

Programul de studii:

Informatica Aplicata

Lucrare de licență

Online Book Reader

|  |  |
| --- | --- |
| Autor: | **Mincu Alexandra Andreea** |
| Coordonator ştiințific: | **lect. dr. Răzvan Bocu** |

Braşov, 2022

Conținut

[1.INTRODUCERE 3](#_Toc105442154)

[2. TEHNOLOGII FOLOSITE 4](#_Toc105442155)

[2.1 ANDROID STUDIO 4](#_Toc105442156)

[2.1.1 Caracteristici Android 4](#_Toc105442157)

[2.1.2 Arhitectura Android 4](#_Toc105442158)

[2.1.3 Android SDK 5](#_Toc105442159)

[2.2 Java 5](#_Toc105442160)

[2.2.1 Caracteristici Java 5](#_Toc105442161)

[2.2.2 Componentele Java 5](#_Toc105442162)

[2.3 Firebase 6](#_Toc105442163)

[2.3.1 Firebase Authentication 6](#_Toc105442164)

[2.3.2 Autentificare SDK Firebase 7](#_Toc105442165)

[2.3.3 Autentificarea FirebaseUI Auth 8](#_Toc105442166)

[2.3.4 Realtime Database 8](#_Toc105442167)

[2.3.5 FirebaseUI Database 9](#_Toc105442168)

[2.3.6 Storage 9](#_Toc105442169)

[2.3.7 Firebase Analytics 10](#_Toc105442170)

[2.4 APIs utilizate 10](#_Toc105442171)

[2.4.1 PDF Page Viewer 11](#_Toc105442172)

[2.4.2 Image Slideshow 11](#_Toc105442173)

[2.4.3 Glide 11](#_Toc105442174)

[3.DESCRIEREA SISTEMULUI 12](#_Toc105442175)

[3.1 Autentificarea 12](#_Toc105442176)

[3.1.1 Pagina de autentificare 13](#_Toc105442177)

[3.1.2 Pagina de înregistrare 14](#_Toc105442178)

[3.1.3 Pagina de recuperare a parolei 17](#_Toc105442179)

[3.2 Pagina principală 19](#_Toc105442180)

[3.2.1Meniul 19](#_Toc105442181)

[3.2.2 Prezentarea cărților 21](#_Toc105442182)

[3.3 Genurile de cărți 26](#_Toc105442183)

[3.3.1 Alegerea categoriei 26](#_Toc105442184)

[3.3.2 Filtrarea cărților după categoria aleasa 27](#_Toc105442185)

[3.4 Deschiderea documentului 27](#_Toc105442186)

[3.5 Opțiunea de căutare 29](#_Toc105442187)

[3.6 Deconectarea contului 30](#_Toc105442188)

[3.7 Modul zi/noapte 30](#_Toc105442189)

[3.8 Modificarea documentelor afișate din Firebase 31](#_Toc105442190)

[3.9 Analiza valorilor din Firebase Analytics 32](#_Toc105442191)

[4. Utilizarea Sistemului 37](#_Toc105442192)

[4.1 Use case pentru utilizator nou 37](#_Toc105442193)

[4.2 Use case pentru utilizator deja autentificat 43](#_Toc105442194)

[4.2.1 Vizualizarea paginii principale 43](#_Toc105442195)

[4.2.2 Alegerea genului cărții si filtrarea documentelor 45](#_Toc105442196)

[4.2.3 Opțiunea de căutare 48](#_Toc105442197)

[4.2.4: Deconectarea utilizatorului 50](#_Toc105442198)

[4.2.5 Modul zi/noapte 52](#_Toc105442199)

[5. Concluzii si perspective de dezvoltare 54](#_Toc105442200)

[5.1 Adaptarea pentru mai mulți clienți 54](#_Toc105442201)

[5.2 Funcția de favorite 54](#_Toc105442202)

[5.3 Pagina de prezentare 55](#_Toc105442203)

[5.4 Filtrarea după limita de vârstă 55](#_Toc105442204)

[Bibliografie 56](#_Toc105442205)

# 1.INTRODUCERE

Aplicația pe care am dezvoltat-o este una de vizualizare de cărți și reviste cu ajutorul conexiunii la Internet. Aceasta își propune sa faciliteze atât dorința cititorului de a avea acces la diferite documente online, cât și a publicațiilor de a-și prezenta pe o arie mai largă produsele. Fiind necesara conexiunea la internet, mi-am dorit sa dezvolt o aplicație care sa ocupe un spațiu cat mai redus in dispozitivul android, pentru a fi ușor de instalat de cat mai mulți utilizatori.

În ultimii ani s-a făcut remarcată o preferință a tinerilor față de e-books în detrimentul formatului clasic, printat. Această predilecție pentru formatul digital este datorată gradului de accesibilitate ridicat pentru e-books fată de cel clasic, prin accesul mult mai ușor la dispozitivele electronice. Apariția device-urilor de tip Kindle a contribuit la creșterea rapidă a popularității cărților digitale prin portabilitate, capacitatea de a customiza tipul de font si dimensiunea acestuia, luminozitatea si capacitatea de a deține pe un singur device mai multe documente.

Am ales aceasta tema datorita pasiunii mele pentru citit, pentru a împărtăși si facilita acest mod de a petrece timpul liber si cu alte persoane. Consider ca aceasta aplicație are o întrebuințare reală in propagarea interesului pentru citit pentru că este ușor accesibilă pentru orice utilizator android, telefonul reprezentând un obiect aproape indispensabil în viața de zi cu zi. Astfel, majoritatea oamenilor vor avea acces mai ușor la informație si la această formă de recreere numită citit.

Această lucrare are ca obiectiv facilitarea înțelegerii tehnologiei folosite în dezvoltarea aplicației.

# 2. TEHNOLOGII FOLOSITE

## 2.1 ANDROID STUDIO

[2][3]Android Studio este IDE-ul oficial folosit pentru dezvoltarea aplicațiilor Android, care conține toate componentele necesare realizării aplicațiilor Android. Acesta deține funcții care facilitează productivitatea în timpul dezvoltării aplicațiilor Android. Pentru acest sistem de operare, Android Studio folosește un sistem de build bazat pe Gradle, șabloane si emulator. Acesta are încorporat Google Cloud Platform, care permite integrare ușoara pentru Google Cloud Messaging si App Engine, Github, Firebase etc. Cu ajutorul Android Studio se pot realiza aplicatii pentru majoritatea dispozitivelor cu sistem de operare Android, precum: telefon, tableta, smart watch, smart TV, automobile.

### 2.1.1 Caracteristici Android

Android este un sistem de operare open-source gratuit, utilizat pentru realizarea de aplicații.[1] Acesta deține următoarele trăsături:

* Suport Hardware: GPS, camera, senzor de proximitate, busola digitala
* Suport Media: MP3, MP4, JPEG, PNG, GIF, BMP
* Conexiune: Internet prin hotspot
* Mesaje: SMS, MMS
* Stocare a datelor: SQL Lite
* Multi-touch: Ecrane multi-touch
* Multi-tasking: Aplicatii multi-tasking
* Conectivitate: GSM sau EDGE, Bluetooth, Wi-Fi, IDEN, CDMA, EV-DO

### 2.1.2 Arhitectura Android

[1]Ca sistem de operare, Android are 5 componente:

* Componenta Kernel Linux- pe care este bazat acest sistem de operare
* Componenta Bibliotecilor- care conțin cod ce cuprinde cele mai importante caracteristici ale acestui sistem de operare
* Componenta Runtime Android- deține un set de librarii care permit realizarea unei aplicații Android folosind limbajul Java
* Componenta Framework a aplicației
* Componenta aplicații- care contine aplicațiile de baza ale dispozitivului( Contacts, Phone, Browser), dar si alte aplicații instalate de utilizator

### 2.1.3 Android SDK

SDK-ul(Software Development Kit) este cea mai importanta componenta a soft-ului si deține instrumentele si pachetele necesare dezvoltării unei aplicații Android funcționale.

## 2.2 Java

Android Studio ajuta la dezvoltarea aplicațiilor Android în doua limbaje de programare: Kotlin si Java. Am ales Java pentru realizarea acestei aplicații deoarece este un limbaj de programare bine cunoscut de majoritatea dezvoltatorilor de aplicații. Se găsește ușor ajutor pentru problemele întâmpinate în timpul realizării aplicației în limbajul Java, în timp ce Kotlin, un limbaj nou, nu dispune de acces la fel de facil la acesta.

Java, limbaj de programare OOP, dispune de un set de tehnologii care facilitează dezvoltarea rapida de aplicații complexe si securizate.

Fiind un limbaj de programare bine cunoscut, Java se folosește la realizarea aplicațiilor Android, având ca avantaj lipsa de complicații cauzate de pointeri. Folosind o mașină virtuala, nu este necesara recompilarea codului în cadrul fiecărui dispozitiv utilizat.

Sunt numeroase avantaje ale utilizării acestui limbaj de programare, fiind o platforma independenta si ușor de folosit, care conține librarii prin intermediul cărora se realizează rapid aplicații. Prin platforma Java se poate asigura o securitate ridicata a aplicațiilor dezvoltate si scalabilitate prin volumul mare de date suportat si capacitatea de extindere.

### 2.2.1 Caracteristici Java

[5]Este un limbaj de programare utilizat pe multe sisteme de operare. În urma compilării, codul sursa devine executabil, rezultând în problema utilizării codului pe fiecare platforma în parte. Compilarea codului în limbaj Java este compilat ca bytecode. Acesta nu se poate executa singur, pentru ca nu este cod nativ (codul care este configurat pentru a rula pe un anumit procesor, care în general nu va rula pe alt procesor, decât daca se va folosi un emulator). Astfel, poate fi executat doar pe o mașină virtuala (Java Virtual Machine). JVM este o aplicație care interpretează bytecode-ul, rezultând în disponibilitatea de a se putea utiliza pe multiple sisteme de operare.

### 2.2.2 Componentele Java

[4][5]Pentru a programa în Java este necesar un compilator pentru Java numit javac. Acesta generează fișierele de clase Java din codul sursa. Fișierele rezultate sunt scrise în bytecode si sunt independente de platforma. JVM-ul (Java Virtual Machine) încarcă aceste fișiere si interpretează bytecode-ul sau compilează in limbaj mașină folosind compilatorul JIT (Just in time).

JRE (Java Runtime Environment) este necesar pentru a executa programe si aplicații Java. Fiecare sistem de operare are un JRE propriu. JRE include următoarele componente: bibliotecile necesare pentru clase, property settings, fișierele resursa, DLL-urile, extensii Java, fișiere necesare pentru managementul securității.

JDK (Java Development Kit) este componenta principala al mediului de lucru Java. Acesta conține JRE javac, Java debugger si alte clase esentiale. Acesta este folosit pentru dezvoltarea de aplicații Java deoarece oferă atât executabilul si binery-urile, cat si instrumentele necesare compilării si debugging-ului unui program în limbajul Java.

## 2.3 Firebase

Firebase este o platforma Google care oferă servicii pentru dezvoltarea aplicațiilor web sau mobile.

Sunt necesare un cont si un proiect care se conectează la aplicația din Android Studio. Se pot grupa mai multe proiecte Android pe un proiect în Firebase, în timp ce un proiect Android Studio poate conține o singura aplicație.

Am utilizat Realtime Database pentru a stoca datele referitoare la utilizatori, tokenii de acces la cărțile stocate în Storage, genurile si titlurile cărților si documentele afișate în cadrul paginii Home.

Am folosit serviciul Firebase Authentication pentru autentificarea pe baza emailului si a parolei. Aceasta se ocupa de securitatea utilizatorilor si se pot seta permisiuni pentru fiecare utilizator.

Firebase Cloud Storage este un serviciu de stocare de obiecte oferit de Google Cloud Platform. Stocând date utilizând acest serviciu ai acces la masurile de securitate Google si dispui de posibilitatea de a gestiona într-un mod sigur datele.

Firebase Analytics este un instrument care te ajuta sa obții informații despre performanta aplicației. Acesta oferă un set complet de informații despre utilizatori si modul în care aceștia utilizează aplicația.

### 2.3.1 Firebase Authentication

Firebase Authentication este o facilitate adusa de Firebase care are ca scop autentificarea utilizatorilor pe baza tokenilor. Aceasta este ușor de integrat in aplicație si oferă servicii de backend, interfețe pentru utilizatori si SDK-uri ușor de folosit. Încadrarea în proiect a acestei facilitați se face printr-un API dat de Firebase.

Firebase Authentication prezinta mai multe opțiuni de autentificare:

* Email si parola (Firebase SDK)
* Federated identity provider integration (prin Facebook, Google, Twitter, Github, Apple)
* Număr de telefon
* Personalizata (printr-un sistem existent la SDK-ul de autentificare pentru accesul la Realtime Database si alte servicii)
* Anonima (utilizatorul nu își creează un cont, ci se creează un cont temporar, anonim, care se va actualiza într-un cont permanent în cazul în care utilizatorul se va înregistra)

Firebase Autentication dispune de 2 tipuri de autentificare:

* Firebase SDK
* FirebaseUI Auth

[8]Ambele tipuri de autentificare utilizează următoarele clase:

* FirebaseAuth Instance-pentru SDK-ul Firebase. Se folosește la crearea de conturi, conectarea sau deconectarea utilizatorilor, modificarea informațiilor utilizatorului curent.
* AuthUI Instance- conține metode de sign-in si de sign-out. Dispune de opțiuni pentru modificarea interfeței, precum tema, dar si opțiuni de autentificare a ușerului.
* FirebaseUser Classes- folosită la încapsularea datelor utilizatorului curent și la returnarea unui obiect tip user. Datele păstrate diferă prin tipul autentificării utilizate. Deține metode de actualizare a datelor utilizatorului, verificarea sa, dar si ștergerea contului din baza de date.
* Authentication Provider Class- metodele de autentificare dispun de clase care sunt utilizate în timpul procesului de autentificare pentru crearea obiectului tip AuthCredential
* AuthCredential Classes- folosită pentru autentificarea cu un cont de pe alta platforma. Aplicația primește un token de la aplicația respectiva si îl transforma într-un obiect de tip AuthCredential si îl transmite către Firebase pentru crearea unui cont.

### 2.3.2 Autentificare SDK Firebase

Acest tip de implementare nu pune la dispoziție programatorului interfața grafica pentru utilizator.[8] Pașii implementării metodei sunt următorii:

1. Alegerea tipurilor de autentificare utilizate în consola Firebase
2. Înregistrarea în aplicație a acestora
3. Adăugarea bibliotecilor SDK în Android Studio
4. Dobândirea referinței pentru FirebaseAuth
5. Implementarea si adăugarea AuthStateListener
6. Crearea interfețelor
7. Scrierea codului operațiilor efectuate pentru toate tipurile de autentificări alese
8. Trimiterea tokenilor si manipularea rezultatelor

### 2.3.3 Autentificarea FirebaseUI Auth

Acest tip de autentificare ne pune la dispoziție toate instrumentele necesare implementării autentificării. Pașii implementării metodei sunt următorii:

1. Alegerea tipurilor de autentificare utilizate în consola Firebase
2. Înregistrarea în aplicație a acestora
3. Adăugarea bibliotecilor FirebaseUI Auth în Android Studio
4. Dobândirea referintei pentru FirebaseAuth
5. Utilizarea clasei AuthUI pentru configurarea si construirea FirebaseUI-ului
6. Lansarea activității de autentificare
7. Manipularea rezultatelor

Comparând cele doua metode de autentificare Firebase se observa ca SDK Firebase nu este la fel de eficient din punct de vedere al timpului de implementare, dar oferă programatorului control complet asupra aspectului aplicației si asupra comportamentului acesteia, în timp ce FirebaseUI Auth este mai restrictiv din acest punct de vedere.

Având în vedere aceste criterii, am ales sa folosesc SDK Firebase pentru aplicația :"Book Reader". Astfel, am putut realiza o interfață grafica personalizata.

### 2.3.4 Realtime Database

[7][8]Realtime Database este o baza de date cloud, sincronizata în timp real cu fiecare client conectat. Aceasta permite datelor sa fie distribuite în tre mai multe aplicații (servere web, Android, IOS) în cel mai scurt timp posibil.

[6]Utilizând un API dispunem de funcții care permit manipularea datelor aproape in timp real si permit datelor sa fie salvate în siguranța în serverele Google. Aceste funcții de manipularea datelor ne permit sa facem următoarele :

* Inserarea componentelor în baza de date;
* Actualizarea informațiilor;
* Ștergerea datelor.

Realtime Database folosește NoSQL database (informațiile nu sunt stocate în tabele si nu se accesează cu ajutorul query-urilor SQL) si sun salvate sub forma de JSON (JavaScript Object Notation). Obiectele JSON sunt de forma cheie-valoare, unde cheia este în identificator unic pentru fiecare element salvat în baza de date si valoarea, reprezentând informația adăugată.

În cazul în care nu exista conexiune la internet, datele vor fi salvate local, putând fi accesate de pe dispozitivul respectiv. În momentul reluării conexiunii la internet, datele se vor sincroniza in baza de date principala, eliminând posibilitatea pierderii de informații sau nesincronizării între date introduse offline si revenirea în online.

Platforma poate susține milioane de utilizatori simultan. Aceasta oferă o metoda securizata de păstrare a datelor într-o baza de date cloud, printr-un număr redus de linii de cod. Viteza în care datele sunt sincronizate între aplicații este aproape instantă.

Folosind aceste caracteristici ale Firebase-ului am folosit Realtime Database pentru a stoca : tokenii necesari accesului la documentele pdf utilizate, tokenii coperților, gestionarea în funcție de gen , afișarea de cărți pe pagina Home si datele utilizatorilor.

### 2.3.5 FirebaseUI Database

FirebaseUI Database este o biblioteca open-source pentru Android care permite programatorului sa se conecteze rapid la elementele de interfață la un API Firebase. Astfel se pot vedea în timp real schimbările făcute în Realtime Database si aduce interfețe simple pentru activități uzuale, precum afișarea de liste sau colecții de obiecte.

