existe o soluție pentru astfel de situații, una care să diminueze riscul pierderii unei programări din motive obiective sau subiective.

Astfel, aplicația „Smart Planner” își propune să optimizeze gestionarea si crearea programărilor medicale din punct de vedere al timpului petrecut, efortului depus și al siguranței oferite.

Structura lucrării

Lucrarea este structurată în 4 capitole, astfel:

În **Capitolul I** sunt prezentate tehnologiile folosite, de la limbaje de programare, la framework-uri, API-uri, librării sau alte specificații tehnice care au contribuit la dezvoltarea aplicației împreună cu justificarea utilizării acestora în aplicație bazată pe adaptarea aspectelor tehnice la funcționalitățile aplicației.

În **Capitolul II** conține un raport de analiză a aplicației, prioritizarea sarcinilor înainte de implementare și analiza altor aplicații similare împreună cu factorul diferențiator al proiectului.

Capitolul III descrie arhitectura aplicației, mai precis structura proiectului și structura bazei de date.

În **Capitolul IV** sunt prezentate pe larg funcționalitățile aplicației precum și detalierea implementării lor.

Lucrarea se încheie cu o sinteză a produsului final prin **Concluzii**, precum și descrierea **Perspectivelor** de dezvoltare ulterioară și îmbunătățire a aplicației.

I. Tehnologii folosite

În acest capitol vor fi prezentate tehnologiile care au stat la baza implementării aplicației, precum limbaje de programare, librării, framework-uri, servicii si API-uri folosite, motivând de asemenea alegerea acestora în procesul de dezvoltare, utilitatea lor în context și impactul adus.

I.1 Limbaje de programare

I.1.1 Kotlin

Kotlin este un limbaj de programare modern, dezvoltat de JetBrains și lansat pentru prima dată în Iulie 2011, urmând ca apoi să aibă sursa deschisă publicului larg (*open-source*) din 2012. Este un limbaj cross-platform, poate fi utilizat în diverse domenii de programare, dar este cunoscut cel mai mult pentru aplicațiile mobile ce rulează pe device-uri cu sistemul de operare Android [5].

Kotlin este un limbaj ce îmbină perfect programarea orientată pe obiect cu programarea funcțională [5], oferindu-le dezvoltatorilor ocazia de a aplica cele 4 principii fundamentale ale primei paradigme: abstractizare, încapsulare, moștenire și polimorfism, precum și simplificarea substanțială a unor sarcini complicate prin linii de cod succinte și modularizare.

Principiile programării orientate pe obiect au fost utilizate pe tot parcursul dezvoltării aplicației mele. Încapsularea și abstractizarea au fost de folos în definirea claselor necesare în implementare, modularizarea aplicației și stabilirea unor clasificatori de acces pentru proprietățile, metodele și starea obiectelor, mai precis pentru furnizarea informațiilor strict necesare și nimic mai mult. Prin moștenire, am folosit proprietățile predefinite ale claselor de bază (*Activity, Fragment*) sau ale interfețelor (*OnMapReadyCallback*) pentru a le adăuga apoi proprietăți proprii necesare pentru funcționalitățile dezvoltate. Prin polimorfism, am suprascris diverse metode de bază precum *onCreate(), onCreateView(), onMapReady(), onRequestPermissionsResult()* și multe altele.

Paradigma de programare funcțională a contribuit la scrierea unui cod robust, reutilizabil și ușor de testat datorită utilizării compunerii funcțiilor, a expresiilor lambda și a stărilor imutabile ale obiectelor. Acestea au inclus în principal procesarea și prelucrarea datelor, un exemplu fiind obținerea programărilor din baza de date, transformarea lor în obiecte și ordonarea după anumite criterii, totul în doar câteva linii de cod.

O altă ramură extrem de importantă a limbajului este programarea asincronă sau programarea *non-blocking* [6]. Aceasta oferă utilizatorilor o experiență fără întreruperi și un flow fluid. Folosirea librăriei *kotlinx.couroutines* [7] mi-a fost necesară în procesul de încărcare în aplicație a datelor extrase din baza de date, precum și pentru așteptarea răspunsurilor de la *request-urile* către API-uri.

Null Safety este măsura de precauție pe care dezvoltatorii care scriu cod Kotlin o pot lua împotriva des întâlnitei erori a referinței nule [8]. Astfel, în dezvoltarea aplicației am avut o breșă de securitate suplimentară atunci când am încercat să extrag informații din baza de date sau le-am transmis dintr-un punct al aplicației în altul.

I.1.2 XML

XML (Extensible Markup Language) este un limbaj de marcare similar cu cel HTML, care permite formatarea elementelor din interfață utlizând elemente imbricate. Astfel, pentru crearea *layout-urilor* din aplicație, în fiecare fișier de tip XML am creat un element părinte (de tip View sau ViewGroup) în interiorul căruia am inserat alte elemente cu diverse proprietăți, fie independente, fie relative la părinte [9]. Astfel, am obținut un cod bine structurat, ușor de citit și de modificat.

În cadrul aplicației, în clasele fragmentelor sau activităților, am creat elemente de tip XML în mod dinamic, spre exemplu afișarea unui pop-up în care administratorul este întrebat daca este sigur că vrea să șteargă programarea curentă, după apăsarea butonului. Am modificat, de asemenea, dinamic proprietăți deja existente ale elementelor din fișierele XML, ca de exemplu ascunderea sau afișarea unei secțiuni la apăsarea unui buton sau îndeplinirea unei condiții.

I.2 Sistemul de operare – Android

I.2.1 Introducere

Android este un sistem de operare open-source destinat majoritar smartphone-urilor și tabletelor, având la bază un nucleu Linux în principal pentru funcționalitățile bazale precum firele de execuție sau gestionarea memoriei low-level [10].

Bucurându-se de un public de peste 2,5 miliarde de oameni cu un dispozitiv ce are la bază acest sistem de operare [1], am decis ca aplicația Smart Planner să fie o aplicație Android, pentru a putea fi de ajutor și la îndemâna cât mai multor oameni.

Pentru dezvoltarea acesteia am ales să folosesc Android Studio, Android Gradle Plugin 7.0.2 și Android API 32.

I.2.2 Structura fișierelor unei aplicații Android

Folosind ca mediu de dezvoltare Android Studio, aplicația mea a respectat structura unei aplicații standard [3] ce poate fi observată în figura 1.1:

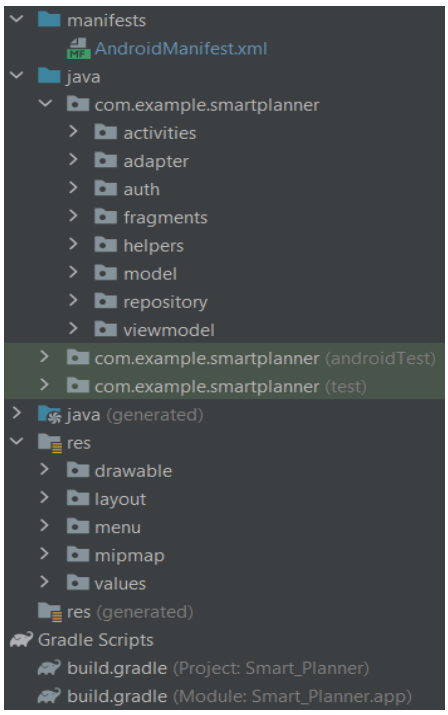

Figura 1.1 Structura aplicației

- În fișierul *AndroidManifest.xml* se definesc componentele aplicației, specificațiile necesare precum permisiunile de care aceasta va avea nevoie pe parcursul experienței utilizatorului, cât și structura și conținutul metadata. Aici se configurează activitatea care inaugurează aplicația, tema sau icon-ul aplicației.
- În folderul *java* se găsesc fișierele cu codul sursă, în subfoldere numite *package-uri*. Acestea funcționează ca un *controller* pentru fișierele UI.
- Folderul *res* conține toate fișierele care nu au cod sursă, împărțite în subfoldere în funcție de categorie, astfel:

  - În directorul *drawable* se găsesc toate fișierele de tip *drawable resource*, adică cele de tip grafic, care pot fi desenate pe ecran. Ele pot fi imagini Bitmap (PNG, JPEG), vector (XML), layers, etc.

  - În folderul *layout* se găsesc toate fișierele XML care construiesc interfața aplicației

  - În folderul *mipmap* pot fi generate icon-uri pentru bara de acțiuni, de navigare sau notificări.

  - Folderul *values* este și el împarțit in subcategorii, pentru resurse de tip string, culori, stiluri sau dimensiuni.
- Fișierul *build.gradle(Project)* definește configurarea tuturor modulelor proiectului.
- Fișierul *build.gradle(Module: app)* conține toate dependințele necesare pentru implementarea funcționalităților.

I.2.3 Android Jetpack

Android Jetpack este o colecție de librării ce ajută la dezvoltarea unor aplicații Android cu cele mai bune practici de implementare, cod robust și o arhitectură stabilă.

Pentru a fi posibilă utilizarea pachetului, este necesară includerea dependinței *androidx.*package* în fișierul *build.gradle(Module: app)* al proiectului.

Acesta ajută la simplificarea cerințelor complexe, reduce redundanța codului și ajută la dezvoltarea funcționalităților care lucrează în mod consistent între dispozitive cu versiuni diferite ale sistemului de operare Android [11].

Din colecție am inclus în proiect majoritar librării din componenta de arhitectură, mai precis librării din pachetul *lifecycle* sau librării care contribuie la implementarea interfeței aplicației (*cardview, recyclerview, modelview*).

O compontentă importantă pe care am integrat-o de asemenea în cadrul proiectului a fost *Navigation Component* ce a permis navigarea între ecranele aplicației prin intermediul butoanelor sau a barei de navigare.

I.3 Firebase

Firebase este o platformă dezvoltată de Google care este menită să ajute la dezvoltarea aplicațiilor web și mobile, punând la dispoziție diverse instrumente de back-end precum sistemul de autentificare, stocarea datelor în Cloud sau baza de date de tip NoSQL. Oferă, de asemenea, tehnologii de monitorizare a aplicațiilor ce includ statistici despre stabilitate și performanță [4], pentru ca dezvoltatorii să asigure o experiență cât mai bună utilizatorilor.

Firebase oferă soluții scalabile pentru stocarea datelor și a fișierelor întrucât timpul de răspuns este foarte rapid, oferă disponibilitatea datelor, un lucru esențial în dezvoltarea unei aplicații de uz zilnic și dispune de asemenea de o multitudine de servicii cu ajutorul cărora pot fi dezvoltate funcționalități elementare în orice proiect.

I.3.1 Firebase Authentication

Firebase Authentication este un serviciu de back-end care oferă un sistem de autentificare securizat incorporabil în aplicații web sau mobile [12]. Simplitatea integrării serviciului în raport cu valoarea adusă aplicației este un avantaj care a dus la utilizarea acestuia în cadrul proiectului.

Ca primă opțiune, am integrat autentificarea utilizatorului folosind email și parolă. Firebase securizează stocarea informațiilor cu caracter sensibil, mai precis folosește un algoritm

numit *scrypt* [13] cu ajutorul căruia parolei utilizatorului îi este aplicată o funcție hash care se concatenează apoi cu valoarea arbitrară *salt* necesară pentru a oferi un plus de securitate.

Există de asemenea opțiunea de autentificare folosind numărul de telefon sau un cont existent furnizat de mari platforme precum Facebook, Gmail, GitHub, Twitter sau Apple. Dintre aceste servicii am ales autentificarea cu conturile de Facebook sau Gmail.

