



UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ**



SPECIALIZAREA INFORMATICĂ

Lucrare de licență

**Aplicație Android pentru schimb de obiecte
și donații**

Absolvent

Diana-Maria Ifrosă

Coordonator științific

Lect. dr. Ana Cristina Iova

București, iunie - iulie 2023

Rezumat

Aplicația “Swap” vine în ajutorul oamenilor care doresc să își îmbrospăteze locuința, fie prin a dona obiectele pe care nu le mai folosesc, fie prin a le schimba cu altele. În acest fel, aceștia contribuie la un mediu înconjurător mai puțin poluat și totodată întregesc spiritul unei comunități orientate spre întrajutorare.

Prin intermediul “Swap”, utilizatorii pot posta obiectele de care nu mai au nevoie sau pot vedea obiectele celorlalți, iar atunci când consideră un schimb ca fiind potrivit, îl pot propune altor utilizatori prin intermediul *chat*-ului din aplicație; acest procedeu se aplică și în cazul donațiilor, iar *chat*-ul poate fi folosit și pentru a cere informații suplimentare. În plus, aplicația pune la dispoziție o pagină dedicată în care aceștia își pot salva postările favorite, o hartă în timp real în care pot vizualiza mai ușor ce obiecte sunt disponibile în jurul lor, dar și o secțiune de tip Istoric unde le sunt salvate schimburile/donațiile din trecut, precum și obiectele postate, încă disponibile în prezent.

Mai mult decât atât, au acces la o pagină specială, personalizată pentru fiecare utilizator în parte, destinată recomandărilor, în care pot vizualiza obiecte pe baza preferințelor salvate în setări, care respectă anumite criterii (pot fi ridicate din anumite orașe favorite, sunt postate de către anumiți proprietari ai căror obiecte îi atrag, conțin niște cuvinte-cheie sau fac parte din categorii specifice).

Deoarece ideea este materializată printr-o aplicație *mobile* pentru Android, este vizat un grup larg de utilizatori, dat fiind faptul că este un sistem de operare larg răspândit, iar prin acest fapt inițiativa spre reciclare și re folosire a obiectelor deja existente este cu ușurință transmisă majorității utilizatorilor de *smartphone*.

Abstract

The “Swap” mobile application comes to the aid of people who want to refresh their home, either by donating the objects they no longer use, or by exchanging them for others. In this way, they contribute to a less polluted environment and at the same time they take part of a community oriented towards helping others.

Through “Swap”, users can post unnecessary objects, see other people's objects, and when they wish to make an exchange, they can propose it to other users through the chat built within the application; this process also applies to donations and the chat can also be used to request more information. In addition, the application provides a dedicated page where they can save their favorite posts, a real-time map where they can more easily see what objects are available around them, but also a History section where their exchanges/donations from the past are saved, as well as their own still available items.

Moreover, they have access to a special page, personalized for each individual user, intended for recommendations, where they can view items based on the preferences previously saved in the settings, that respect some constraints (can be picked up from certain favorite cities, are posted by owners whose objects attract them, contain certain keywords or fall under specific categories).

Since the idea is materialized in a mobile application for Android, a wide group of users is targeted, given the fact that it is a widespread operating system, and by this fact the initiative to recycle and reuse already existing objects is easily transmitted to the majority of smartphone users.

Cuprins

1. Introducere.....	8
1.1. Contextul problemei	8
1.2. Motivație.....	9
1.3. Comparație cu alte aplicații similare	10
1.4. Structura lucrării	11
2. Tehnologii utilizate	12
2.1. Android.....	12
2.1.1. Backend.....	13
2.1.2. Frontend.....	14
2.2. Firebase	15
2.2.1. Cloud Firestore	16
2.2.2. Cloud Storage	18
2.2.3. Firebase Authentication.....	19
2.2.4. Cloud Functions & Firebase Cloud Messaging	19
2.3. Maps SDK & Places SDK.....	20
2.4. Espresso	21
2.5. Alte librării open-source.....	22
3. Arhitectura aplicației	23
3.1. Principii	23
3.2. Model View ViewModel.....	24
3.3. Navigarea între fragmente	26
3.4. Structura bazei de date.....	28
4. Prezentarea aplicației	31
4.1. Identitatea vizuală.....	31
4.2. Autentificarea	32
4.3. Profilul personal	35
4.3.1. Editarea profilului.....	36
4.3.2. Preferințele pentru recomandări	36
4.3.3. Notificările.....	38
4.4. Pagina principală (Acasă).....	39
4.5. Adăugarea unui obiect.....	40

4.6. Pagina unui obiect	41
4.7. Pagina unui alt utilizator.....	43
4.8. Recomandări.....	44
4.9. Harta	44
4.10. Favorite.....	45
4.11. Conversații.....	46
4.11.1. Aspecte generale.....	46
4.11.2. Propunerile.....	47
4.12. Istoric	48
4.13. Pagina de contact (feedback).....	50
4.14. Schimbarea limbii.....	50
4.15. Controlul administratorului folosind Firebase.....	51
4.16. Alte detalii	52
5. Testarea aplicației.....	53
5.1. Testarea de interfață (UI testing).....	53
5.2. Noțiuni generale despre Espresso.....	53
5.3. Cazurile de test	54
5.4. Raport final și concluzii.....	57
6. Concluzii și direcții de dezvoltare	58
Bibliografie.....	59
Anexa A – figuri adiționale.....	62
Anexa B – teste de interfață.....	63

Listă de figuri

Figura 2.1 – Distribuția versiunilor de API pe Android, din cadrul Android Studio.....	13
Figura 3.1 – Șablonul MVVM aplicat în Swap (figură inspirată după cea din [14]).....	24
Figura 3.2 – Flow-ul datelor la trimiterea unui mesaj.....	25
Figura 3.3 – Meniul principal de tip drawer.....	26
Figura 3.4 – Navigare prin butonul săgeata înapoi	26
Figura 3.5 – Graf de navigare al profilului personal	27
Figura 3.6 – Meniul aplicației pentru un vizitator.....	27
Figura 3.7 – Navigarea în cadrul aplicației pentru un vizitator.....	28
Figura 3.8 – Structura din Cloud Storage	30
Figura 3.9 – Structura folder-ului unui utilizator din Cloud Storage	30
Figura 4.1 – Logo-ul aplicației	31
Figura 4.2 – Logo-ul aplicației în meniul telefonului	31
Figura 4.3 – Culori folosite în “Swap”	32
Figura 4.4 – Splash screen la deschiderea aplicației	32
Figura 4.5 – Ecrane introductive	33
Figura 4.6 – Pagina de autentificare.....	34
Figura 4.7 – Pagina de creare cont	34
Figura 4.8 – Email de bun-venit.....	35
Figura 4.9 – Pagina de login.....	35
Figura 4.10 – Validările câmpurilor la autentificare	35
Figura 4.11 – Profilul personal.....	35
Figura 4.12 – Modificarea fotografiei de profil și a parolei	36
Figura 4.13 – Setarea preferințelor pentru recomandări	37
Figura 4. 4 – Setarea preferințelor pentru notificări.....	38
Figura 4.15 – Notificare la adăugarea unei postări de interes	38
Figura 4.16 – Modificarea notificărilor din setările telefonului.....	38
Figura 4.17 – Filtrarea principală din pagina acasă	39
Figura 4.18 – Opțiunile de sortare și filtrare	39

Figura 4.19 – Pagina de adăugare obiect.....	40
Figura 4.20 – Secțiune dedicată preferințelor de schimb	40
Figura 4.21 – Pagina de alegere a locației.....	41
Figura 4.22 – Informații adiționale pentru câmpul fotografii	41
Figura 4.23 – Pagina unui obiect pentru schimb/donație	42
Figura 4.24 – Butonul de raportare postare și pagina destinată acestui lucru	43
Figura 4.25 – Pagina unui alt utilizator	43
Figura 4.26 – Recomandările pe baza preferințelor	44
Figura 4.27 – Pagina hărții	45
Figura 4.28 – Pagina favoritelor cu cele 2 categorii	45
Figura 4.29 – Pagina conversațiilor.....	46
Figura 4.30 – Trimiterea mesajelor text și a imaginilor.....	46
Figura 4.31 – Confirmarea unei propuneri de oferire donație.....	48
Figura 4.32 – Obiectele disponibile din istoric și obiectele indisponibile (evenimentele)	49
Figura 4.33 – Evenimentele de primire donație, donare și schimb	49
Figura 4.34 – Pagina de contact	50
Figura 4.35 – Vizualizarea datelor în consola Firestore.....	51
Figura 4.36 – Vizualizarea și gestionarea utilizatorilor în Firebase Authentication	52
Figura 4.37 – Mesaje explicative când nu sunt disponibile date.....	52
Figura 5.1 – Raportul testelor, generat de Android Studio	57
Figura A.1 – Navigarea în cadrul aplicației pentru un utilizator conectat	62

1. Introducere

1.1. Contextul problemei

Sustenabilitatea și dorința de a aborda un stil de viață mai prietenos cu mediul înconjurător sunt teme adese discutate din ultima perioadă. Un semnal de alarmă care ne-a făcut să ne îndreptăm atenția și spre astfel de teme este avansarea accentuată a nivelului de poluare, ce produce efecte nocive atât asupra oamenilor, cât și asupra naturii în general.

Conform United Nations Environment Programme, industria produselor de îmbrăcăminte încălțăminte și accesorii produce aproximativ 10% din cantitatea de dioxid de carbon emisă în natură, un procent semnificativ mai mare decât cel produs de industriile de zboruri și de curierat combinate [15]. În același timp, este important de menționat faptul că atât această industrie a modei, cât și altele, au abordat o manieră accelerată de producție pentru a satisface nevoia consumatorilor de noutate și a capta constant atenția lor. Astfel, nivelul de poluare și risipă produs ca urmare a stilului de viață accelerat al omului modern este în continuă creștere, fapt ce poate fi confirmat de studii care atestă că piesele de îmbrăcăminte sunt aruncate, în medie, după 7-10 purtări.

Însă, această speță a comerțului nu este singura care provoacă daune mediului înconjurător. Precum fenomenul *fast fashion* prezentat anterior, și fenomenul *fast furniture* [7] ia amploare de la o zi la alta. Potrivit Environmental Protection Agency (EPA), la nivelul Statelor Unite s-a constatat că în anul 2018 au fost aruncate 12 milioane de tone de mobilă, un procent de mai puțin de 20% neajungând în gropi de gunoi. Comparând cu 2.2 milioane de tone înregistrate în 1960, se poate observa o scădere masivă în orientarea omului spre reciclare și refolosire în alte scopuri a celor ajunse nenecesare.

Soluții însă spre a diminua efectul nociv al obișnuințelor omului modern sunt diverse, precum schimbul obiectelor între persoane, donarea obiectelor pe care le considerăm nefolositoare pentru noi și, ca un efect secundar al celor deja menționate, reducerea cumpărării de produse noi. Toate aceste măsuri reduc emisiile de gaze toxice în natură, contribuie la o mai bună gestionare a bugetului personal (întrucât obiectele de tip *second-hand* sau de tip *outlet* sunt adeseori mai ieftine decât cele noi) și previn poluarea câmpurilor prin transformarea lor în gropi de gunoi. Mai mult decât atât, cel puțin din punct de vedere al donațiilor, este de menționat și efectul la nivelul

componentei moral-sociale, întrucât astfel de acțiuni întregesc comunitatea și oferă sprijin celor aflați în nevoie.

1.2. Motivație

Motivația creării unei aplicații mobile precum **“Swap”**, a venit în urma unei călătorii în Amsterdam în 2021. Scopul meu era să îmi vizitez verișoara, și cu această ocazie să observ și stilul de viață al oamenilor din acele părți.

Cu toate că am putut observa multe alte diferențe de mentalitate și de stil al vieții comparativ cu România în excursia mea, ceea ce mi s-a părut cu adevărat interesant a fost modul în care studenții (și nu numai) “comunicau” și se ajutau între ei, fără să interacționeze direct. Astfel, la parterul clădirilor ce îi găzduiau, se găseau adesea piese de mobilier, accesorii de casă ori cărți, pe care ei nu le mai considerau utile, fie din considerente de gust, fie din motivul pragmatic al mutării în altă locuință, și pe care alegeau să le “ofere comunității”. În acest fel, cei ce aveau nevoie de diverse obiecte spre a-și întregi locuința le puteau găsi uneori doar făcând o plimbare prin oraș. Am observat de asemenea și rafturi de cărți amenajate special cu acest scop: lași ce nu mai ai nevoie, iei ce ți se pare interesant. Tot acest ecosistem de repunere în funcțiune al obiectelor, făcut într-o manieră atât de tacită, m-a făcut să mă gândesc la utilitatea unei platforme special concepută pentru schimburi și donații de obiecte, atât la nivelul României, cât și internațional.

Ideea implementării ei ca o aplicație de *mobile* a venit organic: prea puține lucruri folosesc oamenii din zilele noastre mai mult decât telefonul. Astfel, am considerat adecvat să implementez o aplicație adresată deținătorilor de *smartphone* ce are ca sistem de operare Android, dat fiind faptul că reprezintă o majoritate. Prima jumătate de semestru a anului 2023 înregistrează la nivelul pieței de telefoane o vânzare de aproximativ 72% de telefoane care rulează folosind Android, comparativ cu 27% ce au ca sistem de operare iOS [3]. Este așadar indubitabil faptul că Android rămâne unul dintre cele mai populare sisteme de operare pentru telefoanele inteligente.

1.3. Comparație cu alte aplicații similare

Google Play pune la dispoziție aplicații care gravitează în jurul aceleiași idei: ce ție probabil nu îți mai trebuie, pentru altcineva poate fi o dorință.

Printre astfel de aplicații se numără și **“SwappyVerse”**. În cadrul acesteia, utilizatorii pot vedea obiectele postate de ceilalți, le pot salva la Favorite și pot conversa în cadrul aplicației. În plus, la prima deschidere a aplicației utilizatorul trebuie să aleagă niște categorii de interes dintr-o mulțime prestabilită. Elementul de noutate este dat de existența banilor virtuali, ce se pot acumula în urma postărilor sau a recomandării aplicației la prieteni și au ca scop facerea postărilor proprii mai ușor de găsit pentru ceilalți, acționând ca un amplificator al vizibilității postărilor. Este de menționat că aplicația este disponibilă doar în limba engleză.

O altă aplicație este și **“trash nothing”**, o platformă în limba engleză destinată celor interesați de a da obiecte de care nu mai au nevoie sau de a primi obiecte pe care le cer prin intermediul postărilor. Nu este concepută în mod special pentru a favoriza și schimburile, însă utilizatorii se pot bucura de obiecte obținute pe gratis de la membrii din propria comunitate. Aplicația conține secțiuni de *chat*, de favorite și pune la dispoziție posibilitatea de a crea sau de a intra în grupuri, ceea ce dă acces utilizatorilor la o comunitate mai largă, cu mai multe postări. “trash nothing” mai conține, printre altele, și o secțiune în care utilizatorii povestesc din experiența lor cu aplicația. Țările în care platforma este cea mai populară sunt Regatul Unit și Statele Unite ale Americii.

Similară celei precedente, **“HaveNeed”** este o aplicație disponibilă pe Google Play în care utilizatorii postează ce au disponibil spre schimb și ce au nevoie, incluzând atât bunuri, cât și servicii. Dinamica aplicației este distinctă prin faptul că sunt create grupuri (*loops*) din care fac parte 2-5 utilizatori ce dețin obiecte dorite de ceilalți participanți. Astfel, schimbul nu se limitează doar la o acțiune reciprocă între 2 persoane. Mai mult, aplicația conține o pagină de *chat*, una de favorite, un intro explicativ la deschiderea ei și este disponibilă doar în limba engleză.

Comparând cu cele enunțate anterior, aplicația prezentată în lucrare, **“Swap”**, se remarcă prin faptul că este disponibilă și în limba română, dar și în engleză, aducând conceptul de *freecycle* și în România, putând fi totodată folosită și pe plan internațional. Pe de altă parte, se axează pe interacțiunea a numai 2 persoane, însă prin acest fapt face întregul proces de schimb/donație mai accesibil și rapid. În plus, conține o hartă cu ajutorul căreia se pot vizualiza mai ușor postările și

permite o filtrare mai amănunțită a acestora, prin intermediul paginii de recomandări, datorită numeroaselor preferințe pe care utilizatorii le pot selecta. “Swap” permite atât schimbul cât și donațiile de obiecte și nu limitează posibilitățile utilizatorilor folosind monede virtuale.

1.4. Structura lucrării

Capitolele ce urmează vor analiza în detaliu fiecare element ce a contribuit la realizarea proiectului final.

Astfel, **capitolul 2** prezintă amănunțit tehnologiile folosite atât pe partea de *backend*, cât și pe cea de *frontend*, și nu în cele din urmă, la nivel de bază de date. Tot în acest capitol vor fi menționate și diferitele librării/*plugin*-uri utilizate pentru îmbunătățirea aspectului vizual, dar și pentru o mai bună gestionare a datelor din cadrul aplicației. Fiecare tehnologie va fi însoțită de justificări în privința alegerii ei.

Capitolul 3 este destinat arhitecturii generale a proiectului. Este explicat șablonul folosit la nivelul interacțiunii dintre clase (“Model View ViewModel”), precum și modul de navigare între fragmentele/ecranele aplicației. În plus, este detaliată structura bazei de date *cloud-based*. Întreg capitolul este însoțit de scheme și imagini reprezentative pentru o mai bună înțelegere a întregului sistem.

În **capitolul 4** se descrie amănunțit fiecare funcționalitate a aplicației, explicată cu ajutorul capturilor de ecran. Această secțiune a lucrării este concepută ca un manual de utilizare al aplicației, subliniindu-se scenariile de utilizare aferente.

Testarea minimală a aplicației la nivel de interfață este prezentată în **capitolul 5**, împreună cu particularitățile *framework*-ul folosit, Espresso.

În final, **capitolul 6** cuprinde concluziile rezultate din urma realizării proiectului, atât din punct de vedere personal, incluzând provocările întâmpinate, cât și din punct de vedere obiectiv, raportat la eventualele direcții de dezvoltare a aplicației.

2. Tehnologii utilizate

2.1. Android

Android este un sistem de operare dezvoltat în cadrul companiei Android Inc. începând cu 2003, companie cumpărată de către Google în 2005. Este cel mai popular sistem de operare, bazându-ne pe rezultatele prezentate anterior în lucrare [3], putând fi regăsit nu numai în telefoanele inteligente, cât și în tablete, ceasuri (Wear OS este bazat pe Android), televizoare sau chiar și mașini (Android Automobile) [17]. Pe lângă multitudinea de dispozitive în care poate fi remarcat ca sistem de operare de bază, există și alte sisteme de operare care pot rula aplicații dezvoltate folosind Android, spre exemplu ultimele versiuni de Chrome OS.