FirebaseUI Database oferă următoarele funcții:

* FUITableViewDataSource-Data source care leagă o interogare Firebase de un UITableView
* FUICollectionViewDataSource-Data source care leagă o interogare Firebase de un UICollectionView
* FUIIndexCollectionViewDataSource-Data source care populează un collection view cu date indexate din Firebase Database
* FUIIndexTableViewDataSource-Data source care populează un table view cu date indexate din Firebase Database
* FUIArray-Mentine un array sincronizat cu o interogare Firebase
* FUISortedArray-Un array sincronizat sa își ordoneze automat conținutul
* FUIIndexArray-Mentine un array sincronizat cu datele indexate de la doua referințe Firebase

### 2.3.6 Storage

Cloud Storage este o funcție Firebase care are ca scop stocarea datelor pentru dezvoltatorii de aplicații. Aceasta poate păstra imagini, video-uri, audio-uri, documente etc. Unul din atuurile funcției este securitatea manipulării si păstrării datelor, asigurata de Google, indiferent de calitatea internetului.

Ca proprietăți principale regăsim:

* Operații robuste-SDK-urile Firebase pentru Cloud Storage efectuează încărcări si descărcări indiferent de calitatea internetului. Astfel, aceste operații se reiau de unde s-au oprit, reducând timpul de execuție al operațiilor.
* Securitate puternica- SDK-urile Firebase pentru Cloud Storage se integrează cu Firebase Authentication pentru a aduce programatorilor autentificări simple si intuitive. Se poate utiliza modelul de securitate declarative pentru a permite accesul bazat pe numele fișierului, dimensiune, tip de conținut etc.
* Scalabilitate-Cloud Storage este realizat prin conceptul de creștere al aplicației dezvoltate. Acesta permite evoluția aplicației, păstrând aceeași infrastructura.

Cloud Storage păstrează fișierele într-un bucket Google Cloud Storage, făcându-le accesibile si prin Firebase si prin Google Cloud. Acest lucru oferă programatorului flexibilitatea de a manipula datele pentru aplicații mobile prin SDK Firebase pentru Cloud Storage. În plus, se poate realiza procesare prin server precum filtrare de imagini si video transcoding folosind Google Cloud Storage API. SDK Firebase pentru Cloud Storage se integrează ușor cu Firebase Authentication pentru identificarea utilizatorilor si oferă un limbaj declarativ securizat care permite dezvoltatorilor de aplicații sa acceseze si sa controleze fișiere individuale sau grupări de fișiere, putând astfel sa le faci publice sau private.

### 2.3.7 Firebase Analytics

Firebase Analytics este o componenta Google Analytics care se poate integra într-o aplicație Android prin SDK Firebase. Aceasta furnizează rapoarte care te ajuta sa înțelegi mult mai bine comportamentul utilizatorilor în timpul utilizării aplicației, astfel dezvoltatorul poate face decizii informate cu privire la marketingul aplicației si la optimizările de performanta.

SDK-ul cuprinde un număr de evenimente si proprietăți ale utilizatorilor si te ajuta sa iți definești evenimente personalizate pentru a măsura elementele legate de activitatea întreprinsă. După acumularea datelor, acestea sunt valabile în dashboard-ul Firebase. Acesta dispune de o reprezentare detailată a analizei datelor, de la scheme sumare pentru utilizatorii activi si date demografice, pana la informații detaliate precum identificarea obiectelor cela mai bine vândute.

Analytics integrează si caracteristici din Firebase. De exemplu, se pot înregistra automat evenimente care corespund mesajelor de notificare trimise prin Notifications composer si prelucrează rapoarte în privința impactului fiecăruia.

## 2.4 APIs utilizate

Am utilizat mai multe API-uri pentru afișarea imaginilor si a documentelor stocate în Firebase Database, preluându-le tokenii din Realtime Database. Astfel, toate elementele ce țin de fișiere media se pot schimba in timp real din baza de date cloud, reducând din dimensiunea aplicației si ajutând la extinderea acesteia din punctul de vedere al volumului de date. Elementele afișate în aplicație se auto generează în funcție de informațiile găsite în baza de date, imaginile si documentele afișate putând fi schimbate fără nicio modificare asupra codului.

### 2.4.1 PDF Page Viewer

PDF Page Viewer este un widget Android open source dezvoltat de voghDev utilizata pentru afișarea documentelor tip PDF în activități sau fragmente.

Acest API poate afișa documente aflate fie local, în assets, fie remote, printr-un link către zona de stocare a acestuia. În acest fel am putut folosi tokenii generați de Firebase Storage, preluați din Realtime Database pentru a încărca în aplicație documentele dorite de tip PDF.

Pentru utilizarea url-ului la încărcarea pdf-ului, trebuie adăugate în AndroidManifest.xml permisiuni pentru internet, scrierea pentru external storage si citirea pentru external storage.

Widget-ul permite schimbarea paginilor pe orizontala, verticala, zoom în si zoom out (ambele prin atingere dubla sau pinch).

### 2.4.2 Image Slideshow

Image Slideshow este o biblioteca dezvoltata de denzcoskun, utilizata pentru afișarea imaginilor într-un slideshow. Aceasta permite schimbarea automata a imaginilor după un timp ce care îl poate seta dezvoltatorul, adăugarea unui titlu specific fiecărei imagini, setarea unui corner radius, scalarea imaginilor în cadrul dimensiunilor specificate in interfață.

Încărcarea imaginilor se poate face local sau prin url. Afișarea acestora în cadrul elementului din interfață se poate face în mai multe feluri:

* centerCrop
* centerInside
* fit
* No Scale

### 2.4.3 Glide

Glide este dezvoltat de bumptech si reprezintă un open source framework de management de media si încărcare de imagine pentru Android. Acesta cuprinde decodarea media, caching de memorie si disk si resource pooling.

Scopul frameworkului este de a încărca o lista de imagini cat mai rapid si calitativ posibil, putând astfel sa susțină operațiile de încărcare, redimensionare si afișare a unei imagini remote.

# 3.DESCRIEREA SISTEMULUI

## 3.1 Autentificarea

Atunci când deschide aplicația, utilizatorul are de ales între trei opțiuni:

* Sa se autentifice (daca contul este înregistrat si validat);
* Sa își creeze un cont nou;
* Sa își recupereze parola.

|  |
| --- |
| signUpButton.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  openRegister();  }  });  signInButton= findViewById(R.id.sign\_in);  signInButton.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  **if** (rememberMe.isChecked()){  editor.putBoolean("checked",true);  editor.apply();  StoreDataUsingSharedPref(email,password);  loginUser();  }  **else** {  getSharedPreferences(FILE\_EMAIL,MODE\_PRIVATE).edit().clear().commit();  loginUser();  }  }  });  forgotPassword = findViewById(R.id.forgot\_password\_link);  forgotPassword.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  openForgotPassword();  }  }); |

Figura 3.1: Opțiunile la deschiderea aplicației

Codul din *Figura 3.1* realizează îndeplinirea acestor acțiuni.

Daca utilizatorul apasă pe butonul de înregistrare, se va apela funcția *openRegister*, care va deschide pagina de înregistrare.

Daca utilizatorul dorește resetarea parolei, se va apela funcția *forgotPassword*, care va deschide pagina de resetare a parolei.

În momentul în care utilizatorul dorește sa se autentifice, se verifica daca opțiunea de reținere a datelor este bifata sau nu, pentru a păstra pentru sesiunea următoare datele introduse și apoi se apelează funcția de *loginUser*.

### 3.1.1 Pagina de autentificare

Pagina de autentificare se ocupa cu realizarea unei legături cu restul funcțiilor de manipulare a datelor utilizatorului, dar si verificarea si validarea datelor, acțiuni urmate de deschiderea restului aplicației.

**Funcția de autentificare**

|  |
| --- |
| **public void** loginUser(){  String em, pass;  em=email.getText().toString().trim();  pass=password.getText().toString().trim();  **if** (!em.equals("") && !pass.equals("")){  mAuth.signInWithEmailAndPassword(em,pass).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  @Override  **public void** onComplete(@NonNull Task<AuthResult> task) {  if(task.isSuccessful()){  FirebaseUser user= FirebaseAuth.getInstance().getCurrentUser();  **if** (user.isEmailVerified()){  Toast.makeText(LogIn.this,"User logged in successfully", Toast.LENGTH\_SHORT).show();  Intent intent = new Intent(LogIn.this, com.example.bookreader.AppPages.Home.class);  intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  }  **if** (!user.isEmailVerified()){  user.sendEmailVerification();  Toast.makeText(LogIn.this,"Check your email to verify your account!", Toast.LENGTH\_SHORT).show();  }  }  **else** {  Toast.makeText(LogIn.this,"Account doesn't exist or credentials do not match", Toast.LENGTH\_SHORT).show();  }  }  });  }  **if** (em.equals("") || pass.equals("")){  Toast.makeText(LogIn.this,"You must fill all credentials", Toast.LENGTH\_SHORT).show();  }  } |

Figura 3.2: Validarea datelor

În *Figura 3.2* se observa preluarea datelor introduse de utilizator, iar după se verifica.

Prima data se verifica daca ambele câmpuri sunt completate. Daca nu sunt completate, vom primi mesajul *" You must fill all credentials ".*

În pasul următor se caută emailul si parola in Firebase Authentication. În cazul în care acestea sunt găsite se trece la pasul de verificare a validității contului. În cazul contrar, se va primi notificarea *" Account doesn't exist or credentials do not match ".*

Daca contul exista, se verifica daca acesta este validat sau nu. În cazul în care contul nu este validat se trimite un email de validare si primim o notificare pentru a verifica mail-ul și pentru a valida contul. După încheierea procesului de validare se poate continua către restul aplicației cu datele introduse. Astfel se va deschide pagina *Home* când poate condițiile menționate anterior sunt îndeplinite.

### 3.1.2 Pagina de înregistrare

Pagina de înregistrare este alcătuita din patru câmpuri unde se introduc:

* username-ul;
* email-ul;
* parola;
* rescrierea parolei.

De asemenea, avem si opțiunea de resetare a parolei.

|  |
| --- |
| submitUpButton = findViewById(R.id.submit\_data);  submitUpButton.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  registerUser();  }  });  forgotPassword = findViewById(R.id.forgot\_password\_link2);  forgotPassword.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  openForgotPassword();  }  }); |

Figura 3.3: Funcțiile paginii Register

În *Figura 3.3* se observa ca în urma apăsării pe *"Forgot Password"* se va apela funcția *openForgotPassword* , iar la apăsarea butonului *Register* se va apela funcția *registerUser*, care va realiza procesul de înregistrare.

**Funcția de înregistrare**

|  |
| --- |
| **private void** registerUser() {  EditText username, email, password, confirmPassword;  username = (EditText) findViewById(R.id.register\_username\_input);  email = (EditText) findViewById(R.id.register\_email\_input);  password = (EditText) findViewById(R.id.register\_password\_input);  confirmPassword = (EditText) findViewById(R.id.register\_repeat\_password\_input);  String uname, em, pass, cpass;  uname = username.getText().toString().trim();  em = email.getText().toString().trim();  pass = password.getText().toString().trim();  cpass = confirmPassword.getText().toString().trim();  **if** (!uname.equals("") && !em.equals("") && !pass.equals("") && !cpass.equals("")) {  **if** (pass.equals(cpass)) {  mAuth.createUserWithEmailAndPassword(em, pass)  .addOnCompleteListener(new OnCompleteListener<AuthResult>() {  @Override  **public void** onComplete(@NonNull Task<AuthResult> task) {  **if** (task.isSuccessful()) {  User user = new User(uname,em);  FirebaseDatabase.getInstance()  .getReference("Users")  .child (FirebaseAuth.getInstance()  .getCurrentUser()  .getUid())  .setValue(user)  .addOnCompleteListener(new OnCompleteListener<Void>() {  @Override  **public void** onComplete(@NonNull Task<Void> task) {  **if** (task.isSuccessful())  {  Toast.makeText(Register.this, "User registered successfully", Toast.LENGTH\_SHORT).show();  Intent intent = new Intent(Register.this, LogIn.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  }  }  });  }  **else** {  Toast.makeText(Register.this, "Account already exists", Toast.LENGTH\_SHORT).show();  }  }  });  }  **if** (!pass.equals(cpass))  Toast.makeText(Register.this, "The passwords do not match", Toast.LENGTH\_SHORT).show();  }  **if** (uname.equals("") || em.equals("") || pass.equals("") || cpass.equals(""))  Toast.makeText(Register.this,"You must fill all credentials", Toast.LENGTH\_SHORT) .show();  } |

Figura 3.4: Funcția de înregistrare

După cum se poate observa în *Figura 3.4*, la fiecare apel al funcției se preiau valorile din câmpurile paginii si se verifica datele introduse în acestea.

Primul pas este de a verifica daca toate câmpurile date sunt completate. Astfel, daca rămâne un câmp necompletat vom primi mesajul *" You must fill all credentials ".*

Al doilea pas este de comparare a celor doua câmpuri în care introducem parola, respectiv repetarea parolei. Acestea trebuie sa fie identice pentru a trece la pasul următor. Altfel, vom primi mesajul *"The passwords do not match"* .

După ce ne asiguram ca toate câmpurile sunt completate si cele doua parole coincid, urmează verificarea existentei unui alt cont cu același email. Acest lucru se face cu ajutorul lui Firebase Authentication, care nu permite, prin setările făcute în platforma de pe browser, ca doua conturi sa existe cu aceeași adresa de email, așa cum se observa în *Figura 3.5.*

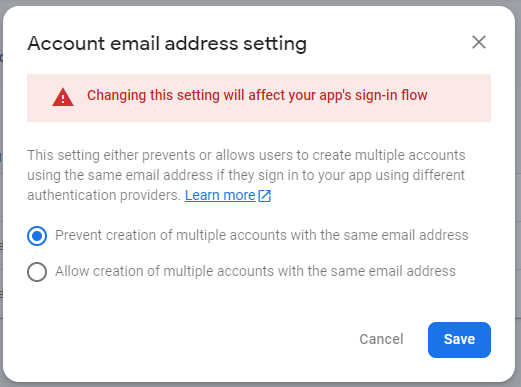


Figura 3.5: Crearea de conturi cu aceeași adresa de email

În urma acestor setări, se poate concluziona faptul ca imposibilitatea de a realiza înregistrarea unui cont în Firebase Authentication se datorează existentei în prealabil al unui cont înregistrat cu aceeași adresa de email. În aceasta situație se afișează mesajul *"Account already exists".*

Pentru realizarea înregistrării se creează un obiect nou tip *User* si se populează cu informațiile introduse anterior.

Se face o cerere pentru actualizarea datelor din tabela *Users* în care se adaugă o instanță noua cu datele introduse. În urma completării acestei acțiuni, vom primi mesajul *"User registered successfully"*.

### 3.1.3 Pagina de recuperare a parolei

În aceasta pagina se afla un câmp unde utilizatorul trebuie sa introducă un email valid pentru a-si reseta parola.

|  |
| --- |
| submitUpButton.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  openLogIn();  }  }); |

Figura 3.6: Resetarea parolei

Pentru a începe procesul de resetare a parolei, se apăsa pe butonul de *Reset Password* ,care apelează funcția *openLogIn*.

|  |
| --- |
| **public void** openLogIn(){  EditText email ;  email = (EditText)findViewById(R.id.forgot\_password\_email\_input);  String em;  em=email.getText().toString().trim();  **if** (em.equals("")) {  email.setError("Email is required!");  email.requestFocus();  return;  } |

Figura 3.7: Verificarea conținutului

La apelarea funcției se preia emailul introdus anterior si se vor executa mai multe verificări acestuia. În primul rând se verifica daca câmpul este completat. În cazul în care acesta este gol se va primi mesajul *"Email is required!".* Apoi se va supune unei verificări de tip Regex .

|  |
| --- |
| **if** (!Patterns.EMAIL\_ADDRESS.matcher(em).matches()){  email.setError("Please provide a valid email!");  email.requestFocus();  return;  } |

Figura 3.8: Verificarea Regex

*"Patterns.EMAIL\_ADDRESS.matcher(em).matches()"*

Aceasta compara datele introduse cu șablonul unei adrese de email. Daca funcția returnează *False*, atunci datele introduse nu urmează forma unui email si se primește mesajul *"Please provide a valid email!"* .

|  |
| --- |
| mAuth.sendPasswordResetEmail(em).addOnCompleteListener(new OnCompleteListener<Void>() {  @Override  **public void** onComplete(@NonNull Task<Void> task) {  **if** (task.isSuccessful()){  Toast.makeText(ForgotPassword.this, "Check your email to reset your password!", Toast.LENGTH\_SHORT).show();  Intent intent=new Intent(ForgotPassword.this,LogIn.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  }  **if** (!task.isSuccessful()){  Toast.makeText(ForgotPassword.this, "Try again!Something wrong happened!", Toast.LENGTH\_SHORT).show();  }  }  }); |

Figura 3.9: Funcția de resetare a parolei

În cazul în care s-a introdus un tip de email valid, urmează pasul de căutare al emailului în Firebase Authentication. Pentru a face acest lucru, se trimite adresa de email prin funcția *sendPasswordResetEmail* si daca apelul este reușit, se va primi mesajul *"Check your email to reset your password!"* si un email conținând un link pentru resetarea parolei. Apoi se va deschide pagina de autentificare.

## 3.2 Pagina principală

Pagina de *Home* este pagina care se deschide imediat dupa autentificarea utilizatorului. Aceasta este alcatuita din doua componente: meniul si prezentarea cartilor. Aceasta are rolul de a sugera utilizatorului carti pentru a-si largi orizonturile literare.

### 3.2.1Meniul

Meniul este prezent pe toate paginile utilizatorului autentificat. Acesta este modul în care se realizează trecerea între paginile principale ale aplicației.