Pentru fiecare utilizator este asociată automat o valoare unică numită *User uid* care este vizibilă în Firebase Console, împreună cu alte date despre cont precum emailul, furnizorul, data creării contului și data ultimei autentificări. Parolele utilizatorilor nu sunt vizibile dezvoltatorilor în consolă, conform figurii 1.2, ceea ce oferă un plus de securitate. Asocierea id-ului unic fiecărui utilizator mi-a fost de folos ulterior pentru stocarea datelor suplimentare despre aceștia în baza de date, folosindu-mă de această valoare drept intrare unică, evitând riscul de suprascriere a datelor din cauza coliziunilor.

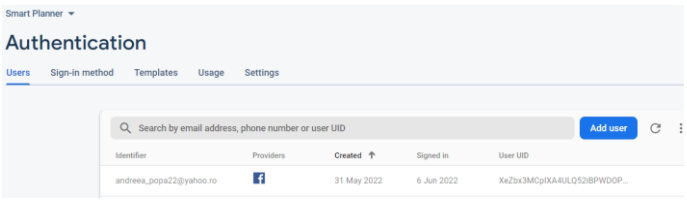

Figura 1.2 Stocarea informațiilor în Firebase Authentication

I.3.2 Firebase Realtime Database

Firebase Realtime Database este o bază de date cloud-hosted în care datele sunt sincronizate în timp real pentru toți utilizatorii conectați și stocate sub formă de JSON.

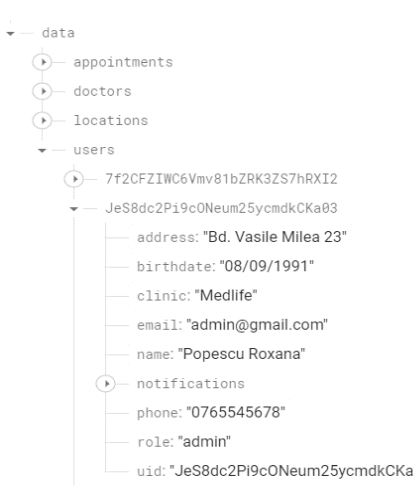

Figura 1.3 Stocarea datelor în Realtime Database

Un serviciu asemănător este Cloud Firestore ce pornește de la baza serviciului de Realtime Database, aducând optimizări semnificative în contextul scalabilității și al duratei de execuție a interogărilor în baza de date. Acesta este conceput pentru aplicațiile care necesită gestionarea unui volum mare de date, cu operații tranzacționale complexe, motiv pentru care am considerat potrivit pentru aplicația mea serviciul de Realtime Database, neavând un flux generos de date cu care lucrează, iar astfel am ales soluția robustă și cu o latență scăzută pentru sincronizarea stărilor între clienți în timp real [14].

Principiul după care Firebase Realtime Database funcționează este diferit de cel bazat pe *request-urile* de tip HTTP, în așa fel încât de fiecare dată când se realizează o modificare în

baza de date, datele se sincronizează în termen de câteva milisecunde pe toate device-urile conectate [15].

Realtime Database este o bază de date NoSQL care stochează datele într-o structură arborescentă de tip JSON, astfel informațiile fiind ușor de accesat în adâncime.

Securitatea bazei de date se realizează prin definirea unor reguli în consola Firebase ce stabilesc cine are drepturi de citire sau scriere de date sau cum acestea sunt structurate. Regulile *default* nu oferă acces nimănui la baza de date, acestea fiind utile când aplicația este inactivă. Totuși, acest scenariu fiind puțin probabil, configurația regulilor de securitate se poate face pe baza a patru tipuri (.read, .write, .validate, .indexOn) [16] cărora li se pot da valori booleene, dependente de autentificarea utilizatorului în aplicație, de tipul datelor care sunt introduse sau de anumiți indecși specificați.

I.3.3 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) este un serviciu oferit de Firebase prin care se pot crea diverse campanii prin care sunt trimise fără cost mesaje sau notificări utilizatorilor pentru a spori angajarea acestora în folosirea frecventă a aplicației și pentru a le stârni loialitatea.

Crearea unei campanii se realizează în consola Firebase și permite analiza audienței căreia i se adresează prin calcularea procentului utilizatorilor care se încadrează în cerințele stabilite, programarea campaniei și stabilirea unei date de sfârșit. De asemenea, este posibilă programarea notificărilor trimise și recurența lor, cât și previzualizarea acestora ca și cum ar fi primite de utilizatori, după cum se poate observa în figura ilustrată mai jos.

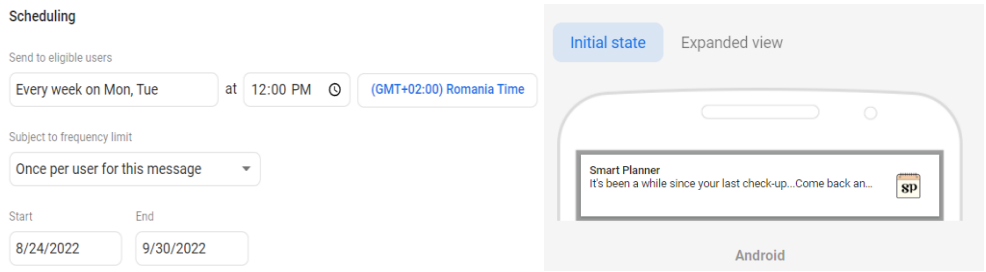

Figura 1.4 Programarea campaniei de notificări

I.3.4 Cloud Storage for Firebase

Cloud Storage for Firebase este un spațiu de stocare rentabil, ușor de utilizat și securizat pentru conținutul generat în aplicație de utilizatori, precum imagini, fișiere audio sau videoclipuri, fișiere care în mod normal au o dimensiune mare.

Avantajele serviciului sunt rapiditatea cu care fișierele sunt încărcate sau descărcate pentru o experiență cât mai facilă a utilizatorilor, precum și securizarea care se realizează prin Firebase Authentication. Se poate personaliza configurația sistemului pentru a stabili cine are acces la date despre fișierele stocate precum nume, tipul de conținut sau dimensiunea [17].

Am folosit acest serviciu pentru stocarea fotografiilor de profil ale utilizatorilor, după cum se poate observa în figura 1.5:

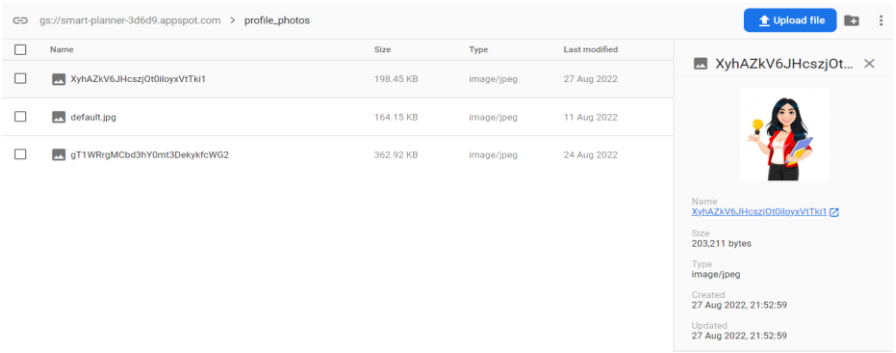

Figura 1.5 Stocarea imaginilor în Cloud Storage

I.4 Maps SDK for Android

Maps SDK este un kit de dezvoltare software ce permite integrarea și configurarea serviciilor de locație folosind date din Google Maps [18]. Astfel, cu acest kit pot fi incorporate hărți în cadrul fragmentelor sau activităților din aplicație.

Pentru integrarea acestui serviciu în proiect, am avut nevoie de o cheie numită *MAPS_API_KEY* pe care am obținut-o din consola Google Cloud, după crearea unui proiect corespondent aplicației mele. Această cheie are caracter confidențial și trebuie păstrată în siguranță. Am adăugat-o în fișierul *local.properties*, fișier exclus din instrumentele de control de versionare, din motive clare de securitate.

Pentru a furniza utilizatorilor aplicației Smart Planner cât mai multe informații utile, în pagina cu detaliile programării selectate am introdus un fragment ce conține harta în care este vizibilă locația curentă a utilizatorului bazată pe locația dispozitivului în cazul acceptării permisiunilor necesare. Harta include și locația clinicii la care este programată vizita medicală, fiind fixată pe hartă cu ajutorul unui *Marker*. La apăsarea pe *marker-ul* corespondent locației clinicii de pe hartă, este afișată o fereastră cu caracter informativ ce conține detalii despre locație precum adresa acesteia

Figura 1.6 Harta cu locația clinicii

sau distanța de la locația curentă a utilizatorului până la destinație, dar și opțiunea de a afișa traseul recomandat între cele două locații, după cum se poate remarca în figura 1.6. Această fereastră a fost creată folosind un *InfoWindow* căruia i-am atașat un adaptor pentru personalizarea ferestrei în funcție de cerințele aplicației.

I.5 Directions API

Directions API este un instrument care calculează traseul recomandat între două locații introduse și care returnează răspunsul în format JSON sau XML cuprinzând direcțiile traseului, în urma unui *request* HTTP.

Acesta cuprinde mai multe moduri de utilizare, oferind informații pe baza mijlocului de transport ales de utilizator dintre: mers pe jos, cu bicicleta, cu mașina sau în tranzit [19].

Am integrat acest serviciu în aplicație pentru ca utilizatorii să poată vedea în fragmentul cu harta traseul recomandat între locația lor curentă și clinca asignată programării curente, pe modul *driving*. Acest lucru se întâmplă doar la solicitarea lor, atunci când aceștia apasă pe butonul *Show route* din fereastra informativă despre clinică, traseul fiind desenat direct pe hartă.

I.6 Librării

I.6.1 Glide

Glide este o librărie dezvoltată de *bumptech* ce permite încărcarea și afișarea imaginilor, imaginilor-videoclip (*video stills*) sau fișierelor de tip GIF în dezvoltarea aplicațiilor Android, fiind recomandată de Google [20].

Principalul avataj adus de Glide este viteza de încărcare a conținutului datorită integrării sistemului de *lifecycle* pentru a prioritiza fișierele din fragmentele sau activitățile active, precum și reutilizarea resurselor de tip vector de bytes sau Bitmaps, minimizând colectarea *garbage* sau fragmentarea *heap-urilor*.

Încărcarea unei imagini se poate realiza într-o singură linie de cod cu ajutorului *url-ului* unde este stocată, în cazul proiectului actual Cloud Storage for Firebase, și al resursei din layout de tip *ImageView*.

Am integrat această librărie pentru încărcarea din Cloud a imaginii de profil în pagina dedicată detaliilor utilizatorului, respectiv a unei imagini *default* în cazul în care acesta nu a setat una preferențial.

I.6.2 OkHttp

OkHttp este o librărie dezvoltată de *Square* care oferă un client de tip HTTP cu ajutorul căruia aplicația Android poate comunica printr-o conexiune cu un server, pentru a putea trimite si primi *request-uri* [21].

Integrarea librăriei a făcut posibilă comunicarea între aplicație și Directions API, creând un client HTTP prin intermediul căruia au fost trimise *request-uri* de tip GET cu cele două destinații între care să fie calculat traseul, primind înapoi un răspuns de tip JSON cu coordonatele direcțiilor.

I.6.3 Android Lifecycle (Jetpack)

Pachetul *lifecycle* din *androidx.lifecycle* furnizează o multitudine de clase și interfețe predefinite pentru a creea componente ce își schimbă comportamentul în funcție de modificări ale ciclului de viață al unui fragment sau al unei activități.

Aceste componente se numesc *lifecycle-aware components* și ajută la simplificarea și îmbunătățirea organizării codului. Ciclurile de viață sunt atașate majorității componentelor unei aplicații Android, fiind gestionate de sistemul de operare [22]. Întreruperea funcționării normale a acestora poate cauza blocarea aplicației sau alte evenimente neprevăzute.

