



COPERTA

[Document subtitle]



[DATE]

[COMPANY NAME]

[Company address]

Conținut

1.INTRODUCERE	3
2. TEHNOLOGII FOLOSITE	4
2.1 ANDROID STUDIO	4
2.1.1 Caracteristici Android	4
2.1.2 Arhitectura Android	4
2.1.3 Android SDK	5
2.2 Java	5
2.2.1 Caracteristici Java	5
2.2.2 Componentele Java	5
2.3 Firebase	6
2.3.1 Firebase Authentication	6
2.3.2 Autentificare SDK Firebase	7
2.3.3 Autentificarea FirebaseUI Auth	7
2.3.4 Realtime Database	8
2.3.5 FirebaseUI Database	9
2.3.6 Storage	9
2.3.7 Firebase Analytics	10
2.4 APIs utilizate	10
2.4.1 PDF Page Viewer	10
2.4.2 Image Slideshow	11
2.4.3 Glide	11
3.DESCRIEREA SISTEMULUI	12
3.1 Autentificarea	12
3.1.1 Pagina de autentificare	13
3.1.2 Pagina de înregistrare	14
3.1.3 Pagina de recuperare a parolei	17
3.2 Pagina principală	19
3.2.1 Meniul	19
3.2.2 Prezentarea cărților	21
3.3 Genurile de cărți	25
3.3.1 Alegerea categoriei	25
3.3.2 Filtrarea cărților după categoria aleasa	26
3.4 Deschiderea documentului	26
3.5 Opțiunea de căutare	27
3.6 Deconectarea contului	29

Bibliografie	30
--------------------	----

1.INTRODUCERE

Aplicația pe care am dezvoltat-o este una de vizualizare de cărți și reviste cu ajutorul conexiunii la Internet. Aceasta își propune să 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.

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 față 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 și dimensiunea acestuia, luminozitatea și capacitatea de a deține pe un singur device mai multe documente.

Am ales această temă datorită pasiunii mele pentru citit, pentru a împărtăși și facilita acest mod de a petrece timpul liber și cu alte persoane. Consider că această aplicație are o întrebuințare reală în 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 și 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

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 și emulator. Acesta are încorporat Google Cloud Platform, care permite integrare ușoară pentru Google Cloud Messaging și App Engine, Github, Firebase etc. Cu ajutorul Android Studio se pot realiza aplicații 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. Acesta deține următoarele trăsături:

- Suport Hardware: GPS, camera, senzor de proximitate, busolă digitală
- 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: Aplicații multi-tasking
- Conectivitate: GSM sau EDGE, Bluetooth, Wi-Fi, IDEN, CDMA, EV-DO

2.1.2 Arhitectura Android

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 librării care permit realizarea unei aplicații Android folosind limbajul Java
- Componenta Framework a aplicației

- Componenta aplicații- care conține aplicațiile de bază ale dispozitivului(Contacts, Phone, Browser), dar și alte aplicații instalate de utilizator

2.1.3 Android SDK

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

2.2 Java

Android Studio ajută la dezvoltarea aplicațiilor Android în două limbaje de programare: Kotlin și 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 rapidă de aplicații complexe și 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 virtuală, nu este necesară recompilarea codului în cadrul fiecărui dispozitiv utilizat.

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

2.2.1 Caracteristici Java

Este un limbaj de programare utilizat pe multe sisteme de operare. În urma compilării, codul sursă devine executabil, rezultând în problema utilizării codului pe fiecare platformă în parte. Compilarea codului în limbaj Java este compilat ca bytecode. Acesta nu se poate executa singur, pentru că 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 dacă se va folosi un emulator). Astfel, poate fi executat doar pe o mașină virtuală (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

Pentru a programa în Java este necesar un compilator pentru Java numit javac. Acesta generează fișierele de clase Java din codul sursă. Fișierele rezultate sunt scrise în bytecode și sunt independente de platformă. JVM-ul (Java Virtual Machine) încarcă aceste

fișiere și interpretează bytecode-ul sau compilează în limbaj mașină folosind compilatorul JIT (Just in time).

JRE (Java Runtime Environment) este necesar pentru a executa programe și 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 principală al mediului de lucru Java. Acesta conține JRE, javac, Java debugger și alte clase esențiale. Acesta este folosit pentru dezvoltarea de aplicații Java deoarece oferă atât executabilul și binery-urile, cât și instrumentele necesare compilării și debugging-ului unui program în limbajul Java.