Implementarea acestuia se face printr-un *BottomNavigationView*, care preia din resurse fișierul *item\_menu.xml*. Acesta conține cele cinci butoane introduse prin Vector Assets în folderul *res/drawable*.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <menu xmlns:android="http://schemas.android.com/apk/res/android">  <item android:id="@+id/nav\_home"  android:title="@string/home\_text"  android:icon="@drawable/ic\_home"/>  <item android:id="@+id/nav\_genres"  android:title="@string/genres\_text"  android:icon="@drawable/ic\_genres"/>  <item android:id="@+id/nav\_search"  android:title="@string/search\_text"  android:icon="@drawable/ic\_search"/>  <item android:id="@+id/nav\_profile"  android:title="@string/profile\_text"  android:icon="@drawable/ic\_profile"/>  <item android:id="@+id/nav\_settings"  android:title="@string/settings\_text"  android:icon="@drawable/ic\_settings"/>  </menu> |

Figura 3.10: Realizarea meniului in xml

Acest meniu se introduce în *BottomNavigationMenu* prin *app:menu="@menu/item\_menu".* Se poate observa in *Figura 3.9* această atribuire în *home.xml*.

|  |
| --- |
| <com.google.android.material.bottomnavigation.BottomNavigationView  android:id="@+id/home\_bottom\_navigation"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:layout\_margin="30dp"  android:elevation="2dp"  app:menu="@menu/item\_menu"  android:background="@drawable/round\_corners"  app:itemRippleColor="@android:color/transparent"  app:itemIconSize="30dp"  app:labelVisibilityMode="unlabeled"  app:itemIconTint="@drawable/item\_selector"/> |

Figura 3.11: Utilizarea meniului

Funcționalitatea acestuia se realizează în Home.java, unde se integrează deschiderea paginilor caracterizate anterior, cu excepția paginii curente, care nu prezinta modificare la apăsare.

|  |
| --- |
| navigationView=findViewById(R.id.home\_bottom\_navigation);  navigationView.setOnItemSelectedListener(item -> {  Intent intent;  **switch** (item.getItemId()){  case R.id.nav\_home:  break;  **case** R.id.nav\_genres:  intent=new Intent(Home.this, Genres.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  break;  **case** R.id.nav\_search:  intent=new Intent(Home.this, Search.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  break;  **case** R.id.nav\_profile:  intent=new Intent(Home.this, Profile.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  break;  **case** R.id.nav\_settings:  intent=new Intent(Home.this, Settings.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  startActivity(intent);  break;  }  return true;  }); |

Figura 3.12: Deschiderea paginilor din meniu

Codul din *Figura 3.10* este asemănător si în fișierele *java* ale celorlalte pagini, cu o implementare precum cea din figura.

### 3.2.2 Prezentarea cărților

Pe lângă meniu, pagina *Home* cuprinde mai multe prezentări de cărți, alese după diferite criterii.

Primul mod de prezentare este printr-un Image Slideshow, adăugat cu ajutorul unui API. Acesta se instalează adăugând la modulul *build.gradle*, în *dependencies,* *"implementation 'com.github.denzcoskun:ImageSlideshow:0.1.0'".*

În *home.xml* se adaugă folosind tagul *<com.denzcoskun.imageslider.ImageSlider>, așa cum se poate vedea în Figura 3.11* si cu ajutorul unui widget *Card View.*

|  |
| --- |
| <androidx.cardview.widget.CardView  android:layout\_width="320dp"  android:layout\_height="420dp"  android:layout\_centerHorizontal="true"  app:cardUseCompatPadding="true">  <com.denzcoskun.imageslider.ImageSlider  android:id="@+id/home\_first\_image\_slider"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  app:iss\_auto\_cycle="true"  app:iss\_period="1000"  app:iss\_delay="0"/>  </androidx.cardview.widget.CardView> |

Figura 3.13: Image Slideshow în home.xml

Acesta este ciclic, astfel încât sa reia de la început imaginile atunci când trece de ultima poza.

În Home.java se implementează folosind *ImageSlider*. Pentru a putea fi ușor de schimbat documentele prezentate, am optat pentru preluarea imaginilor si a documentelor din baza de date Realtime Database. Acolo am creat o categorie speciala pentru agest slideshow, unde am introdus titlul cărții, genul literar în care se încadrează, linkul către imaginea de coperta si linkul către documentul tip *pdf*, stocate în Firebase Storage. Aceste linkuri sunt tokeni de acces prin care se preiau documentele pentru afișare, respectiv deschidere.

Pentru a prelua din baza de date remote aceste documente si imagini, este nevoie ca în AndroidManifest.xml sa se introducă permisiunile pentru acces la internet , așa cum se observa în *Figura 3.12*.

|  |
| --- |
| <uses-permission android:name="android.permission.INTERNET"/>  <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/> |

Figura 3.14: Permisiunile de acces la internet

Cu ajutorul unei liste de tipul *SlideModel* am introdus în slideshow coperta si titlul, așa cum se regăsesc în baza de date remote.

Pentru a deschide la click documentul asociat cu coperta aleasa, am introdus un *ItemClickListener*, care deschide o pagina noua cu afișarea pdf-ului corespunzător, comparând titlurile din baza de date cu titlul documentului. Atunci când se găsește elementul căutat se creează un intent de tip Document în care se adaugă prin funcția *putExtra* titlul, linkul către documentul din Firebase Storage si genul literar al acestuia.

|  |
| --- |
| final List<SlideModel> remoteimages=new ArrayList<>(); FirebaseDatabase.getInstance().getReference().child("HomeImageSlideshow").addListenerForSingleValueEvent(new ValueEventListener() {  @Override  **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  **for** (DataSnapshot data:dataSnapshot.getChildren()) {  remoteimages.add(new SlideModel(Objects.requireNonNull  (data.child("cover").getValue()).toString(), Objects.requireNonNull(data.child("title")  .getValue()).toString(), ScaleTypes.FIT));  }  mainslider.setImageList(remoteimages, ScaleTypes.FIT);  mainslider.setItemClickListener(new ItemClickListener() {  @Override  **public void** onItemSelected(int i) {  String title=remoteimages.get(i).getTitle().toString();  **for** (DataSnapshot data:dataSnapshot.getChildren()){ **if**(Objects.requireNonNull(data.child("title").getValue()).toString().equals(title)){  String cover = Objects.requireNonNull(data.child("cover") .getValue()).toString();  String url = Objects.requireNonNull(data.child("url") .getValue()).toString();  String genre = Objects.requireNonNull(data.child("genre") .getValue()).toString();  Intent intent = new Intent(Home.this, Document.class);  intent.putExtra("title", title);  intent.putExtra("url", url);  intent.putExtra("genre", genre);  startActivity(intent);  }  }  }  });  }  @Override  **public void** onCancelled(@NonNull DatabaseError error) {  }  }); |

Figura 3.15: Adăugarea imaginilor si deschiderea documentelor

Al doilea mod de prezentare de cărți este prin *HorizontalScrollView*, la care adăugam *ImageViews* generate din elementele găsite în categoria asociata în Realtime Database.

Conectându-ne la Realtime Database, preluam pentru fiecare categorie titlul, genul si linkurile către coperta si către documentul pdf. Vom încărca cu ajutorul API-ului *Glide* imaginile în *ImageView*. Acesta se instalează adăugând la modulul build.gradle, în dependencies, *" implementation 'com.github.bumptech.glide:glide:4.8.0'* ".

După Preluarea datelor din baza de date se aleg parametrii si tipul de scalare al imaginii. Acest lucru se face prin funcțiile *setLayoutParams*, unde setam lățimea imaginii si înălțimea sa. Apoi , prin *setScaleType* ne asiguram ca imaginea preluata din baza de date se încadrează in parametrii pe care i-am atribuit în linia anterioara si acordam funcției *setAdjustViewBounds* valoarea *true*. Pentru a avea o distanțare între documente am folosit *setPadding* cu valoarea doi pe toate direcțiile. Se adaugă un *OnClickListener* pentru deschiderea documentului într-o pagină nouă.

|  |
| --- |
| FirebaseDatabase.getInstance().getReference().child("HomePopular").addListenerForSingleValueEvent(new ValueEventListener() {  @Override  **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  **for** (DataSnapshot data:dataSnapshot.getChildren()) {  String title = Objects.requireNonNull(data.child("title") .getValue()).toString();  String cover = Objects.requireNonNull(data.child("cover") .getValue()).toString();  String url = Objects.requireNonNull(data.child("url").getValue()).toString();  String genre = Objects.requireNonNull(data.child("genre") .getValue()).toString();  ImageView image =new ImageView(Home.this);  image.setLayoutParams(new ViewGroup.LayoutParams(350, ViewGroup.LayoutParams.MATCH\_PARENT));  image.setScaleType(ImageView.ScaleType.CENTER\_CROP);  image.setAdjustViewBounds(true);  image.setPadding(2,2,2,2);  image.setOnClickListener(new View.OnClickListener() {  @Override  **public void** onClick(View v) {  Intent intent = new Intent(Home.this, Document.class);  intent.putExtra("title", title);  intent.putExtra("url", url);  intent.putExtra("genre", genre);  startActivity(intent);  }  });  RequestOptions requestOptions=new RequestOptions();  Glide.with(Home.this).load(cover).apply(requestOptions).into(image);  firstLinearLayout.addView(image);  }  }  @Override  **public void** onCancelled(@NonNull DatabaseError error) {} }); |

Figura 3.16: Inserarea de imagini în HorizontalScrollView

Folosind Realtime Database, toate schimbările care se fac în baza de date se aplica în timp real în aplicație, elementele modificându-se foarte ușor de la distanta. Prezentarea cărților este adaptabila la datele introduse in baza de date, indiferent de numarul elementelor găsite la căutare. Toate imaginile si documentele tip pdf sunt stocate în Firebase Storage pentru a asigura securitatea datelor si pentru a reduce dimensiunea aplicației, unele cărți având dimensiuni considerabil mai mari fata de altele. Firebase Storage oferă accesul la elementele stocate doar prin tokenii de acces pe care i-am generat si adăugat în Realtime Database.

Am folosit aceasta metoda de afișare pentru trei tipuri de categorii: Popular, Recent si Discover. Fiecare are propria sa categorie în Realtime Database.



Figura 3.17: Organizarea în Realtime Database

Am ales introducerea elementelor prin numere pentru a asigura ordinea dorita la afișare. Realtime Database ordonează alfabetic automat, iar introducerea elementelor prin titlurile lor ar însemna prezentarea cărților într-un mod pe care nu ni l-am dori întotdeauna.

Pentru a putea accesa datele din Realtime Database, trebuie sa setam anumite reguli pe platforma din browser. Pentru a controla cine are accesul la aceste date si pentru securitatea aplicației, vom acorda drepturi de modificare si citire a datelor din Realtime Database doar utilizatorilor autentificați. În pagina de *Rules* am introdus codul din *Figura 3.18* .

|  |
| --- |
| { "rules": {  ".read": "auth != null",  ".write": "auth != null",  }} |

Figura 3.18: Regulile de acces la baza de date

## 3.3 Genurile de cărți

Unul din cele mai importante moduri de a realiza o ordonare a documentelor este prin împărțirea lor în categorii. Astfel, se pot caută mult mai ușor obiecte ale viitoarelor lecturi, alegând genul literar pe care utilizatorul dorește sa îl citească.

Cu acest scop am implementat o pagina de prezentare a categoriilor, care trimite mai departe cititorul către toate cărțile aparținând genului ales.

### 3.3.1 Alegerea categoriei

Apăsând în meniul din partea de jos a paginii suntem trimiși către pagina care cuprinde toate genurile literare pe care le avem in aplicație. Acestea sun prezentate într-un *Recycler View* , împărțit pe doua coloane folosind *GridLayoutManager* cu *spanCoun*t cu valoarea 2, ce conține elemente de tipul *item\_books.xml*. Folosind o lista de tip *CategoryModel* în care adăugam toate categoriile afișate (fantasy, lifestyle, fashion, romance, mystery, historic, sci-fi, classic, modern, horror, young adult, thriller, manga) si un adapter care preia aceasta lista pentru a popula pagina.

*CategoryAdapter* creează obiecte tip *View* cu ajutorul listei trimise ca parametru la constructor. Acest adapter se trimite către *Grid View* pentru a încarcă obiectele în pagina.

|  |
| --- |
| @Override  **public View** getView(int position, View convertView, ViewGroup parent) {  View myView;  if(convertView == null){  myView= LayoutInflater  .from(parent.getContext())  .inflate(R.layout.genre\_item\_layout,parent,false);  }  **else{**  myView=convertView;  }  TextView genreName=myView.findViewById(R.id.genre\_name);  genreName.setText(genre\_list.get(position).getName());  return myView;  } |

Figura 3.19: Crearea unui View

Se vor afișa într-un *Recycler View*, în ordinea pe care am dat-o în lista inițială, toate genurile literare introduse.

Pentru a merge mai departe către documentele aparținând genului respectiv am implementat un *OnItemClickListener* pentru *Grid View*, unde , folosind un *switch*, am descris cazurile apăsării fiecărui element.

Folosind un *String Static* preluam categoria aleasa pentru a o folosi în pagina următoare si deschidem pagina *Genre List* printr-un nou intent.

La fel ca în cazul paginii Home, avem si aici un meniu in partea inferioara a paginii, care ne face legătura cu restul paginilor principale ale aplicației. În cazul de față, avem funcții implementate pentru apăsarea butonului de Home, pe când butonul de Genres nu are.

### 3.3.2 Filtrarea cărților după categoria aleasa

În urma alegerii unui gen de carte se va deschide o pagina noua unde se afișează toate cărțile aparținând acelui gen. Acest lucru se face păstrând valoarea din clasa *Genres.java* si folosind-o in *GenreList.java*.

În cadrul funcției *onCreate* din *GenreList.java* se parcurg obiectele din baza de date, din categoria *Books*, se extrage genul fiecăruia si se compara cu variabila reținută în clasa Genres.java. Cele care, în urma comparatiei, returneaza valoarea *True*, sunt pastrate într-o lista de tip *Books*.

Se declara un adapter de tipul *BookListAdapterGenres* la care se trimite lista ca parametru.

Acest adapter asigura afișarea cărților în pagina, în doua coloane, fiecare obiect conținând coperta si numele documentului. Acestui obiect i se mai asociază si link-ul pentru deschiderea pdf-ului.

## 3.4 Deschiderea documentului

Acțiunea de deschidere a documentelor se realizează într-o pagina separata, în Document.java si ViewPDF.java. În clasa Document.java declaram un ProgressBar care se afișează la începerea activității si se oprește atunci când se încheie încărcarea cărții.

Folosind un API pentru vizualizarea documentelor pdf încărcam în aplicație cărțile prin url.

PdfViewPager este un API dezvoltat de voghDev, instalat prin inserarea în fișierul build.gradle, in dependencies a liniei din figura 3.20.

|  |
| --- |
| implementation 'es.voghdev.pdfviewpager:library:1.1.3' |

Figura 3.20: Instalarea API-ului PdfViewPager

Acest API se implementează în document.xml prin codul din figura 3.21 si un linear layout.

|  |
| --- |
| <androidx.viewpager.widget.ViewPager  android:id="@+id/document\_book"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"/> |

Figura 3.21: Implementarea in document.xml

Linear layout-ul este trimis ca parametru pentru crearea unui ViewPDF împreuna cu link-ul către document, luat din Bundle, împreuna titlul si genul cărții, si cu progress bar-ul menționat anterior.

În *ViewPDF.java* se încarcă documentul de tip pdf prin link-ul trimis în *Document.java* folosind un PDFPageAdapter aparținând API-ului utilizat. Se extrage cartea folosind funcția *FileUtil.extractFileNameFromURL(url)).*

|  |
| --- |
| **public** ViewPDF(String url, LinearLayout pdfLayout, ProgressBar loader, Activity activity){  DownloadFile.Listener listener=new DownloadFile.Listener() {  @Override  **public void** onSuccess(String url, String destinationPath) {  pagerAdapter=new PDFPagerAdapter(activity, FileUtil .extractFileNameFromURL(url));  remotePDFViewPager.setAdapter(pagerAdapter);  refreshLayout(pdfLayout);  loader.setVisibility(View.GONE);  }  @Override  **public void** onFailure(Exception e) {  }  @Override  **public void** onProgressUpdate(int progress, int total) {  }  };  remotePDFViewPager=new RemotePDFViewPager(activity,url,listener);  }  **private void** refreshLayout(LinearLayout pdfLayout) {  pdfLayout.addView(remotePDFViewPager  ,LinearLayout.LayoutParams.MATCH\_PARENT  ,LinearLayout.LayoutParams.MATCH\_PARENT);  } |

Figura 3.22: Încărcarea documentului

Folosind biblioteca RemotePDFViewer din API, aplicam adapterul creat anterior, apelam funcția refreshLayout, care afișează în pagina documentul si setam vizibilitatea progress bar-ului la GONE.

## 3.5 Opțiunea de căutare

Una dintre cele mai importante funcții ale aplicației este reprezentata de căutarea cărții dorite. Aceasta pagina este accesata prin alegerea paginii de Search din cadrul meniului, marcata printr-o lupă.

Pagina *search.xml* este alcătuită din widgetul SearchView si din widgetul RecyclerView. În cadrul widgetului de SearchView se pot introduce cuvintele cheie pentru găsirea documentului căutat după titlu, iar în cel de-al doilea widget se afișează, pe doua coloane, cărțile care conțin String-urile introduse în bara de search.

Căutarea si afișarea în pagina sunt implementate în Search.java cu ajutorul unui adapter numit *BookListAdapterSearch.java*. Se declara în *Search.java* adapter-ul de tipul menționat anterior si se implementează o funcție *onDataChange* pentru SearchView. Aceasta trimite către adapter la fiecare modificare a textului o noua lista de obiecte de tip Books cu elementele găsite inițial, folosind un String gol, reprezentând toate cărțile din baza de date. La adăugarea unui String nou in SearchView se apelează funcția de filtrare a adapter-ului.

BookListAdapterSearch.java implementează o funcție de tip *Filter* , regăsită în figura 3.23, care realizează filtrarea elementelor din lista. Prima verificare se face asupra lungimii String-ului din SearchView. Daca acesta este zero, se va afișa lista cu toate cărțile din baza de date. În cazul contrar, se va verifica ce titluri conțin String-ul introdus. Acest lucru se face prin funcția *contains* aplicata titlurilor si cu trimiterea String-ului ca parametru. Funcția va verifica daca parametrul trimis se regăsește în titlu. În cazul adevărat se va adăuga obiectul de tip Book unei liste goale, altfel se va trece la elementul următor.

La finalul parcurgerii listei inițiale, se va lua lista nou creata si aceasta se va afișa în pagina, în RecyclerView, prin funcția *publishResults*.

Aceasta funcție de filtrare se va apela la fiecare modificare a textului introdus în SearchView, iar la colirea acestuia se va popula lista filtrata cu lista inițiala, care conține toata elementele din tabelul Books din Realtime Database.