LiveData este o astfel de componentă care reacționează la modificări ale datelor din interiorul său, trimițând notificări unui observator din cadrul aceluiași ciclu de viață.

Componenta *ViewModel* crează, stochează și încarcă date pe care le comunică apoi altor componente din același ciclu de viață.

Acestea au fost esențiale în comunicarea aplicației cu baza de date întrucât au făcut posibile extragerea și menținerea datelor persistente, în timp real, fără ca firul natural de funcționare al aplicației să fie alterat.

Astfel, firul de prelucrare a datelor asteaptă până la declanșarea evenimentului de extragere a programărilor medicale din baza de date, pentru a evita un posibil *blocking-point* în aplicație cauzat de încercarea folosirii datelor indisponibile la momentul în cauză. Atunci, utilizatorului îi este afișat un UI informativ, corespunzător figurii 1.7.

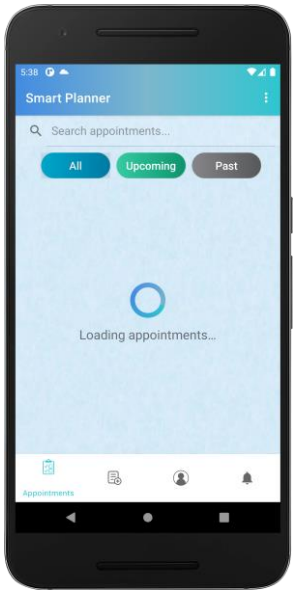

Figura 1.7 Interfața în timpul încărcării programărilor

I.6.4 Material Components for Android (MDC-Android)

Material Components for Android este o librărie care oferă componente și stiluri pentru construirea interfețelor în aplicațiile Android cu scopul de a creea un design modern care să atragă atenția utilizatorilor [23].

Din această librărie au fost integrate în aplicația Smart Planner diverse stiluri care intră în componența temelor și pe care le-am inclus și definit în fișierul *themes.xml*, precum *Theme.MaterialComponents.Light.NoActionBar*, temă pe care am folosit-o pentru ecranul de Splash Screen, pentru a elimina bara de acțiuni.

De asemenea, componente precum *Button*, *Card*, *Bottom Navigation*, *Text Input Layout* mi-au fost folositoare în construirea ecranelor din aplicație pentru a oferi utilizatorilor o interfață cât mai aspectuoasă și intuitivă, ca aceștia să folosească aplicația cu usurință.

II. Analiza aplicației

Înaintea dezvoltării oricărui produs software este necesară întocmirea unei analize amănunțite referitoare la diverse aspecte importante legate de scopul acestuia. Câteva astfel de aspecte sunt: stabilirea clară a unui grup țintă căruia i se adresează produsul, utilitatea acestuia și impactul pe care îl are asupra utilizatorilor, gradul de scalabilitate sau alte variabile ce formează o viziune clară complet necesară a prori.

Acestea sunt aspecte care conduc la o implementare cu etape bine definite și alese corect din punct de vedere cronologic și tehnologic. Cu alte cuvinte, un raport de analiză a aplicației este un pas esențial ce trebuie parcurs înaintea implementării efective, având o influență asupra succesului produsului și utilității sale.

O astfel de analiză va fi detaliată în acest capitol despre aplicația Smart Planner, cuprinzând de asemenea o analiză a pieței și o comparație cu alte produse software cu tematică similară, precum și punerea în discuție a punctelor forte și slabe ale soluției propuse.

II.1 Grupul țintă și nevoile pe care aplicația le satisface

Un studiu realizat de *European Observatory on Health Systems and Policies* din anul 2021 arată că în România, raportat la 100.000 de locuitori ai țării, există 306 cazuri de mortalitate evitabilă prin prevenție și 210 cazuri de cauze tratabile ale mortalității, România numărându-se printre țările cu cele mai ridicate rate [2].

Acest lucru îngrijorător vine la pachet cu știri periodice despre frecvența redusă a românilor la controlul medical de rutină, o mare parte mergând la doctor doar la nevoie sau după depășirea termenului recomandat.

Amânarea vizitei medicale poate avea repercursiuni puternice asupra pacienților, iar specialiștii recomandă prevenția si măsurile de precauție necesare pentru o viață sănătoasă și fericită.

Cu alte cuvinte, fiecare dintre noi ar trebui să aibă grijă de sănătatea noastră și a celor din jurul nostru mai presus de orice. Prin intermediul aplicației Smart Planner am dorit să încurajez cât mai tare acest lucru, oferindu-le oamenilor o modalitate mai eficientă de a-și programa o vizită medicală de orice fel, printr-un proces rapid și la îndemâna oricui, astfel ca aceștia să nu mai perceapă această sarcină drept costisitoare ca timp și energie.

Prin urmare, aplicația se adresează unui public larg format din adulți și adolescenți care dispun de un dispozitiv smart cu sistem de operare Android. Fie că vor să economisească timpul

petrecut în telefon pentru programarea controlului medical sau nu se încadrează în programul de lucru al clinicii, fie că în agenda lor intervin periodic evenimente neprevăzute ce le pot pune programarea în pericol, sau doar vor să țină o evidență clară a vizitelor medicale, pacienților le vine în ajutor aplicația Smart Planner.

Pe scurt, principalele beneficii aduse unui pacient de utilizarea aplicației sunt:

- economisirea timpului petrecut la telefon pentru programarea vizitei medicale
- reducerea riscului de a rata o programare din cauza unor evenimente neprevăzute
- evidența și gestionarea programărilor de orice tip într-un singur loc
- remindere când se apropie o programare
- încurajarea creșterii frecvenței vizitelor la medic

II.2 Prioritizarea funcționalităților

În urma clarificării scopului și utilității aplicației, pot fi stabilite funcționalitățile de bază ale aplicației de care aceasta are nevoie pentru a satisface nevoile propuse. În mod evident, impactul adus de acestea diferă de la o funcționalitate la alta, fiind nevoie de categorizarea acestora pentru a stabili o ordine de priorități.

Eu am ales două criterii de analiză a funcționalităților și anume: importanța și gradul de dificultate. Astfel, acestea vor putea fi împărțite în 4 categorii: ușoare-valoroase, grele-valoroase, ușoare-nevaloroase, grele-nevaloroase, categorii pe baza cărora poate fi stabilită ordinea în care se vor implementa funcționalitățile, cea enumerată anterior, în masura în care acest lucru este posibil ținând cont și de detaliile tehnice și dependența dintre acestea.

Graficul din figura 2.1 ilustrează scorurile atribuite fiecărei cerințe.

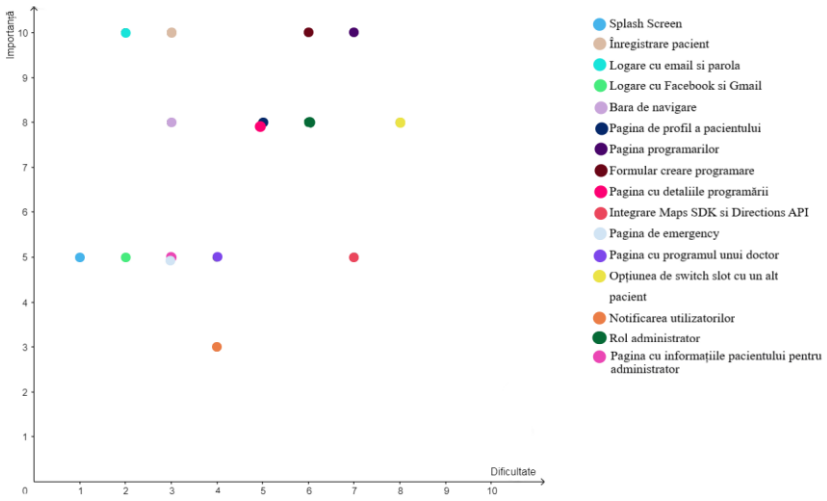

Figura 2.1 Graficul priorității funcționalităților

În urma raportului, am obținut următoarea ordine de implementare a funcționalităților:

- Splash Screen
- Înregistrare pacient
- Logare cu email și parola
- Bara de navigare
- Logare cu Facebook și Gmail
- Pagina programărilor
- Pagina cu detaliile programării
- Pagina de profil a pacientului
- Formular creare programare
- Rol administrator
- Pagina de emergency
- Pagina cu programul unui doctor
- Opțiunea de switch slot cu un alt pacient
- Notificarea utilizatorilor
- Pagina cu informațiile pacientului pentru administratori
- Integrare Maps SDK și Directions API

De menționat este faptul că pe parcusul dezvoltării această ordine a mai suferit modificări din diverse motive tehnice sau de logistică, însă esența s-a păstrat și a reprezentat un bun punct de plecare și de reper pe parcursul implementării funcționalităților.

II.3 Puncte forte și puncte slabe

Stabilirea punctelor forte și punctelor slabe ale aplicației este un pas important în decursul dezvoltării unei aplicații deoarece acesta ajută la formarea unei viziuni realiste asupra produsului pentru a evidenția plusurile acestuia și pentru a avea în vedere elementele negative care pot fi îmbunătățite, evitând astfel unele situații neașteptate sau așteptări nerealiste.

După o analiză amănunțită a produsului, am ajuns la identificarea umătoarelor puncte forte:

- Utilizatorii își pot creea cu ușurință un cont, un profil de informații sau o programare. Aplicația este ușor de folosit și are o interfață prietenoasă.
- Este un produs care înglobează mai multe servicii: crearea unei programări, evidența tuturor programărilor indiferent de categorie sau clinica asignată, informații despre

locația clinicilor și despre programul doctorilor, notificări de reamintire, opțiunea de a suna personalul medical / pacientul direct din aplicație, adăugarea automată a programării direct în calendarul personal, opțiunea de schimbare slot cu alt pacient. Astfel, aducându-le atâtea beneficii utilizatorilor, aplicația are un adevărat potențial.

- Este un produs care economisește timp atât pacienților care îl folosesc, cât și personalului medical care se ocupă de gestionarea programărilor.
- Are potențial de scalabilitate, fiind un produs ce poate fi utilizat în diverse regiuni, cu o capacitate mare de utilizatori, folosit de un număr mare de clinici.
- Aduce o funcționalitate nouă față de alte aplicații cu temă similară, prin opțiunea de schimbare a programării cu alt pacient, fapt ce ar putea evita așteptarea mai mare pentru o reprogramare.

Am găsit, de asemenea, următoarele puncte slabe:

- Există alte aplicații asemănătoare, specializate pe diferite rețele mari de sănătate. Astfel, aplicația intră pe o piață cu ofertă semnificativă, dar totuși se distinge tocmai prin caracterul general care permite utilizarea ei ca agendă medicală.
- Chiar dacă aplicația are un potențial de scalabilitate, aceasta ar putea avea nevoie de ajustări sau funcționalități suplimentare, spre exemplu particulare clinicilor care pot avea politici de funcționare diferite.

II.4 Aplicații similare

II.4.1 Evexia

Evexia [24] este o aplicație pentru managementul programărilor și al cabinetelor medicale care aduce trei beneficii comune cu aplicația Smart Planner: reduce numărul programărilor neonorate prin recurența *reminder-elor* către pacienți, ușurează procesul administrativ de creare și gestionare de programări și oferă un design modern și intuitiv.

Aplicația se adresează cabinetelor printr-un abonament lunar plătit, cu scopul de a crește veniturile cabinetului prin găsirea de noi pacienți, păstrarea recurenței celor actuali și îmbunătățirea imaginii clinicii. Aceasta oferă în plus accesul la rapoarte cu diferite analize și statistici. Cât pentru pacienți, aceasta este gratis.