Android este la bază *open-source*, ceea ce îi justifică popularitatea, însă nu toate elementele sale *software* încorporate sunt disponibile publicului, amintind Google Play Store sau Gmail. Din acest punct de vedere, al libertății cu care poate fi modificat, există numeroși comercianți ale caror produse, deși sunt bazate pe Android, au diverse forme. Un exemplu în acest sens este interfața unui telefon Samsung, comparativ cu unul LG.

Prima versiune de Android (1.0) a fost lansată în anul 2008 și a conținut versiuni incipiente ale aplicațiilor cunoscute din ziua de azi, Gmail, Youtube, Maps [10]. Ultima versiune lansată a fost cea cu numărul 13, în august 2022. Fiecare lansare a sistemului de operare a fost însoțită de un *API level* distinct, pentru a marca diferențele la nivelul *framework*-ului.

Aplicația prezentată în această lucrare poate fi rulată pe telefoane care au **API level minimum 26**, dar este concepută pentru a rula pe **API Level 33** (*target SDK version*). Această decizie a fost influențată de funcționalitățile disponibile și de distribuția cumulativă la nivel internațional a telefoanelor care au la bază Android OS, prezentată de IDE-ul pe care l-am folosit (Android Studio) la crearea proiectului, cum poate fi observat în figura 2.1. Reiese din aceasta că, alegând API level minimum 26, aplicația “Swap” va putea fi folosită pe aproximativ 90.7% din telefoanele care rulează Android OS. Este indicat la crearea unei aplicații mobile ca aceasta să fie concepută pentru o gamă cât mai vastă de telefoane, pentru a acoperi un procent cât mai mare de utilizatori.

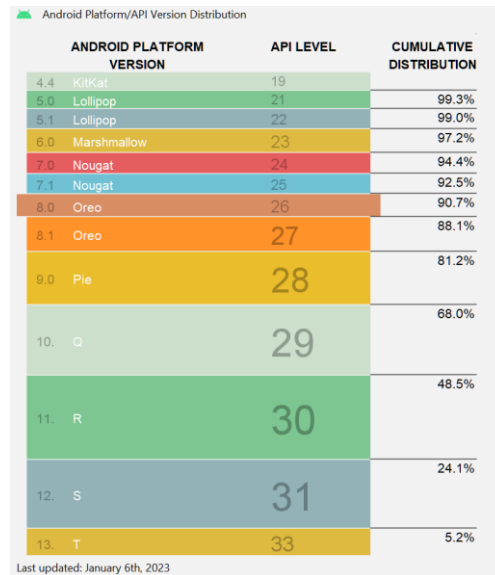


Figura 2.1 – Distribuția versiunilor de API pe Android, din cadrul Android Studio

2.1.1. Backend

Pe partea de *backend* am ales să implementez proiectul folosind **tehnologii native**. Comparând cu alte tipuri de tehnologii actuale, spre exemplu cele *cross-platform*, implementarea unei aplicații Android folosind tehnologiile native vine cu avantaje, precum faptul că sunt mai mature din punct de vedere al dezvoltării, sunt mai performante la nivel de interacțiune cu componentele *hardware* (de exemplu camera) și folosindu-le pe acestea, dezvoltarea per total a aplicației este mai ușoară și mai eficientă datorită unei plaje mari de librării *3rd party* pusă la dispoziție. Pe de altă parte, alegând această metodă de implementare, aplicația rezultată va putea fi folosită doar pe un singur sistem de operare (Android) și va fi necesară o altă implementare pentru o eventuală extindere la un alt sistem de operare, precum iOS [2].

Proiectul este construit folosind **AndroidX**, o îmbunătățire a librăriei Android Support Library (*deprecated*) și care are ca scop reorganizarea într-o manieră mai ușor de înțeles a pachetelor și ierarhiei aferente. Parte din această librărie fac elementele ce țin strict de dezvoltarea în Android, precum Fragment, RecyclerView sau Toolbar și nu de Android OS [20].

Pentru dezvoltarea unei aplicații Android native sunt multiple opțiuni în ceea ce privește limbajul de programare. Dintre acestea cele mai populare sunt Java, Kotlin și C/C++. Pentru

crearea proiectului prezentat am ales să folosesc **Kotlin** din multiple considerente. Kotlin este un limbaj de programare *open-source* dezvoltat în 2010 de către JetBrains, folosit în numeroase companii populare precum Amazon, Netflix sau Uber [33][5]. Acesta se poate combina în cadrul aceluiasi proiect cu Java, întrucât este 100% compatibil cu JVM (Java Virtual Machine) și tot din acest punct de vedere, este foarte ușor de învățat având o bază de cunoștințe în Java. Comparativ cu Java, Kotlin se poate folosi atât în manieră *object oriented*, cât și *functional*, având particularități precum *lambda functions* sau *higher-order functions*. Mai mult, un avantaj față de Java îl reprezintă și numărul de linii de cod necesare scrierii unei aceleiași funcționalități. Din acest punct de vedere, Kotlin este considerat mai concis, necesitând cu aproximativ 40% mai puține linii de cod. O altă îmbunătățire cu care Kotlin se distinge față de competitorii săi este proprietatea de *null-safety*, dată de tipurile de date special concepute să facă aplicația mai puțin predispusă la erori de tip de date, precum *Null Pointer Exception*. Este de asemenea un limbaj de programare mai ușor de folosit prin faptul că nu pune accent pe *boilerplate code* (*setters*, *getters* etc.) și lasă programatorul să se concentreze pe lucrurile cu adevărat importante din cadrul aplicației.

Build tool-ul folosit la crearea proiectului a fost **Gradle**, care este cel oficial integrat în Android Studio (IDE-ul recomandat dezvoltării aplicațiilor Android). Cu ajutorul acestuia se gestionează dependențele, versiunile de SDK, *plugin*-urile de la nivelul proiectului și se pot seta diverse proprietăți care să ajute la partea de *build* a proiectului într-un timp mai scurt, precum mecanismul de *caching* [30].

2.1.2. Frontend

La nivel de *frontend*, am folosit fișiere **XML** (eXtensible Markup Language) pentru conturarea fiecărei pagini sau a unui element individual de UI (de exemplu, un *card* reprezentând o postare din pagina principală) folosind elemente de bază de Android (TextView, ProgressBar etc.) sau din Android Material (ShapeableImageView, TextInputLayout etc.).

Fișierele XML conțin tag-uri organizate sub formă de ierarhie și proprietăți specifice elementului de UI folosit, precum *textColor*, *layout_gravity*, *padding* și multe altele.

Cu ajutorul acestora, am integrat elementele de UI disponibile în Android și în alte librării *open-source*, am separat codurile culorilor și textele folosite în fișiere separate și am creat iconițe cu simboluri specifice (inimă pentru favorite, coș de gunoi pentru ștergere etc.).

2.2. Firebase

Firebase reprezintă un set de *tool*-uri dezvoltat de Google cu scopul de a ușura munca programatorilor de Android și nu numai [11]. Aceste instrumente sunt *cloud-based* și sunt folosite pentru a face etapele de *build*, *deploy* sau *test* ale unui proiect mai ușor de gestionat. *Tool*-urile sunt împărțite în 4 categorii principale: Build, Release & Monitor, Analytics și Engage, fiecare abordând un sector diferit din procesul de dezvoltare *software*. Funcționalitățile oferite de către Firebase sunt numeroase, iar printre acestea se număra RealtimeDatabase, Cloud Messaging, Authentication, Test Lab și Crashlytics. Cel mai comun pachet oferit de Firebase este Blaze Plan, de tipul *pay-as-you-go* care permite amatorilor să dezvolte aplicații mici în mod gratuit, întrucât nu folosesc prea multe resurse.

În cadrul proiectului, am decis să folosesc Firebase datorită multitudinii de funcționalități oferite. Astfel, prin intermediul acestuia am reușit să mențin baza de date în *cloud*, ceea ce mi-a permis să mă pot concentra mai mult pe dezvoltarea efectivă a aplicației decât pe *setup*-ul bazei de date. În plus, cu ajutorul Firebase Authentication am identificat și gestionat cu ușurință utilizatorii înregistrați, iar notificările au fost implementate folosind Cloud Functions.

O altă alternativă disponibilă pentru stocarea datelor, populară în dezvoltarea aplicațiilor mobile Android este SQLite, o baza de date SQL *open-source*, ușor de folosit și care permite o stocare structurată a datelor [9]. Pentru proiectul prezentat, însă, nu este o alternativă fezabilă deoarece presupune stocarea locală a datelor, pe telefoanele Android, iar utilizatorii nu au acces la datele celorlalți, spre exemplu postările lor.

Voi descrie în subcapitolele următoare fiecare instrument în parte și modul în care l-am folosit, pentru a evidenția utilitatea pachetului de servicii Firebase, urmând ca în capitolul 3.4 să explic arhitectura generală a bazei de date și modul de împărțire a datelor pe categorii.

2.2.1. Cloud Firestore

Cloud Firestore este unul din *tool*-urile oferite de Firebase și reprezintă o bază de date în *cloud* ce folosește NoSQL și care poate fi utilizată în dezvoltarea aplicațiilor mobile sau web [23]. Avantajele acestuia sunt flexibilitatea în folosire, scalabilitatea bazei de date, cât și faptul că oferă *realtime updates* prin intermediul *realtime listeners*. În acest fel, datele modificate în Firestore sunt sincronizate între toate dispozitivele care rulează aplicația respectivă. În plus, oferă *offline support*, ceea ce face ca aplicația să poată fi folosită chiar și fără o conexiune la internet, sincronizând modificările în *cloud* când dispozitivul revine la o conexiune de internet.

În cadrul “Swap”, Cloud Firestore are rolul de a reține datele obiectelor postate, ale utilizatorilor, tranzacțiile încheiate, propunerile trimise și alte date destinate administratorilor (raportările postărilor, *feedback*, statusul email-urilor trimise la înregistrarea unui nou utilizator).

Toate acestea sunt organizate în colecții cu scopuri bine definite care au în componență documente ce memorează perechi de tip cheie-valoare [24]. O colecție poate fi privită ca o clasă din Kotlin, iar un document ca un obiect din acea clasă. Documentele pot memora tipuri de date clasice precum string-uri sau numere, însă pentru a memora un obiect mai complex, este necesară îmbinarea mai multor tipuri de date, spre exemplu un map care are mai multe chei reprezentând câmpurile obiectului iar valori fiind efectiv valorile câmpurilor. Documentele au id-uri unice și nu sunt limitate în a conține aceleași câmpuri, însemnând că mai multe documente din cadrul aceleiași colecții pot avea câmpuri diferite. Există de asemenea opțiunea de a crea subcolecții în cadrul unei colecții, însă acestea vin cu dezavantaje din punct de vedere al apelării lor deoarece nu se pot face la fel de multe operații.

Interacțiunea cu baza de date se face cu ajutorul operațiilor asincrone de citire, scriere, actualizare sau ștergere (CRUD), disponibile prin metodele aferente puse la dispoziție de către librăria Cloud Firestore Android (“com.google.firebase:firebase-firestore-ktx”). Un exemplu de citire și extragere a datelor dintr-un document, parte a unei colecții numită ExchangeItems, poate fi observat în fragmentul de cod de mai jos.

În exemplul descris, prin intermediul metodei `getExchangeItem`, care primește ca parametru un id al obiectului vizat și un *callback* pentru a întoarce obiectul (deoarece operația asupra bazei de

date se face în manieră asincronă), se salvează datele în obiectul item de tipul ItemExchange și se notifică metoda apelantă prin intermediul *callback*-ului.

```
val db = Firebase.firestore
val EXCHANGE_COLLECTION = "ExchangeItems"

fun getExchangeItem(itemId: String, callback: OneParamCallback<Item>) {
    db.collection(EXCHANGE_COLLECTION).document(itemId).get().addOnCompleteListener { task ->
        if (task.isSuccessful) {
            val item = task.result.toObject(ItemExchange::class.java)
            callback.onComplete(item)
        } else {
            callback.onError(task.exception)
        }
    }
}
```

Pentru a “asculta” modificările dintr-o anumită colecție se folosesc *listeners* speciali care se activează atunci când apare o modificare în *cloud*. Elementul de la baza funcționării lor este *snapshot*-ul datelor care se creează la inițializarea *listener*-ului și se actualizează la fiecare schimbare apărută în baza de date [24]. Un astfel de procedeu este descris în fragmentul de cod de mai jos și este aplicat aceleiași colecții, însă de această dată se returnează prin intermediul *callback*-ului toate obiectele (documentele) din cadrul colecției, sub formă de ArrayList.

```
fun getExchangeItemsAndListen(callback: ListParamCallback<Item>) {
    itemsExchangeListenerRegistration =
        db.collection(EXCHANGE_COLLECTION).addSnapshotListener { snapshot, error ->
            if (error != null) {
                Log.w(TAG, "Listen failed for exchange items", error)
                return@addSnapshotListener
            }
            if (snapshot != null) {
                val allItems = ArrayList<Item>()
                val documents = snapshot.documents
                documents.forEach {
                    val item = it.toObject(ItemExchange::class.java)
                    if (item != null) {
                        Log.d(TAG, "Retrieved exchange item ${it.data}")
                        allItems.add(item)
                    }
                }
                callback.onComplete(allItems)
            } else {
                Log.w(TAG, "No such snapshot")
            }
        }
}
```

2.2.2. Cloud Storage

Cloud Storage din cadrul Firebase are rolul de a salva fotografii sau videoclipuri într-un mediu *cloud*. Similar cu Firestore, acest instrument este foarte ușor scalabil și oferă metode dedicate manipulării datelor media cu ajutorul librăriei Cloud Storage Android (“com.google.firebase:firebase-storage-ktx”) [12]. Organizarea datelor nu se face în colecții și documente, ci în simple foldere, având ca elemente de ierarhie conceptele de părinte și copil. Un exemplu de adăugare a unei imagini în Cloud Storage, având URI-ul acesteia, este ilustrat în codul de mai jos în care se întoarce cu ajutorul unui *callback* URL-ul imaginii adăugate.

```
private val firebaseStorage: FirebaseStorage = FirebaseStorage.getInstance()
private val storageReference: StorageReference = firebaseStorage.reference

private fun uploadItemPhoto(
    owner: String,
    itemId: String,
    imageUri: Uri,
    callback: OneParamCallback<String>
) {
    val itemPhotosReference = storageReference.child(owner).child(itemId)
    imageUri.let {
        val imageName = UUID.randomUUID().toString()

        itemPhotosReference.child(imageName).putFile(it).addOnCompleteListener { task ->

            if (task.isSuccessful) {
                itemPhotosReference.child(imageName).downloadUrl.addOnSuccessListener { url ->
                    val imageUrl = url.toString()
                    callback.onComplete(imageUrl)

                }.addOnFailureListener {
                    callback.onError(task.exception)
                }
            } else {
                callback.onError(task.exception)
            }
        }
    }
}
```

2.2.3. Firebase Authentication

Pentru gestionarea utilizatorilor în procesul de autentificare am folosit Firebase Authentication, un instrument ce permite stocarea utilizatorilor și autentificarea lor folosind email & parolă, *3rd party apps*, precum Google sau Facebook, SMS și altele. Permite de asemenea *multi-factor authentication* și opțiunea de schimbare a parolei. Protocoalele folosite în cadrul Firebase Authentication sunt OAuth 2.0 și OpenID Connect [27].

În cadrul “Swap”, am folosit opțiunile de autentificare prin email & parola și cea prin intermediul Google. Integrarea acestora a fost una ușoară, deoarece ca în cazurile menționate anterior, am folosit o librărie specială. În fragmentul de cod de mai jos e descris procesul de înregistrare al unui utilizator, folosind Firebase Authentication.

```
val auth = Firebase.auth

fun registerUser(email: String, pass: String, callback: OneParamCallback<FirebaseUser>) {
    auth.createUserWithEmailAndPassword(email, pass)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Log.w(TAG, "Register successful for user: $email")
                callback.onComplete(auth.currentUser)
            } else {
                Log.w(TAG, "Register failed for user: $email")
                callback.onError(task.exception)
            }
        }
    }
}
```

2.2.4. Cloud Functions & Firebase Cloud Messaging

Cloud Functions este un *tool* aparținând ecosistemului Firebase prin intermediul căruia se pot rula anumite acțiuni implementate în Node.js (în proiectul curent prin Javascript) la nivel de *server* atunci când are loc o modificare în baza de date *cloud* [25].

Firebase Cloud Messaging (FCM) este un instrument care permite programatorilor să trimită mesaje sau notificări utilizatorilor având în prealabil niște elemente identificatoare ale dispozitivelor pe care este instalată aplicația respectivă (“FCM tokens”) [28].

În cadrul “Swap”, prin îmbinarea celor 2 tehnologii prezentate am implementat notificările din cadrul aplicației. Acestea se împart în 2 categorii: notificări din cadrul *chat*-ului și cele ca urmare a adăugării unei postări de către alți utilizatori. În cazul amândurora a fost necesar să reținem *token*-urile dispozitivelor înregistrate, astfel că la *login* se salvează într-un document separat email-ul folosit și *token*-ul dispozitivului, iar la *logout* se șterge acea intrare. Este de menționat că *token*-urile se generează la instalarea aplicației însă sunt salvate local în Shared Preferences până la conectarea la un cont valid.

În cazul notificărilor pentru conversații, am setat un *trigger* pentru Firestore la orice modificare din colecția “Chats” pentru a monitoriza primirea mesajelor noi de către utilizatori: în urma acestuia se preia din baza de date numele expeditorului și folosind *token*-urile salvate se trimite o notificare destinatarului. La *click* pe această notificare se va deschide pagina conversațiilor, iar cea vizată va fi prima din listă.

Pentru notificările postărilor, *trigger*-urile sunt setate pe colecțiile “DonationItems” și “ExchangeItems” la adăugarea unui nou document. După verificarea datelor postării cu preferințele utilizatorilor conectați se va trimite o notificare ce conține un titlu sugestiv, denumirea și descrierea obiectului postat. Mai multe detalii despre preferințele notificărilor se pot găsi în capitolul 4.3.

2.3. Maps SDK & Places SDK

Pentru a putea integra harta live și opțiunea de *autocomplete* a locațiilor la căutare, am folosit SDK-uri destinate lucrului cu hărți, din platforma Google Play services.