2.3 Firebase

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

Sunt necesare un cont și 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 singură aplicație.

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

Am folosit serviciul Firebase Authentication pentru autentificarea pe baza emailului și a parolei. Aceasta se ocupă de securitatea utilizatorilor și 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 măsurile de securitate Google și dispui de posibilitatea de a gestiona într-un mod sigur datele.

Firebase Analytics este un instrument care te ajută să obții informații despre performanța aplicației. Acesta oferă un set complet de informații despre utilizatori și modul în care aceștia utilizează aplicația.

2.3.1 Firebase Authentication

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

Firebase Authentication prezintă mai multe opțiuni de autentificare:

- Email și 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 in cazul in care utilizatorul se va înregistra)

Firestore Authentication dispune de 2 tipuri de autentificare:

- FirebaseAuth
- FirebaseAuthUI

Ambele tipuri de autentificare utilizează următoarele clase:

- FirebaseAuth Instance- pentru SDK-ul FirebaseAuth. Se folosește la crearea de conturi, conectarea sau deconectarea utilizatorilor, modificarea informațiilor utilizatorului curent.
- FirebaseAuthUI Instance- conține metode de sign-in si de sign-out. Dispune de opțiuni pentru modificarea interfeței, precum tema, dar si opțiuni de autentificare a ușurului.
- FirebaseAuthUser 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 in 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 FirebaseAuth pentru crearea unui cont.

2.3.2 Autentificare SDK FirebaseAuth

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

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

2.3.3 Autentificarea FirebaseAuthUI

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

1. Alegerea tipurilor de autentificare utilizate in consola Firebase
2. Înregistrarea in aplicație a acestora
3. Adăugarea bibliotecilor FirebaseAuth in Android Studio
4. Dobândirea referinței pentru FirebaseAuth
5. Utilizarea clasei FirebaseAuth pentru configurarea si construirea FirebaseAuth-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, in timp ce FirebaseAuth este mai restrictiv din acest punct de vedere.

Având in 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

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

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

- Inserarea componentelor in baza de date;
- Actualizarea informațiilor;
- Ștergerea datelor.

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

In cazul in care nu exista conexiune la internet, datele vor fi salvate local, putând fi accesate de pe dispozitivul respectiv. In momentul reluării conexiunii la internet, datele se vor sincroniza in baza de date principala, eliminând posibilitatea pierderii de informații sau nesincronizării intre date introduse offline si revenirea in 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 in care datele sunt sincronizate intre 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 copertilor, gestionarea in funcție de gen , afișarea de cărți pe pagina Home si datele utilizatorilor.

2.3.5 FirebaseUI Database

FirebaseUI Database este o bibliotecă open-source pentru Android care permite programatorului să se conecteze rapid la elementele de interfață la un API Firebase. Astfel se pot vedea în timp real schimbările făcute în Realtime Database și 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 să își ordoneze automat conținutul
- `FUIindexarray`-Mentine un array sincronizat cu datele indexate de la două 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 și păstrării datelor, asigurată de Google, indiferent de calitatea internetului.

Ca proprietăți principale regăsim:

- Operații robuste-SDK-urile Firebase pentru Cloud Storage efectuează încărcări și descărcări indiferent de calitatea internetului. Astfel, aceste operații se reiau de unde s-au oprit, reducând timpul de execuție al operațiilor.
- Securitate puternică- SDK-urile Firebase pentru Cloud Storage se integrează cu Firebase Authentication pentru a aduce programatorilor autentificări simple și 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 infrastructură.

Cloud Storage păstrează fișierele într-un bucket Google Cloud Storage, făcându-le accesibile și prin Firebase și 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 și video transcoding folosind Google Cloud Storage API. SDK Firebase pentru Cloud Storage se integrează ușor cu Firebase Authentication pentru identificarea utilizatorilor și oferă un

limbaj declarativ securizat care permite dezvoltatorilor de aplicații să acceseze și să controleze fișiere individuale sau grupări de fișiere, putând astfel să le facă publice sau private.

2.3.7 Firebase Analytics

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

SDK-ul cuprinde un număr de evenimente și proprietăți ale utilizatorilor și te ajută să îți 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 detaliată a analizei datelor, de la scheme sumare pentru utilizatorii activi și date demografice, până la informații detaliate precum identificarea obiectelor cea mai bine vândute.