Pe Google Play este disponibilă sub două forme: Evexia Doctor și Evexia Client, prima având puțin peste 100 de descărcări și ultima actualizarea publicată pe 5 aprilie 2021, iar a doua peste 10 descărcări și ultima actualizare pe 19 ianuarie 2020. Totuși, în descrierea celor două

aplicații este menționat faptul că sunt destinate exclusiv programărilor stomatologice. Aceste informații sunt furnizate din Google Play.

II.4.2 DocBook – Programări la doctor

DocBook [25] este o aplicație pentru programări medicale direct de pe telefon, ce oferă accesul la sute de doctori din clinici medicale private și multiple specializări. Se bazează pe abonament personalizat pentru fiecare pacient și oferă informații utile despre clinici și medici, având vizibile diverse recenzii pentru serviciile oferite.

Aceasta s-a bucurat de un succes considerabil, având peste 10.000 de descărcări pe Google Play, însă ultima actualizare a fost publicată în anul 2018, iar ultimele recenzii au fost postate cu ani în urmă, ceea ce denotă faptul că aplicația este puțin folosită în momentul actual.

II.4.3 Regina Maria

Regina Maria [26] este aplicația cu cel mai mare succes în domeniul medical, având peste 100.000 de descărcări pe Google Play și recenzii foarte bune, beneficiind de asemenea de actualizări frecvente.

Aceasta își propune să digitalizeze dosarul medical prin gestionarea digitală a programărilor medicale. Oferă pacienților posibilitatea de a-și crea o programare în aplicație sau de a trimite o cerere către Call Center prin intermediul acesteia pentru a lua legătura cu un reprezentant. Pacienții au acces la istoricul tuturor investigațiilor făcute în cadrul clinicii, iar datele lor sunt protejate.

Aceasta dispune de asemenea de Clinica Virtuală prin intermediul căreia se pot efectua vizite medicale online pentru a evita contactul direct în situații de risc precum cele cauzate de simptome ale virusului Covid 19.

Totuși, aplicația este destinată exclusiv pacienților care aleg să beneficieze de serviciile oferite de rețeaua privată de sănătate Regina Maria.

Concluzii

În urma analizei pieței, nu am găsit prea multe aplicații active cu tematică asemănătoare cu Smart Planner, cu excepția aplicației Regina Maria ce se bucură de un succes remarcabil, dar care se adresează unui public restricționat, mai precis pacienților abonați la serviciile acestei rețele de sănătate.

Totuși, în domeniul aplicațiilor web gama este mult mai bogată, însă acestea nu fac subiectul discuției deoarece aplicația Smart Planner promovează mai întâi de toate

accesibilitatea către utilizatori oferită de folosirea frecventă a telefoanelor mobile, numărul oamenilor sau frecvența cu care aceștia folosesc un laptop sau un calculator fiind semnificativ mai scăzute.

II. 5 Ce aduce nou?

Aplicația Smart Planner are avantajul de a aduce mai multe servicii disponibile în același loc. Funcționalități precum crearea rapidă a unei vizite medicale, gestionarea tuturor programărilor curente, recurența notificărilor de tip *reminder* precum și furnizarea de informații ajutătoare oferă utilizatorilor un bun instrument prin care pot realiza mai multe sarcini, astfel economisind timp și energie. Afișarea locației clinicii pe hartă și traseul recomandat până la destinație sunt, de asemenea, funcționalități utile, dar nu foarte des întâlnite în alte aplicații.

Pe de altă parte, din pespectiva personalului clinicii, pacienții au acces la mai multe informații care le pot fi de folos, fapt ce reduce de asemenea timpul petrecut de aceștia pentru obținerea lor.

Un avantaj în plus adus utilizatorilor Smart Planner este diversitatea serviciilor disponibile în aplicație, orice clinică având posibilitatea să se înroleze pentru un cont de administrator.

Totuși, o funcționalitate pe care nu am întâlnit-o în alte aplicații similare este cea care vizează gestiunea situațiilor neprevăzute ale pacienților în care aceștia ar fi nevoiți să anuleze o programare cu puțin timp înainte de data stabilită. Astfel, prin opțiunea de cere de schimbare a intervalului programării cu un alt pacient, poate fi redus numărul programărilor neonorate sau anulte cu scurt timp înainte, oferindu-le și pacienților o șansă în plus de a nu rata vizita medicală.

III. Arhitectura aplicației

În acest capitol va fi descrisă structura aplicației și a bazei de date, având o viziune clară asupra desfășurării implementării și a necesităților aplicației în urma redactării raportului de analiză. Astfel, vor fi motivate alegerile făcute asupra structurii proiectului și va fi de asemenea descris modul de lucru.

III.1 Structura aplicației

Pentru ca proiectul să aibă o structură cât mai stabilă, bine definită și ușor de înțeles, alegerea unui model arhitectural care să stea la baza acestuia este un pas esențial.

Pentru aplicația Smart Planner am folosit șablonul arhitectural **MVVM** (*Model, View, ViewModel*) ce constă în separarea componentelor de prezentare a datelor ce se realizează prin *View-uri*, de logica aplicației.

Cele trei componente care formează șablonul de proiectare sunt:

- **Model** – componenta responsabilă pentru datele aplicației, nu are acces direct către *View*, lucrează împreună cu *ViewModel* pentru manipularea datelor.
- **View** – componenta care transmite informațiile referitoare la acțiunile utilizatorului către *ViewModel*, poate fi considerată ca un observator și nu conține aspecte care țin de logica aplicației.
- **ModelView** – este considerată puntea de legătură între *Model* și *View* și este responsabilă pentru trimiterea fluxurilor de date de la *Model* către *View* și pentru actualizări în cazul modificării acestora.

Figura de mai jos ilustrează procesul prin care cele trei componente comunică între ele:

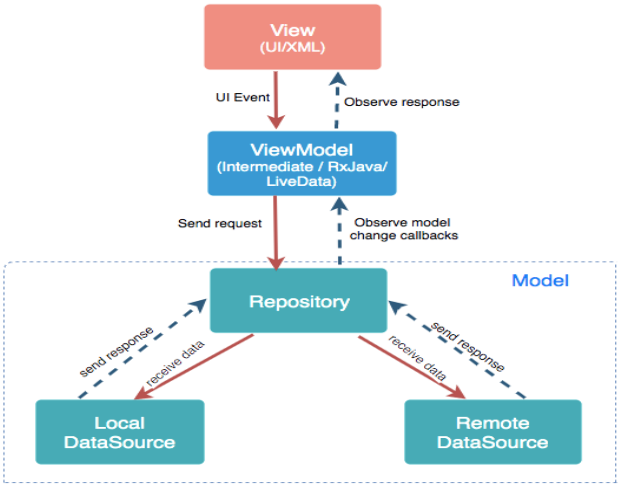

Figura 3.1 Arhitectura MVVM [27]

Pentru implementarea șablonului am utilizat componenta de *lifecycle* pe care am menționat-o în capitolul cu tehnologii, **LiveData**, care se comportă ca un observator în jurul datelor din model care sunt în concordanță cu conceptul de *lifecycle*. Ca proces de funcționare, în momentul în care datele suferă modificări, componenta *ViewModel* acționează și le transmite *View-ului* pentru a-și actualiza conținutul. Această abordare mi-a fost utilă în extragerea datelor din baza de date și pentru păstrarea consistenței acestora la modificările rezultate din acțiunile utilizatorilor.

Un alt aspect legat de structura proiectului este modul în care am implementat ecranele aplicației, folosind pentru ecranele principale (cele din bara de navigație) o singură activitate drept container în care vor fi interschimbate fragmente corespunzătoare fiecărui buton din bara de navigație.

Am ales această abordare pentru a păstra o perfomanță cât mai ridicată și pentru a eficientiza procesul de extragere a datelor din baza de date, ce reprezintă o sarcină costisitoare, atât ca timp, cât și ca resurse. Astfel, la instanțierea activității container numită *MainActivity*, am încărcat toate datele care îmi sunt necesare pentru navigarea între cele patru fragmente principale (*AppointmentsFragment, CreateAppointmentFragment, ProfileFragment, NotificationsFragment*). Comparativ cu încărcarea datelor în fiecare fragment, acestea ar fi fost extrase de fiecare dată când utilizatorul navighează pe alt ecran din cauza reinstanțierii fragmentelor la fiecare navigație.

Totuși, proiectul conține și alte activități (*AppointmentDetailsActivity, EmergencyPageActivity, PatientProfileActivity, ScheduleActivity*) care sunt accesibile doar prin anumite acțiuni ale utilizatorului precum apăsarea unui buton sau element din meniul din bara de acțiuni. Am considerat convenabilă această abordare, dat fiind transferul mare de date și dinamicitatea acestora de care am avut nevoie în ecranele respective, precum și nevoia de navigare facilă în ecranele anterioare folosind butonul de *back*, aceste activități neavând inclusă bara de navigație. În figura alăturată este ilustrată navigarea între ecranele aplicației:

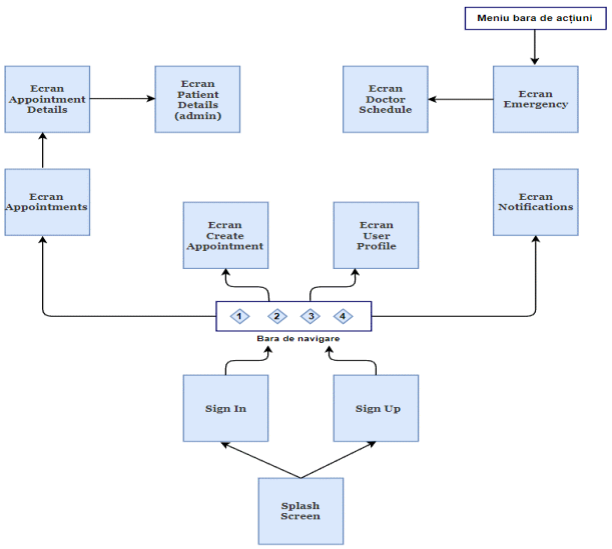

Figura 3.2 Ecranele aplicației

III.2 Structura bazei de date

Aplicația folosește o bază de date de tip NoSQL, furnizată de serviciul Realtime Database din Firebase. La baza alegerii a stat rapiditatea extragerii și manipulării datelor cu care vine la pachet, precum și integrarea facilă a acesteia în cadrul proiectului. Folosind alte trei servicii furnizate de Firebase, a fost la îndemână să folosesc această plaftormă și pentru stocarea datelor din aplicație. În continuare voi prezenta structura tabelelor din baza de date.

Tabelul **appointments**:

- *id*: format prin concatenarea uid-ului pacientului asignat, a datei și a orei de început a programării, pentru a asigura unicitatea intrărilor.
- *name*: numele programării
- *description*: scurtă descriere a programării
- *date*: data programării
- *doctor*: numele doctorului asignat
- *start*: ora de început a programării
- *end*: ora de sfârșit a programării (este setată default ca fiind start + 1h, însă ajută la calcularea mai rapidă a altor date precum intervalele disponibile ale unui doctor într-o anumită zi)
- *location*: numele clinicii asignate programării
- *patient*: uid-ul pacientului
- *type*: tipul programării (Radiologie, Cardiologie, etc.)

Figura 3.3 Tabelul appointments

Tabelul **doctors**:

- *name:* numele complet al doctorului
- *contact*: numărul de telefon
- *schedule*: are o structură arborescentă și reprezintă programul medicului împărțit pe zile, iar intrările sunt de tip cheie:valoare, data fiind cheia și valoarea un string format din concatenarea intervalelor în care acesta are programări, separate prin ";"; dacă o anumită dată nu are intrare în tabel, înseamnă că doctorul nu are nicio programare în ziua respectivă.