Android Maps SDK permite programatorilor să integreze Google Maps în proiectele lor, oferind hărți în timp real, *markers* pentru marcarea locurilor pe hartă și modalități de a personaliza aspectul hărților [19]. Cu ajutorul acestui SDK, în aplicația “Swap”, utilizatorii pot vizualiza obiectele din jurul lor cu o mai mare ușurință, raportându-se la poziția acestora. Totodată, harta este folosită și în procesul de adăugare a unui obiect, pentru a alege locația de întâlnire. Integrarea în proiect a fost făcută folosind librăria “com.google.android.gms:play-services-maps:18.1.0” și un *FragmentContainerView* pentru *layout*.

Atunci când utilizatorii folosesc harta, ei au opțiunea de a căuta în bara de căutare o locație de interes, urmând ca pe hartă să se afișeze acea locație. Pentru a ușura căutarea utilizatorilor, am introdus și opțiunea de *autocomplete* a locațiilor la căutare. Acest lucru a fost posibil datorită **Android Places SDK** (“com.google.android.libraries.places:places:3.1.0”) [35]. Cu ajutorul *autocomplete widget*-ului, la căutare utilizatorii primesc sugestii de locații pe care le pot selecta, iar în urma acestei acțiuni în aplicație se salvează locația aleasă. Utilizatorii nu sunt limitați în a alege o opțiune dintre recomandări și pot continua să scrie în bară locația dorită.

2.4. Espresso

Espresso este un *framework open-source* creat de Google, folosit pentru a scrie teste automate pentru testarea de interfață (UI testing) [8]. *Test case*-urile se scriu în Java sau Kotlin, cele mai populare limbaje de programare pentru aplicațiile Android, făcând *framework*-ul ușor de folosit și de învățat. În plus, este construit plecând de la JUnit, un *framework* cunoscut pentru testarea unitară în Android.

Espresso are un mecanism de sincronizare care se asigură că activitățile/paginile sunt complet încărcate înainte de a rula testele, astfel încât simulează scenariile de folosire a aplicației din viața reală și evită erorile de tipul “*object not detected*”. Mai mult decât atât, Espresso închide activitățile testate după ce testele au luat sfârșit.

Un alt *framework* popular de testare automată a interfeței este Appium. Acesta este de asemenea *open-source*, însă comparativ cu Espresso, permite crearea de teste nu doar pentru aplicațiile native, ci și pentru aplicații web sau hibride [18]. Appium are o arhitectură de tip “client-server” ce folosește Selenium WebDriver API și este disponibil programatorilor sub formă de librării în Java, Ruby sau Python. Din acest punct de vedere, Appium oferă o gamă largă de opțiuni în ceea ce privește alegerea *stack*-ului, nefiind limitat la Java sau Kotlin, precum Espresso. Totuși, Appium este mai complicat de setat din cauza complexității configurării *server*-ului, în contrast cu Espresso care este deja integrat în Android Studio. Din punct de vedere al stabilității, Espresso este de preferat datorită mecanismului de sincronizare care nu conduce la *flaky tests* (teste care uneori

trec cu succes, alteori pică, într-o manieră nedeterministă, având o aceeași configurare). Viteza este de asemenea un considerent cheie în alegerea *framework*-ului Espresso.

Analizând aceste diferențe, am ales să folosesc Espresso la crearea aplicației “Swap” pentru a scrie teste automate cu scopul de a verifica interfața paginii de adăugare a unui obiect. Mai multe detalii despre testarea propriu-zisă pot fi găsite în capitolul 5, destinat acestui subiect.

2.5. Alte librării open-source

Gson este o librărie *open-source* creată de Google cu scopul de a transforma obiecte Java/Kotlin în reprezentări JSON sau invers [31]. Folosind metodele `toJson()` și `fromJson()` aplicate unui obiect din Kotlin am folosit această librărie (“com.google.code.gson:gson:2.9.0”) pentru a salva local în Shared Preferences datele referitoare la utilizatorul conectat în acel moment.

Pentru încărcarea fotografiilor în elementele de UI (de exemplu `ImageView`) am folosit **Glide** (“com.github.bumptech.glide:glide:3.8.0”), o librărie *open-source* ce are scopul de a manipula imaginile și de a gestiona memoria folosită pentru ele [29].

O altă librărie *open-source* utilizată aparține unui membru din comunitatea Android, **ImageSlideShow** (“com.github.denzcoskun:ImageSlideShow:0.1.0”), și are rolul de a prezenta imaginile sub forma unui carusel derulabil [32]. O altă funcționalitate a librăriei este de a seta o imagine de tip *placeholder* până când imaginile se încarcă sau o imagine de eroare când acestea nu s-au încărcat cu succes. Am folosit acest element de frontend pentru paginile care conțin imagini ale produselor postate.

CircularProgressButton este o librărie *open-source* creată tot de un membru din comunitatea Android [22]. Prin intermediul acesteia am afișat un buton care se transformă într-un *progress bar* la apăsarea lui și care transmite utilizatorului ideea că datele lui sunt procesate (la autentificare, adăugare a unei postări, raportare a unei postări și trimitere de *feedback*). Versiunea folosită în proiect este 1.0.11.

3. Arhitectura aplicației

3.1. Principii

Aplicațiile mobile sunt alcătuite din diverse componente (activități, fragmente, servicii etc.) care interacționează între ele. Modul în care se produce interacțiunea este la decizia programatorului, deoarece o aplicație poate fi construită în diverse moduri, însă pentru a fi scalabilă, ușor de testat și de întreținut, este recomandat să se urmărească niște principii arhitecturale [21].

Unul dintre cele mai importante principii este “*separation of concerns*”. Acesta atestă faptul că este necesar ca partea de logică a aplicației să fie separată de partea de UI (activități, fragmente), al cărei scop nu ar trebui să vizeze decât interacțiunea cu elementele vizuale (butoane, câmpuri de text etc.), afișarea datelor în UI și interferența cu sistemul de operare (precum lucrul cu camera). Deoarece activitățile și fragmentele au cicluri de viață controlate de sistemul de operare, atunci când condițiile dispozitivului sunt neprielnice (de exemplu puțină memorie disponibilă), acestea sunt distruse și, nerespectând principiul “*separation of concerns*”, datele stocate în ele se pierd.

“*Single source of truth*” este un principiu ce face referire la modul în care sunt gestionate datele în cadrul aplicației. Este indicat să existe un singur *owner* al datelor, care să le protejeze, și care să interacționeze cu restul aplicației prin metode speciale de expunere a datelor sau de modificare a lor. *Owner*-ul este de cele mai multe ori baza de date, într-o aplicație *offline-first*, sau un ViewModel în celelalte cazuri.

Un alt principiu arhitectural recomandat este “*unidirectional data flow*” și se referă la direcția în care datele parcurg componentele aplicației. Spre exemplu, atunci când utilizatorul apasă butonul de “Salvează”, se produce un *trigger* în activitatea/fragmentul aferent, iar acesta este transmis părții din sistem responsabilă cu logica. Acolo se produce o modificare a datelor (fie prin a face o acțiune la nivelul bazei de date sau nu), iar rezultatul se întoarce către partea de UI, care este actualizată.

3.2. Model View ViewModel

“**Model View ViewModel**” este un șablon arhitectural care urmărește principiile prezentate anterior, fiind alcătuit din 3 elemente principale [14]:

1. **View**: partea vizuală la care are acces în mod direct utilizatorul; sunt incluse aici elementele de interfață grafică, activitățile și fragmentele.
2. **ViewModel**: liantul dintre partea de *frontend* și date (View și Model), altfel spus locul unde se aplică logica proiectului. Are ciclu de viață independent față de fragmente și are rol de strat de *caching*.
3. **Model**: elementul responsabil cu gestionarea în mod direct a datelor, prin intermediul *repositories* care preiau datele dintr-o bază de date locală sau *cloud* prin operații CRUD. În “Swap”, fiecare *repository* este responsabil de o colecție din Firestore, sau de fișierele din Cloud Storage.

Aplicația “Swap” este construită folosind acest șablon (figura 3.1), iar interacțiunile dintre elemente au loc cu ajutorul metodelor cu scop unic, *callback*-urilor și componentelor *lifecycle-aware* precum LiveData.

LiveData este o clasă specială care reține tipuri de date generale (String) sau *custom* (obiecte aparținând unor clase create de programator) ce are rolul de a notifica fragmentele sau activitățile când apar schimbări de date, prin intermediul observatorilor. În plus, este *lifecycle-aware*, însemnând că notifică fragmentele/activitățile aferente doar dacă acestea încă sunt “în viață”, adică nu au fost distruse de sistemul de operare sau de utilizator [34].

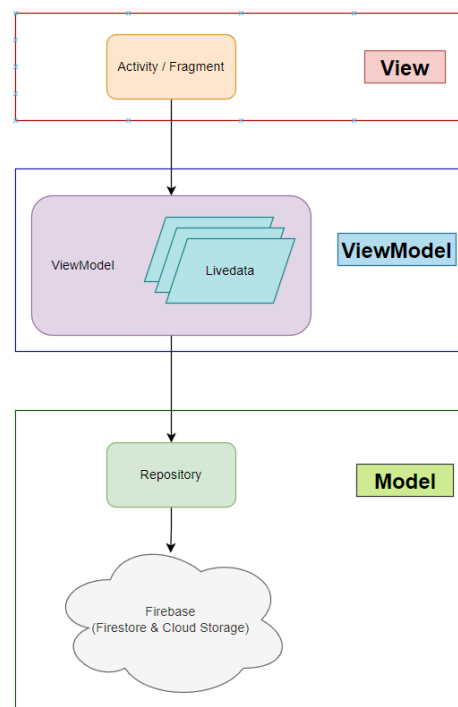


Figura 3.1 – Șablonul MVVM aplicat în Swap
(figură inspirată după cea din [14])

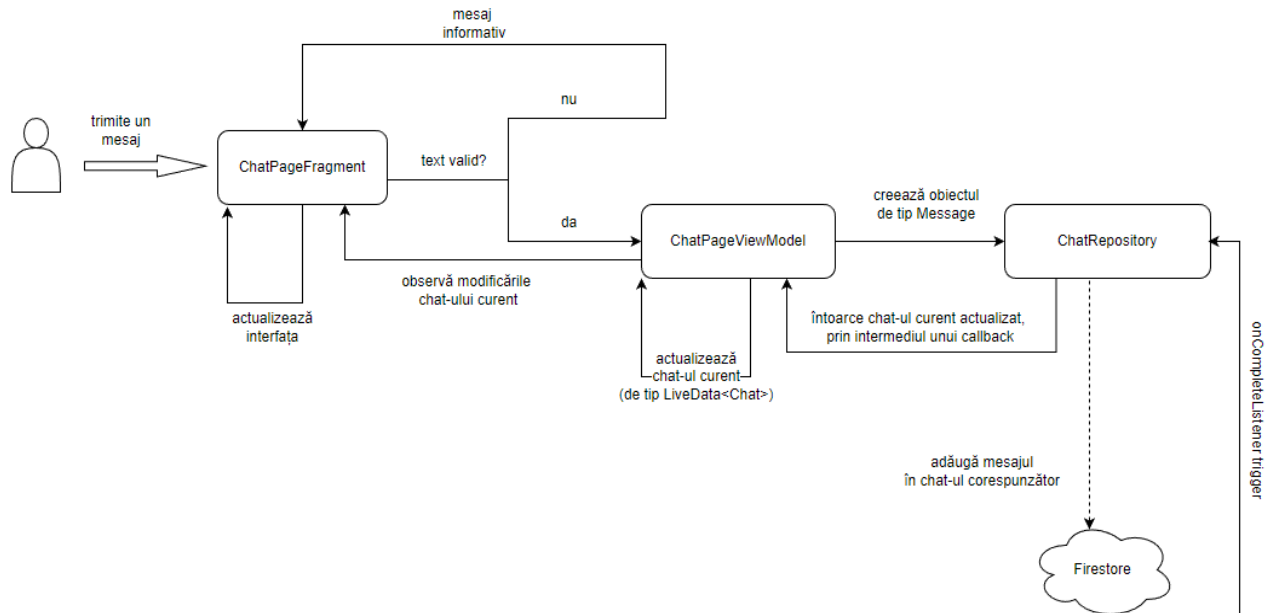


Figura 3.2 – Flow-ul datelor la trimiterea unui mesaj

Pentru a ilustra *flow*-ul datelor prin cele 3 componente cu ajutorul elementelor de tip LiveData voi prezenta un exemplu (descriș schematic și în figura 3.2): să presupunem că utilizatorul dorește să trimită un mesaj nou în *chat*. După tastarea acestuia, el apasă butonul de trimitere ce declanșează un *trigger* în *listener*-ul de tip “onClickListener” al butonului, din fragmentul ChatPageFragment.

```

@Parcelize
data class Message(
    val date: Timestamp = Timestamp(0, 0),
    val photoUri: String? = null,
    val proposalId: String? = null,
    val senderEmail: String = "",
    val text: String? = null
): Parcelable
  
```

Dacă acel câmp de text este gol, fragmentul va afișa un mesaj informativ și *flow*-ul se va opri până la următoarea acțiune a utilizatorului. Dacă nu, se va apela metoda addMessageToChat din cadrul ViewModel-ului aferent fragmentului în care se creează un obiect de tip Message (modelul său e descriș în fragmentul de cod de mai sus).

ViewModel-ul va apela o metodă din *repository*-ul destinat mesajelor, ChatRepository, pentru a adăuga mesajul în *chat*-ul corespunzător din baza de date din Firestore (cu id-ul compus din

email-urile celor 2 participanți la conversație). La finalizarea adăugării în baza de date, se întoarce un *callback* (cu parametru *chat*-ul actualizat) din funcția apelată în *repository* către funcția *addMessageToChat*, iar la primirea acestuia se atribuie obiectului de tip *LiveData* valoarea primită. În momentul schimbării datelor din obiectul *LiveData* în *ViewModel* se declanșează un observator setat către acel obiect în fragmentul de la care se plecase. În urma acestui proces, se actualizează corespunzător elementele de UI din fragment.

3.3. Navigarea între fragmente

Proiectul este construit și după modelul *Single-Activity Architecture*, având o singură activitate și mai multe fragmente, pentru a ușura interacțiunile dintre ele. Activitatea este echivalentul unui *container*, iar fiecare fragment este destinat unui ecran (unei pagini) cu scop bine definit. Pe lângă tranzacțiile efective dintre pagini, acest șablon vine și cu avantajul transmiterii mai ușoare de date între fragmente, prin intermediul unor clase *ViewModel* comune sau prin *Navigation component* [4].

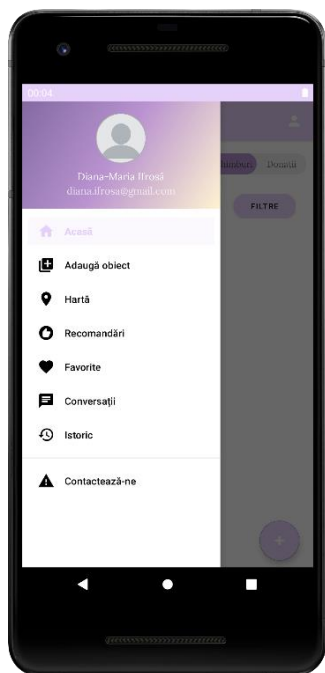


Figura 3.3 – Meniul principal de tip *drawer*



Figura 3.4 – Navigare prin butonul săgeata înapoi

Navigarea în cadrul aplicației se poate face fie prin meniul de tip *drawer menu* (figura 3.3), fie prin apăsarea unor elemente din UI (spre exemplu, pentru navigarea către pagina unui produs, se apasă pe *card*-ul acesteia din listă, iar pentru a naviga înapoi se apasă săgeata din partea de sus a paginii ca în figura 3.4, sau butonul de *back* fizic al telefonului).

Prin intermediul Navigation component din Android Jetpack, am creat un graf de navigare (schițat în figura A.1.) cu ajutorul căruia am gestionat navigarea între fragmente, setând parametrii fiecărui fragment în parte, acolo unde este cazul [13]. Adiacent celui schițat în figura precizată este și cel din figura 3.5, pentru profilul personal.

Un exemplu în acest sens este fragmentul paginii unui obiect postat, care primește sub formă de parametri la navigare obiectul ale cărui detalii trebuie afișate. Acest lucru a fost posibil datorită *plugin*-ului SafeArgs din AndroidX (“androidx.navigation.safeargs”). O altă funcționalitate utilă a componentei de navigare este personalizarea comportamentului butonului de *back*, ca în cazul fragmentului de adăugare obiect: după adăugarea cu succes a unui obiect, utilizatorul este redirecționat către pagina acestuia, iar la apăsarea butonului *back*, el se întoarce la pagina de dinaintea celei de adăugare obiect, pentru că cea de adăugare în sine nu ar mai avea niciun scop în navigare.

Pentru utilizatorii care nu sunt conectați la un cont (vizitatori) am restricționat accesul către pagini (figura 3.6) prin intermediul activității principale, lăsând vizibile doar opțiunile de Acasă, Adăugare obiect și Hartă, ultimele 2 fiind doar sugestive, întrucât apăsând pe ele, utilizatorul va fi redirecționat spre autentificare. Graful de navigare în acest caz este schițat în figura 3.7.