Afișarea din widgetul RecyclerView se face precum în pagina *grenre\_list.xml,* în două coloane, fiecare obiect având titlul si coperta ca identificatori ai cărții si un link asociat pentru deschiderea documentului.

|  |
| --- |
| **private final Filter** bookFilter=new Filter() {  @Override  protected FilterResults performFiltering(CharSequence constraint) {  ArrayList<Books> filteredBooksList=new ArrayList<>();  **if** (constraint == null || constraint.length()==0){  filteredList.clear();  filteredList.addAll(list);  filteredBooksList.addAll(list);  }  **else**{  String filterPattern=constraint.toString().toLowerCase().trim();  **for** (Books books: list){  **if** (books.getTitle().toLowerCase().contains(filterPattern)){  filteredBooksList.add(books);  }  }  }  FilterResults results=new FilterResults();  results.values=filteredBooksList;  results.count=filteredBooksList.size();  return results;  }  @SuppressLint("NotifyDataSetChanged")  @Override  **protected void** publishResults(CharSequence constraint, FilterResults results) {  filteredList.clear();  filteredList.addAll((ArrayList) results.values);  notifyDataSetChanged();  }}; |

Figura 3.23: Filtrarea căutării

## 3.6 Deconectarea contului

În secțiunea *Profile* a aplicației se poate utiliza funcția de deconectare a contului. Pagina conține o recunoaștere a username-ului utilizatorului curent si un buton care apelează funcția de deconectare a utilizatorului curent de la aplicație. Acest lucru se realizează folosind un OnClickListener, în care se deschide pagina *login.xml*. După deconectarea contului, butonul de *Back* nu mai face trecerea la o pagina anterior deschisa, în acest caz *Profile*, ci închide aplicația.

Utilizatorul aplicației este informat printr-un mesaj că deconectarea contului a fost încheiata cu succes.

## 3.7 Modul zi/noapte

Pagina *Settings* are ca rol trecerea aplicației din modul zi în modul noapte, respectiv trecerea din modul noapte în modul zi. Pentru a realiza acest lucru, am definit în *res/values/themes* două fișiere de tip xml numite *value/themes.xml* si *value-night/themes.xml*. În fiecare dintre acestea doua am definit doi itemi numiți *background\_color* si *text\_color*. Pentru modul de zi, *background\_color* a primit valoarea *#FFFFFF*, reprezentând culoarea alba, iar *text\_color* a primit valoarea *#000000*, reprezentând culoarea neagra. În cazul modului de noapte, valorile sunt inversate. Toate paginile din aplicație preiau de aici culorile in care sunt reprezentate paginile. Astfel, în cadrul metodelor *onCreate* din fiecare clasa, mai întâi se afla daca este activ modul de noapte sau cel de zi.

Detecția modului activ de face printr-o comparație între *AppCompatDelegate .getDefaultNightMode()* si *AppCompatDelegate.MODE\_NIGHT\_YES* sau între *AppCompatDelegate.getDefaultNightMode*() si *AppCompatDelegate .MODE\_NIGHT\_AUTO\_BATTERY*. Daca rezultatul uneia dintre comparații este adevărat, atunci de folosește modul de noapte in pagina respectivă. În cazul contrariu, se va folosi modul de zi.

În pagina *Settings* avem un *SwitchCompat* care ne arata modul curent al aplicației. Atunci când acesta are valoarea *False*, modul curent este cel de zi. Atunci când are valoarea *True*, modul curent este cel de noapte. Schimbarea valorii de adevăr a acestui *SwitchCompat* va schimba tema utilizată în cadrul întregii aplicații. Aceasta opțiune se păstrează într-un *SharedPreferencse* sub denumirea *"save",* care își schimba valoarea odată cu *SwitchCompat-ul*. Schimbarea din *SharedPreferences* se face prin intermediul unui Editor, care inserează o valoare de adevăr în funcție de modul rămas în urma click-ului.

Un obiect de tipul *SharedPreferences* ne face legătura cu un fișier care conține perechi de tipul cheie-valoare si ne asigura metode de citire si modificare a acestor date. Aceste perechi se păstrează într-un fișier xml în memoria dispozitivului.

## 3.8 Modificarea documentelor afișate din Firebase

Pentru păstrarea datelor utilizate în cadrul aplicației am folosit *Realtime Database* furnizat de *Firebase*, o platforma *Google*. Aceasta păstrează datele sub forma de document de tip *JSON*. Din acest motiv nu se folosește limbajul *SQL* pentru accesarea, introducerea sau modificarea datelor. Platforma vine cu propriile biblioteci si funcții pentru manipularea datelor si accesul la acestea.

În interiorul bazei de date, obiectele sunt introduse prin modelul cheie-valoare. Astfel, categoriile principale, cheile, pot avea denumiri generice, valorile diferențiind de la un obiect la altul. Acest lucru permite efectuarea de operații generalizate, cu rezultate diverse în funcție de valoarea cheilor.

Platforma este intuitiva, ușor de utilizat, permițând chiar si unui utilizator neexperimentat utilizarea acesteia, cat timp cunoaște șablonul folosit pentru buna rulare a aplicației.

Unul din plusurile platformei este modificarea în timp real a datelor din cadrul aplicației. Schimbarea unui link sau a unei denumiri în *Realtime Database* se va remarca aproape instant în aplicație. În acest fel se pot schimba coperțile cărților, se pot modifica genuri sau denumiri si se pot asocia link-uri către alte documente doar prin schimbarea textului asociat cheii.

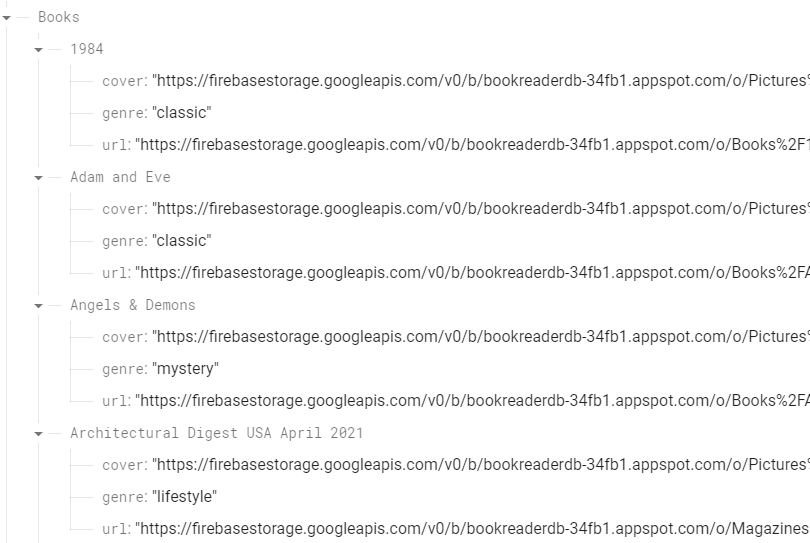


Figura 3.24: Realtime Database

## 3.9 Analiza valorilor din Firebase Analytics

Folosind *Firebase Analytics*, putem urmări modul în care se utilizează aplicația si putem folosi datele furnizate de aceasta platforma pentru a prezice modul de dezvoltare a aplicației. Astfel putem anticipa si preveni viitoare probleme si costuri.

Firebase are diferite planuri pentru funcțiile pe care le poate îndeplini. Acestea implica anumite costuri după depășirea unor valori alese de Google. Printre acestea se regăsesc si o parte din tehnologiile folosite, având următoarele costuri lunare:

* Realtime Database
  + Gratuit pentru stocarea de pana la 1GB de date, apoi 5$/GB
  + Gratuit pentru descărcarea de pana la 10GB de date, apoi 1$/GB
* Cloud Storage
  + Gratuit pentru stocarea de pana la 5GB de date, apoi 0.026/$/GB
  + Gratuit pentru descărcarea de pana la 1GB de date, apoi 0.12$/GB
  + Gratuit pentru realizarea a maximum 20k operatii de incarcare zilnice, apoi 0.05$/10k
  + Gratuit pentru realizarea a maximum 50k operatii de descarcare zilnice, apoi 0.004/10k
* Authentication
  + Gratuit pentru autentificarea a maximum 10k în SUA, Canada si India, apoi 0.01$/verificare
  + Gratuit pentru autentificarea a maximum 10k în restul tarilor, apoi 0.06$/verificare

Având în vedere aceste costuri, ne putem folosi de *Firebase Analytics* pentru a ne

pregăti pentru posibile viitoare costuri si pentru a lua decizii în ceea ce privește dezvoltarea aplicației pe viitor.

*Firebase Analytics* generează mai multe grafice pentru o reprezentare statistica a datelor preluate din aplicație. Am preluat graficele oferite de platforma pentru ultimele 30 de zile pentru a exemplifica modul în care pot apărea datele în cadrul acestei platforme.

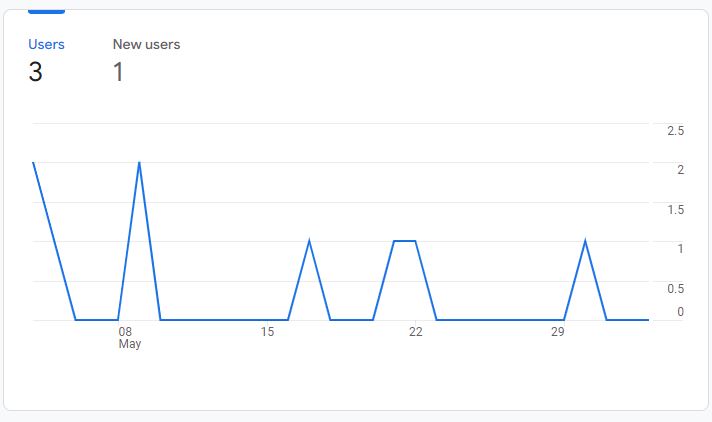


Figura 3.25: Utilizatorii din ultimele 30 de zile

Graficul din figura 3.25 reprezintă evoluția adăugării de conturi în ultimele 30 de zile. Se poate observa o creștere mai mare a numărului de noi utilizatori în perioada 4 mai - 9 mai, apoi crearea de cate un cont nou la intervale de aproximativ o săptămână distanta .

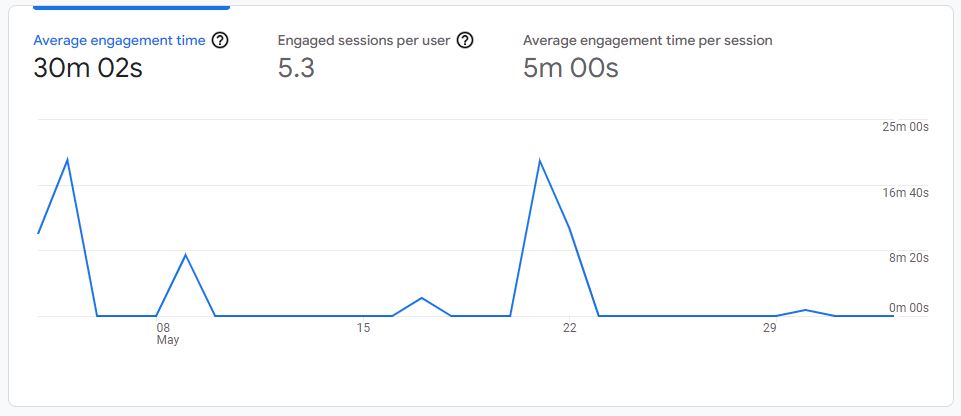


Figura 3.26: Timpul mediu se interacțiune a fiecărui utilizator activ

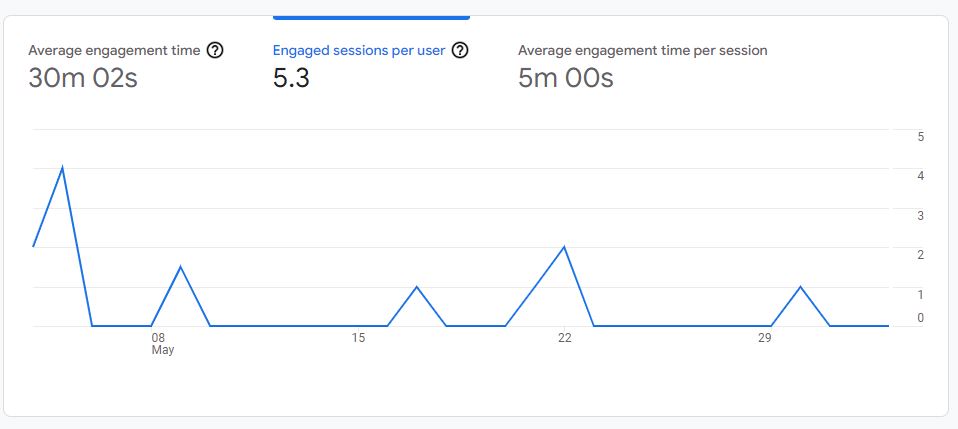


Figura 3.27: Numărul mediu de sesiuni a fiecărui utilizator activ

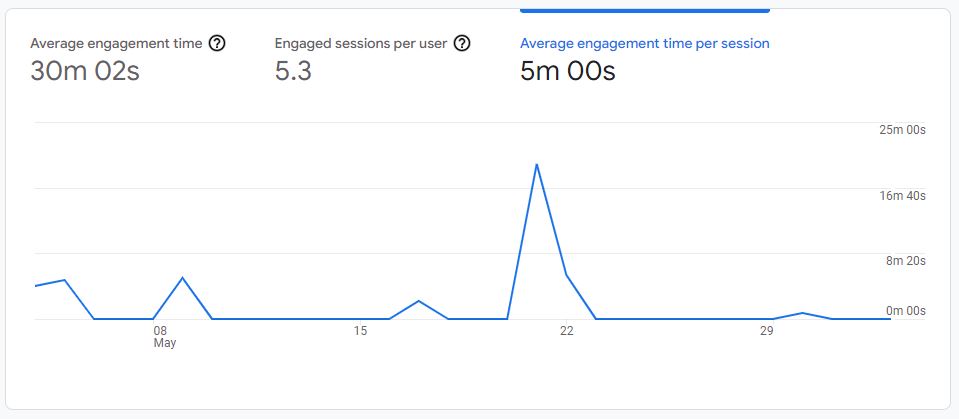


Figura 3.28: Timpul mediu de interacțiune per sesiune

Din figurile 3.26, 3.27, 3.28 se poate observa ca exista o corelare între creșterea timpului de utilizare mediu a aplicației de către fiecare utilizator si numărul mediu de sesiuni. Aceasta creștere coincide si cu apariția unor noi utilizatori reprezentata în figura 3.25. Acest lucru se poate remarca si în figura 3.29, unde se prezinta activitatea utilizatorilor pe 30 de zile, 7 zile si o zi.

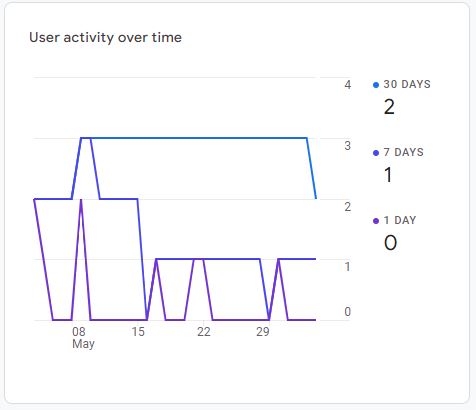


Figura 3.29: Activitatea utilizatorilor pe diferite intervale de timp

Se observa câți utilizatori au folosit aplicația în intervalele de timp menționate anterior si parcursul descendent al graficului.



Figura 3.30: Raportarea zi-săptămână-lună

În figura sunt reprezentate rapoartele între DAU(daily active users), MAU(monthly active users) si WAU(weekly active users). Se remarcă o foarte mare asemănare între DAU/MAU si DU/WAU.

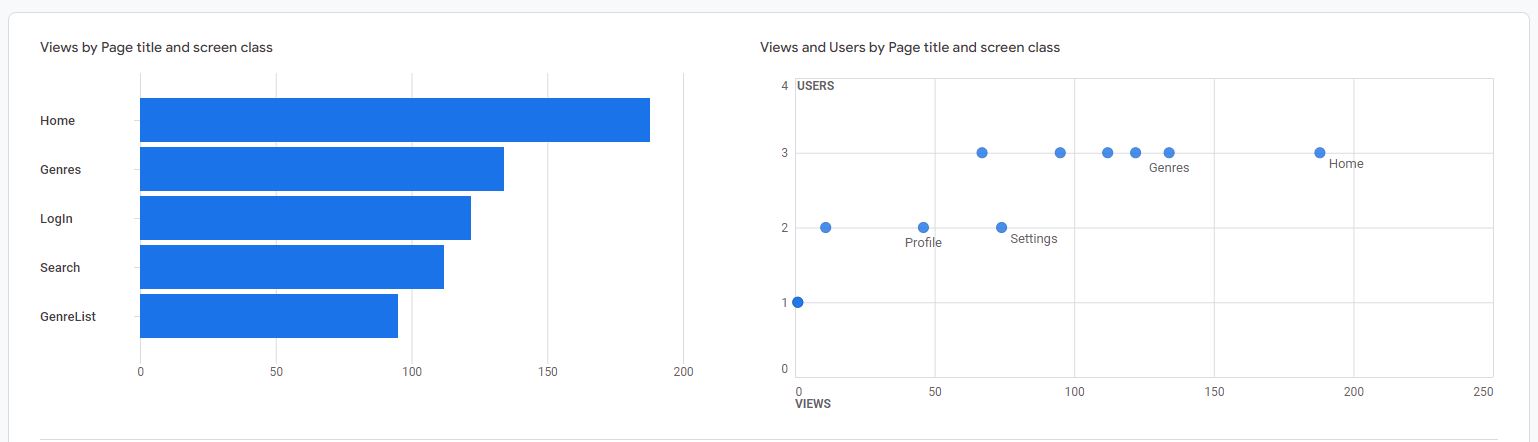


Figura 3.31: Grafic vizualizări pagini



Figura 3.32: Tabel vizualizări pagini

În figurile 3.31 si 3.32 sunt marcate grafic si sub forma de tabel numărul de accesări si timpul de vizualizare a paginilor aplicației. Observam ca paginile cele mai accesate sunt: pagina de *Home*(unde se deschide aplicația după autentificare), pagina *Genres*(de unde alegem genul literar pe care dorim sa îl citim), pagina *LogIn*(unde se deschide aplicația), pagina *Search* si pagina *GenreList*(care apare ca urmare a selectării unui gen literar în pagina *Genres*).

În dreapta tabelului din figura 3.32 se afișează costul fiecărei pagini, el fiind 0$ în prezent la toate categoriile deoarece nu s-au atins încă limitele menționate anterior.