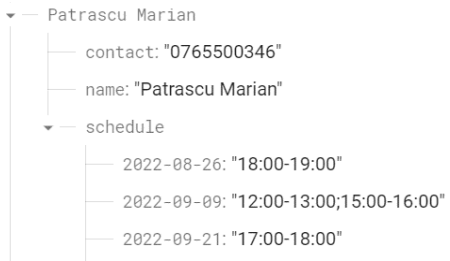

Figura 3.4 Tabelul doctors

Tabelul **locations**:

- *name*: numele clinicii
- *contact*: numărul de telefon
- *address*: adresa clinicii

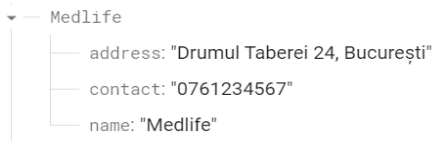

Figura 3.5 Tabelul locations

Intrările în acest tabel se fac prin numele clinicii, valoare care nu poate fi identică pentru două clinici diferite. Astfel, fiecare clinică este sub forma unui arbore cu coloanele menționate anterior.

Tabelul **users**:

- *uid*: user uid-ul unic generat automat de Firebase la înregistrarea în aplicație
- *name*: numele complet al utilizatorului
- *email*: adresa de email a utilizatorului
- *birthdate*: data nașterii
- *phone*: numărul de telefon mobil
- *address*: adresa utilizatorului
- *role*: rolul utilizatorului în aplicație (poate fi "admin" sau "user")

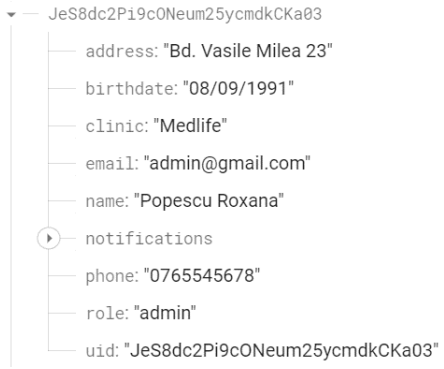

Figura 3.6 Tabelul users

- *clinic*: numele clinicii la care lucrează administratorul (rămâne gol, mai precis un string vid, în cazul pacienților)
- *notifications*: un arbore în care sunt stocate notificările active ce aparțin utilizatorului curent

Tabelul **requests**:

- *id*: id-ul cererii este generat prin concatenarea uid-urilor pacienților între care se trimite, precum și intervalul programării ce trebuie schimbată

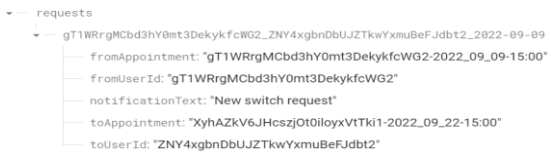

Figura 3.7 Tabelul requests

- *notificationText*: textul creat dinamic pentru trimiterea personalizată a notificării
- *fromUserUid*: uid-ul pacientului care trimite cererea
- *fromAppointment*: id-ul programării pe care pacientul vrea să o schimbe
- *toUserId*: uid-ul pacientului care primește cererea

- *toAppointment*: id-ul programării cu care se cere schimbul

Securizarea bazei de date a fost realizată cu ajutorul definirii unor reguli numite Security Rules furnizate de Firebase prin care accesul de scriere și citire este autorizat doar utilizatorilor autentificați și am stabilit dimensiuni maxime pentru anumite coloane din tabele, după cum se poate observa în figura alăturată.

Figura 3.8 Regulile de securitate ale bazei de date

Pentru a mă asigura că datele nu vor fi suprascrise sau pierdute din cauza unor posibile inserții greșite în timpul dezvoltării aplicației, aceste acțiuni fiind ireversibile, am descărcat periodic versiunea curentă a bazei de date și am stocat-o local. Am realizat acest lucru prin opțiunea *Export JSON* din consola Firebase.

IV. Funcționalitățile aplicației

În capitolul curent vor fi descrise funcționalitățile implementate în aplicație și va fi de asemenea detaliat procesul dezvoltării acestora din punct de vedere tehnic și conceptual, fiind atașate diverse imagini cu ecranele în care au fost aplicate.

IV.1 Splash Screen

Splash Screen este primul ecran pe care utilizatorul îl vede atunci când deschide aplicația. Rolul acestuia este de a ilustra o interfață prietenoasă în timp ce configurația și resursele de care aplicația are nevoie pentru funcționare sunt încărcate. La terminarea acestui proces, aplicația își poate începe cursul normal și celalalte funcționalități ale ei pot fi folosite.

Ecranul splash screen are o activitate corespondentă care este setată ca *launcher* în fișierul *AndroidManifest.xml*, și care, după ce toate resursele necesare au fost încărcate, creează un *Intent* către ecranul principal în cadrul căreia utilizatorul își poate începe activitatea.

Construirea acestui ecran a fost realizată prin definirea unei teme personalizate în fișierul de resurse în care am dezactivat bara de acțiuni, am setat un fundal simplu și bara de status ca fiind transparentă. Interfața acestuia este una simplistă, ilustrând logo-ul aplicației, după cum se poate vedea în figura de mai jos.

Figura 4.1 Splash Screen

IV.2 Înregistrarea și logarea în aplicație

După ce ecranul *splash screen* dispare, utilizatorul este redirecționat către ecranul de înregistrare. Această etapă este esențială pentru utilizarea ulterioară a aplicației.

Pentru început, utilizatorul va trebui să introducă doar numele complet, un email valid pe care îl deține și o parolă pe care o va introduce de două ori, pentru a confirma corectitudinea acesteia. Câmpurile parolei și al confirmării parolei au default conținutul ascuns din motive de securitate, însă acestea pot fi afișate apăsând icon-ul din dreapta câmpului.

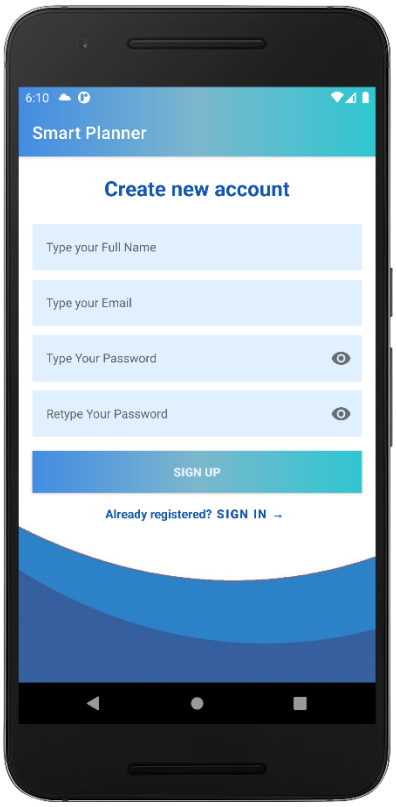

Figura 4.2 Înregistrarea în aplicație

În cazul în care utilizatorul nu introduce un email valid, nu completează un câmp deloc sau cele două câmpuri pentru parolă nu corespund, este afișat un scurt *pop-up* informativ corespunzător motivului pentru care nu a putut fi realizată înregistrarea contului.

De asemenea, dacă un utilizator încearcă să creeze un cont cu un email care există deja în baza de date, deci are un cont existent asociat, acesta este înștiințat și sfătuit să meargă către pagina de logare. Acest lucru previne suprascrierea datelor deja existente în contul utilizatorului cu câmpuri vide, adică resetarea contului și pierderea informațiilor.

După ce datele introduse trec de aceste validări, contul utilizatorului este creat, iar acesta este înștiințat și redirecționat către pagina cu programări pentru a-și continua activitatea.

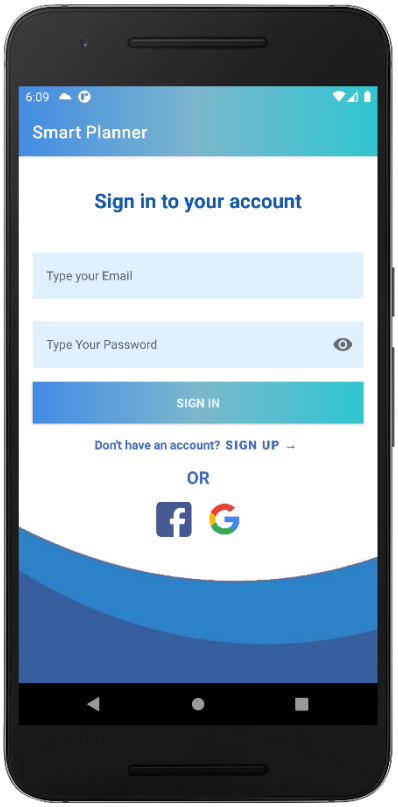

Figura 4.3 Logarea în aplicație

În ceea ce privește logarea în aplicație, aceasta constă într-un formular asemănător cu cel de înregistrare și se realizează cu ajutorul adresei de email și al parolei folosite la înregistrare, date cunoscute de Firebase Authentication. În cazul lipsei completării sau completării

greșite a câmpurilor, autentificarea eșuează și utilizatorul este înștiințat printr-un *pop-up*.

Utilizatorul are de asemenea opțiunea de a se loga cu un cont deja existent pe una din platformele Facebook sau Gmail. Pentru a selecta oricare din aceste opțiuni, este necesară simpla apăsare a logo-ului platformei corespunzătoare afișat în pagina de autentificare. După prima logare cu un cont de acest tip, credențialele vor fi de asemenea stocate în Firebase împreună cu tipul platformei terțe, iar contul va fi recunoscut de device și utilizat default pentru autentificările viitoare.

IV.3 Profilul utilizatorului

În bara de navigare se află butonul care redirecționează utilizatorul către profilul său. În această pagină sunt afișate date relevante în contextul aplicației precum numele complet, numărul de telefon, adresa și data nașterii.

În mod evident, imediat dupa crearea contului, exceptând câmpul corespunzător emailului și al numelui pacientului, toate celelalte sunt goale, nefiind incluse în formularul de înregistrare. Astfel, utilizatorul va trebui să completeze celelalte informații în pagina de profil, pentru a putea fi accesibile personalului de la clinicile la care va programa vizite medicale. Din acest motiv, după crearea unui cont de orice tip din cele trei posibile, utilizatorul primește automat o notificare prin care i se comunică necesitatea completării profilului.

În pagina de profil, utilizatori își pot seta o fotografie de profil, care va fi ulterior stocată în Firebase și încărcată de acolo în sesiunile următoare. În lipsa alegerii unei fotografii, va fi încărcată o poză *default* din Cloud, din motive care țin de estetica paginii.

Pagina de profil are ca scop, atât afișarea informațiilor curente despre utilizator, cât și posibilitatea de actualizare a acestora în cazul unor modificări precum folosirea unui nou număr de telefon sau schimbarea adresei.

Actualizarea datelor se realizează la orice modificare făcută în câmpurile text urmată de apăsarea butonului *Save*.