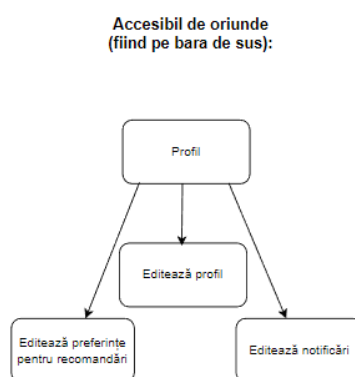


Figura 3.5 – Graf de navigare al profilului personal

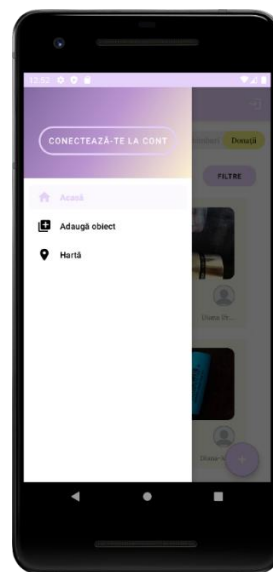


Figura 3.6 – Meniul aplicației pentru un vizitator

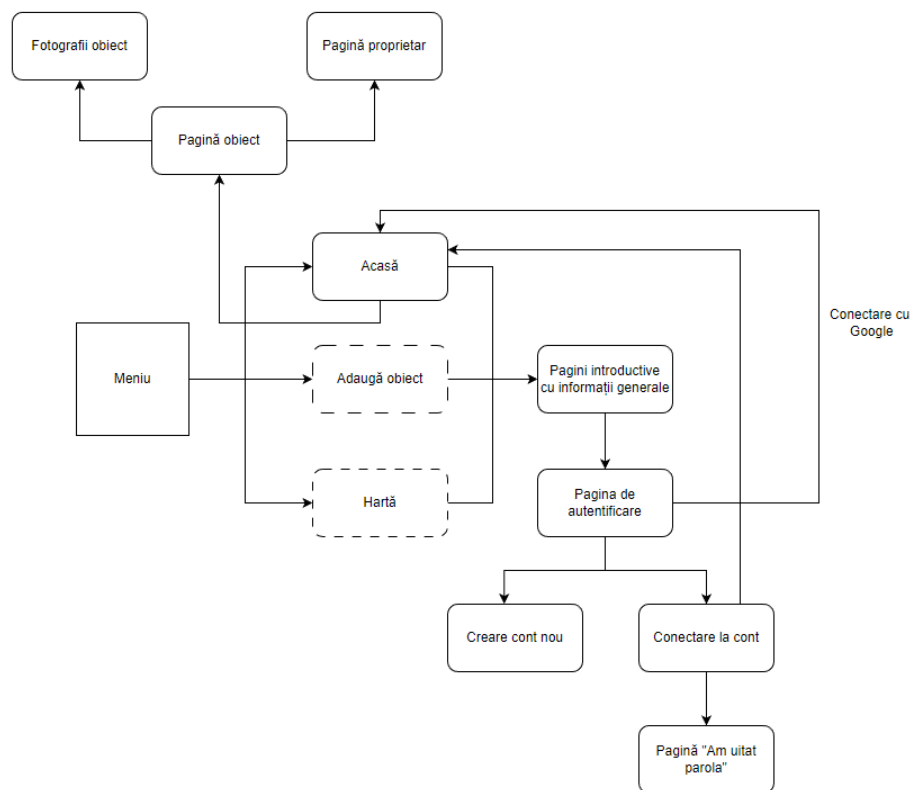


Figura 3.7 – Navigarea în cadrul aplicației pentru un vizitator

3.4. Structura bazei de date

Baza de date a aplicației este ținută integral pe *cloud*, prin intermediul Firestore și Cloud Storage din intermediul Firebase. Deoarece Firestore menține datele în colecții și documente sub forma NoSQL, am ales să folosesc câte o colecție pentru fiecare element distinct din aplicație. Colecțiile rezultate sunt după cum urmează:

1. **Users:** conține câte un document pentru fiecare utilizator înregistrat, având ca id email-ul acestuia. Documentele conțin numele, 2 liste cu id-urile postărilor favorite (pentru schimb și donații), o listă cu id-urile *chat*-urilor din care face parte utilizatorul, un link spre poza de profil salvată în Cloud Storage, preferințele pentru recomandări și notificări (liste cu categorii, orașe preferate, cuvinte, utilizatori de interes).
2. **DonationItems:** conține documente pentru fiecare obiect postat cu scopul donației, având id-ul obiectului (generat aleator), denumire, descriere, categorie, condiție (dacă există),

- oraș, locație exactă (latitudine și longitudine), email-ul proprietarului, o listă cu link-uri către fotografiile salvate în Cloud Storage, data postării, anul fabricării (dacă există), id-ul evenimentului din istoric (dacă există).
3. **ExchangeItems**: alcătuită din documente pentru fiecare obiect postat cu scopul schimbului, având în plus față de colecția anterioară și o listă de categorii preferate pentru schimb (dacă există).
 4. **History**: conține documente cu id-uri generate aleator pentru fiecare eveniment încheiat (schimb sau donație), iar acestea sunt alcătuite din dată, id-ul primului obiect, id-ul celui de-al doilea obiect (doar în cazul schimbului), email-ul celui ce a primit donația (dacă este cazul).
 5. **Chats**: are în componență documente pentru fiecare conversație distinctă, având ca id-uri concatenarea email-urilor celor 2 utilizatori, separate prin spațiu. Fiecare document conține o listă cu mesajele din conversație, cu dată, text (dacă e cazul), link din Cloud Storage către fotografia trimisă (dacă e cazul), id-ul propunerii trimise (dacă e cazul), email-ul utilizatorului ce a trimis mesajul.
 6. **Proposals**: alcătuită din documente cu id-uri generate aleator, care conțin id-ul primul obiect (cel care este dorit de utilizatorul care face propunerea), id-ul celui de-al doilea obiect (doar în cazul schimbului), email-urile celor 2 utilizatori implicați, 2 câmpuri pentru confirmări (de tip boolean).
 7. **Reports**: o colecție care reține raportările postărilor făcute de utilizatori. Fiecare raport este sub forma unui document care are un id generat aleator, email-ul celui ce a raportat, dată, id-ul postării raportate, un text explicativ și un câmp boolean pentru a identifica postarea ca fiind una de donație sau de schimb (completat automat).
 8. **Feedback**: conține documente fiecare reprezentând o problemă/sugestie semnalată de utilizatori. Câmpurile documentelor sunt id generat aleator, email-ul utilizatorului ce a trimis *feedback*-ul, dată și un mesaj explicativ.
 9. **Mail**: în această colecție se salvează statusul email-urilor trimise la înregistrarea unui nou utilizator, fiind populată automat de extensia din Firebase “Trigger Email from Firestore”.
 10. **CategoriesObjects**: alcătuită din documente pentru fiecare categorie de obiecte din aplicație, iar fiecare document reține numărul de obiecte disponibile (care nu au fost încă schimbate sau donate) și numărul total de obiecte care au fost postate în acea categorie.

Este o colecție utilă mai ales pentru administratori pentru a avea date generale despre aplicație.

11. **UserTokens**: reține email-urile și *token*-urile FCM corespunzătoare pentru utilizatorii conectați.

Fotografiile sunt salvate folosind Cloud Storage, care nu este structurat pe documente și colecții, ci pe foldere. Fiecărui folder îi corespunde un utilizator sau un *chat*, având ca denumiri email-ul utilizatorului sau id-ul *chat*-ului. Structura poate fi observată și în figura 3.8. Tot aici sunt salvate și logo-ul aplicației și o fotografie de profil de tip *placeholder*, pentru situațiile când utilizatorii nu își setează o altă fotografie.

Folderurile utilizatorilor sunt create automat la înregistrare și conțin pe lângă fotografia de profil și câte un alt folder pentru fiecare obiect postat, în care se află fotografiile acestuia (figura 3.9).

În cazul folderelor pentru conversații, sunt salvate pozele trimise în cadrul acestora.

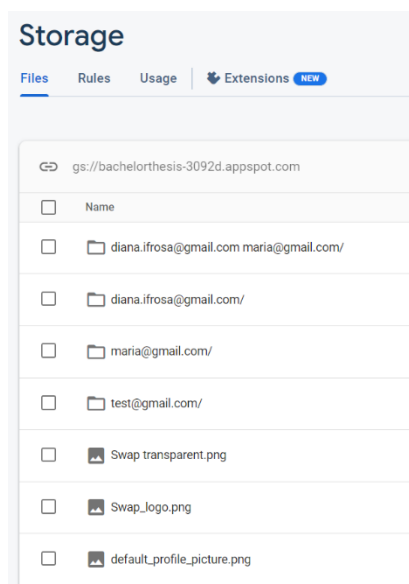


Figura 3.8 – Structura din Cloud Storage

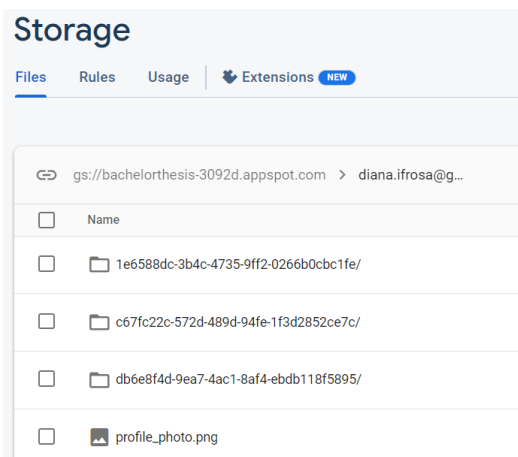


Figura 3.9 – Structura folder-ului unui utilizator din Cloud Storage

4. Prezentarea aplicației

4.1. Identitatea vizuală

Prima interacțiune cu o aplicație are o însemnătate aparte, deoarece pe lângă funcționalitățile propriu-zise, utilizatorul descoperă elementele ce compun identitatea vizuală a aplicației. Printre acestea se numără numele, logo-ul, culorile folosite, font-urile textelor, cât și imaginile/simbolurile din componența aplicației [6].

Rolul lor este de a ajuta utilizatorul să identifice mai ușor o aplicație, să înțeleagă mesajul pe care aceasta îl promovează și totodată să creeze o experiență de navigare memorabilă.

Aplicația prezentată în această lucrare se numește, așa cum am mai menționat, “Swap” – o denumire ce relevă scopul ei și motivația din spatele creării acesteia, schimbul de obiecte. Logo-ul cuprinde denumirea aplicației și se evidențiază prin săgețile din cadrul literei S, așa cum se poate observa în figura 4.1.



Figura 4.1 – Logo-ul aplicației



Figura 4.2 – Logo-ul aplicației în meniul telefonului

În meniul telefonului, logo-ul aplicației este format din prima literă a numelui (figura 4.2), urmând modelul clasic de *branding* al aplicațiilor mari, precum Facebook sau Google.

Mai mult decât atât, logo-ul are culorile principale ale aplicației, mov și galben, care sunt folosite pentru a diferenția mai ușor postările cu obiecte pentru schimb de cele pentru donație (mov = schimb, galben = donație). Pe lângă aceste culori și variații ale lor (figura 4.3), paleta aleasă include alb, negru și alte culori precum roșu, albastru în nuanțe pastelate. Font-ul folosit pentru crearea logo-ului este League Spartan, regăsit și pe bara de sus unde sunt titlurile paginilor, pentru a păstra consistența. Celelalte font-uri folosite în cadrul aplicației sunt IstokWeb și Lora. Toate fonturile folosite au fost alese astfel încât să cuprindă și diacriticele din limba română.



Figura 4.3 – Culori folosite în “Swap”

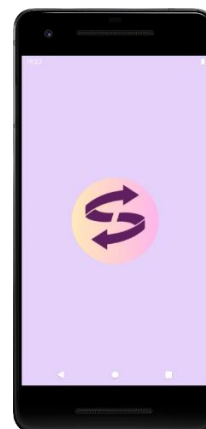


Figura 4.4 – Splash screen la deschiderea aplicației

La deschiderea aplicației, se lansează un *splash screen* ce conține logo-ul (figura 4.4), care maschează elegant inițializarea datelor și a componentelor interne, iar la finalizarea procesului *splash screen*-ul dispare, lăsând vizibilă pe ecran pagina principală (*dashboard*). Prima interacțiune cu aplicația prin intermediul acestui tip de ecran este importantă, deoarece prezența unui ecran alb până la inițializarea componentelor interne îi poate sugera în mod greșit utilizatorului că s-a produs o eroare sau îl poate face să își piardă răbdarea. Având un element vizual pe ecran, el își poate îndrepta atenția către acesta cât timp aplicația se pregătește de lansare.

Fotografiile folosite au fost preluate de pe www.freepik.com.

4.2. Autentificarea

“Swap” permite utilizatorilor să vizualizeze postările fără a fi autentificați, însă pentru a folosi celelalte funcționalități precum adăugarea unei postări, *chat*-ul sau salvarea la favorite, ei trebuie să își creeze un cont. Pentru a naviga spre autentificare, este suficient să fie apăsat simbolul din partea dreapta-sus a barei principale. O alternativă a acestuia este butonul “Conectează-te la cont” din cadrul meniului, de sus.

Înainte de a ajunge la paginile dedicate autentificării, utilizatorii trec printr-o serie de ecrane introductive (figura 4.5) în care le sunt prezentate pe scurt funcționalitățile de adăugare obiect, recomandări și hartă.

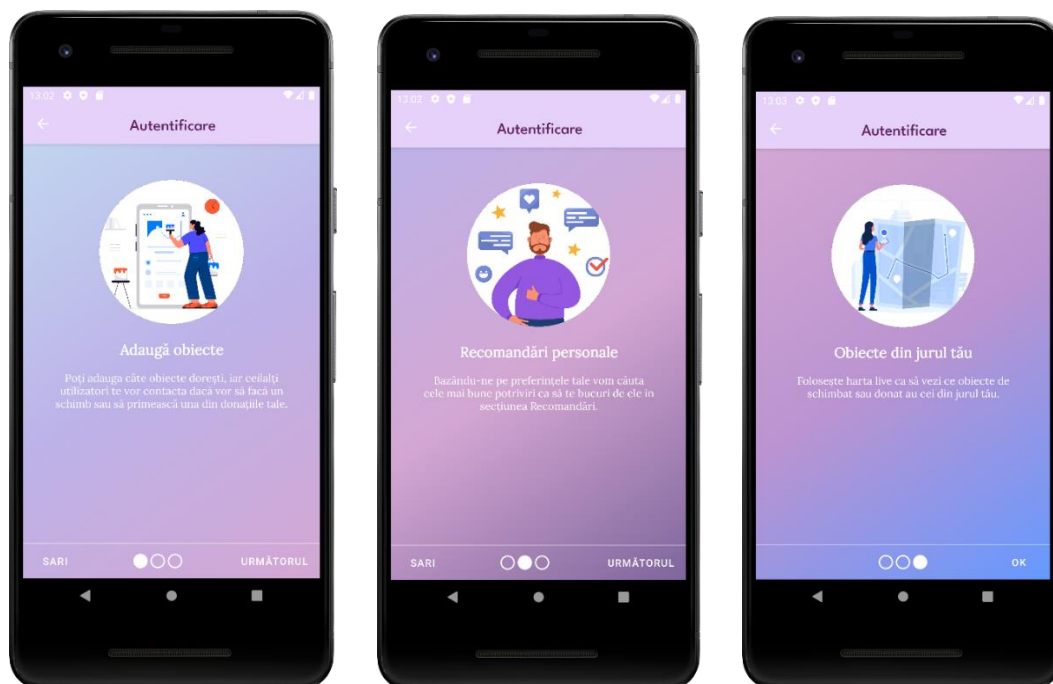


Figura 4.5 – Ecrane introductive

Desigur, ei pot alege să sară peste vizualizarea acestor ecrane, apăsând butonul “Sari” din partea de jos. Pentru a vizualiza conținutul lor, se poate avansa fie apăsând butonul “Următorul”, fie prin glisarea ecranului.

După ce acestea au fost parcurse, utilizatorii pot alege modul în care să continue autentificarea: prin crearea unui cont dacă nu au deja unul, prin conectarea la un cont existent sau prin conectarea prin intermediul Google (figura 4.6).

Dacă opțiunea aleasă este de a continua cu Google, după ce își aleg contul dorit, vor fi redirecționați spre pagina principală (Acasă).

Pe de altă parte, dacă doresc să își creeze un cont (figura 4.7), va fi necesară introducerea unor date: o adresă de email, numele complet și o parolă (de minimum 8 caractere dintre care măcar o literă mare, o literă mică și o cifră).



Figura 4.6 – Pagina de autentificare

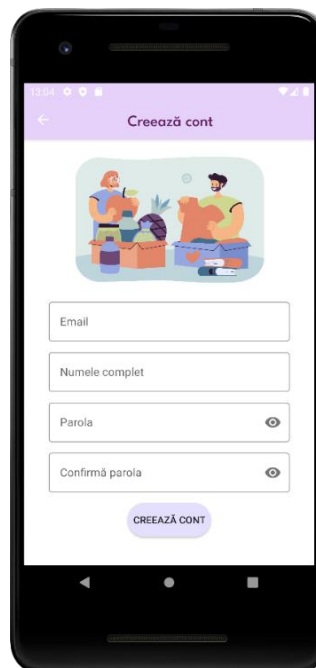


Figura 4.7 – Pagina de creare cont

După crearea contului, vor trebui să se conecteze cu email-ul și parola setate, în pagina de *login*. În plus, la crearea contului aceștia primesc pe adresa de email înregistrată un email de bun-venit, precum în figura 4.8. Din cadrul acestuia ei pot naviga direct către aplicație (datorită *deep links*), făcând *click* pe butonul din email, dacă acesta este deschis pe telefon. Dacă email-ul este deschis pe un laptop sau un calculator, vor fi redirecționați spre site-ul Google Play, ca o sugestie că aplicația s-ar putea descărca de acolo (momentan nu este postată).

Pentru conectarea cu un cont deja existent, sunt necesare email-ul și parola (figura 4.9), utilizatorii având de asemenea posibilitatea să își reseteze parola dacă au uitat-o, alegând opțiunea aceasta din pagină. Ei vor primi un email cu un link pentru setarea noii parole.

După conectare, redirecționarea se va face tot către pagina de Acasă.

Atât la *login* cât și la *register*, utilizatorii sunt atenționați dacă nu completează câmpurile corespunzător (figura 4.10). Spre exemplu, la crearea contului, mesajele de atenționare apar dacă email-ul nu este valid, numele conține doar numele de familie sau doar prenumele, parola nu este conform cerințelor sau nu corespunde cu parola confirmată.



Figura 4.8 – Email de bun-venit

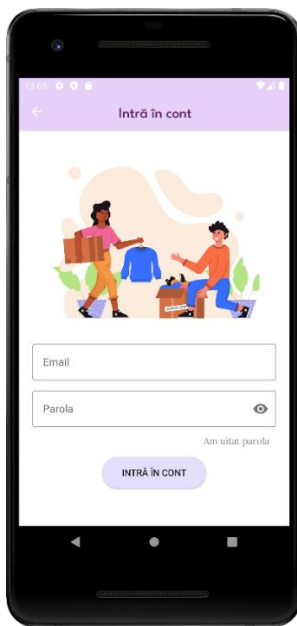


Figura 4.9 – Pagina de login

 A form showing validation errors for the login fields.
 - Email field: 'myemail' with a red 'X' icon and error message 'Adresa de email este invalidă'.
 - Numele complet field: 'Elena' with a red 'X' icon and error message 'Numele complet este obligatoriu'.
 - Parola field: 'pass' with a red 'X' icon and error message 'Ar trebui să aibă măcar 8 caractere, o literă mare, o literă mică și o cifră.'
 - Confirmă parola field: empty with a red 'X' icon and error message 'Confirmarea parolei e obligatorie'.

Figura 4.10 – Validările câmpurilor la autentificare

4.3. Profilul personal

Iconița din partea dreapta-sus conduce spre profilul personal al utilizatorului conectat, ilustrat în figura 4.11. Aceeași pagină se poate accesa și prin apăsarea secțiunii din partea de sus a meniului principal, ca în figura 3.3.