Aceste grafice si tabele prezentate anterior ajuta la înțelegerea comportamentului utilizatorilor si la estimarea costurilor generate un urma folosirii aplicației.

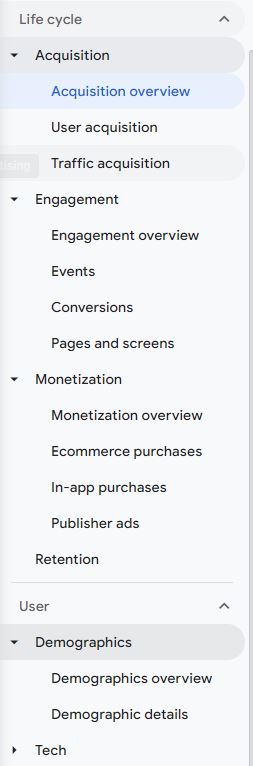


Figura 3.33: Meniul generării de rapoarte

După cum se poate vedea în figura 3.33, platforma dispune de multe alte rapoarte care se pot genera, care nu sunt încă viabile pe aplicație în starea ei curenta. Printre acestea s-ar regăsi toata categoria *Monetization*, deoarece nu exista în aplicație in momentul acesta un mod de plata a unor servicii sau afișare de reclame.

# 4. Utilizarea Sistemului

Pentru a prezenta desfășurarea unor evenimente din cadrul aplicației, am simulat parcursul unor operații efectuate de utilizator prin intermediul unor imagini.

## 4.1 Use case pentru utilizator nou

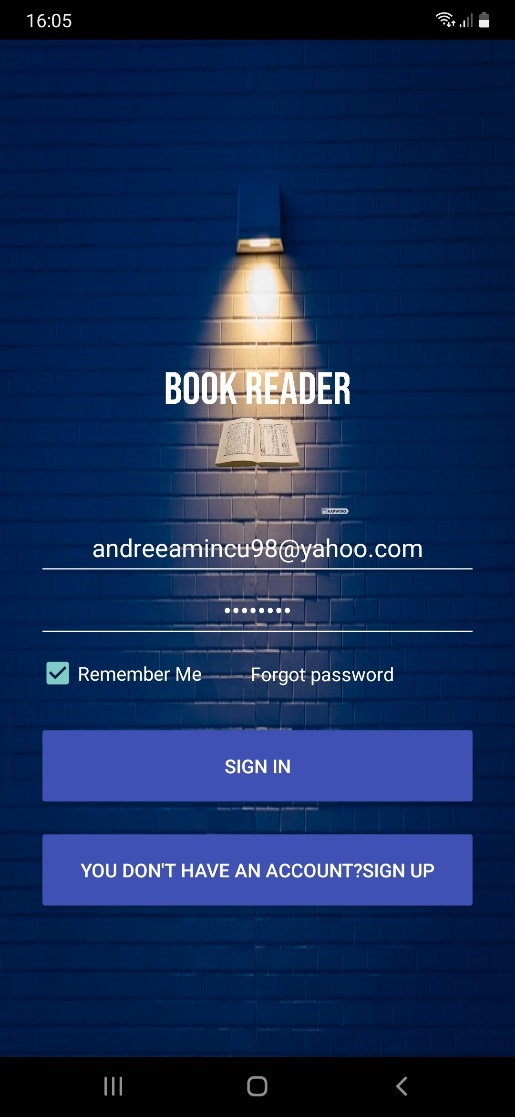
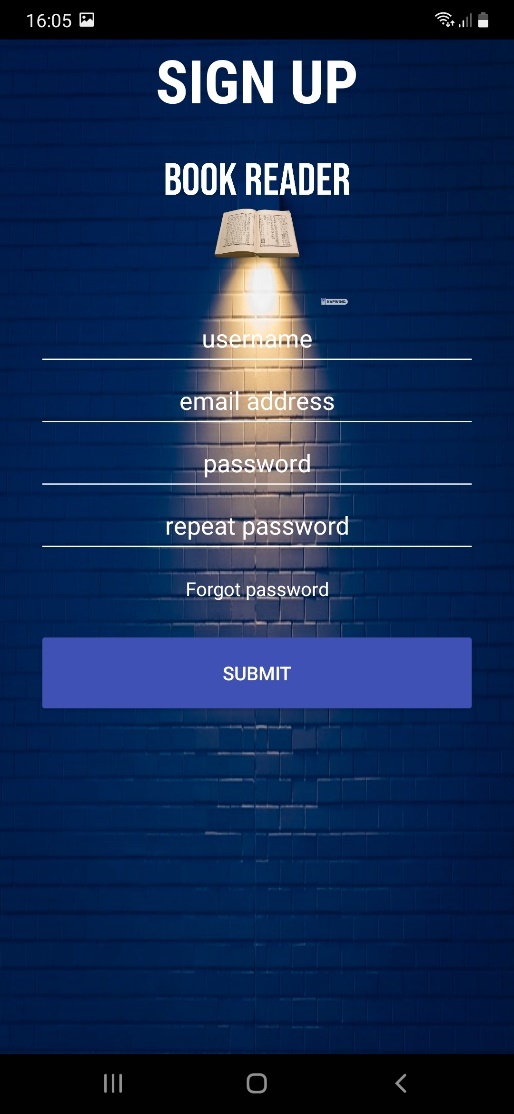


Figura 4.1: Pagina de autentificare Figura 4.2: Pagina de înregistrare

Pagina din figura 4.1 reprezintă interfața primei paginii a aplicației. Aici utilizatorul poate sa se autentifice, sa acceseze pagina de înregistrare si pagina de resetare a parolei.

Bifarea checkbox-ului de *Remember Me* asigura păstrarea datelor introduse în câmpul de email si câmpul de parolă. Lăsarea acestuia nebifat va goli aceste câmpuri la redeschiderea aplicației.

Apăsarea butonului de *Sign Up* va deschide pagina de înregistrare, prezentată în figura 4.2. Utilizatorul completează aceste câmpuri pentru crearea unui cont nou.

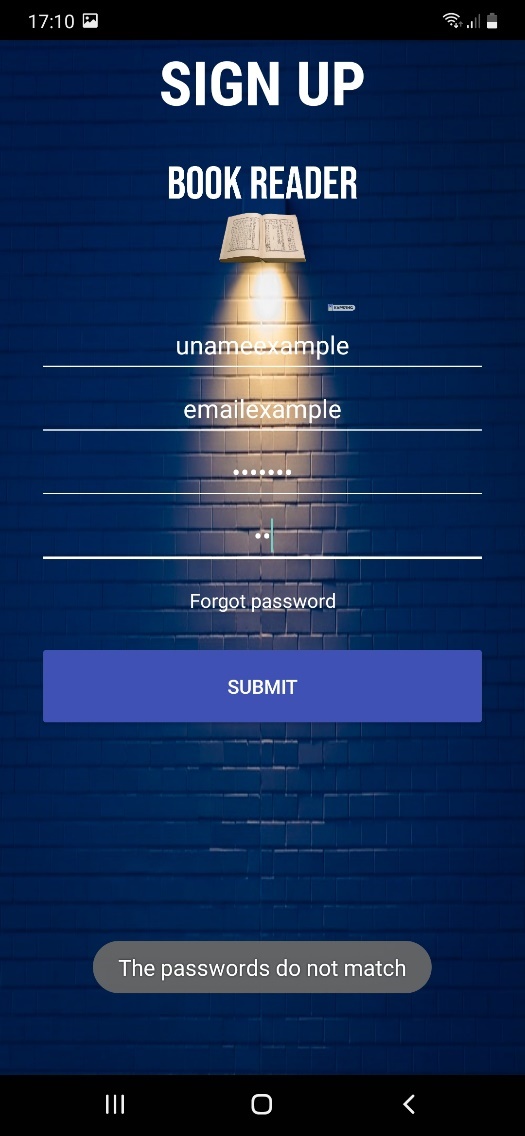
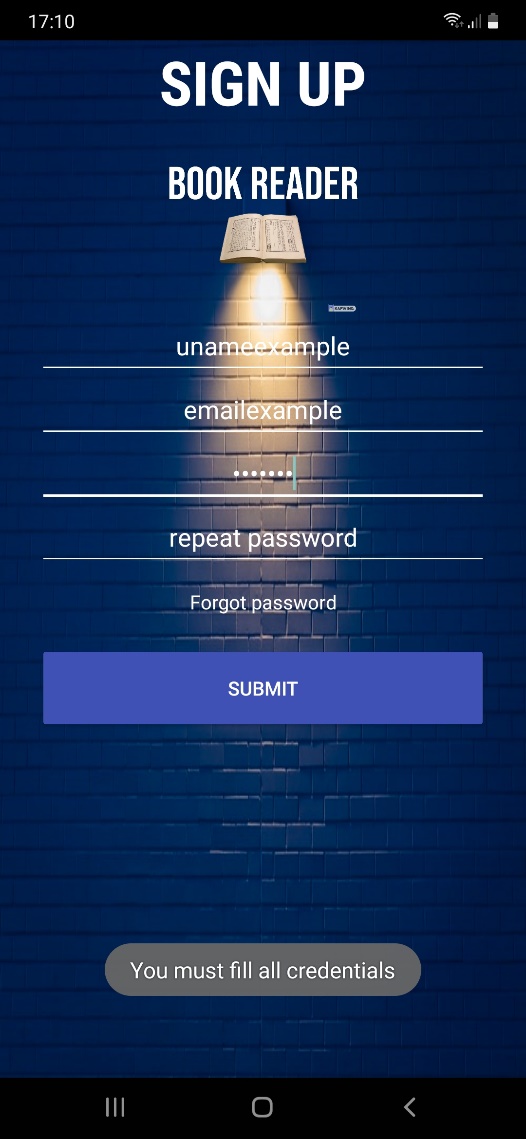


Figura 4.3: Lipsa câmpurilor completate Figura 4.4: Parolele nu coincid

În cazul în care utilizatorul nu a completat toate câmpurile, se va afișa mesajul din figura 4.3, iar operația de înregistrare nu se va mai efectua.

Daca parolele nu coincid, utilizatorul va primi mesajul din figura 4.4.



Figura 4.5: Reținerea datelor în Firebase Auth

În figura 4.5 este reprezentat modul în care se rețin datele de autentificare în Firebase Autentication. Pe acest cont se pot efectua operații de resetare a parolei, dezactivare a contului sau ștergere a contului direct din platforma.

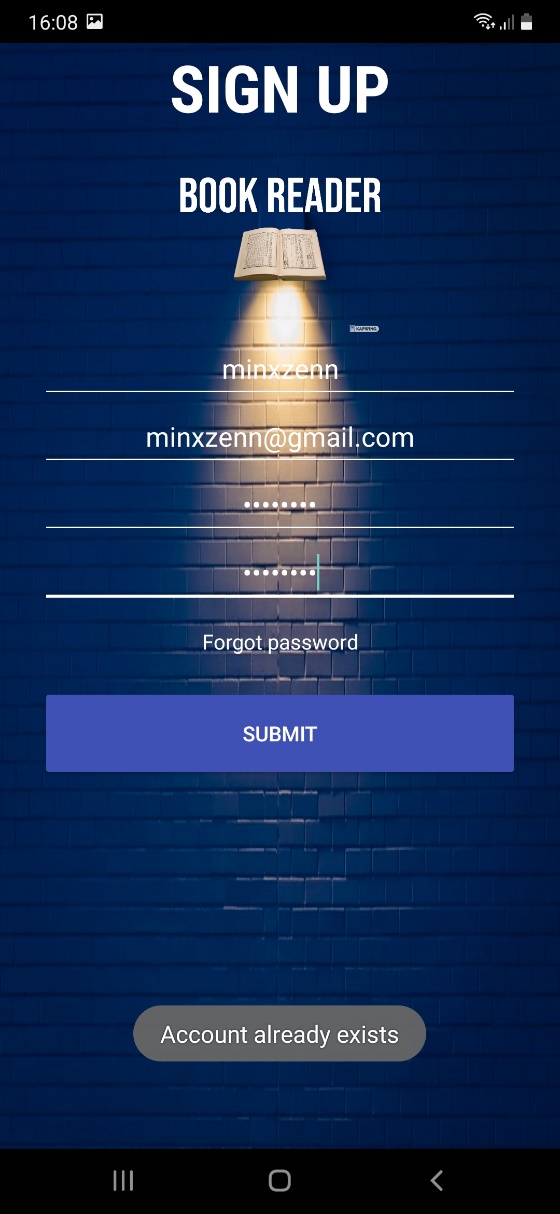
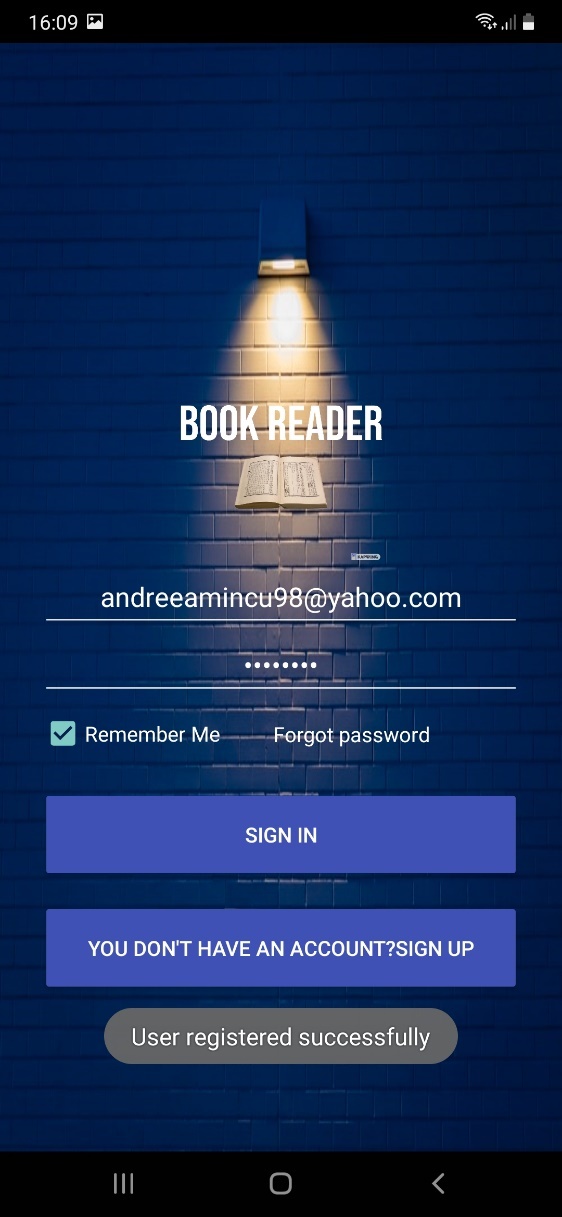


Figura 4.6: Contul exista deja Figura 4.7: Contul a fost înregistrat cu succes

În cazul în care contul exista deja cu adresa de email introdusa, utilizatorul va fi notificat cu mesajul din figura 4.6. După completarea corecta a câmpurilor cu un email valid se va reveni la pagina de autentificare si va fi trimis mesajul din figura 4.7.

Pentru a realiza autentificarea, se completează câmpul de email si câmpul de parolă cu date asociate unui cont existent. Daca nu exista acest cont sau parola nu este cea corecta, utilizatorul este notificat precum în figura 4.8. Daca datele introduse sunt corecte, iar contul nu este activat, ni se va cere sa ne verificam adresa de email si sa activam contul urmând pașii menționați în mail.

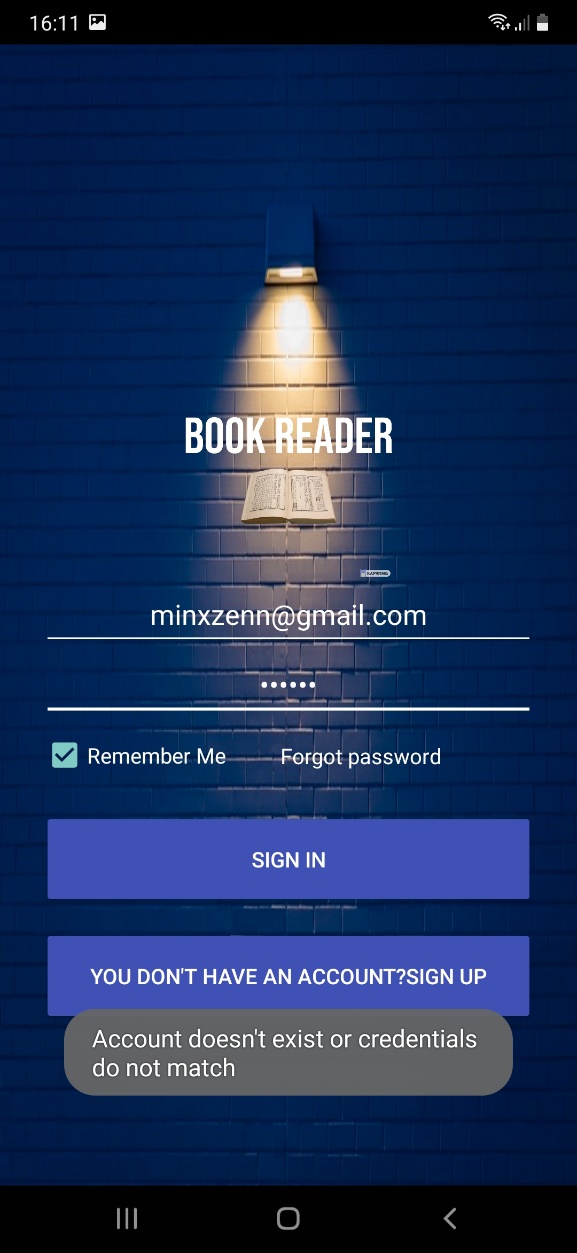
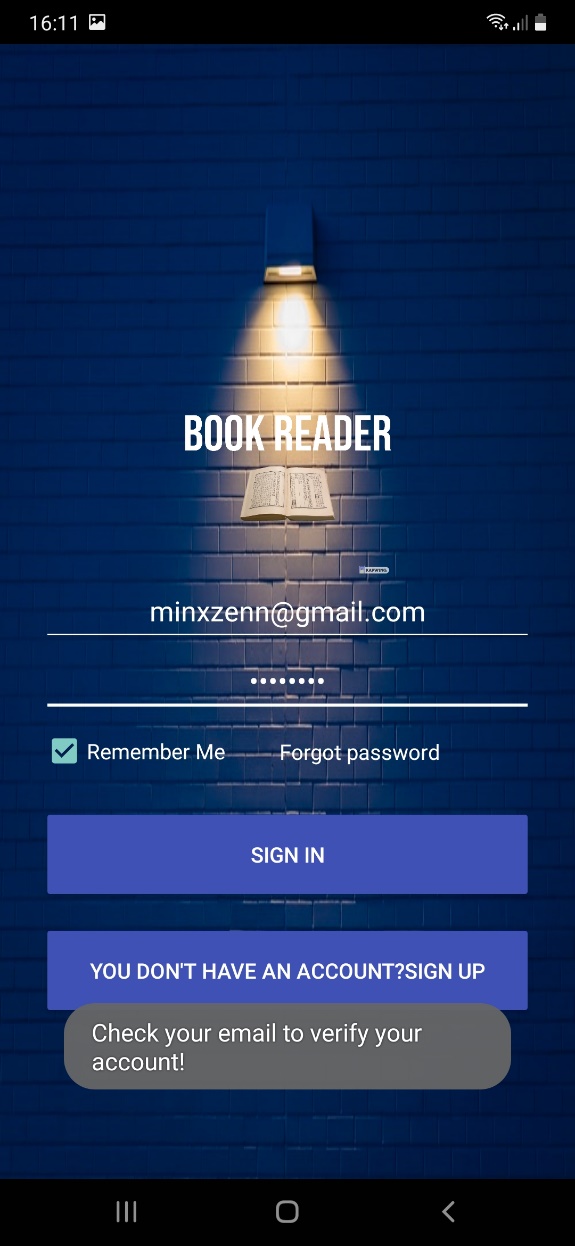


Figura 4.8: Nu exista cont cu aceste date Figura 4.9: Verificați adresa de email

pentru validarea contului

La accesarea adresei de email se va regăsi un mail precum cel din figura 4.10, unde se va da click pe link-ul menționat în instrucțiuni si se va primi mesajul din figura 4.11.