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

2.4 APIs utilizate

Am utilizat mai multe API-uri pentru afișarea imaginilor și a documentelor stocate în Firebase Database, preluându-le tokenii din Realtime Database. Astfel, toate elementele ce țin de fișiere media se pot schimba în timp real din baza de date cloud, reducând din dimensiunea aplicației și ajutând la extinderea acestora 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 și 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 utilizat 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 și citirea pentru external storage.

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

2.4.2 Image Slideshow

Image Slideshow este o biblioteca dezvoltata de denzcoskun, utilizata pentru afişarea 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 in cadrul dimensiunilor specificate in interfață.

Încărcarea imaginilor se poate face local sau prin url. Afişarea acestora in cadrul elementului din interfață se poate face in 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 afişare 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.

In momentul in 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

Page 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 observă preluarea datelor introduse de utilizator, iar după se verifică.

Prima dată se verifică dacă ambele câmpuri sunt completate. Dacă nu sunt completate, vom primi mesajul *"You must fill all credentials"*.

În pasul următor se caută emailul și parola în 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"*.

Dacă contul există, se verifică dacă acesta este validat sau nu. În cazul în care contul nu este validat se trimite un email de validare și 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 toate condițiile menționate anterior sunt îndeplinite.

3.1.2 Pagina de înregistrare

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

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

De asemenea, avem și 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 observă că 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 și se verifică datele introduse în acestea.

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

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

După ce ne asigurăm că toate câmpurile sunt completate și cele două parole coincid, urmează verificarea existenței 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 două conturi să existe cu aceeași adresă de email, așa cum se observă în *Figura 3.5*.

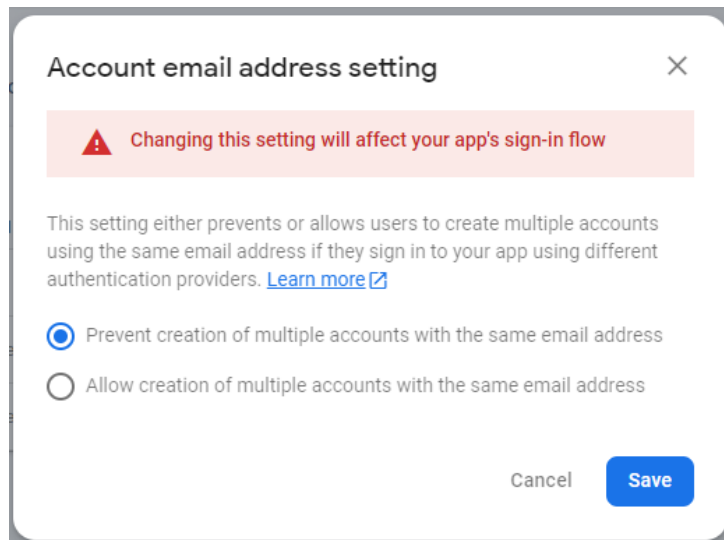


Figura 3.5: Crearea de conturi cu aceeași adresa de email

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

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

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

3.1.3 Pagina de recuperare a parolei

În această pagină se află un câmp unde utilizatorul trebuie să introducă un email valid pentru a-și 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 apăsă 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 și se vor executa mai multe verificări acestuia. În primul rând se verifică dacă 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 compară datele introduse cu șablonul unei adrese de email. Dacă funcția returnează *False*, atunci datele introduse nu urmează forma unui email și 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* și dacă apelul este reușit, se va primi mesajul *"Check your email to reset your password!"* și 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 după autentificarea utilizatorului. Aceasta este alcătuită din două componente: meniul și prezentarea cartilor. Aceasta are rolul de a sugera utilizatorului cărți pentru a-și lărgi orizonturile literare.

3.2.1 Meniul

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 în xml

Acest meniu se introduce în *BottomNavigationMenu* prin *app:menu="@menu/item_menu"*. Se poate observa în *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 prezintă 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 și 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 și 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 să 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 și a documentelor din baza de date Realtime Database. Acolo am creat o categorie specială pentru acest slideshow, unde am introdus titlul cărții, genul literar în care se încadrează, linkul către imaginea de copertă și 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 și imagini, este nevoie ca în *AndroidManifest.xml* să se introducă permisiunile pentru acces la internet, așa cum se observă 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 copertă și 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 in 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").addListene
rForSingleValueEvent(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 in categoria asociata in 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 in *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 in 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 in HorizontalScrollView

Folosind Realtime Database, toate schimbările care se fac in baza de date se aplica in timp real in 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 in 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 in Realtime Database.