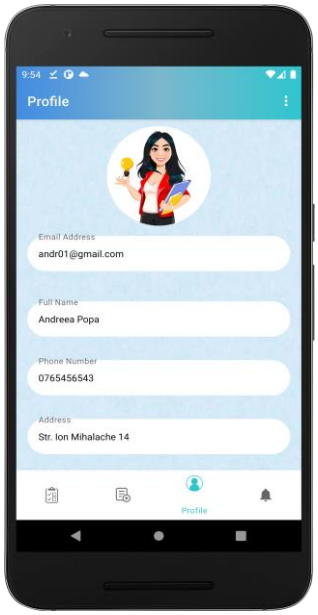

Figura 4.4 Profilul utilizatorului 1

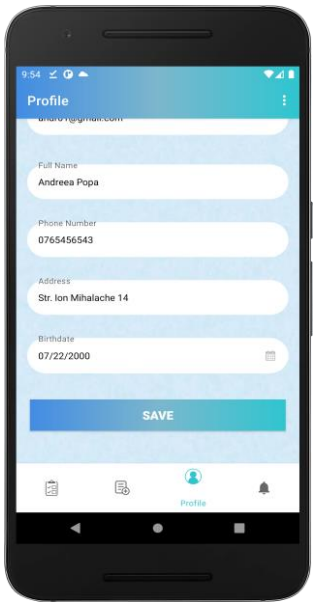

Figura 4.5 Profilul utilizatorului 2

IV.4 Vizualizarea tuturor programărilor

Ecranul Appointments este cel pe care utilizatorul îl vede pentru prima dată după logarea în aplicație. Din perspectiva pacientului, aici sunt afișate toate programărilor pe care acesta le-a creat în aplicație. Din perspectiva administratorului, în această pagină sunt afișate toate programările clinicii la care acesta lucrează. În ambele cazuri programările sunt ordonate după dată și au culori diferite în funcție de data acestora, astfel: cele din ziua curentă sunt verzi, cele viitoare sunt albastre, iar cele care din istoric sunt gri.

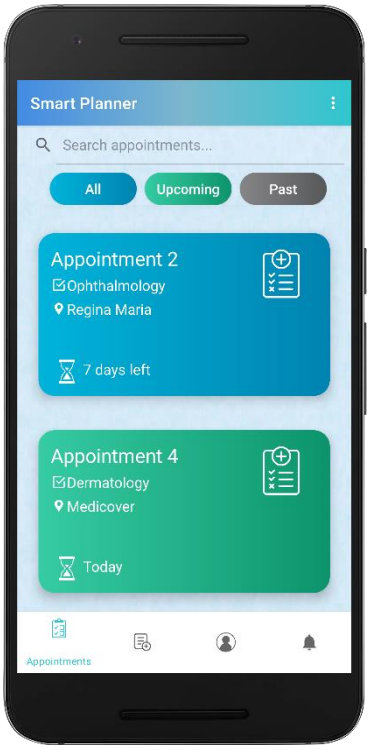

Figura 4.6 Pagina programărilor pacientului

Acest ecran conține trei filtre utile pentru a vizualiza mai ușor programările de interes, astfel este setat *default* filtrul *All* care afișează toate programările, dar mai pot fi folosite și filtrele *Upcoming* pentru cele viitoare, sau *Past* pentru istoricul programărilor.

Programările din pagină sunt afișate cu ajutorul unui *recyclerview* prin care acestea se încarcă și descarcă în mod dinamic pentru optimizarea resurselor folosite, astfel nu se vor încărca toate programările în același timp, ci pe măsură ce utilizatorul derulează lista. Din perspectiva pacienților, nu putem spune că ar fi o îmbunătățire semnificativă. Însă, ca administrator într-o clinică în care numărul programărilor poate fi de ordinul sutelor, iar încărcarea simultană a acestora ar putea dura chiar câteva minute, această optimizare este absolut necesară pentru o bună funcționare a aplicației.

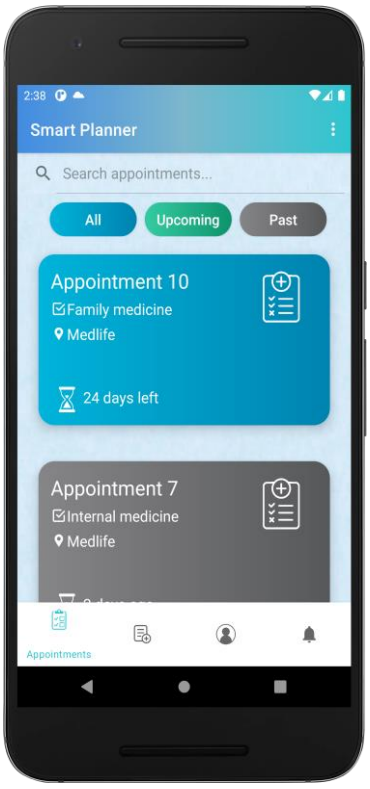

Figura 4.7 Pagina programărilor din clincă

În pagină există de asemenea o bară de căutare pentru economisirea timpului de căutare prin *scroll*, astfel în cazul în care utilizatorul dorește să vadă o programare anume, el poate căuta numele acesteia sau o secvență din el.

În pagină sunt afișate programările sub forma unui card cu design simplist pentru a nu încărca ecranul cu prea

multe informații. Astfel, pentru fiecare programare sunt afișate doar numele, tipul, clinica și câte zile mai sunt sau au trecut de la data programării.

Pentru a vedea toate informațiile necesare, atât pacienții, cât și administratorii trebuie doar să apese pe programarea dorită, acțiune care îi va redirecționa către pagina cu detaliile programării selectate.

IV.5 Crearea unei programări noi

IV.5.1 Completarea formularului

Create appointment este ecranul în care pacienții pot crea programări noi.

Acest lucru se realizează prin completarea unui formular ce conține câmpuri de tip *input text* pentru următoarele informații: nume pentru programare și o scurtă descriere. Pentru setarea datei pentru programare am implementat un input de tip *DatePicker* pentru a face introducerea acesteia mai rapidă și mai plăcută pentru utilizator. Pacientul poate alegere doar ziua curentă sau una viitoare, cele din trecut fiind dezactivate. De asemenea, nu pot fi alese decât date din zile lucrătoare, fără sâmbătă sau duminică. Urmează apoi tipul programării care reprezintă aria medicală în care aceasta se încadrează, și apoi doctorul, iar alegerea se face prin afișarea unei liste *dropdown*, numită *Spinner*, ce are atașat un adaptor pentru încărcarea dinamică a opțiunilor, astfel informațiile fiind încărcate din baza de date și apoi în listă.

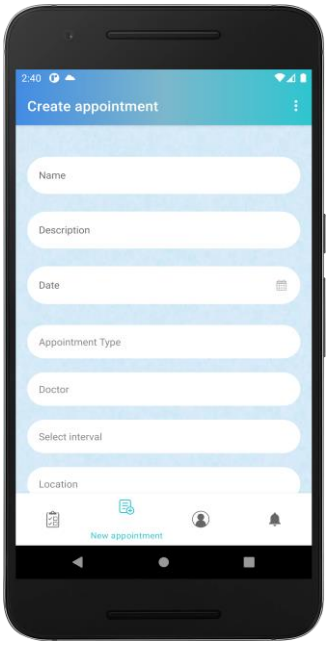

Figura 4.8 Formularul pentru crearea programării 1

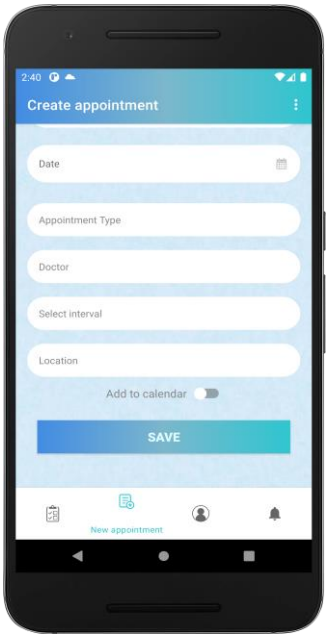

Figura 4.9 Formularul pentru crearea programării 2

Un input asemănător este intervalul programării care este tot de tip Spinner, însă ce trebuie menționat aici este că opțiunile acestuia sunt calculate în back-end, pe baza datei alese și a doctorului asignat. Astfel, se elimină complet cazul în care s-ar putea programa două vizite simultan la același medic.

În cazul în care, la selectarea listei de intervale nu au fost introduse data și doctorul, se afișează pe ecran un mesaj informativ, iar lista rămâne nepopulată. De asemenea, dacă în

ziua respectivă doctorul ales are deja tot programul ocupat, este adăugat în listă un mesaj corespunzător, neselectabil.

La apăsarea butonului *Save*, programarea este salvată în baza de date, iar utilizatorul este redirecționat apoi automat către pagina de programări unde poate vizualiza noua programare creată.

IV.5.2 Adăugarea programării în calendarul personal

După introducerea datelor necesare pentru crearea programării, pacientul are opțiunea de a adăuga în calendarul personal un eveniment corespunzător, prin glisarea *switch-ului* „Add to calendar", cu data și durata corespunzătoare.

IV.6 Vizualizarea detaliilor unei programări

IV.6.1 CardView cu detaliile programării

Detaliile unei programări se pot vizualiza în ecranul dedicat la care utilizatorul ajunge prin apăsarea programării dorite din lista de pe pagina Appointments.

Din perspectiva administratorului, în pagina cu detaliile programării, în interiorul unui card sunt afișate următoarele informații: numele și descrierea programării, numele complet al pacientului, precum și data, ora, tipul și doctorul programării. În plus, acesta poate intra pe pagina cu detaliile pacientului printr-un simplu click pe numele acestuia, din acest motiv fiind subliniat.

Din perspectiva utilizatorului, într-un card asemănător sunt afișate următoarele informații: numele și descrierea programării, data, ora, tipul, doctorul și clinica.

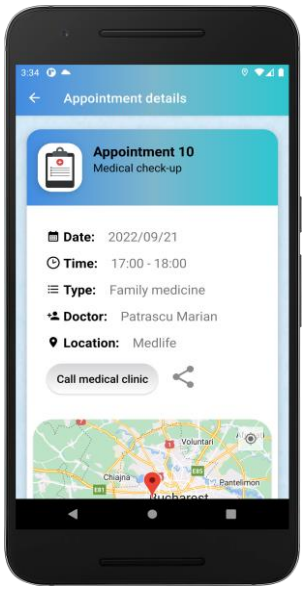

Figura 4.10 Detaliile programării pentru pacient

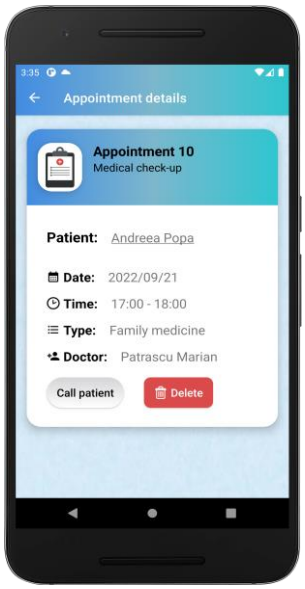

Figura 4.11 Detaliile programării pentru administrator

IV.6.2 Apelare din aplicație

Prin apăsarea butonului *Call patient*, administratorul poate suna pacientul direct din aplicația Smart Planner, fiindu-i completat automat numărul de telefon al acestuia.

Pacientul are și el opțiunea de a face un apel direct către clinica unde este programată vizita medicală.

Acest lucru s-a realizat prin crearea unui *Intent* de tip *ACTION_DIAL*, după acceptarea permisiunii, cu numărul de telefon extras din baza de date și completat automat pentru utilizator.

IV.6.3 Share la detaliile programării

Pacientul poate să facă *share* la detaliile programării prin apăsarea butonului cu icon sugestiv, fie copiind textul formatat, fie apăsând pe una din aplicațiile disponibile precum aplicația Gmail. Această funcționalitate a fost implementată prin crearea unui *Intent* de tip *ACTION_SEND* cu un text format dinamic drept parametru.