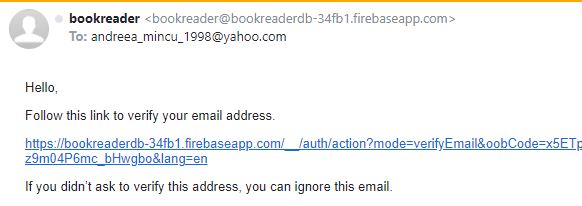


Figura 4.10: Mail-ul de verificare

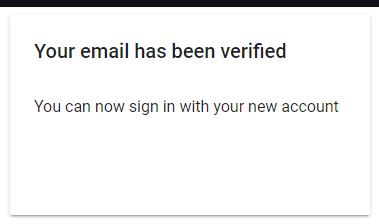


Figura 4.11: Confirmarea verificării contului

O alta funcție importanta este cea de resetare a parolei. Pagina unde se poate realiza acest lucru este accesata prin accesarea *Forgot password*, fie din pagina de autentificare, fie din pagina de înregistrare.

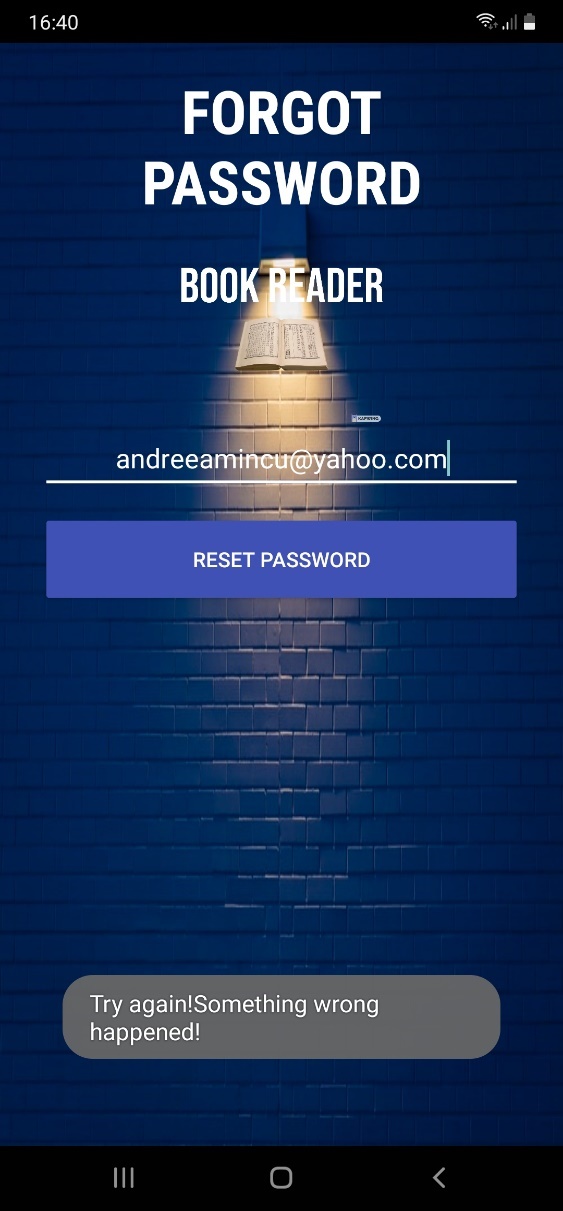
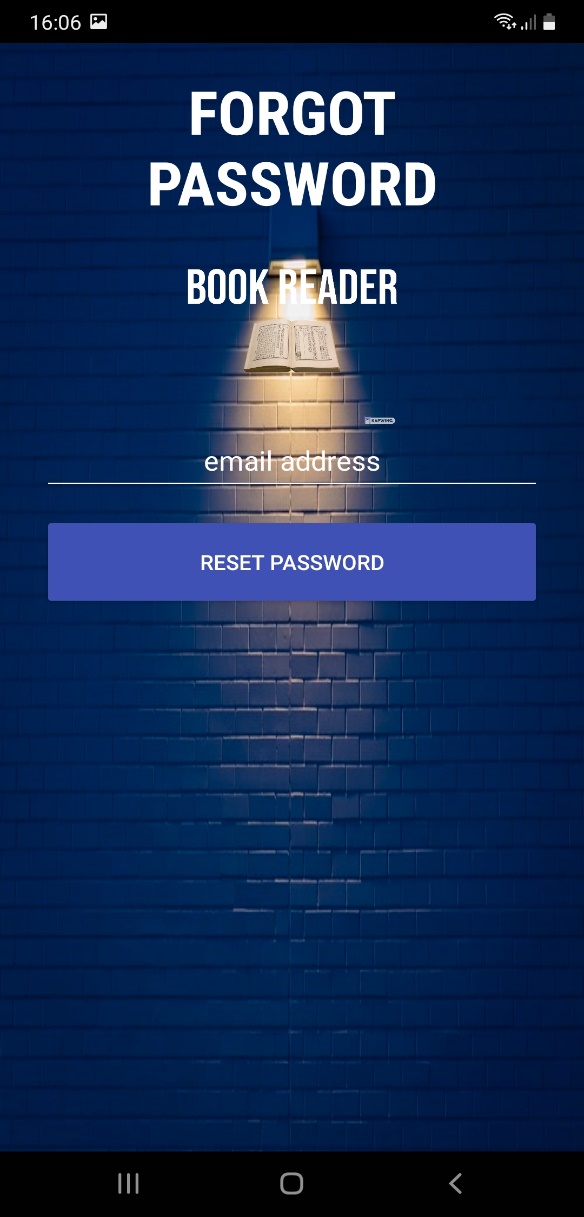


Figura 4.12: Pagina de resetare a parolei Figura 4.13: Nu exista cont cu aceasta

adresa de email

În pagina de înregistrare exista un câmp în care utilizatorul trebuie sa introducă adresa de email a contului la care vrea sa schimbe parola. Daca nu exista niciun cont asociat acestei adrese de email, se va primi notificarea din figura 4.13.

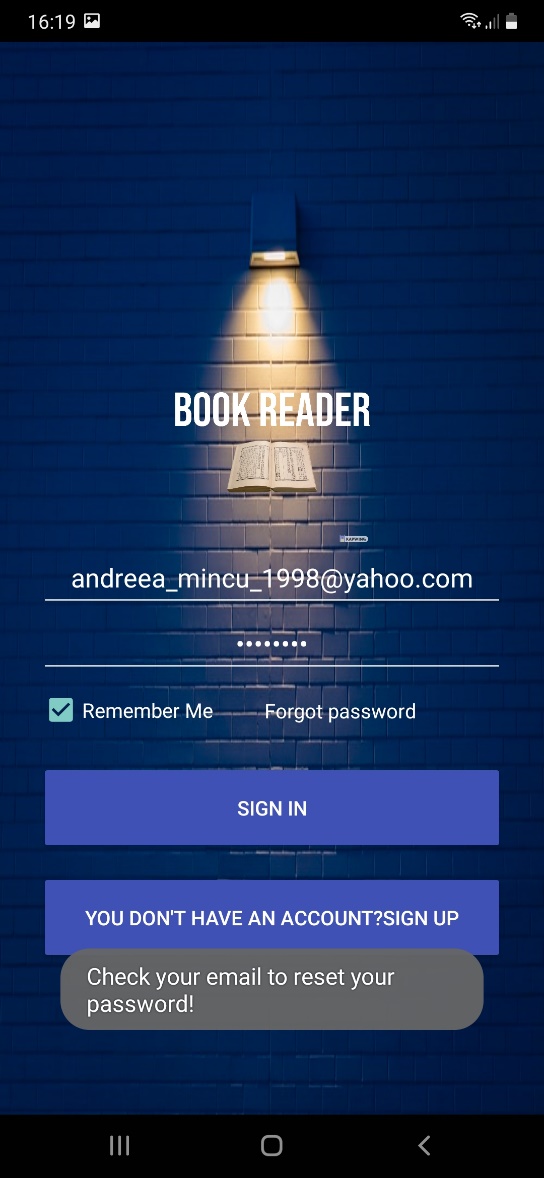


Figura 4.14: Mesaj de verificare a adresei de email pentru resetarea parolei

După introducerea unei adrese de email valida si apăsarea butonului de *Reset Passowrd*, utilizatorul va fi redirecționat la pagina de autentificare si va primi un mesaj de verificare a adresei de email în scopul resetării parolei, așa cum se observa în figura 4.14.

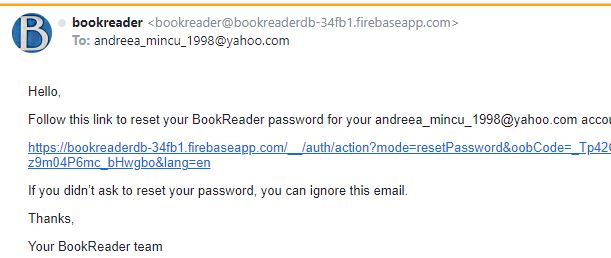


Figura 4.15: Emailul de resetare a parolei

La verificarea adresei de email se va intra pe mail-ul din figura 4.15 si se va accesa link-ul dat pentru a putea reseta parola.

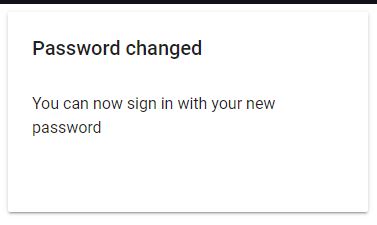


Figura 4.16: Introducerea noii parole

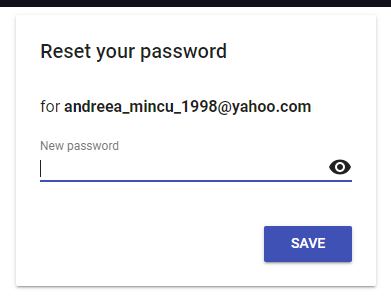


Figura 4.17: Confirmarea schimbării parolei

Link-ul ne va redirecționa către o pagina în care trebuie introdusa noua parola asociata adresei de email utilizate, iar în urma apăsării butonului *Save* vom primi confirmarea schimbării făcute.

## 4.2 Use case pentru utilizator deja autentificat

În acest subcapitol am prezentat pașii pe care ii poate realiza un utilizator autentificat, după validarea contului.

### 4.2.1 Vizualizarea paginii principale

După autentificare, utilizatorul este redirecționat către pagina principala, primind mesajul din figura 4.18. Acolo poate vizualiza sugestiile de cărți, prezentate în mai multe categorii. Acesta poate observa mai multe documente, atât prin slideshow-ul principal, unde se derulează automat cărțile, sau în listele de mai jos, prin mișcări către stânga sau către dreapta.

De asemenea, în partea inferioara a paginii se regăsește meniul principal, care face legătura cu funcțiile principale ale aplicației:

* Home;
* Genres;
* Search;
* Profile;
* Settings.

Alegerea unuia dintre elementele meniului va redirecționa utilizatorul către pagina corespunzătoare, fiecare pagina fiind marcata printr-o imagine sugestiva funcționalității.

În cazul în care în setări este selectata ca tema principala tema de noapte, culorile paginii se vor schimba, având fundalul negru si scrisul alb precum în figura 4.20.

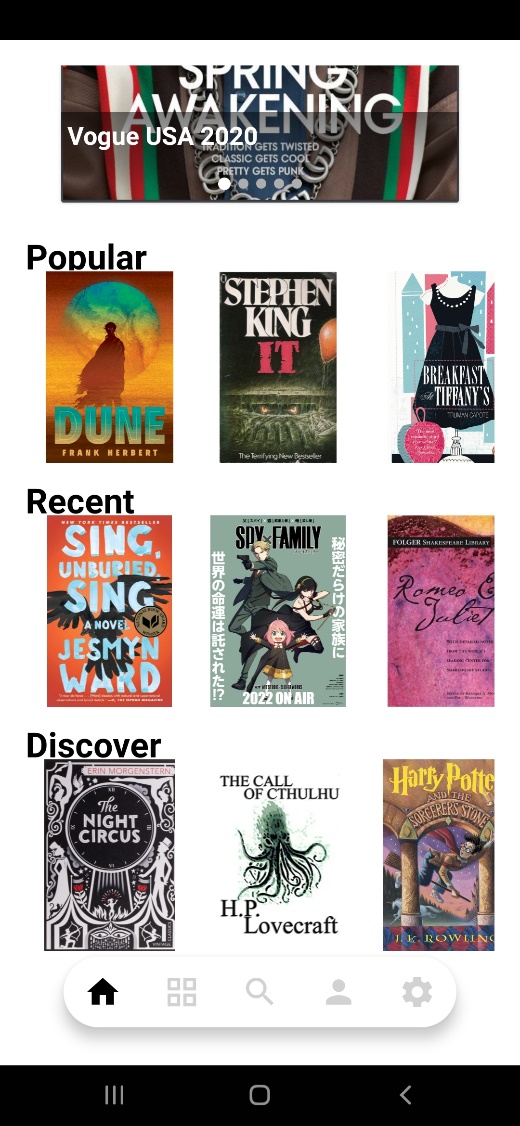
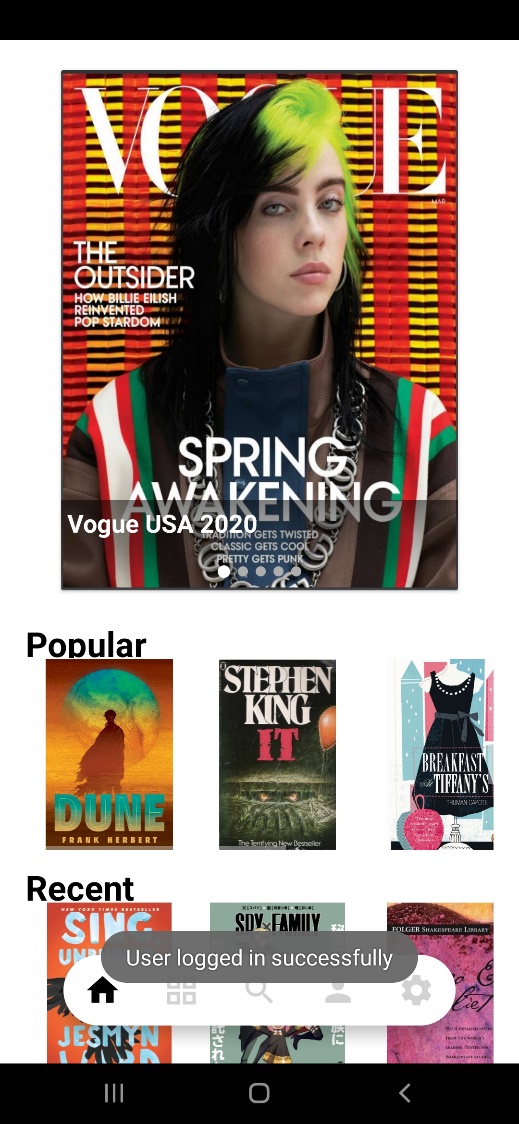


Figura 4.18: Autentificare reușită Figura 4.19:Categorii pagina principală

La alegerea uneia din cărțile prezentate, utilizatorul va fi redirecționat către pagina în care va putea citi documentul. La deschiderea paginii va apărea o animație de încărcare, care va dispărea în momentul afișării cărții. Aici va putea schimba paginile glisând stânga sau dreapta si va putea realiza un *zoom in* prin apăsare dubla sau ciupire. Acest tip de deschidere a documentelor este prezent în cadrul întregii aplicații.

Revenirea la pagina inițială se va face prin apăsarea butonului back.