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

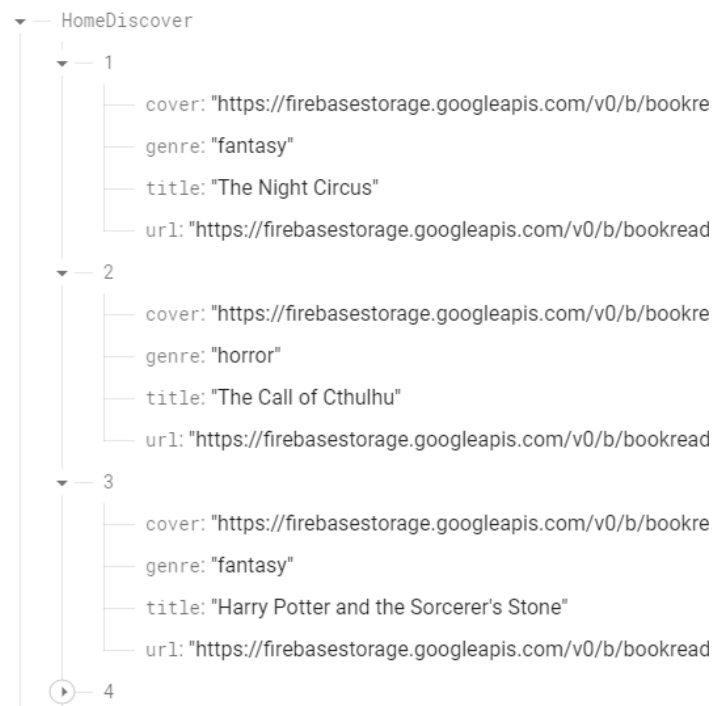


Figura 3.17: Organizarea in Realtime Database

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. In 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 cauta mult mai ușor obiecte ale viitoarelor lecturi, alegând genul literar pe care utilizatorul dorește să îl citească.

Cu acest scop am implementat o pagină 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 în aplicație. Acestea sunt prezentate într-un *RecyclerView*, împărțit pe două coloane folosind *GridLayoutManager* cu *spanCount* cu valoarea 2, ce conține elemente de tipul *item_books.xml*. Folosind o listă de tip *CategoryModel* în care adăugăm toate categoriile afișate (fantasy, lifestyle, fashion, romance, mystery, historic, sci-fi, classic, modern, horror, young adult, thriller, manga) și un adapter care preia aceasta listă 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 încărca 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 *RecyclerView*, 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 in pagina următoare si deschidem pagina *Genre List* printr-un nou intent.

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

3.3.2 Filtrarea cărților după categoria aleasa

In 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*.

in 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 in clasa *Genres.java*. Cele care, in urma comparatiei, returneaza valoarea *True*, sunt pastrate intr-o lista de tip *Books*.

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

Acest adapter asigura afișarea cărților in pagina, in 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, in *Document.java* si *ViewPDF.java*. In clasa *Document.java* declarăm 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 in aplicație cărțile prin url.

PdfViewPager este un API dezvoltat de *voghDev*, instalat prin inserarea in 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 in *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.

In *ViewPDF.java* se încarcă documentul de tip pdf prin link-ul trimis in *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 in 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. In cadrul widgetului de SearchView se pot introduce cuvintele cheie pentru găsirea documentului căutat după titlu, iar in cel de-al doilea widget se afișează, pe doua coloane, cărțile care conțin String-urile introduse in bara de search.

Căutarea si afișarea in pagina sunt implementate in Search.java cu ajutorul unui adapter numit *BookListAdapterSearch.java*. Se declara in *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 in 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. In 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 in titlu. In 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 in pagina, in RecyclerView, prin funcția *publishResults*.

Aceasta funcție de filtrare se va apela la fiecare modificare a textului introdus in 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 in pagina *genre_list.xml*, in doua 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;
    }
}
```

```
@SuppressWarnings("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

Bibliografie

2.3.4-

<https://firebase.google.com/docs/database/?msclkid=d8c21cddd06e11eca05a5f9ddfe4bfb6>

2.3.5- <https://firebaseopensource.com/projects/firebase/firebaseui-ios/database/readme/?msclkid=edbfb51d07111ec9ba52a1efa703369>

2.3.6- <https://firebase.google.com/docs/storage>