În cadrul profilului sunt afișate fotografia de profil, numele și email-ul folosit la conectare, cât și alte opțiuni de configurare a profilului, prezentate în subcapitolele ce urmează. În plus, din cadrul aceleiași pagini utilizatorul se poate deconecta de la contul curent prin apăsarea butonului aferent din partea de jos.

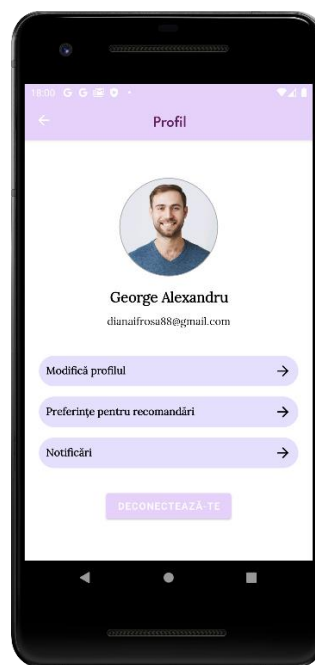


Figura 4.11 – Profilul personal

4.3.1. Editarea profilului

Prima opțiune de configurare a profilului personal include schimbarea fotografiei de profil prin alegerea alteia din galeria telefonului sau schimbarea parolei, ca în figura 4.12. Este de menționat că, atunci când un utilizator se conectează folosind Google, nu își va putea schimba parola din considerente de securitate, astfel că cele 2 câmpuri destinate acestui scop vor fi înlocuite de un mesaj explicativ.

Schimbarea parolei este însoțită de mesaje de validare cu rol explicativ dacă parola veche nu este validă sau corectă sau dacă parola nouă nu este validă. Condițiile de validitate sunt minimum 8 caractere dintre care măcar o literă mare, o literă mică și o cifră.

Schimbările efectuate sunt salvate după apăsarea butonului “OK” din partea de sus a paginii.

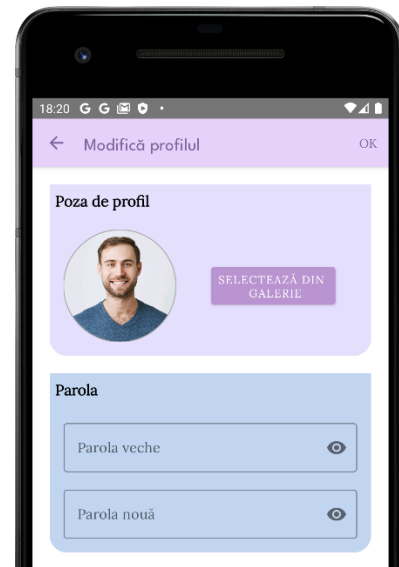


Figura 4.12 – Modificarea fotografiei de profil și a parolei

4.3.2. Preferințele pentru recomandări

Tot în cadrul profilului se pot seta și preferințele în baza cărora se aleg recomandările din pagina corespunzătoare. Categoriile ce se pot personaliza după interesul utilizatorului sunt:

1. **Cuvinte:** aici se introduc, prin intermediul unei liste, cuvinte sau sintagme pe care utilizatorul vrea să le regăsească în titlul sau descrierea unei postări. Prin intermediul simbolului “+” se adaugă o nouă intrare în listă, iar prin simbolul “X” se șterge.
2. **Proprietari:** folosind de asemenea o listă, utilizatorul poate adăuga alți utilizatori ale căror postări ar vrea să le vadă printre recomandări. Bara de căutare include funcționalitatea de *autocomplete*, astfel că poate alege din lista utilizatorilor deja existenți sau poate introduce un nume de sine stătător.
3. **Categorii:** o mulțime prestabilită de categorii prezentată prin *checkboxes* oferă posibilitatea de a marca anumite categorii de interes.

4. **Orașe:** similar proprietarilor, utilizatorul își poate alege orașele preferate folosind bara de căutare și opțiunile de *autocomplete* puse la dispoziție. De asemenea, nu este limitat de sugestiile afișate, iar adăugarea și ștergerea obiectelor din listă se fac folosind aceleași butoane amintite.
5. **Preferințe de schimb:** dacă utilizatorul plănuiește să facă schimburi cu obiectele sale, această secțiune îi vine în ajutor prin a seta categoriile de interes pentru schimb, prin intermediul *checkboxes*.

Aspectul paginii este ilustrat în figura 4.13, fiind unul colorat și intuitiv datorită culorilor diferite ale categoriilor și mesajelor explicative de la începutul lor.

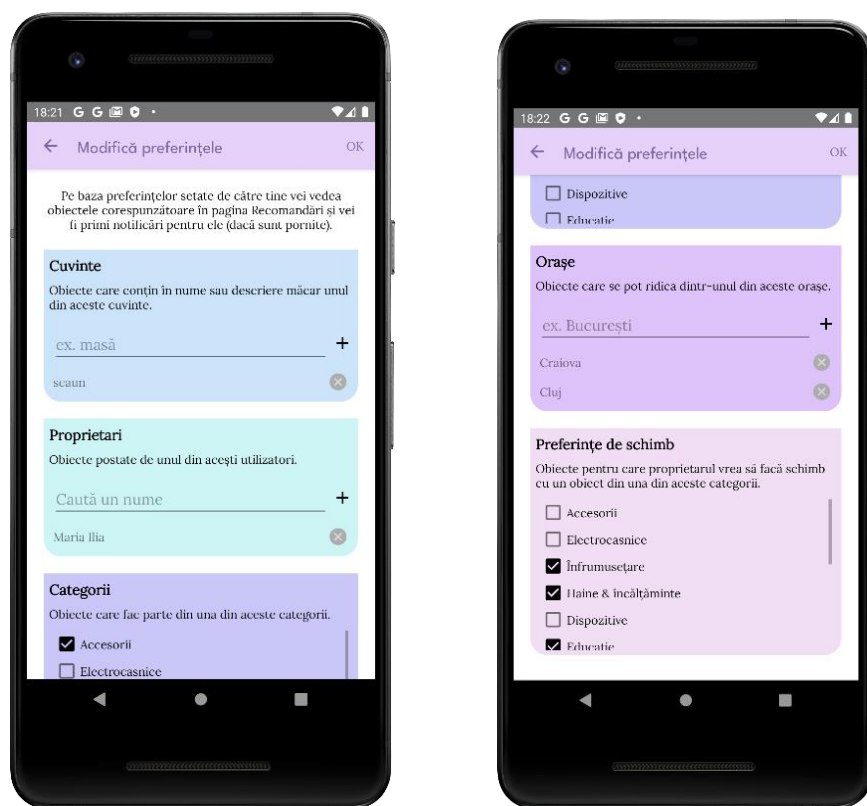


Figura 4.13 – Setarea preferințelor pentru recomandări

4.3.3. Notificările

Așa cum am amintit într-un capitol anterior, notificările sunt împărțite în 2 categorii: pentru mesajele din conversații și pentru postările noi. Utilizatorii pot seta preferințele pentru cele din urmă în cadrul profilului, accesând secțiunea “Notificări”. Opțiunile puse la dispoziție sunt: să primească notificări când se postează orice obiect, când se postează un obiect în conformitate cu preferințele setate pentru recomandări sau să nu primească deloc. Aspectul paginii (figura 4.14) este unul simplu, cu *radio buttons* pentru alegerea unei opțiuni. Salvarea opțiunii alese se face prin apăsarea butonului “OK”.

Un exemplu de notificare pentru o postare nouă de interes este ilustrat în figura 4.15, aceasta conținând logo-ul și numele aplicației, cât și datele referitoare la postare (denumire obiect și descriere).



Figura 4.15 – Notificare la adăugarea unei postări de interes

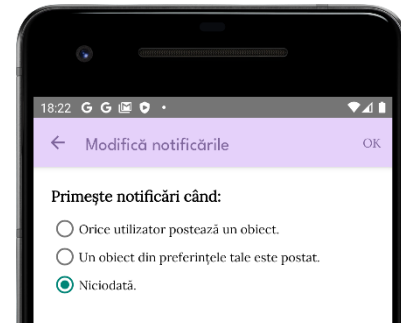


Figura 4.14 – Setarea preferințelor pentru notificări

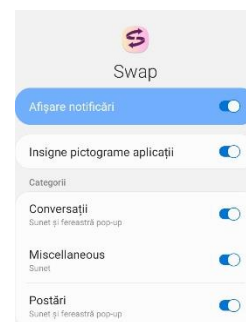


Figura 4.16 – Modificarea notificărilor din setările telefonului

Notificările pentru conversații pot fi activate sau dezactivate din setările telefonului. De asemenea, și cele pentru postări pot fi dezactivate și de acolo. Prin crearea a 2 canale diferite de notificări (*notification channels*), în cadrul setărilor telefonului, utilizatorii pot modifica după preferințe cele 2 categorii de notificări, precum în figura 4.16.

4.4. Pagina principală (Acasă)

Această pagină este prima care se afișează la deschiderea “Swap”. Rolul ei este de a vizualiza postările (toate în cazul vizitatorilor și doar cele care nu sunt proprii în cazul utilizatorilor conectați). Principala opțiune de filtrare este la nivel de scop al postărilor (schimburi sau donații) și este implementată folosind un buton de tip *toggle*, din partea dreaptă-sus (figura 4.17).

Bara de căutare are rolul de a reduce postările afișate doar la cele care conțin în denumire sau descriere cuvintele specificate. De asemenea, butoanele “Filtre” și “Sortează” au rolul de a ajuta utilizatorul să găsească postările de interes mai rapid. Opțiunile lor sunt ilustrate în figura 4.18. Sortarea se poate face după dată (ascendent sau descendent) sau după denumire (ascendent sau descendent). Filtrarea poate fi făcută după oraș (având sugestii de orașe dintre cele ale obiectelor postate) sau după categoriile predefinite de obiecte (accesorii, electrocasnice, înfrumusețare, haine & îmbrăcăminte, dispozitive, educație, mâncare & băuturi, mobilă, jocuri, grădină, decorațiuni, bijuterii, medical). Atunci când se modifică textul din bara de căutare, opțiunile setate la sortare și filtre se resetează automat, pentru a facilita o nouă căutare.

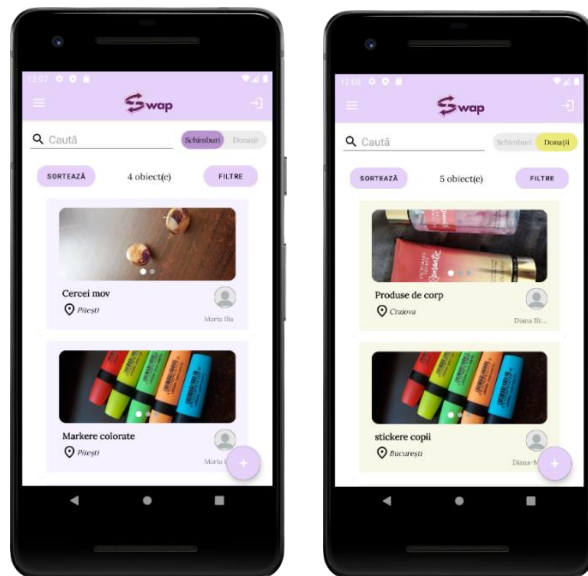


Figura 4.17– Filtrarea principală din pagina acasă

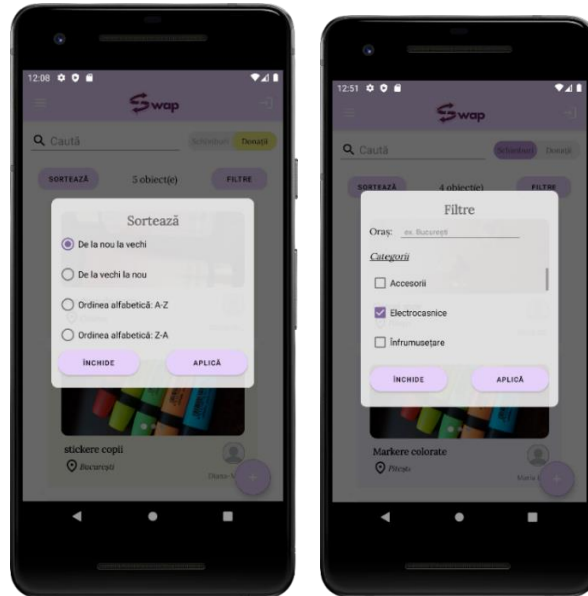


Figura 4.18 – Opțiunile de sortare și filtrare

4.5. Adăugarea unui obiect

Această funcționalitate este accesibilă din meniu sau prin apăsarea butonului *floating* “+” din partea de jos a paginii principale. Pentru a adăuga o postare sunt necesare următoarele date: denumirea obiectului (minim 3 caractere), descrierea lui (minim 5 caractere), scopul (schimb sau donație), categoria (dintre cele menționate anterior), minim 2 fotografii, locația (aleasă prin intermediul unei hărți). Alte detalii opționale sunt categoriile de obiecte cu care utilizatorul vrea să facă schimb (dacă este cazul), data fabricării, starea obiectului (nou/ca nou, folosit, pentru componente/ nu funcționează). Interfața paginii de adăugare obiect este prezentată în figura 4.19.

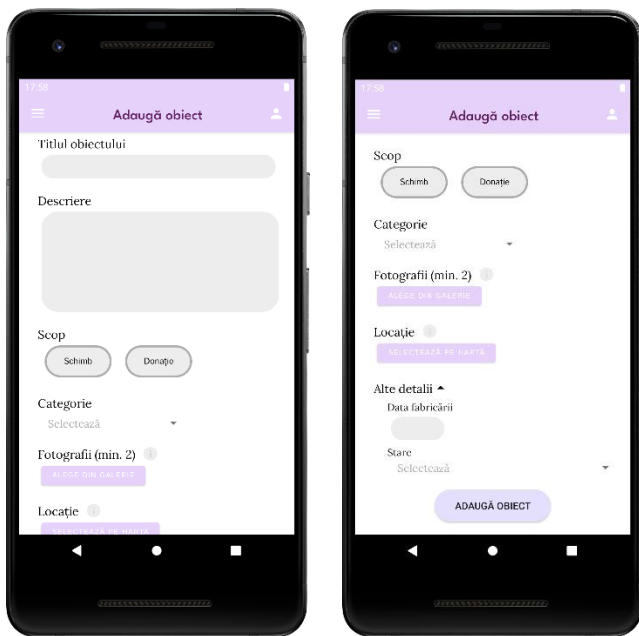


Figura 4.19 – Pagina de adăugare obiect

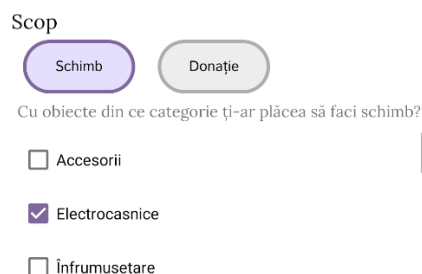


Figura 4.20 – Secțiune dedicată preferințelor de schimb

La selectarea scopului, dacă este schimb, se va afișa o secțiune specială pentru a alege categoriile preferate de schimb, dacă este cazul, precum în figura 4.20.

În plus, și această pagină prezintă diverse mesaje de atenționare pentru a valida datele introduse, fiind de culoare roșie, în dreptul câmpului vizat. În figura 4.21 este prezentată și harta prin intermediul căreia se alege locația, cu sugestii de locații la nivelul barei de căutare. Salvarea locației selectate se face prin apăsarea butonului “OK”.

O altă funcționalitate a paginii este de a prezenta informații referitoare la caracterul obligatoriu al câmpurilor locație și fotografii, cât și indicații de urmat, prin apăsarea iconițelor din dreptul acestora, ca în figura 4.22. După adăugarea cu succes a unui obiect, utilizatorii vor fi redirecționați către pagina acestuia.

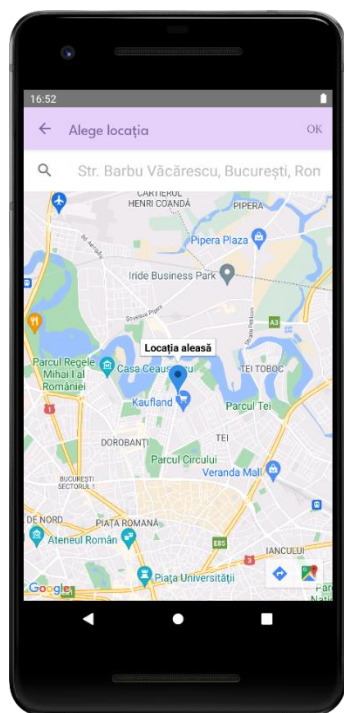


Figura 4.21 – Pagina de alegere a locației

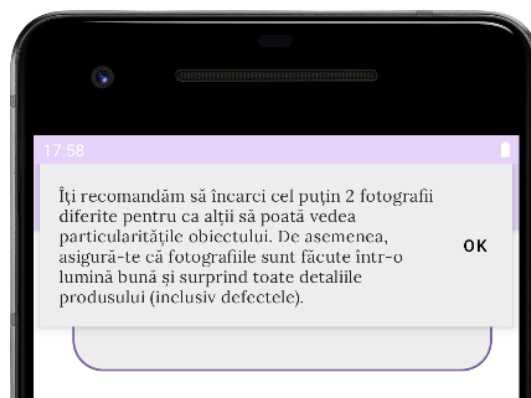


Figura 4.22 – Informații adiționale pentru câmpul fotografii

4.6. Pagina unui obiect

Pentru a vizualiza o postare, este suficient să fie apăsată *card*-ul ei din pagina principală sau din Istoric, secțiunea Disponibile. Pagina care se va deschide va conține detaliile obiectului respectiv plus numele și poza proprietarului. Se vor afișa datele obligatorii de la adăugarea postării, plus cele opționale dacă există. Categoria din care face parte obiectul nu va fi afișată, fiind doar un element util în filtrare/oferire de recomandări. Fotografiile pot fi vizualizate în detaliu făcând *click* pe caruselul din partea de sus a paginii.

Pentru a recunoaște cu ușurință scopul postării, sub fotografii este afișată o iconiță sugestivă și pe lângă aceasta, fundalul paginii este în raport cu regula prezentată în capitolul dedicat identității

vizuale, adică mov pentru schimburi și galben pentru donații. Cele 2 tipuri de pagini sunt ilustrate în figura 4.23.