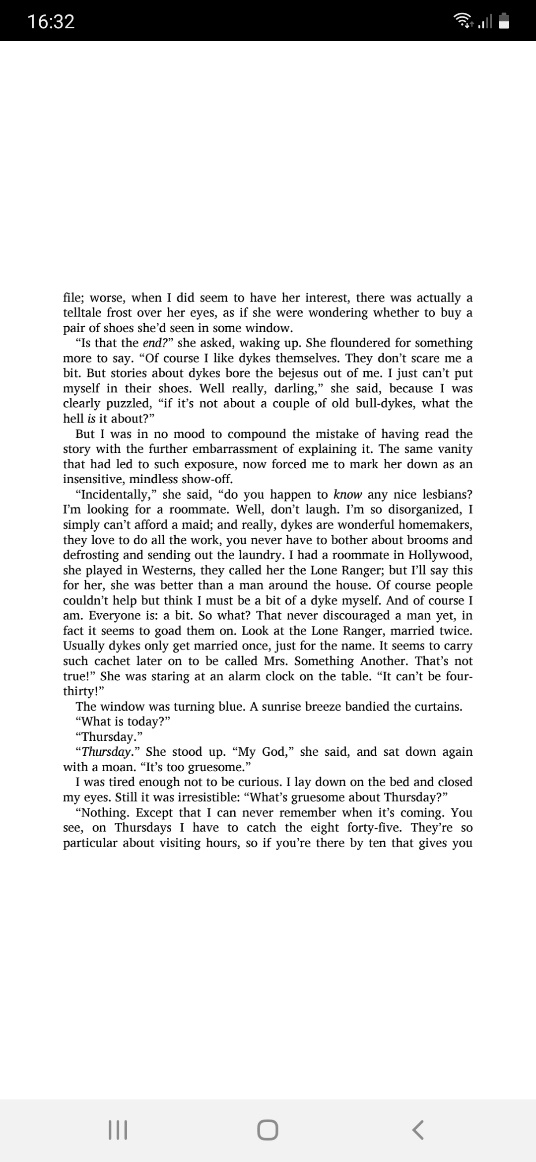
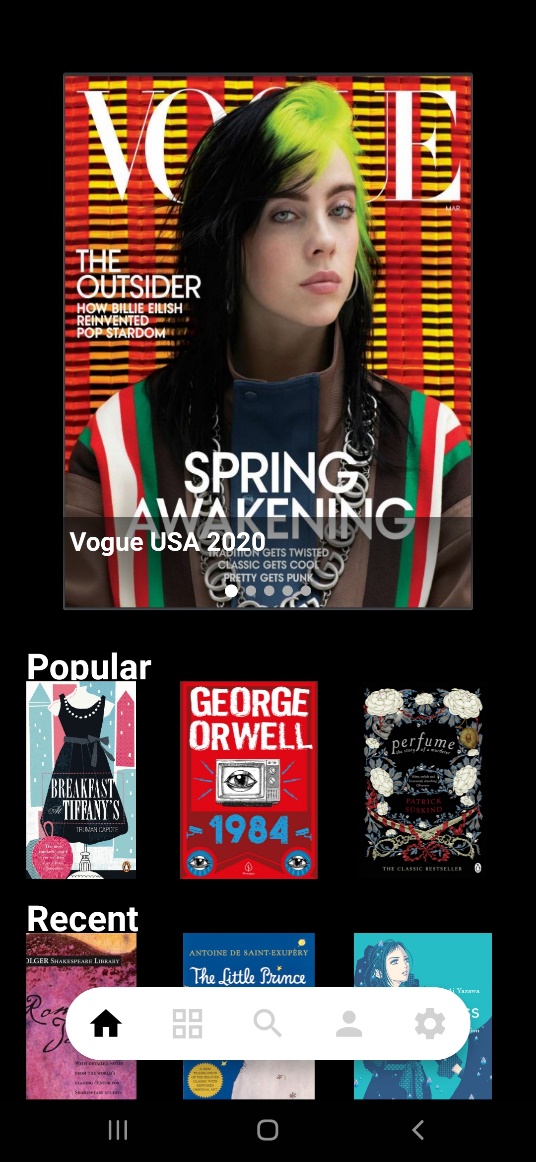


Figura 4.20: Modul de noapte pagina Home Figura 4.21: Deschiderea documentului

### 4.2.2 Alegerea genului cărții si filtrarea documentelor

La selectarea celei de-a doua opțiuni a meniului din partea inferioara a paginii principale, suntem redirecționați către pagina Genres. Aceasta are ca funcționalitate principala filtrarea listei de cărți de care dispune aplicația după genul în care acestea se încadrează.

Pagina inițială este reprezentata de o lista de genuri utilizate la filtrarea documentelor si meniul principal regăsit si aici în partea inferioara a paginii. Lista categoriilor din care putem alege cuprinde:

* Fantasy;
* Lifestyle;
* Fashion;
* Romance;
* Mystery;
* Historic;
* Sci-Fi;
* Classic;
* Modern;
* Horror;
* Young Adult;
* Thriller;
* Manga.

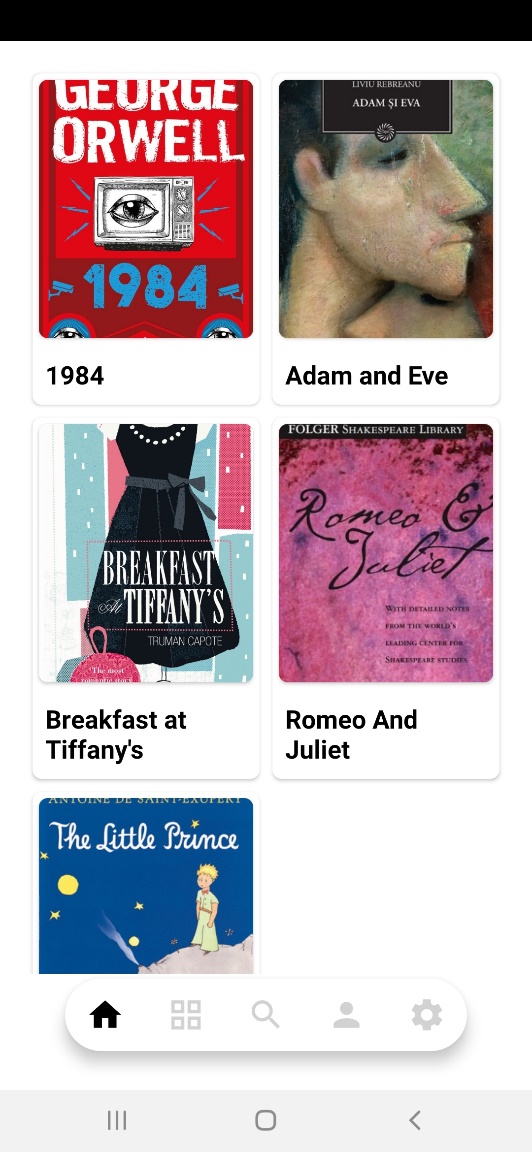
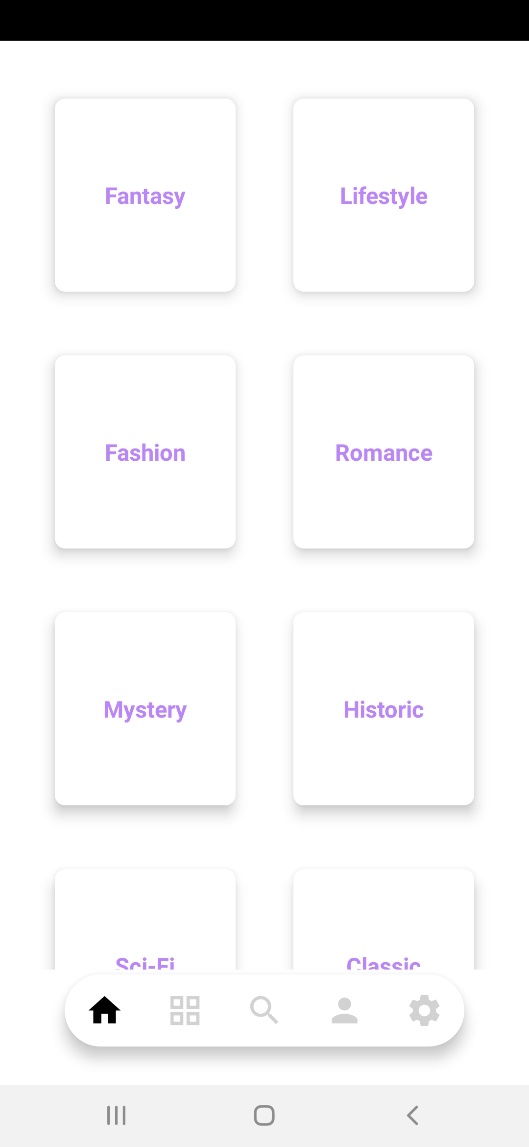


Figura 4.22: Lista de selecție a genurilor Figura 4.23: Cărțile filtrate

După alegerea unuia dintre genurile listate, se va deschide o noua pagina cu toate cărțile filtrate după decizia făcută anterior. În Figura 4.23 se afișează cărțile încadrate în categoria Classic, prezentate sugestiv prin coperta si titlu. La alegerea uneia dintre ele, utilizatorul va fi redirecționat către pagina de deschidere a documentului, asemănător figurii 4.21, după o animație de încărcare, având proprietățile menționate anterior.

Cele doua pagini reprezentative acestei funcții de clasificare dispun si ele de o tema de noapte. Si în acest caz vor fi pagini cu fundal negru, dar cu un scris violet, respectiv alb pentru pagina cărților filtrate. Meniul din partea inferioara a paginii va rămâne alb, atât în modul de zi, cât si în modul de noapte.

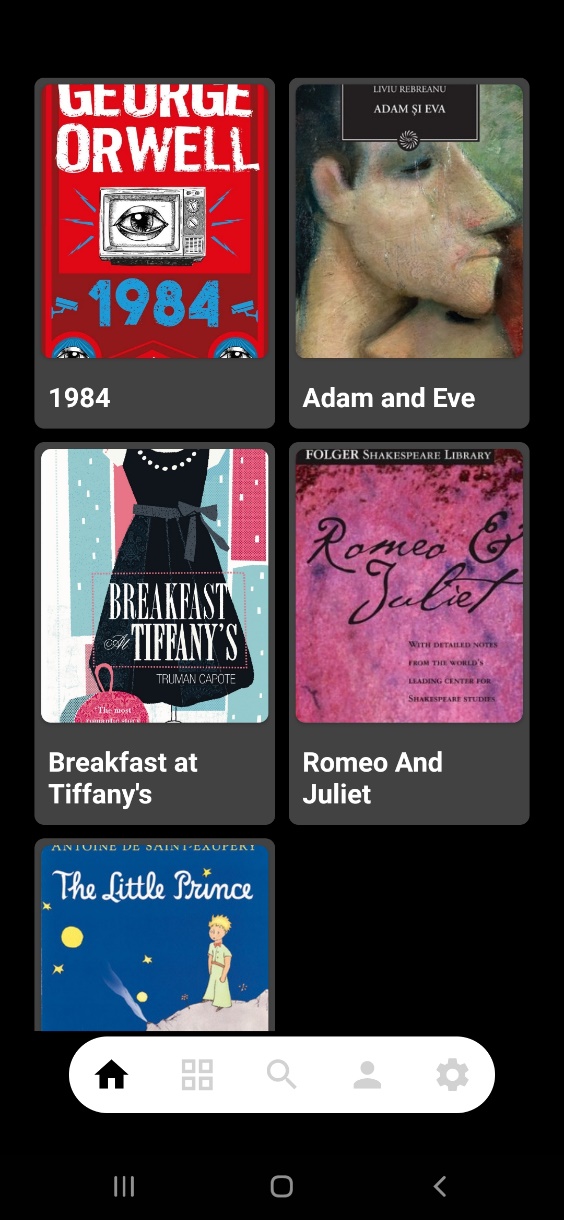


Figura 4.24: Mod noapte lista categorii Figura 4.25: Mod noapte cărți filtrate după

categorie

### 4.2.3 Opțiunea de căutare

A treia imagine din meniul din partea inferioara a paginii reprezintă pagina de căutare. Aici se pot căuta cărțile după nume.

Aceasta pagina este alcătuită din trei componente:

* Câmpul de căutare
* Zona de afișare a rezultatelor
* Meniul principal



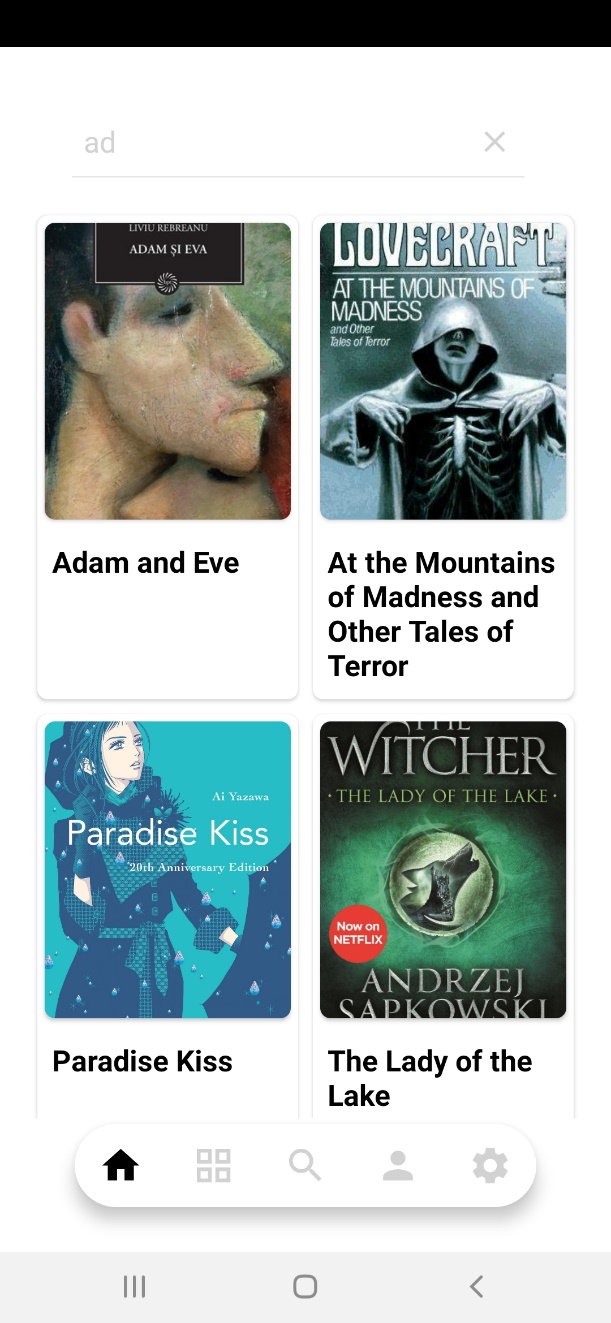


Figura 4.26: Începerea căutării Figura 4.27:Afisarea căutării

Pentru a începe căutarea unui document trebuie apăsată lupa din partea superioara a paginii, care va deschide câmpul în care se pot introduce cuvintele cheie, lucru observat în figura 4.26. Acestea vor fi căutate printre titlurile cărților din baza de date si se vor afișa mai jos cărțile găsite, prezentate sugestiv prin copertă si titlu, precum in figura 4.27.

La fiecare modificare a textului din câmpul de căutare se va schimba rezultatul afișat, iar la golirea acestuia se vor lista toate cărțile conținute de baza de date.

La atingerea unuia dintre documente se va deschide si în acest caz pagina de afișare a documentului, la fel ca in figura 4.21 si va păstra aceleași proprietăți.

Pagina de căutare are si ea varianta proprie pentru tema de noapte, unde câmpul de navigație devine negru cu scris alb, rezultatele căutării sunt încadrate într-un chenar gri cu scris alb, iar meniul din partea inferioara a paginii rămâne si în cazul acesta alb.

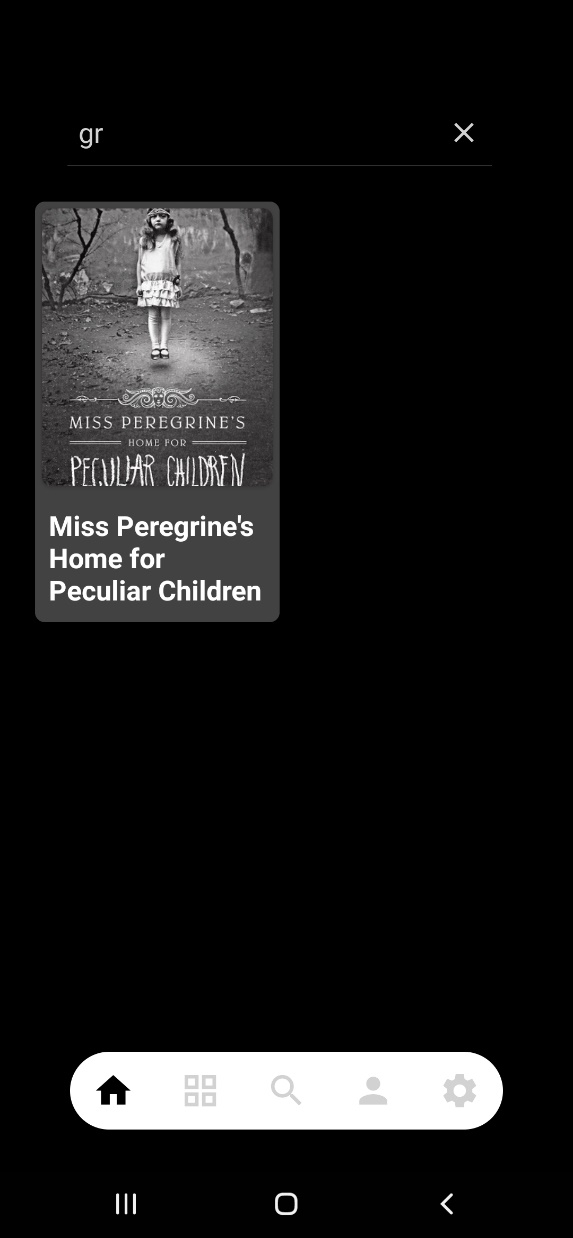
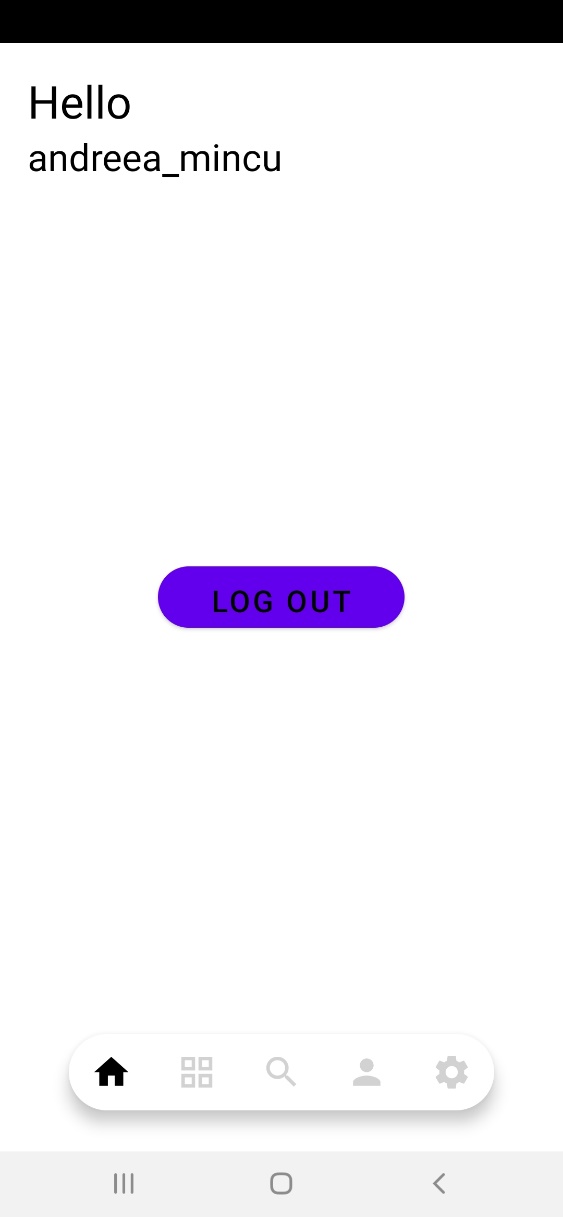


Figura 4.28: Modul noapte căutare

### 4.2.4: Deconectarea utilizatorului

A patra opțiune din cadrul meniului din partea inferioara a paginii este modulul Profile. Acesta are ca funcție de baza deconectarea utilizatorului curent.