IV.6.4 Ștergerea unei programări

Tot în acest ecran, administratorul are posibilitatea de a șterge programarea prin apăsarea butonului *Delete*, în urma căruia este afișat un *pop-up* de confirmare, pentru a evita acțiunile neintenționate.

Am decis ca acest buton să fie disponibil doar pentru administratorul clinicii pentru a evita abuzul acestei acțiuni din perspectiva pacienților. Astfel, în situația în care un pacient dorește să anuleze o programare, acesta poate iniția un apel către clinică de pe butonul din pagina cu detaliile programării, ia legătura cu administratorul, motivând de asemenea alegerea făcută, ca ulterior cel din urmă să șteargă programarea din baza de date.

IV.6.5 Harta cu locația clinicii

Pentru pacienți am implementat funcționalitatea de integrare a unui fragment **Maps** în pagina cu detaliile programării. Această hartă este vizibilă doar pentru utilizatorii care acceptă permisiunea ca aplicația să le utilizeze locația în timpul rulării.

Pe hartă sunt afișate locația curentă a pacientului și un *pin* al locației clinicii. La apăsarea pe *pin*, se afișează o fereastră informativă (*InfoWindow*) specifică hărților, însă cu design personalizat. Pentru interfața acesteia am creat un nou fișier *layout* în care am construit fereastra, adăugând informații utile, o imagine și un buton.

În cadrul acestei ferestre este afișată distanța dintre locația curentă a pacientului și clinica unde a programat vizita medicală. Distanța este calculată folosind funcția *AndroidLocation.distanceBetween()* care primește ca parametri cele două perechi de coordonate, latitudine și longitudine, pentru cele două locații între care se dorește calcularea distanței. Pentru

Figura 4.12 Fereastra informativă din hartă

a obține coordonatele geografice ale unei locații căreia îi cunosc doar adresa, am folosit funcția *Geocoder.getFromLocationName()*.

Mai mult decât atât, în fereastra cu informațiile clinicii, există un buton numit *Show Route* care, apăsat, afișează traseul recomandat între locația pacientului și clinică. Traseul este desenat direct pe hartă și este calculat cu ajutorul API-ului **Directions API**, unde sunt trimise request-uri HTTP către un link creat dinamic dependent de cele două locații.

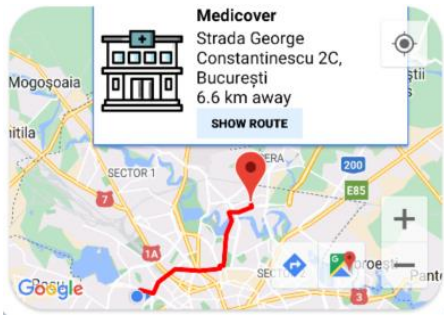

Figura 4.13 Afișarea traseului recomandat

IV.7 Vizualizarea detaliilor pacientului

Ecranul cu detaliile pacientului este accesibil doar pentru administratori, această funcționalitate nefiind necesară pentru pacienți deoarece aceștia își pot vedea și modifica datele personale în pagina de profil.

Dacă un administrator, atunci când navighează în pagina cu detaliile programării, dorește să vadă mai multe informații despre pacient, el poate naviga către un ecran dedicat printr-un simplu click pe numele acestuia.

Figura 4.14 Detaliile pacientului

În pagina dedicată pacientului sunt afișate următoatele informații: numele complet, adresa de email, numărul de telefon, data nașterii și adresa pacientului.

În cazul în care pacientul nu a introdus toate aceste informații după crearea contului, administratorul îi poate trimite, prin apăsarea butonului *Request patient info*, o notificare ca reminder în care acesta să fie înștiințat că va trebui să furnizeze informațiile necesare pentru a nu își pierde programările.

IV.8 Schimbarea programării cu alt pacient

Această funcționalitate va fi împărțită pe subcapitole pentru a putea explica pe îndelete întregul proces pe care utilizatorii trebuie să îl urmeze pentru a schimba intervalul unei programări cu un alt pacient.

IV.8.1 Ecranul Emergency

Acest ecran conține un chenar informativ pentru îndrumarea pacienților. În cazul în care un pacient ajunge în situația în care nu mai poate onora o programare dintr-un motiv neprevăzut, acesta poate crea o cerere către un alt pacient pentru a schimba intervalele celor două programări. Astfel, după un acord comun, primul nu va fi nevoit să o anuleze pe cea curentă și să creeze una nouă pentru care ar putea aștepta mai mult.

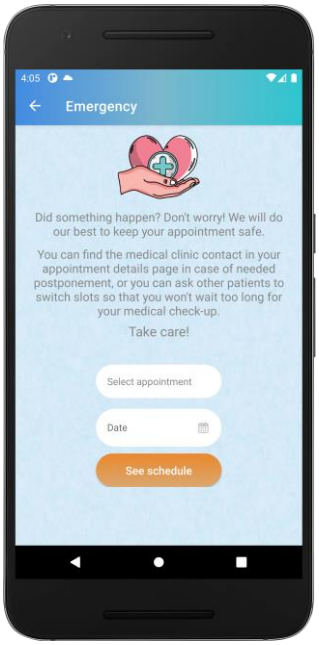

Figura 4.15 Pagina Emergency

Pentru începerea procesului, în pagina Emergency, pacientul va alege programarea cu pricina dintr-o listă de tip *dropdown* populată cu toate programările curente ale sale, precum și data pe care vrea să o aleagă în schimb.

Apăsarea butonului *See schedule* îl va redirecționa pe pacient către pagina în care este afișat programul doctorului în ziua aleasă.

IV.8.2 Pagina cu programul doctorului

În acest ecran sunt afișate toate intervalele în care doctorul poate avea programări, mai precis intervalul 08:00-20:00. Fiecare interval are un titlu corespunzător și o culoare specifică în funcție de disponibilitate astfel: intervalele ocupate cu programări au titlul "Busy" și culoarea gri, iar cele libere au titlul "Available" și culoarea albastră, des întâlnită în aplicație.

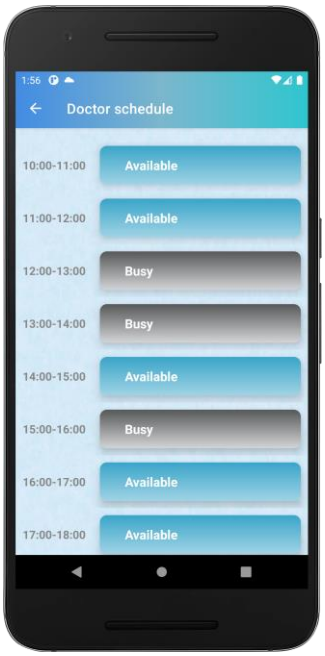

Figura 4.16 Programul doctorului

La apăsarea unui interval disponibil, pacientul este întrebat printr-un *pop-up* de confirmare dacă dorește să schimbe data programării curente cu intervalul ales. În caz afirmativ, modificarea este făcută în baza de date, iar pacientul este redirecționat pe pagina principală.

La apăsarea unui interval ocupat, utilizatorul este întrebat dacă dorește să trimită cererea de schimb pacientului căruia îi aparține programarea din intervalul respectiv. Astfel, datele personale sunt în siguranță chiar și în contextul interacțiunii între pacienți. În cazul confirmării trimiterii, pacientul respectiv primește o notificare pentru a accepta sau refuza cererea.

IV.8.3 Cererea de schimb

Cererea de schimb constă într-o notificare cu textul "New switch request" care, prin click, va deschide o fereastră cu detaliile cererii precum: programarea în cauză și intervalul în care aceasta ar putea fi mutată. Detaliile pacientului care a trimis cererea nu sunt vizibile pentru păstrarea confidențialității, nefiind relevante în context. La acceptarea cererii, intervalele celor două programări sunt interschimbate. În ambele cazuri, fie că pacientul acceptă sau refuză cererea, după răspuns, aceasta este ștearsă și primul pacient este înștiințat.

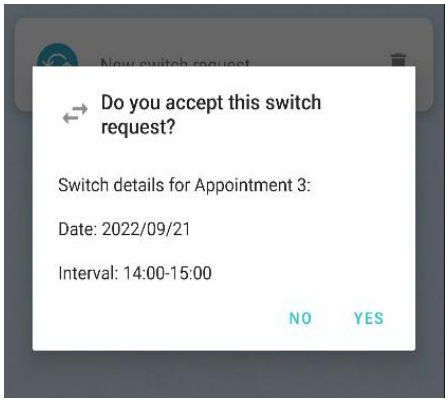

Figura 4.17 Cerere de schimbare a programării

IV.9 Bara de navigare

În partea de jos a ecranului principal, în care utilizatorul este redirecționat imediat după înregistrare sau logare, se află bara de navigare către ecranele care sunt cele mai des frecventate de utilizator.

Figura 4.18 Bara de navigare

Toate elementele incluse în bara de navigare sunt construite cu ajutorul fragmentelor și a unei activități ce are rol de *container* pentru acestea. Astfel, din bara de navigare, utilizatorul poate naviga între ecranele: Appointments, New Appointment, Profile, Notifications.

Acest meniu a fost implementat cu ajutorul componentei *Bottom Navigation* și fiecare element conține un *icon* sugestiv și numele ecranului.

IV.10 Meniul din bara de acțiuni

Meniul din bara de acțiuni constă într-o listă de două elemente ce conduc către funcționalități importante. Una dintre cele două este opțiunea de deconectare din aplicație. Cea de-a doua este accesibilă doar pentru pacienți și constă în redirecționarea către pagina Emergency.

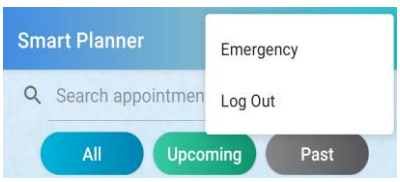

Figura 4.19 Meniul din bara de acțiuni

IV.11 Notificări

În cadrul aplicației, am implementat mai multe tipuri de notificări pe care utilizatorii le primesc pentru o experiență cât mai plăcută și benefică. Atât pacienții, cât și administratorii primesc notificări personalizate cu diverse scopuri ce vor fi detaliate pe rând.

După înregistrarea în aplicație, utilizatorii primesc automat o notificare pentru a-și completa datele personale în pagina de profil.

Un pacient poate primi această notificare și mai târziu, în cazul în care un administrator îi solicită asta prin apăsarea butonului *Request patient info* din pagina cu detaliile pacientului.

La rândul său, administratorul primește o notificare prin care este înștiințat că pacientul și-a actualizat datele personale.

În cazul unei cereri de schimbare a intervalului programării cu un alt pacient, acesta primește o notificare corespunzătoare care îi va deschide ulterior un *pop-up* pentru confirmare sau refuz. De asemenea, pacientul care a trimis cererea va primi o notificare cu răspunsul corespunzător.

Un al tip de notificare este cel care înștiințează pacienții în legătura cu programările viitoare, care sunt programate la mai puțin de trei zile distanță de ziua curentă.

Toate aceste notificări sunt în cadrul aplicației, în ecranul corespunzător. Acestea sunt încărcate într-un *recyclerview* și au opțiune de ștergere.

Pentru a încuraja utilizarea aplicației, am creat de asemenea o campanie de notificări cu ajutorul serviciului Firebase Cloud Messaging. Aceasta se adresează utilizatorilor care nu au mai folosit aplicația de cel puțin 14 zile și trimite o notificare prin care îi încurajează să își programeze o nouă vizită medicală.