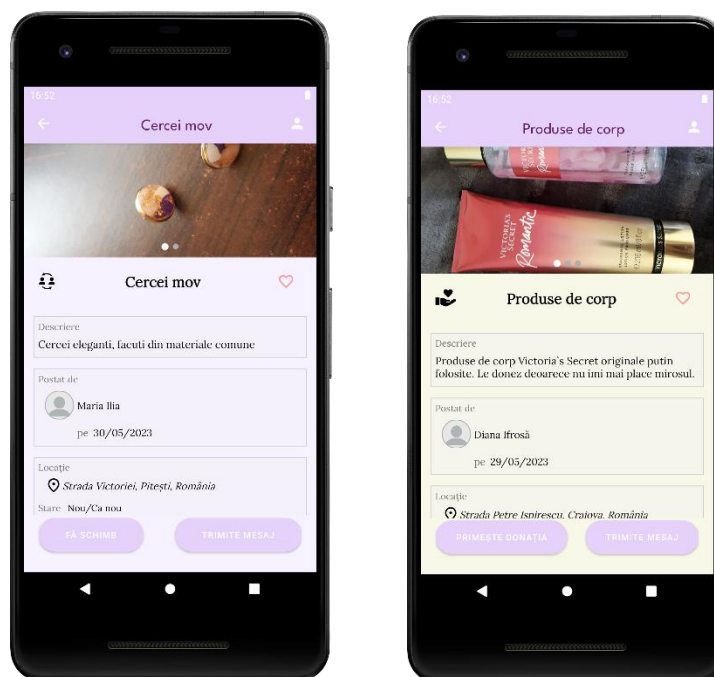


Figura 4.23 – Pagina unui obiect pentru schimb/donație

În partea de sus a paginii, există opțiunea și de a adăuga un obiect la favorite, prin apăsarea butonului în formă de inimă. În partea de jos sunt prezente 2 butoane care au comportamentul după cum urmează:

- butonul “Trimite mesaj” va deschide un *chat* cu proprietarul postării (dacă nu există deja o conversație se creează una, iar dacă există se deschide aceea). Această funcționalitate este gândită pentru a veni în ajutor persoanelor care vor să afle mai multe detalii despre obiectul postat, fără a fi forțați să facă o propunere.
- butonul “Fă schimb” va deschide o pagină în care utilizatorul trebuie să aleagă un obiect din cele personale disponibile, pentru a trimite o propunere de schimb în *chat*.
- butonul “Primește donația” va trimite în *chat* o propunere de primire a donației respective.

Propunerile și aspectul lor în cadrul conversațiilor vor fi abordate în subcapitolul destinat conversațiilor.

De asemenea, sub detaliile obiectului este prezentă și opțiunea de a raporta o postare (figura 4.24). Pagina acesteia este ilustrată în aceeași figură și conține un câmp de text pentru a introduce motivul raportării.

După trimiterea raportului, administratorul va primi un email automat cu datele principale ale raportării, urmând să investigheze situația analizând datele din Firestore/Cloud Storage în consola web pusă la dispoziție de Firebase.

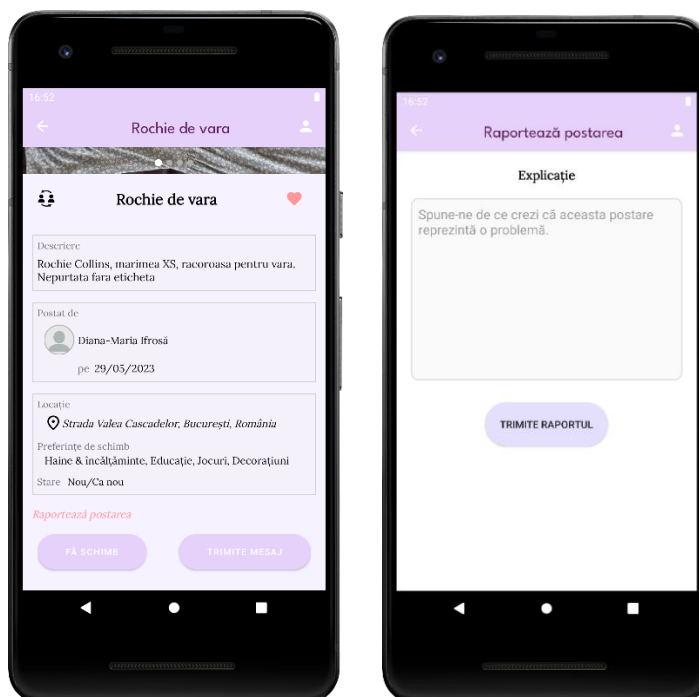


Figura 4.24 – Butonul de raportare postare și pagina destinată acestui lucru

4.7. Pagina unui alt utilizator

Este firesc ca oamenii să fie previzibili. Din acest punct de vedere, există posibilitatea ca unui utilizator să aibă obiecte într-un anumit stil, plăcut altora. În cadrul “Swap”, este ușor să vezi ce alte obiecte au ceilalți.

Pentru a naviga către pagina unui alt utilizator, nu este nevoie decât de un *click* pe numele sau imaginea acestuia. În cadrul profilului, se pot vizualiza fotografia sa, numele complet și toate postările (obiectele disponibile) acestuia într-o ordine aleatoare în caruselul derulabil din partea de jos (figura 4.25).

La apăsarea pe unul din obiectele din carusel, se va deschide pagina acestuia.



Figura 4.25 – Pagina unui alt utilizator

4.8. Recomandări

Recomandările pot fi vizualizate în pagina cu același nume accesibilă din meniu și cuprind postări care corespund preferințelor adăugate în setări (cuvinte de interes, alți utilizatori preferați, orașe favorite și altele descrise în capitolul 4.3). Scopul paginii este de a oferi utilizatorului o selecție de postări cât mai aproape de interesele sale și astfel să mențină interesul asupra aplicației.

Ordinea în care acestea sunt afișate este una aleatoare pentru a crește șansa vizualizării unui număr cât mai mare dintre ele, mai ales dacă sunt numeroase postări în această secțiune. *Design*-ul (figura 4.26) este similar paginii principale, iar la apăsare pe un *card* se deschide pagina obiectului aferent.

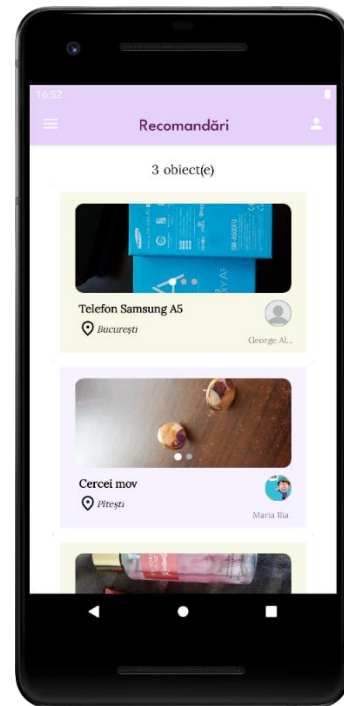


Figura 4.26 – Recomandările pe baza preferințelor

4.9. Harta

Această pagină are rolul de a ajuta utilizatorii să vizualizeze obiectele din jurul lor sau dintr-o anumită zonă geografică. Bara de căutare oferă la completare și sugestii de tip *autocomplete*, pentru a ușura căutarea unei locații. În plus, la apăsarea unui indicator de pe hartă se va deschide în partea de jos o previzualizare a obiectului de la acea locație, urmând ca la o apăsare pe ea, utilizatorul să fie redirecționat către pagina obiectului (figura 4.27).

Pentru a vizualiza obiectele din apropierea locației curente, este necesar ca utilizatorul să își dea acordul folosirii acesteia, fapt ce se reduce la alegerea opțiunii “Allow” din *popup*-ul clasic. Folosind butoanele oferite implicit de Google Maps, se poate controla nivelul *zoom*-ului și se poate “naviga” la locația curentă de pe hartă.

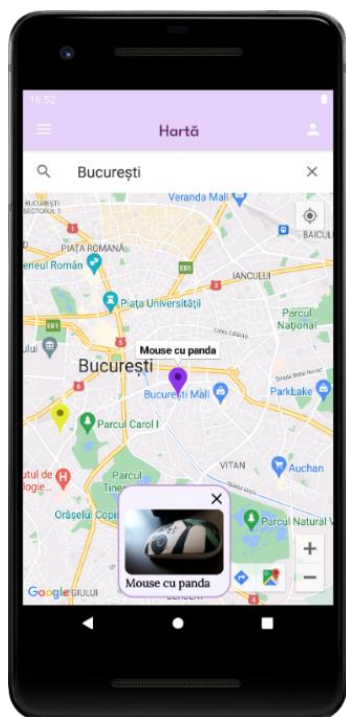


Figura 4.27 – Pagina hărții

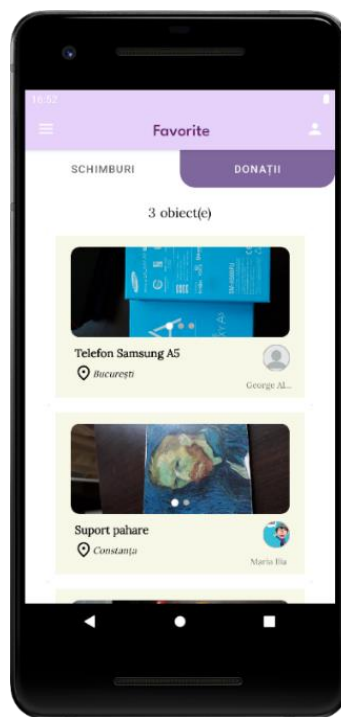


Figura 4.28 – Pagina favoritelor cu cele 2 categorii

4.10. Favorite

Obiectele care prezintă interes pot fi salvate în pagina Favorite, accesibilă din meniul principal. Acolo pot fi regăsite toate postările marcate cu ♥ în pagina obiectului și care încă sunt disponibile (nu au fost date sau șterse). Scopul paginii este de a salva într-un loc postările interesante, pentru a putea reveni la ele de-a lungul timpului, chiar dacă aplicația a fost închisă.

Pagina este împărțită în 2 categorii (schimburi și donații) pentru a facilita vizualizarea (figura 4.28). Ordinea obiectelor în pagină este cea în care s-a făcut salvarea pentru trasabilitate. Design-ul este unul simplu, similar paginii principale și la fel ca la recomandări, la apăsare pe un *card* se deschide pagina obiectului aferent.

4.11. Conversații

4.11.1. Aspecte generale

Partea de conversații a aplicației este una foarte importantă deoarece este mediul în care se fac propunerile și se creează relațiile interumane. Mai mult decât atât, este spațiul în care se cer clarificări, se stabilesc întâlnirile pentru predarea produselor și se negociază. Pagina în care se regăsesc conversațiile este ilustrată în figura 4.29.

Conversațiile sunt ordonate după data ultimului mesaj și au o previzualizare alcătuită din informații despre ultimul mesaj după caz, mesaj text, fotografie sau propunere, sugerând și de către cine a fost trimis mesajul.

Fiecare *card* al unei conversații conține și fotografia de profil, numele interlocutorului, dar și data la care a fost trimis mesajul.

În cadrul conversațiilor utilizatorii pot trimite mesaje text sau fotografii din galeria personală folosind bara din partea de jos. Mesajele sunt ordonate după data primirii și sunt orientate spre dreapta dacă utilizatorul curent le-a trimis sau spre stânga în caz contrar. Un exemplu în acest sens este ilustrat în figura 4.30.

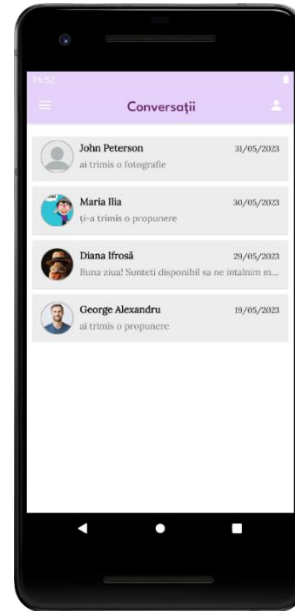


Figura 4.29 – Pagina conversațiilor



Figura 4.30 – Trimiterea mesajelor text și a imaginilor

4.11.2. Propunerile

Tot în cadrul conversațiilor se trimit și propunerile de schimb sau de primire a unei donații. Această funcționalitate este una dintre cele cheie ale aplicației deoarece stă la baza schimburilor și donațiilor. Acest lucru se face din pagina produsului, așa cum a fost deja prezentat în capitolul 4.6.

Propunerile sunt reprezentate prin *card*-uri de culoare corespunzătoare și iconițe sugestive, ce conțin după caz, fotografiile și denumirile celor 2 obiecte (figura 4.28) sau fotografia și denumirea obiectului dat spre donare și profilul utilizatorului ce dorește să primească donația respectivă (figura 4.29). La apăsarea pe o propunere se deschide pagina acesteia, care conține datele schimbului sau donației, cât și un buton pentru a accepta propunerea. Datele afișate sunt cele ale obiectelor implicate (denumire, descriere, fotografii etc.), alături de poza de profil și denumirea celuilalt utilizator. Un exemplu de propunere de oferire donație, din perspectiva proprietarului obiectului, este ilustrat în figura 4.30.



Figura 4.28 – Trimiterea unei propuneri de schimb în chat

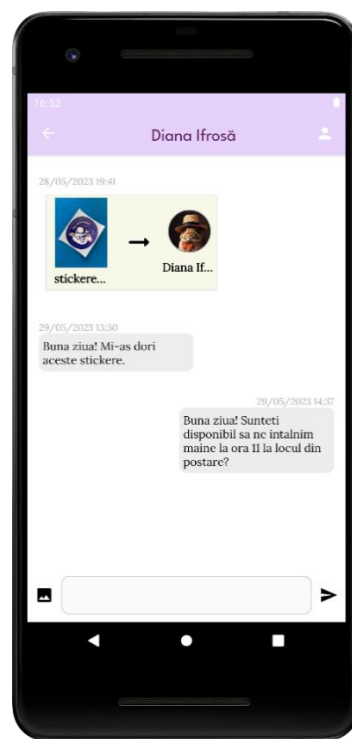


Figura 4.29 – Primirea unei propuneri de oferire donație în chat



Figura 4.30 – Pagina unei propuneri de oferire donație

În cazul utilizatorilor care trimit o propunere, aceasta este automat considerată acceptată, însă pentru cei ce primesc propuneri, ei le pot confirma apăsând butonul din partea de jos a paginii. Aceștia le vor fi prezentate și condițiile în care trebuie să confirme o propunere în aplicație, ca în figura 4.31.

După confirmarea unei propuneri, aceasta va fi afișată în pagina istoricului, iar obiectul (sau obiectele) va fi considerat indisponibil, nemaifiind afișat celorlalți utilizatori. În plus, dacă un obiect este deja dat, propunerea nu va mai putea fi acceptată, fiind însoțită de un mesaj de informare.

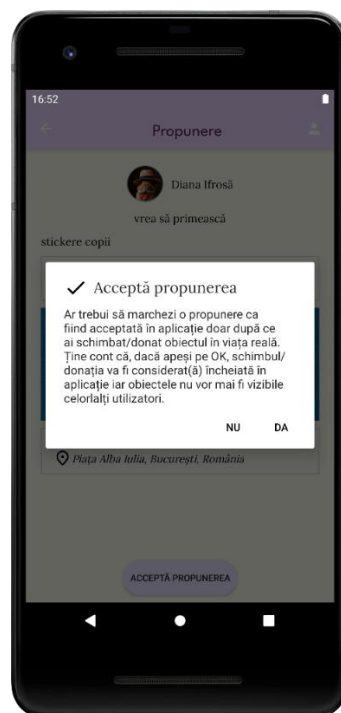


Figura 4.31 – Confirmarea unei propuneri de
oferire donație

4.12. Istoric

Secțiunea istoricului este cea în care utilizatorii își pot găsi propriile postări ale obiectelor încă disponibile, cât și schimburile sau donațiile încheiate. Această pagină este împărțită în 2 categorii principale ("Disponibile" și "Indisponibile") pentru a facilita vizualizarea (figura 4.32).

În prima categorie se regăsesc obiectele postate care încă sunt disponibile, indiferent dacă sunt pentru schimb sau donație, în ordinea postării acestora. Aspectul este similar celui din pagina principală, iar la apăsarea unui *card* se va deschide pagina obiectului respectiv. Pagina obiectului este de această dată puțin modificată, astfel că nu mai sunt prezente butoanele de la baza paginii, nici opțiunile de raportare postare sau adăugare la favorite. În schimb, utilizatorul are opțiunea de a șterge obiectul, apăsând un buton cu o iconiță sugestivă.

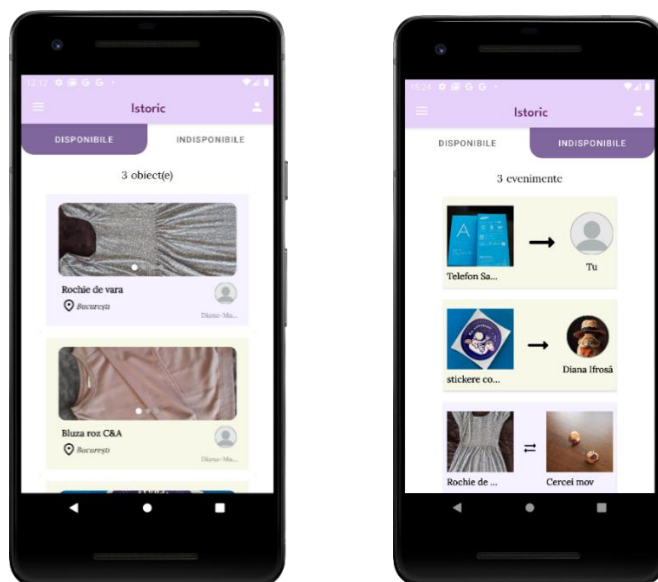


Figura 4.32 – Obiectele disponibile din istoric și obiectele indisponibile (evenimentele)

A doua categorie conține detalii despre schimburile sau donațiile făcute/primate, tot sub formă de listă. Tipul “tranzacției” poate fi dedus din culoarea *card*-ului și simbolul din cadrul acestuia. Toate cele 3 posibilități de “tranzacții” (schimb, donație și primire donație) sunt ilustrate în figura 4.33. Paginile evenimentelor conțin detalii despre obiectele implicate, dar și despre proprietarii lor. Mai mult decât atât, la apăsarea pe fotografia utilizatorului menționat în eveniment se va deschide profilul acestuia.