În partea superioara a paginii este menționat utilizatorul curent si se oferă, printr-un buton, deconectarea acestuia de la aplicație. În partea inferioara a paginii este prezent meniul principal al aplicației.

 Si pagina Profile dispune de o tema de noapte, prezentata în figura 4.30, unde diferă fundalul, culoarea textului si culoarea butonului fata de tema de zi.

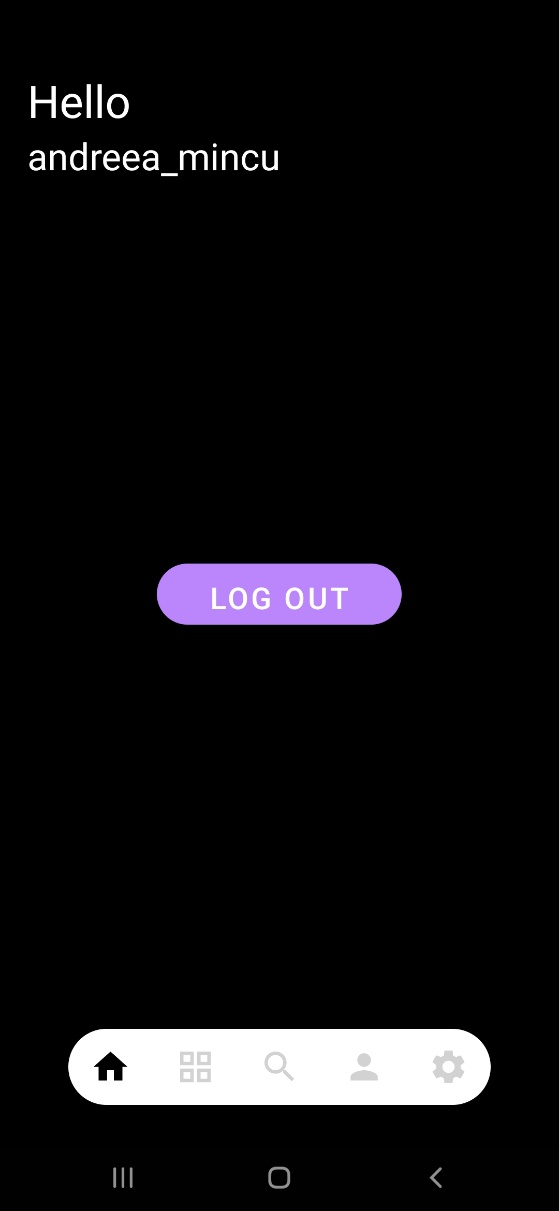


Figura 4.29: Pagina Profile Figura 4.30: Modul noapte pagina Profile

Funcția care permite deconectarea utilizatorului curent este importanta deoarece permite autentificarea altui cont fără a ieși din aplicație.

La apăsarea butonului, utilizatorul va fi redirecționat către pagina de autentificare, unde va primi mesajul de deconectare reușită, precum în figura 4.31.

Daca se apasă butonul înapoi după deconectarea utilizatorului, aplicația nu va reveni la pagina anterioara, pagina Profile, ci se va închide.

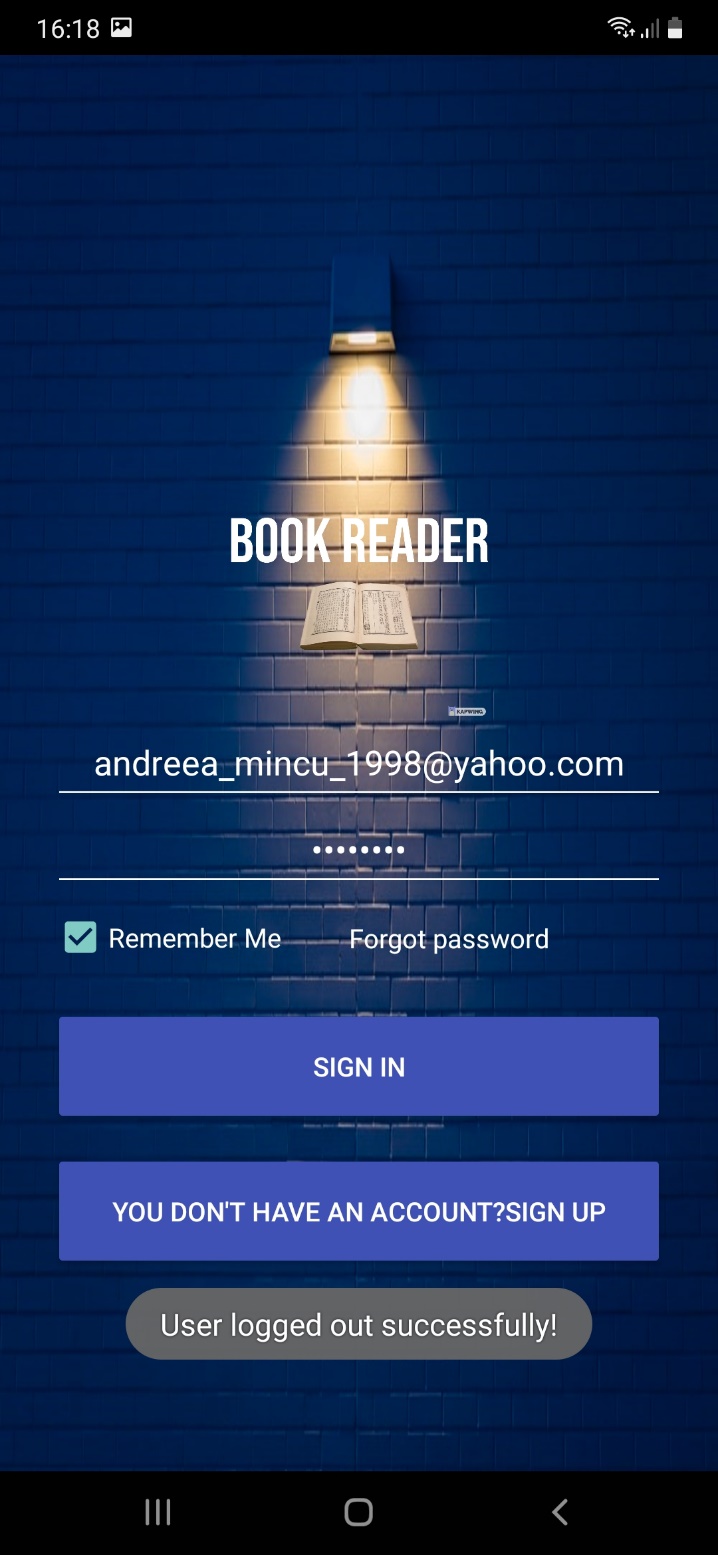


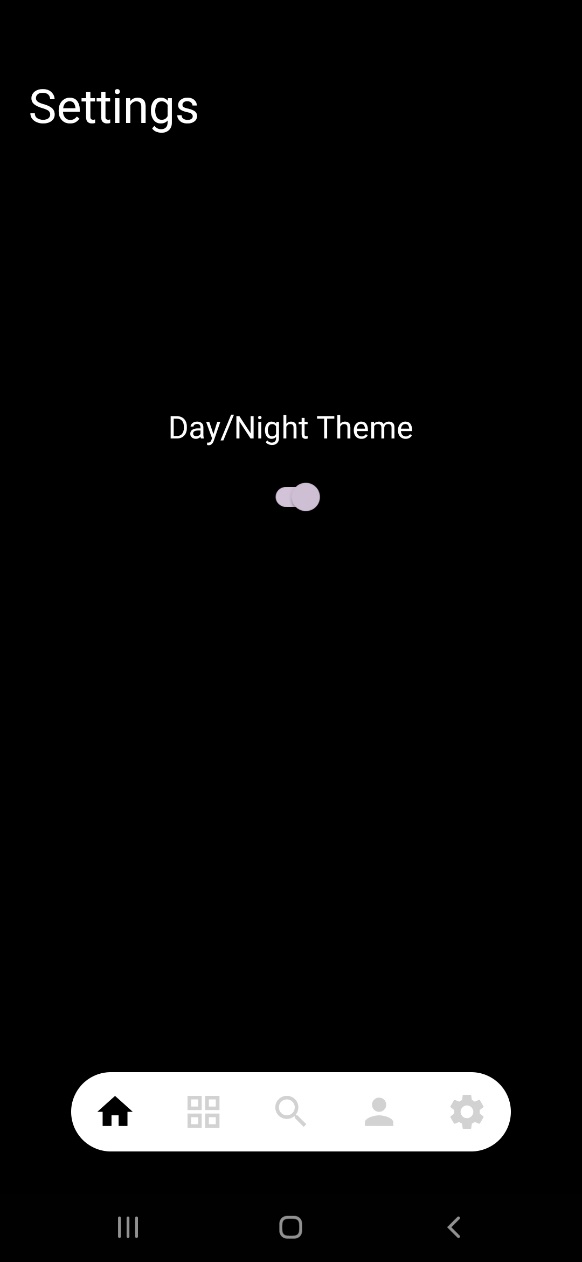
Figura 4.31: Deconectarea reușită

Pentru reconectarea la aplicație, daca se dorește reautentificarea cu același cont, utilizatorul va trebui sa reintroducă datele de autentificare, în cazul în care căsuța de *Remember Me* nu a fost bifata. În cazul contrar se va apăsa doar pe butonul de *Sign In*.

Daca se dorește conectarea cu alt cont, se vor introduce datele altui cont existent pentru a merge mai departe către restul aplicației.

### 4.2.5 Modul zi/noapte

A cincea opțiune a meniului din partea inferioara a paginii este reprezentat prin pagina Settings. Aceasta are un rol important din punct de vedere vizual, făcând tranziția de la modul de zi la modul de noapte, respectiv tranziția de a modul de noapte la modul de zi.



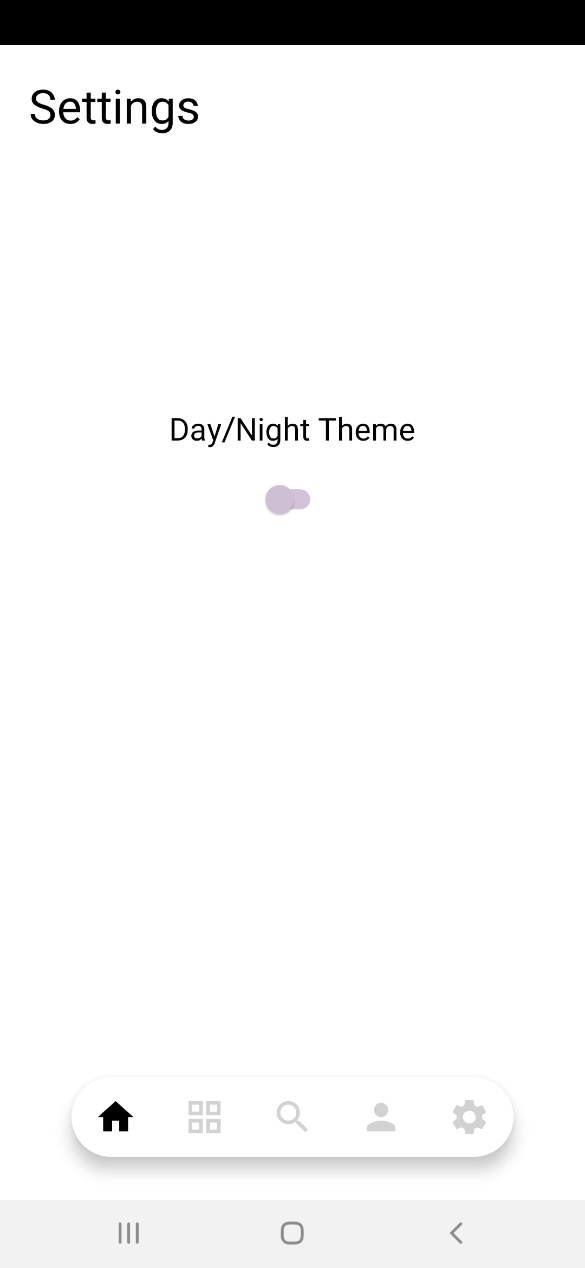


Figura 4.32: Modul de zi Figura 4.33: Modul de noapte

Pagina Settings este alcătuită dintr-un întrerupător si meniul principal al aplicației.

Întrerupătorul are rolul de a schimba la atingerea sa tema aplicației. Atunci când este oprit, aplicația utilizează modul de zi, iar atunci când este pornit, aplicația folosește modul de noapte. Aceste schimbări se manifesta în întreaga aplicație si sunt păstrate pana la încheierea sesiunii.

# 5. Concluzii si perspective de dezvoltare

În concluzie, aplicația satisface nevoia iubitorilor de lectură de a descoperi si citi cărți de toate genurile.

Aceasta aplicație facilitează accesul la astfel de documente prin ușurință de a accesa materialele de pe telefonul mobil, obiect indispensabil in viața de zi cu zi. Aplicația are dimensiuni reduse si legătura către cărți se face cu ajutorul unei conexiuni la internet. Astfel, se încurajează lectura, mai ales în rândul tinerilor, care in ultimii ani au preferat mediile digitale de procurare a cărților în detrimentul formatului clasic, printat.

Pe lângă funcționalitățile prezentate anterior, am plănuit o serie de noi facilitați pentru a îmbunătăți experiența utilizatorilor:

1. Adaptarea pentru mai mulți clienți;
2. Funcția de favorite;
3. Pagina de prezentare;
4. Filtrarea după limita de vârstă.

## 5.1 Adaptarea pentru mai mulți clienți

În momentul actual, aplicația are o forma generala, care sugerează un mod de prezentare a cărților. Aceasta, însă, poate fi adaptata pentru mai mulți clienți. Aceasta aplicație se poate mula foarte ușor pe cerințele unei edituri care dorește sa își extindă si în mediul online publicațiile, astfel focalizându-se pe documente tip text, cu anumite genuri de cărți alese. Alt mod de adaptare a aplicației este pentru o editura de reviste, care dorește sa cuprindă toate edițiile sale, cu accent pe imaginile prezentate. Aceste doua cazuri sunt generalizări ale unor situații posibile.

Exista alt tip de client care ar putea cere o aplicație care sa dețină doar un anumit gen de cărți, din mai multe surse, cu alte tipuri de categorii dorite de aceștia.

Unul dintre punctele forte ale aplicației este, astfel, adaptabilitatea, capacitatea de a se mula pe mai multe tipuri de cerințe ale clientului, respectând în mare același schelet.

## 5.2 Funcția de favorite

O alta actualizare de viitor a aplicației pe care am dezvoltat-o este reprezentata de funcția de favorite. Prin intermediul acesteia, utilizatorul ar putea marca cărțile preferate si vizualiza într-o pagină separata, accesată prin intermediul paginii Profile. Astfel, se va putea tine cont mult mai ușor de documentele deja citite sau care au ieșit în evidenta într-un fel pozitiv. Acest lucru va ajuta utilizatorul sa revină mai ușor asupra acestor cărți.

## 5.3 Pagina de prezentare

Este o pagina pe care doresc sa o adaug intre momentul selecției cărții si deschiderea acesteia in aplicație.

Pagina de prezentare va conține:

* titlul;
* coperta;
* genul;
* rezumatul cărții.

Aceasta pagina va ajuta utilizatorul sa aleagă si sa înțeleagă mai bine documentul pe care este dispus sa îl citească.

Odată cu implementarea funcției de favorite, adăugarea cărții la favorite se va putea realiza în cadrul acestei pagini. Scoaterea acesteia din lista respectiva se va putea realiza tot din aceasta pagină.

## 5.4 Filtrarea după limita de vârstă

Acest lucru se va putea realiza după modificarea formularului de înregistrare, unde se va adăuga data nașterii. Apoi, în baza de date, se va adăuga la fiecare carte vârsta minima necesara citirii fiecărei cărți în parte. În aplicație, la fiecare funcție de afișare, se va aplica un filtru de vârstă , permițând numai celor care îndeplinesc condiția de vârstă minima să poată vizualiza documentul.

# Bibliografie

[1] Jerome DiMarzio: Beginning Android Programming with Android Studio.

[2] Murat Yener, Onur Dundar: Expert Android Studio.

[3] Mike van Drongelen: Android Studio Cookbook.

[4] Ken Arnold, James Gosling, David Holmes: THE Java™ Programming Language, Fourth Edi‐

tion.

[5] Budi Kurniawan: Java for Android, Second Edition

[6] Houssem Yahiaoui: Firebase Cookbook: Over 70 recipes to help you create real‐time web

and mobile applications with Firebase.[7] Bill Stonehem: Google Android Firebase: Learning the Basics.[8] Neil Smyth: Firebase Essentials ‐ Android Edition.

[9] [Online]. Available: https://firebase.google.com/docs/.

[10] [Online]. Available: https://firebaseopensource.com/.

[11] [Online]. Available: https://stackoverflow.com/.

[12] [Online]. Available: https://www.youtube.com/.

[13] [Online]. Available: https://github.com/.