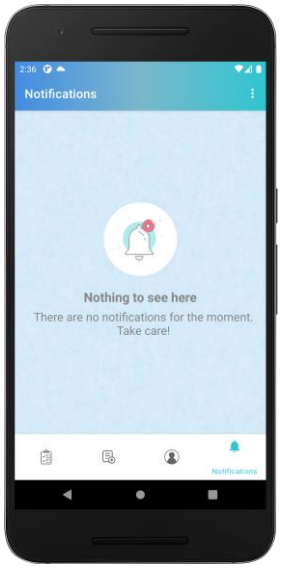

Figura 4.20 Ecranul fără notificări

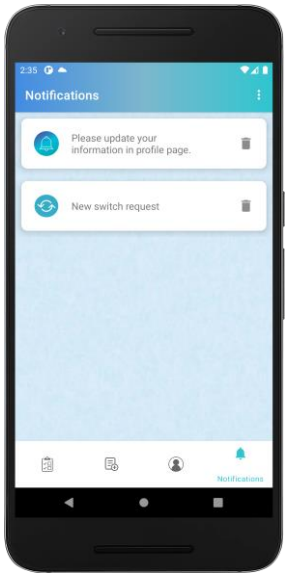

Figura 4.21 Ecranul cu notifcări

IV.12 Mesaje informative, validări și *pop-ups*

Mesajele de informare, ferestrele de tip *pop-up* pentru confirmarea acțiunilor, cât și validările din aplicație au rolul de a oferi utilizatorilor o experiență cât mai plăcută și sigură în folosirea aplicației.

Majoritatea funcționalităților din aplicație au diverse moduri de validare, în principal cu privire la *input-ul* introdus de utilizatori, precum câmpurile din formularul de înregistrare sau din cel de programare nouă. Astfel, cazuri neplăcute precum înregistrarea cu un cont deja existent ce cauzează suprascrierea datelor inițiale, sau selectarea unei date indisponibile pentru o programare, cum ar fi o zi de sâmbătă, sunt prevenite prin aceste măsuri.

Crearea unor elemente speciale în interfață pentru a informa utilizatorii cu privire la starea aplicației este, de asemenea, importantă. În ecranul cu programări, în timpul procesului de extragere a informațiilor din baza de date, utilizatorilor le este afișată o bară de progres și un mesaj "Loading appointments..." pentru a aștepta afișarea datelor.

Un alt caz ce merită menționat este scenariul în care un pacient, încercând sa trimită o cerere de schimb de programare, acesta apasă pe un interval ocupat, iar programarea corespunzătoare intervalului ales este chiar a lui. Modificarea în acest caz ar fi complet irelevantă pentru el, iar prevenirea acesteia se realizează printr-un mesaj de tip *Toast* în care acesta este informat că acel slot este ocupat de el și nu îl poate alege, precum și anularea trimiterii cererii.

De asemenea, în cadrul aplicației au fost implementate diverse ferestre de tip *pop-up* pentru confirmarea anumitor acțiuni. Spre exemplu, administratorul este întrebat dacă este sigur că vrea să șteargă programarea curentă după apăsarea butonului *Delete*. Astfel, pot fi prevenite acțiuni neintenționate care ar putea conduce la situații nedorite.

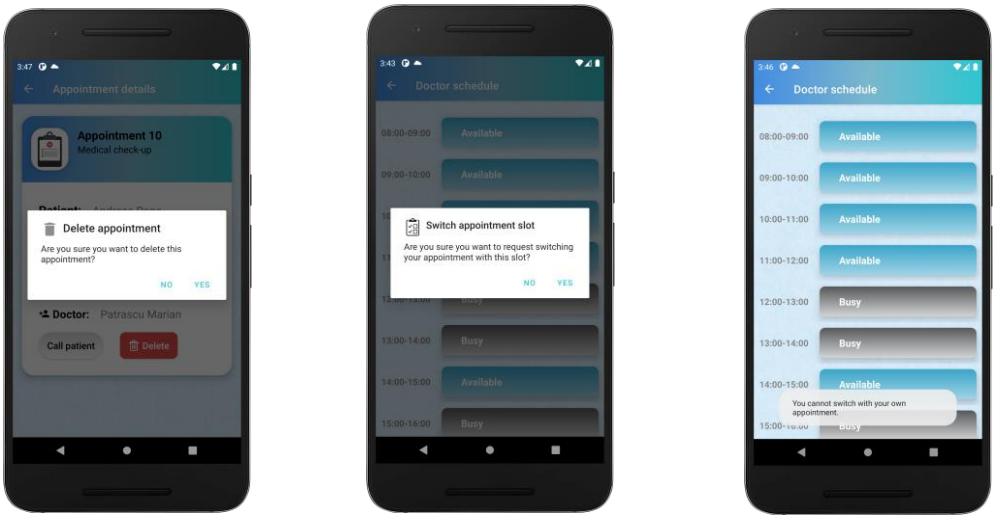

Figura 4.22 Mesaje informative și de confirmare

Concluzii

Este foarte important să avem grijă de sănătatea noastră. Procesul prin care trecem pentru a satisface această nevoie poate avea o influență semnificativă asupra deciziei de a planifica sau de a amâna o vizită medicală. De aceea, aplicația mobile pentru gestionarea programărilor medicale oferă pacienților o soluție prin care își pot crea cu ușurință o vizită la medicul specialist, fără efort și prea mult timp consumat. Aceasta vine, de asemenea, în ajutorul personalului medical prin dispensarea sarcinilor costisitoare ca timp care pot fi optimizate în mod digital.

Aplicația este deschisă publicului larg și își propune să aducă beneficii utilizatorilor încă de la prima utilizare. Caracterul accesibil datorat rulării pe dispozitive mobile contribuie la extinderea grupului țintă și întărește ideea de optimizare a unor sarcini îndelungate.

Interfața modernă și intuitivă contribuie la o experiență cât mai plăcută pentru utilizatori. Integrarea tehnologiilor de actualitate pe parcursul dezvoltării, precum și varietatea funcționalităților implementate, contribuie la calitatea aplicației oferind în același timp simplitate și confort.

Aplicația Smart Planner oferă pacienților un proces simplu și rapid de creare și gestionare a programărilor medicale. Aceasta încurajează pacienții să meargă cu regularitate la medic pentru un control de rutină și promovează prevenția. Aplicația le oferă disponibilitatea datelor de care au nevoie direct pe dispozitivul lor, atâta timp cât aceștia sunt conectați la Internet.

În ceea ce privește personalul medical, aplicația vine în ajutorul administratorilor de programări medicale prin optimizarea sarcinilor repetitive, oferind cât mai multe informații disponibile pentru pacienți. Astfel, interacțiunea dintre ei va fi redusă, iar timpul ambelor părți va fi economisit.

Scopul principal și obiectivele inițiale ale aplicației au fost ca aceasta să ofere o soluție pentru o problemă reală și să vină în ajutorul unui public țintă concret, aspecte ce au fost îndeplinite cu succes.

În concluzie, aplicația Smart Planner este un produs finit, cu un scop bine definit, ce soluționează gestionarea întrevederilor cu scop medical.

Perspective

În ceea ce privește perspectivele de dezvoltare ulterioară, aplicația ar putea fi îmbunătățită prin implementarea a noi funcționalități precum:

- Creșterea aplicabilității pentru mai multe tipuri de servicii medicale, programările actuale având intervalul standard de o oră. Astfel, prin diversitatea duratei, ar putea fi încadrate mai multe tipuri de programări.
- Opțiunea de marcare de către administratori a anumitor zile din programul medicilor ca fiind indisponibile, spre exemplu pentru zilele de concediu de odihnă sau sărbători legale.
- Trimiterea unor notificări cu mesaj personalizat între utilizatori sau implementarea unui *chat*.
- Serviciu de suport unde utilizatorii pot comunica diferite nemulțumiri, probleme sau funcționalități deficitare. Acest serviciu este important pentru sprijinul utilizatorilor, reprezentând, de asemenea, o oportunitate de dezvoltare a calității aplicației prin *feedback* direct.
- Opțiune pentru limba selectată în aplicație (română sau engleză).
- Extinderea domeniilor precum cele de *wellness* sau îngrijire personală. În cazul acestei dezvoltări, aplicația are un potențial considerabil de extindere, crescând atât numărul utilizatorilor, cât și frecvența folosirii aplicației de către aceștia. Această extindere presupune implementarea de funcționalități noi sau adaptarea celor existente pentru aceste servicii.
- Implementarea configurațiilor necesare și dezvoltarea aplicației pentru alte sisteme de operare. Această funcționalitate ar contribui la creșterea publicului țintă.

Aplicația Smart Planner are un potențial ridicat de dezvoltare, iar în forma actuală reprezintă o bază stabilă de pornire prin înglobarea funcționalităților ce vin în sprijinul utilizatorilor.

Bibliografie

[1] David Curry, 10 August 2022, Android Statistics, https://www.businessofapps.com/data/android-statistics/ , Accesat 08 August 2022, 16:43

[2] OCDE/European Observatory on Health Systems and Policies (2021), România: Profilul de țară din 2021 în ceea ce privește sănătatea, State of Health in the EU, OECD Publishing, Paris/ European Observatory on Health Systems and Policies, Bruxelles, pg. 13 https://health.ec.europa.eu/system/files/2022-01/2021_chp_romania_romanian.pdf , Accesat 20 August 21:32

[3] Shubham Khan, 5 Martie 2021, Android | Android Application File Structure, https://www.geeksforgeeks.org/android-android-apps-file-structure/ , Accesat 10 August 18:07

[4] Vrijraj Singh, 12 Decembrie 2018, Introduction to Firebase, https://medium.com/codingurukul/introduction-to-firebase-f9f6ccc8a785 , Accesat 11 August 13:12

[5] https://kotlinlang.org/docs/getting-started.html , Accesat 09 August 20:09

[6] https://kotlinlang.org/docs/async-programming.html , Accesat 09 August 22:11

[7] https://kotlinlang.org/docs/multiplatform-mobile-concurrency-and-coroutines.html , Accesat 09 August 22:27

[8] https://kotlinlang.org/docs/null-safety.html , Accesat 09 August 22:43

[9] https://developer.android.com/guide/topics/ui/declaring-layout , Accesat 10 August 15:36

[10] https://developer.android.com/guide/platform , Accesat 10 August 16:50

[11] https://developer.android.com/jetpack/getting-started , Accesat 10 August 20:21

[12] https://firebase.google.com/docs/auth , Accesat 11 August 14:01

[13]https://firebase.google.com/docs/reference/admin/java/reference/com/google/firebase/auth/hash/Scrypt , Accesat 11 August 15:20

[14] https://firebase.google.com/docs/database/rtdb-vs-firestore , Accesat 11 August 19:21

[15] https://firebase.google.com/docs/database , Accesat 11 August 18:34

[16] https://firebase.google.com/docs/database/security , Accesat 12 August 14:34

[17] https://firebase.google.com/docs/storage , Accesat 12 August 15:29

[18] https://developers.google.com/maps/documentation/android-sdk/overview , Accesat 17 August 14:45

[19] https://developers.google.com/maps/documentation/directions/overview , Accesat 17 August 15:23

[20] https://bumptech.github.io/glide/ , Accesat 18 August 16:40

[21] https://square.github.io/okhttp/ , Accesat 18 August 17:11

[22] https://developer.android.com/topic/libraries/architecture/lifecycle , Accesat 19 August 18:32

[23] https://github.com/material-components/material-components-android , Accesat 20 August 14:56

[24] https://evexiaapp.ro/ , Accesat 20 August 22:40

[25] https://play.google.com/store/apps/details?id=com.docbook.app&hl=ro , Accesat 21 August 14:45

[26] https://play.google.com/store/apps/details?id=com.tremend.reginamaria&hl=en&gl=US , Accesat 25 August 23:05

[27] https://blog.mindorks.com/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide , Accesat 26 August 19:55