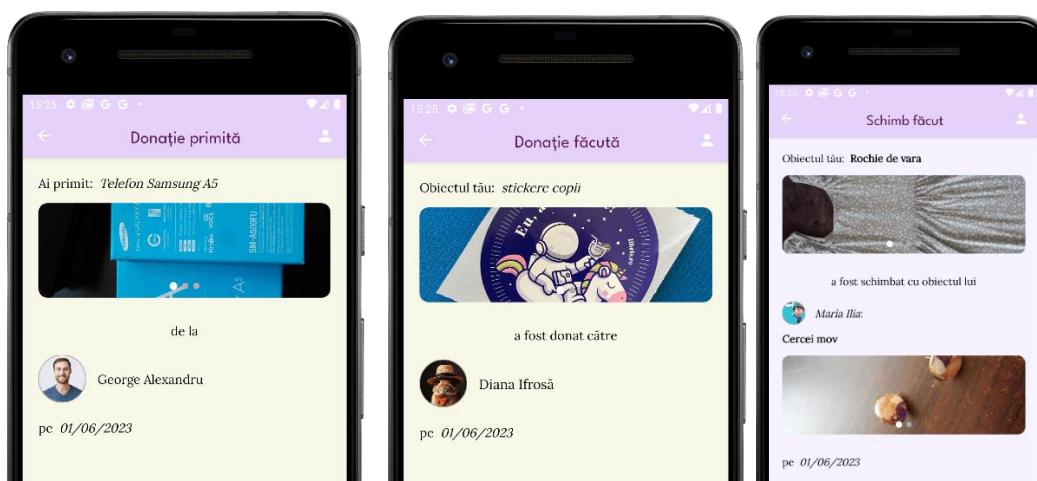


Figura 4.33 – Evenimentele de primire donație, donare și schimb

4.13. Pagina de contact (feedback)

Feedback-ul este un element important în dezvoltarea unui produs *software*, deoarece conectează dezvoltatorul sau administratorul cu nevoile directe ale oamenilor. Astfel, o pagină destinată acestui lucru este o punte de legătură între utilizatori și administratori/dezvoltatori.

“Swap” pune la dispoziție utilizatorilor săi o pagină pentru a oferi sugestii sau a primi ajutor. Aceasta este accesibilă din meniul principal, fiind ultima opțiune, denumită “Contactează-ne”.

Aspectul ei este unul simplu și intuitiv (figura 4.34), conținând doar un câmp de text pentru motivarea sesizării și un buton pentru trimiterea acesteia, email-ul utilizatorului fiind adăugat în sesizare în mod automat.

În plus, de îndată ce a fost trimisă sesizarea, administratorul va primi un email cu datele acesteia, anume id-ul documentului din Firestore pentru trasabilitate, email-ul utilizatorului care a făcut sesizarea și motivul său.

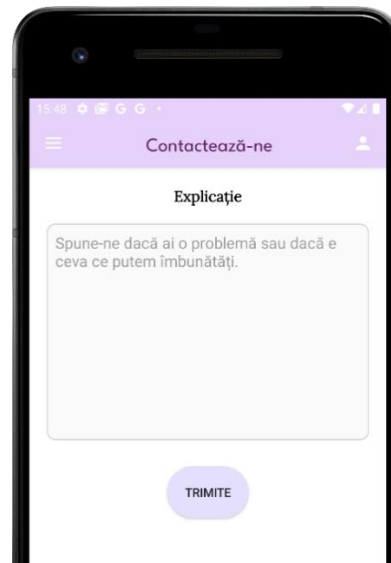


Figura 4.34 – Pagina de contact

4.14. Schimbarea limbii

Pentru a fi accesibilă cât mai multor oameni și a aduce un plus de valoare față de ale aplicației similare, “Swap” poate fi folosită atât în limba română, cât și în limba engleză. Limba aplicației depinde de limba setată pe telefon (din setări). De exemplu, dacă telefonul are ca limbă principală româna, atunci utilizatorii vor putea folosi aplicația “Swap” în limba română, similar fiind și pentru engleză. În schimb, dacă telefonul este setat pe o limbă nesuportată de aplicație, atunci limba implicită va fi engleza.

4.15. Controlul administratorului folosind Firebase

Monitorizarea datelor și a utilizatorilor din cadrul aplicației poate fi făcută de către administrator folosind consola web pusă la dispoziție de Firebase. În plus, acesta este notificat și pe email atunci când se trimite o raportare sau o sesizare, precum este descris în capitolele anterioare.

Vizualizarea și editarea datelor în consola web este una facilă, necesitând doar *click-uri*. Datele pot fi analizate prin intermediul documentelor din colecțiile Firestore, fiind prezentate într-o manieră ușor de citit și fără să necesite cunoștințe tehnice de baze de date (figura 4.35). Fotografiile pot fi deschise direct în *browser* folosind *link-urile* salvate în documente.

Colecția CategoriesObjects este special construită pentru analiza modului de folosire a aplicației, deoarece se pot vizualiza categoriile de obiecte și numărul de obiecte postate sau tranzacționate din fiecare. În urma unei analize pe această colecție, administratorul își poate da seama ce categorie este cea mai populară, sau din contră, cea mai nepopulară, cât și o posibilă nișă spre care se îndreaptă aplicația.

De asemenea, Cloud Firestore oferă în secțiunea “Usage” date despre frecvența operațiilor făcute la nivelul bazei de date (citiri, actualizări, ștergeri) pe anumite intervale de timp. Această secțiune este disponibilă și pentru Cloud Storage și Authentication.

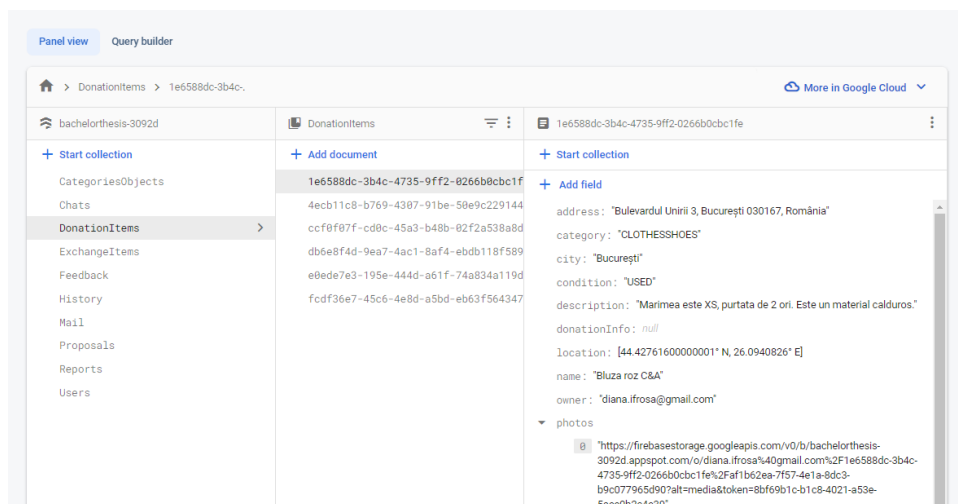
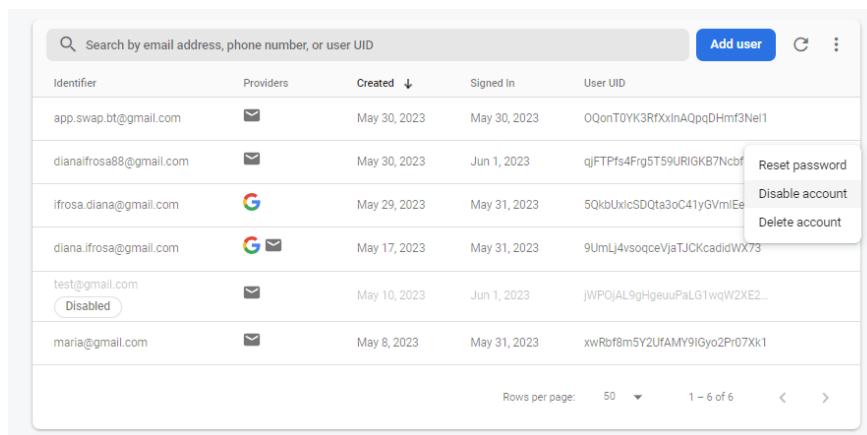


Figura 4.35 – Vizualizarea datelor în consola Firestore

Vizualizarea și gestionarea utilizatorilor se poate face în același mod, în secțiunea Authentication, precum în figura 4.36. Aici administratorul poate șterge sau bloca utilizatori, dar și reseta parolele acestora. Dacă un utilizator este blocat sau șters de către administrator, acesta va fi înștiințat la *login* iar în termen de maxim o oră va fi deconectat.



Identifier	Providers	Created ↓	Signed In	User UID
app.swap.bt@gmail.com	📧	May 30, 2023	May 30, 2023	OQonT0YK3RfXlnAQpQDHmf3NeI1
dianaifrosa8@gmail.com	📧	May 30, 2023	Jun 1, 2023	qFTPfs4Frq5T59URIGKB7Ncbf
ifrosa.diana@gmail.com	🌐	May 29, 2023	May 31, 2023	5QkbUxicSDQta3oC41yGvmIE
diana.ifrosa@gmail.com	🌐 📧	May 17, 2023	May 31, 2023	9UmLj4vsoqceVjaTJCKcadidWX73
test@gmail.com Disabled	📧	May 10, 2023	Jun 1, 2023	JWPOJAL9gHgeuuPaLG1wqW2XE2...
maria@gmail.com	📧	May 8, 2023	May 31, 2023	xwRbf8m5Y2UfAMY9IGyo2Pr07Xk1

Figura 4.36 – Vizualizarea și gestionarea utilizatorilor în Firebase Authentication

4.16. Alte detalii

Lipsa datelor poate apărea, de exemplu, atunci când un utilizator nu are niciun element salvat la favorite sau când nu are niciun schimb sau donație încheiate.

Pentru a evita situațiile când o pagină rămâne goală din pricina lipsei de date, “Swap” înștiințează utilizatorii de acest fapt prin mesaje cu rol explicativ, precum se poate observa și în figura 4.37.

Mai mult decât atât, în timp ce datele se încarcă este afișat un simbol de tipul *loading circle* pentru a evidenția faptul că datele sunt procesate.

Ambele elemente menționate au de asemenea și rolul de a elimina o posibilă bănuială a utilizatorului cum că aplicația s-a blocat. În plus, mesajele oferă și indicații de utilizare a funcționalității aferente.



Figura 4.37 – Mesaje explicative când nu sunt disponibile date

5. Testarea aplicației

5.1. Testarea de interfață (UI testing)

Așa cum am precizat în capitolul 2, destinat tehnologiilor, am folosit *framework*-ul Espresso pentru a testa comportamentul paginii de adăugare a unui obiect, mai exact, pentru a scrie teste automate de UI.

Testarea de interfață este o parte esențială din procesul de testare a unei aplicații mobile, deoarece are ca scop verificarea calității interacțiunii dintre utilizator și aplicația în sine prin intermediul interfeței grafice, astfel încât erorile să fie detectate cât mai rapid [1].

Pe parcursul scrierii testelor de interfață, se urmărește ca interacțiunea să fie una *user-friendly*, ca toate elementele (butoane, imagini, câmpuri de text etc.) să fie poziționate pe pagină așa cum este intenționat și în același timp, ca interfața aplicației să funcționeze corect. Mai mult decât atât, viteza cu care aplicația răspunde la *input*-ul utilizatorului este de asemenea luată în considerare, întrucât reprezintă un factor important în experiența sa generală de navigare. Acest tip de testare poate fi făcut atât manual, cât și automat, folosind dispozitive reale sau virtuale (din Android Emulator).

5.2. Noțiuni generale despre Espresso

Ca în cadrul majorității instrumentelor de testare, sunt necesare, pentru a scrie teste relevante, metode care să returneze starea/obiectul pe care îl testăm, metode care să aplice niște schimbări asupra obiectului testat și alte metode care să verifice starea obiectului în urma modificărilor.

În acest sens, *framework*-ul este alcătuit din 4 componente ce se îmbină pentru a crea teste de interfață [8]:

1. **Espresso:** punctul de start al scrierii de *test cases*, care oferă metode specifice pentru a interacționa cu *view*-urile aplicației, precum `onView()` sau `onData()`, precum și API-uri special concepute să interacționeze cu partea de sistem de operare din *view*-uri, spre exemplu `pressBack()`,

2. **ViewMatchers:** o colecție de obiecte care implementează interfața *Matcher* și permite *tester*-ului să localizeze *view*-ul dorit în ierarhie folosind un id, un text sau alți identificatori; obiectele din această colecție se dau ca parametru metodei *onView()*.
3. **ViewActions:** o colecție de obiecte care definesc acțiunea care trebuie îndeplinită în *view*-ul testat, precum *click()*. Obiectele din această categorie sunt date ca parametru metodei *perform()*.
4. **ViewAssertions:** o colecție de obiecte al căror scop e să valideze starea în care se află *view*-ul testat (sub formă de aserțiuni), spre exemplu, dacă un anumit text a apărut pe ecran sau un alt element este afișat. Metoda *check()* primește ca parametru astfel de obiecte, iar cea mai folosită aserțiune este *matches()*.

Espresso conține de asemenea și niște adnotări speciale care oferă informații adiționale cu privire la modul de execuție al testelor. Printre cele mai cunoscute se numără [16]:

- **@Rule:** indică o regulă alcatuită dintr-o instrucțiune sau mai multe, care trebuie rulată înainte și după fiecare test.
- **@RunWith:** specifică cu ce clasă sa fie rulate testele.
- **@Test:** indică faptul că acea metodă reprezintă un test.
- **@Before:** indică faptul că metoda adnotată trebuie să fie executată înainte de fiecare test din suita de teste.
- **@After:** similar anterioarei, metoda adnotată trebuie să fie executată după fiecare test din suita de teste.

5.3. Cazurile de test

Pagina de adăugare a unui obiect conține numeroase câmpuri de tip *input*, precum *TextViews*, mai multe tipuri de butoane, *spinners* și are comportament diferit în funcție de *input*-ul utilizatorului. Spre exemplu, secțiunea destinată scopului postării ascunde o altă subsecțiune care se afișează doar dacă opțiunea de “Schimb” este selectată.

Regulile pe care un utilizator trebuie să le respecte pentru a adauga cu succes un obiect în aplicație și care au fost urmărite în testele scrise sunt:

- a. Titlul trebuie să aibă cel puțin 3 caractere; în caz contrar, va fi indicat un mesaj de eroare, fie sugerând că titlul nu ar trebui să fie gol, fie că ar trebui să aibă minimum 3 caractere.
- b. Descrierea trebuie să aibă cel puțin 5 caractere; în caz contrar, va fi indicat un mesaj de eroare, fie sugerând că descrierea nu ar trebui să fie goală, fie că ar trebui să aibă minimum 5 caractere.
- c. Câmpurile din secțiunea “Alte detalii”, cât și categoria de schimb preferată (dacă obiectul este postat pentru schimb) sunt opționale, iar la deschiderea/închiderea acestei secțiuni (prin apăsare pe simbolul săgeată) simbolul se schimbă (sageată-sus pentru deschis, săgeata-jos pentru închis) .
- d. Categoria din care face parte obiectul este obligatoriu să fie aleasă din *spinner*-ul pus la dispoziție.
- e. Scopul postării trebuie ales prin apăsarea pe unul dintre butoanele “Schimb” sau “Donație”.
- f. Secțiunea ascunsă, de sub scopul postării, care se referă la categoria de schimb preferată va fi afișată doar dacă scopul selectat este “Schimb”, în caz contrar rămânând ascunsă.
- g. După apăsarea butonul “Salvează”, dacă una din categorii este incorect completată sau nu este completată deloc, un mesaj explicativ de eroare va apărea în dreptul câmpului aferent.

Structura clasei de teste este descrisă în fragmentul de cod de mai jos. Este de menționat că în acesta apare și o regulă de tip `@Before`, care deschide fragmentul aferent paginii de adăugare a obiectului, înainte ca testele să fie rulate.

```
@RunWith(AndroidJUnit4::class)
class AddItemFragmentTest
    : TestCase()
{
    private lateinit var scenario: FragmentScenario<AddItemFragment>

    @Before
    fun setup() {
        scenario = launchFragmentInContainer(themeResId = R.style.Theme_BachelorThesis)
        scenario.moveToState(Lifecycle.State.STARTED)
    }

    // the actual tests are written here
}
```

Testele scrise acoperă toate regulile enunțate, tratând și cazurile de frontieră, spre exemplu când titlul sau descrierea au 3, respectiv 5 caractere. De asemenea, se testează și reacția paginii la un

posibil comportament haotic al utilizatorilor, precum apăsarea butoanelor de scop, “Schimb” și “Donație” alternativ de mai multe ori, sau apăsarea unui singur buton de mai multe ori.

Au fost scrise 14 teste, care se pot împărți în categorii după cum urmează:

1. Teste care urmăresc caracterul obligatoriu al câmpurilor: 1 test.
2. Teste care verifică faptul că lungimea minimă a titlului și a descrierii este respectată: 6 teste, câte 3 pentru fiecare câmp în parte. Spre exemplu, pentru titlu s-au testat cazurile lungime < 3, lungime = 3, lungime > 3. Un exemplu de test din această categorie este ilustrat în fragmentul de cod de mai jos.
3. Teste referitoare la funcționalitatea *spinner*-ului, de a alege o categorie dintre cele disponibile: 1 test.
4. Teste referitoare la selectarea unui scop al postării: 5 teste (urmărind cazurile: selectare “Schimb”, selectare “Donație”, selectare alternantă “Schimb” apoi “Donație” și viceversa, selectare de 4 ori consecutiv “Schimb”, selectare de 4 ori consecutiv “Donație”).
5. Teste care verifică posibilitatea de a deschide secțiunea de “Alte detalii” și schimbarea simbolului indicativ (săgeata-sus, săgeata-jos): 1 test.

```
@Test
fun titleLessCharacters() {
    onView(withId(R.id.item_title_edittext)).perform(typeText("Ti"))

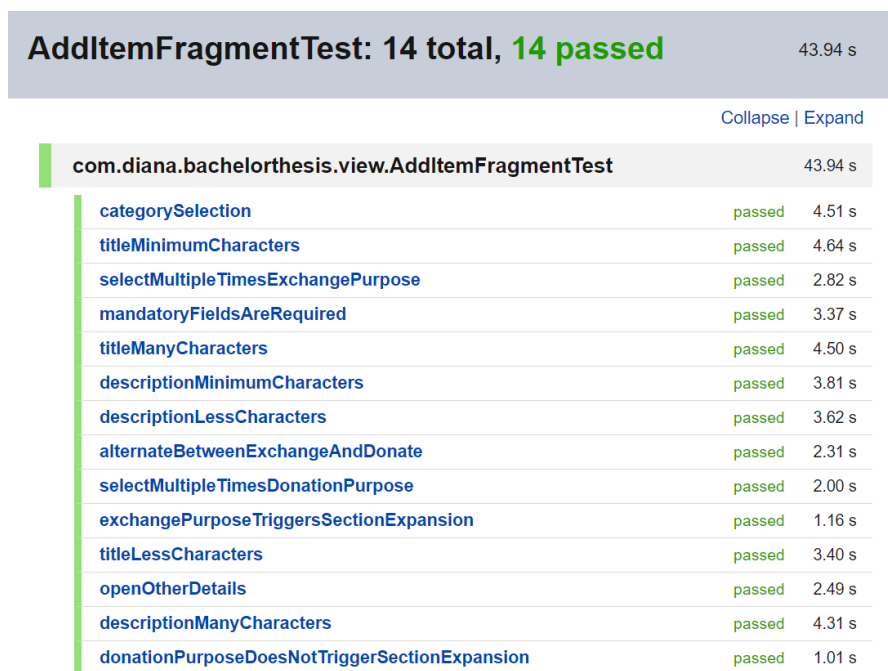
    // click the Save button and wait for the fields validation to be finished
    onView(withId(R.id.save_item)).perform(scrollTo(),click())
    Thread.sleep(1500)

    onView(withId(R.id.item_title)).perform(scrollTo())
    onView(withId(R.id.item_title_status)).check(matches(isDisplayed()))
    onView(withId(R.id.item_title_status)).check(matches(withText(R.string.item_title_short)))
    // item title short = "Obligatory cel puțin 3 caractere"
}
```

Notă: Alte teste pot fi consultate în Anexa B.

5.4. Raport final și concluzii

Testele au fost rulate pe un dispozitiv real, Samsung Galaxy J5, iar rezultatele acestora sunt ilustrate în figura 5.1, care conține un raport al testării generat în urma rulării suitei de teste, de către Android Studio. Se poate observa că toate testele scrise au trecut cu succes și că timpul de rulare aproximativ a fost de 50 de secunde.



AddItemFragmentTest: 14 total, 14 passed		43.94 s
		Collapse Expand
com.diana.bachelorthesis.view.AddItemFragmentTest		43.94 s
categorySelection	passed	4.51 s
titleMinimumCharacters	passed	4.64 s
selectMultipleTimesExchangePurpose	passed	2.82 s
mandatoryFieldsAreRequired	passed	3.37 s
titleManyCharacters	passed	4.50 s
descriptionMinimumCharacters	passed	3.81 s
descriptionLessCharacters	passed	3.62 s
alternateBetweenExchangeAndDonate	passed	2.31 s
selectMultipleTimesDonationPurpose	passed	2.00 s
exchangePurposeTriggersSectionExpansion	passed	1.16 s
titleLessCharacters	passed	3.40 s
openOtherDetails	passed	2.49 s
descriptionManyCharacters	passed	4.31 s
donationPurposeDoesNotTriggerSectionExpansion	passed	1.01 s

Figura 5.1 – Raportul testelor, generat de Android Studio

Provocările întâmpinate în procesul de testare au provenit de la *setup*-ul necesar atât la nivelul IDE-ului, Android Studio, cât și pe dispozitivul folosit, deoarece trebuie dezactivate setările precum *window animation scale*, *transition animation scale* și *duration scale animator* pentru a nu interfera cu testele [26].

Este important de menționat că procesul de testare al unei aplicații mobile este unul mult mai lung și mai detaliat comparativ cu cele prezentate în această lucrare, însă am dorit să acopăr cel puțin la nivel minimal și această etapă a dezvoltării *software* pentru a asigura o robustețe mai ridicată a aplicației.

6. Concluzii și direcții de dezvoltare

“Swap” vine în ajutorul oamenilor ce caută o platformă online pentru a susține inițiativele lor de schimbare sau donare de obiecte. Prin funcționalitățile descrise în cadrul lucrării, aplicația se remarcă prin interfața intuitivă și ușor de folosit de o mare majoritate a utilizatorilor de *smartphone*, indiferent de nivelul de experiență cu alte aplicații similare (spre exemplu prin butoane cu text explicativ și nu doar iconițe sau prin împărțirea celor 2 scopuri, schimb și donație, folosind 2 culori opuse). Mai mult decât atât, este accesibilă și datorită faptului că este disponibilă în 2 limbi, engleza și româna. Aceste lucruri împreună cu unele funcționalități abordate sumar sau deloc în alte aplicații, fac din “Swap” un produs cu potențial ridicat de utilitate, dar și cu numeroase direcții de dezvoltare.

Din punct de vedere al dezvoltării propriu-zise, acest proiect a reprezentat pentru mine o provocare prin faptul că am construit o aplicație funcțională de la 0, abordând atât partea de *backend*, cât și cea de *frontend*. În plus, a reprezentat o oportunitate de învățare a utilizării elementelor din ecosistemul Android, ceea ce constituie un puternic pilon de dezvoltare profesională în această direcție. Erorile întâmpinate pe parcursul creării “Swap” nu au fost puține, însă au avut rolul de a mă ajuta să aprofundez teoria învățată și să observ în practică cum toate componentele aplicației se îmbină, rezultând un produs complet.

Dezvoltările viitoare ale aplicației pot fi atât pe partea de *design*, cât și pe partea mai tehnică prin șlefuirea actualelor funcționalități sau adăugarea unora noi și inovative. Spre exemplu, la nivel de *design* ar putea fi modificată interfața pentru ecranele de tip tabletă pentru a folosi tot spațiul disponibil într-un mod mai eficient. De asemenea, aspectul general al aplicației poate fi modernizat prin introducerea unor animații captivante pentru utilizatori. Câteva dintre funcționalitățile care consider că pot fi îmbunătățite sunt cele de adăugare obiect (prin paginare sau prezența unui *loading bar* care să indice câte câmpuri mai trebuie completate), *chat* (prin adăugarea posibilității de a vedea când mesajul a fost citit, prin crearea de grupuri sau *location sharing*) și acceptarea unei propuneri (impunerea unui număr maxim de donații primite fără a face personal o donație sau un schimb sau acordarea de *feedback* în urma efectuării unei tranzacții).

Bibliografie

- [1] Amrita Vazirani, *What is Android UI Testing?*, <https://www.browserstack.com/guide/what-is-android-ui-testing>, ultima accesare 26 mai 2023.
- [2] Anastasiya Marchuk, *Native Vs Cross-Platform Development: How To Choose*, <https://www.uptech.team/blog/native-vs-cross-platform-app-development>, ultima accesare 25 mai 2023.
- [3] Ash Turner, *Android vs. Apple Market Share: Leading Mobile Operating Systems (OS) (May 2023)*, <https://www.bankmycell.com/blog/android-vs-apple-market-share/>, ultima accesare 23 mai 2023.
- [4] Ben Kitpitak, *Reasons to use Android Single-Activity Architecture with Navigation Component*, <https://oozou.com/blog/reasons-to-use-android-single-activity-architecture-with-navigation-component-36>, ultima accesare 28 mai 2023.
- [5] David Ford, *Who is using Kotlin?*, <https://medium.com/@daveford/who-is-using-kotlin-84b11b4fb51a>, ultima accesare 25 mai 2023.
- [6] Flori Needle, *What Are Brand Identity Elements?*, <https://blog.hubspot.com/marketing/what-are-brand-elements>, ultima accesare 27 mai 2023.
- [7] Francesca Perry, *Like fast fashion, 'fast furniture' is a problem for our planet*, <https://edition.cnn.com/style/article/fast-furniture-problem-for-our-planet/index.html>, ultima accesare 23 mai 2023.
- [8] Gaurav Prabhakar, *What is Espresso Testing? How does it work?*, <https://www.browserstack.com/guide/what-is-espresso-testing-how-does-it-work#toc0>, ultima accesare 26 mai 2023.
- [9] Jessica Clark, *Firebase vs SQLite – What are the differences?*, <https://blog.back4app.com/firebase-vs-sqlite/>, ultima accesare 26 mai 2023.
- [10] JR Raphael, *Android versions: A living history from 1.0 to 14*, <https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html?page=2>, ultima accesare 24 mai 2023.
- [11] Katie Terrell Hanna, *Google Firebase*, <https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase>, ultima accesare 26 mai 2023.

- [12] Kobi Bohbot, *Firebase Storage: What It Is and How It Works*, <https://blog.back4app.com/firebase-storage/>, ultima accesare 25 mai 2023.
- [13] Muhamed Riyas M, *Navigation Component- The Complete Guide*, <https://medium.com/@muhammed.riyas/navigation-component-the-complete-guide-c51c9911684>, ultima accesare 25 mai 2023.
- [14] Priyank Kumar, *Understanding MVVM Architecture in Android*, <https://medium.com/swlh/understanding-mvvm-architecture-in-android-aa66f7e1a70b>, ultima accesare 28 mai 2023.
- [15] Rachael Dottle, Jackie Gu, *The Global Glut of Clothing Is an Environmental Crisis*, <https://www.bloomberg.com/graphics/2022-fashion-industry-environmental-impact/#xj4y7vzkg>, ultima accesare 23 mai 2023.
- [16] Rina Thakkar, *UI Testing with Espresso in Android*, <https://medium.com/mindful-engineering/ui-testing-with-espresso-in-android-10dfbc9f25da>, ultima accesare 26 mai 2023.
- [17] Scott Brown, *What is Android? Here's everything you need to know*, <https://www.androidauthority.com/what-is-android-328076/>, ultima accesare 24 mai 2023.
- [18] Victoria Bezsmolna, *Appium vs. Espresso: Which Framework to Use for Automated Android Testing*, <https://smartbear.com/blog/appium-vs-espresso-which-framework-to-use>, ultima accesare 26 mai 2023.
- [19] Documentație oficială Android Maps, <https://developer.android.com/training/maps>, ultima accesare 26 mai 2023.
- [20] Documentație oficială AndroidX, <https://developer.android.com/jetpack/androidx>, ultima accesare 25 mai 2023.
- [21] Documentație oficială arhitectură Android, <https://developer.android.com/topic/architecture>, ultima accesare 28 mai 2023.
- [22] Documentație oficială CircularProgressBar, <https://github.com/FarhamHosseini/LoadingButton>, ultima accesare 26 mai 2023.
- [23] Documentație oficială Cloud Firestore, <https://firebase.google.com/docs/firestore>, ultima accesare 26 mai 2023.
- [24] Documentație oficială Cloud Firestore data, *Get data with Cloud Firestore*, <https://firebase.google.com/docs/firestore/query-data/get-data>, ultima accesare 26 mai 2023.

- [25] Documentație oficială Cloud Functions, <https://firebase.google.com/docs/firestore/extend-with-functions> , ultima accesare 5 iunie 2023.
- [26] Documentație oficială Espresso, <https://developer.android.com/training/testing/espresso/setup>, ultima accesare 27 mai 2023.
- [27] Documentație oficială Firebase Authentication, <https://firebase.google.com/docs/auth>, ultima accesare 26 mai 2023.
- [28] Documentație oficială Firebase Cloud Messaging, <https://firebase.google.com/docs/cloud-messaging>, ultima accesare 5 iunie 2023.
- [29] Documentație oficială Glide, <https://github.com/bumptech/glide>, ultima accesare 26 mai 2023.
- [30] Documentație oficială Gradle, <https://developer.android.com/build>, ultima accesare 27 mai 2023.
- [31] Documentație oficială Gson, <https://github.com/google/gson>, ultima accesare 26 mai 2023.
- [32] Documentație oficială ImageSlideShow, <https://github.com/denzcoskun/ImageSlideshow>, ultima accesare 26 mai 2023.
- [33] Documentație oficială Kotlin, <https://kotlinlang.org/docs/faq.html#can-i-use-kotlin-for-native-development>, ultima accesare 25 mai 2023.
- [34] Documentație oficială LiveData, <https://developer.android.com/topic/libraries/architecture/livedata>, ultima accesare 28 mai 2023.
- [35] Documentație oficială Place Autocomplete, <https://developers.google.com/maps/documentation/places/android-sdk/autocomplete>, ultima accesare 26 mai 2023.

Anexa A – figuri adiționale

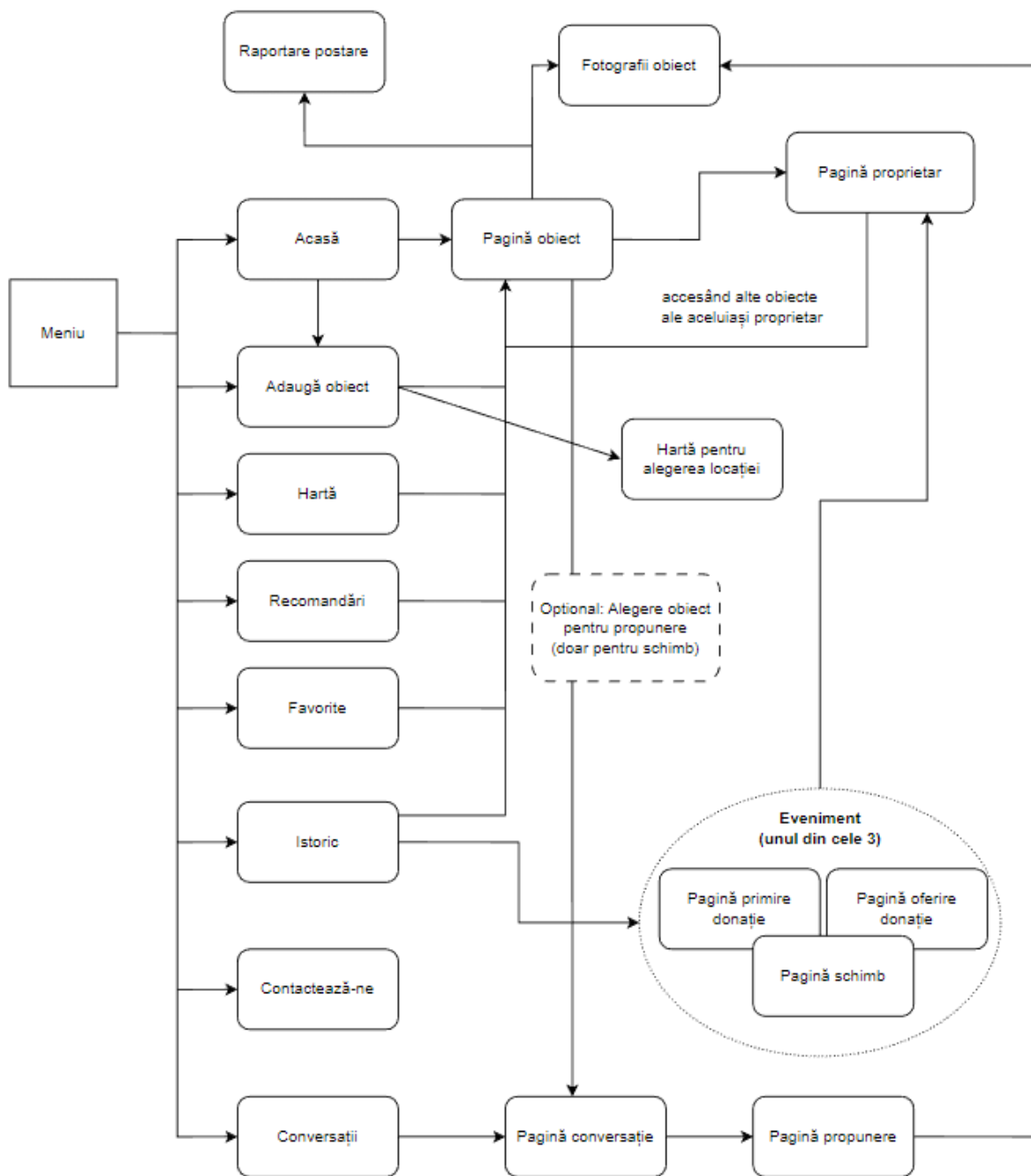


Figura A.1 – Navigarea în cadrul aplicației pentru un utilizator conectat

Anexa B – teste de interfață

Test 1: selectarea unei categorii

```
@Test
fun categorySelection() {
    val targetContext = ApplicationProvider.getApplicationContext<Context>()
    val category = targetContext.resources.getString(R.string.categ_education)

    onView(withId(R.id.item_category)).perform(scrollTo())
    onView(withId(R.id.spinner_categories)).perform(click())
    onData(allOf(`is`(instanceOf(String::class.java)), `is`(category))).perform(click())

    // verify the category chosen is displayed
    onView(withId(R.id.spinner_categories)).check(matches(withSpinnerText(containsString(
        category
    ))))
    // verify the category status field does not have any message
    onView(withId(R.id.item_category_status)).check(matches(not(isDisplayed()))))
}
```

Test 2: deschiderea secțiunii “Alte detalii” folosind iconița săgeată

```
@Test
fun openOtherDetails() {
    onView(withId(R.id.text_other_details)).perform(scrollTo())
    onView(withId(R.id.text_other_details)).perform(click())
    onView(withId(R.id.save_item)).perform(scrollTo())

    Thread.sleep(1000)

    // check hidden layout appears below the text and the drawable icon is changed into arrow up
    onView(withId(R.id.hidden_layout_other_details)).check(matches(isDisplayed()))
    onView(withId(R.id.hidden_layout_other_details)).check(isCompletelyBelow(withId(R.id.text_other_details)))
    onView(withId(R.id.text_other_details)).check(matches(withTagValue(equalTo(R.drawable.ic_arrow_dropup))))

    // check that after clicking again on the icon, the hidden layout disappears and
    // the drawable icon is changed into arrow down
    onView(withId(R.id.text_other_details)).perform(click())
    onView(withId(R.id.hidden_layout_other_details)).check(matches(not(isDisplayed())))
    onView(withId(R.id.text_other_details)).check(matches(withTagValue(
        equalTo(R.drawable.ic_arrow_dropdown))))
}
```

Test 3: selectarea de mai multe ori a butonului de scop “Schimb”

```
@Test
fun selectMultipleTimesExchangePurpose() {
    // check that for multiple selections of 'Exchange' button the section designated to Exchange
    // does appear, and that the button remains checked
    for (i in 1..4) {
        checkExchangePurpose()
    }
}

fun checkExchangePurpose() {
    onView(allOf(withId(R.id.radioButtonExchange), withText(R.string.exchange_purpose))).perform(click())

    onView(allOf(withId(R.id.radioButtonExchange), withText(R.string.exchange_purpose))).check(matches(
        isChecked()
    ))
    onView(withId(R.id.layout_preferences)).check(matches(isDisplayed()))
    onView(withId(R.id.layout_preferences)).check(isCompletelyBelow(withId(R.id.radioButtonExchange)))
}
```