|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ  DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI |  |

PROIECT DE DIPLOMĂ

Romanescu Radu Andrei

COORDONATOR ȘTIINȚIFIC

Prof.univ.dr.ing. Sorin Ilie

Septembrie 2023

CRAIOVA

|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI  ELECTRONICĂ  DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI |  |

Book rent Service

Romanescu Radu Andrei

COORDONATOR ȘTIINȚIFIC

Dr.ing. Sorin Ilie

Septembrie 2023

CRAIOVA

*„Învățătura este o comoară care își urmează stăpânul pretutindeni.”*

Proverb popular

**DECLARAȚIE DE ORIGINALITATE**

Subsemnatul Romanescu Radu Andrei student la specializarea Calculatoare și tehnologia informației în limba Romana din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoştinţă de cele prezentate mai jos şi că îmi asum, în acest context, originalitatea proiectului meu de licenţă:

* cu titlul Book Sharing Service,
* coordonată de Prof. univ. dr. ing. SORIN ILIE,
* prezentată în sesiunea Septembrie 2023

La elaborarea proiectului de licenţă, se consideră plagiat una dintre următoarele acţiuni:

* reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referinţa precisă,
* redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
* prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
* însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

* plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
* indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
* precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
* precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data, Semnătura candidatului,

|  |  |  |
| --- | --- | --- |
|  | UNIVERSITATEA DIN CRAIOVA  Facultatea de Automatică, Calculatoare şi Electronică  Departamentul de latoare și Tehnologia Informației | Aprobat la data de  …………………  Şef de departament,  Prof. dr. ing.  Şef de departament, Prof. dr. ing.  Nicolae Enescu |
|  |  |  |

**PROIECTUL DE DIPLOMĂ**

|  |  |
| --- | --- |
| Numele și prenumele studentului/-ei: | Romanescu Radu Andrei |
| Enunțul temei: | Book sharing service |
| Datele de pornire: | Aplicatie de sharing carti ca o alternativa a librariilor |
| Conținutul proiectului: | 1.Introducere  2.Concepte Teoretice  3.Descrierea Aplicatiei  4.Concluzii |
| Material grafic obligatoriu: | Power Point care conține cod, capturi de ecran, diagrame, explicații |
| Consultații: | Periodice |
| Conducătorul științific  (titlul, nume și prenume, semnătura): | Prof.univ.dr.ing.Sorin Ilie |
| Data eliberării temei: |  |
| Termenul estimat de predare a proiectului: |  |
| Data predării proiectului de către student și semnătura acestuia: |  |

**REZUMATUL PROIECTULUI**

Aplicația de *Book* *Sharing* *Serivce* iși are ca scop de a oferi o alternativă pentru librăriile fizice. Utilizatorii iși pot posta pe aplicație colecția de carți pe care alți utilizatori o vor putea vizualiza si daca sunt atrași de o carte o pot împrumuta. Metoda livrarii se va face la acordul dintre utilizatorul care deține cartea si utilizatorul care vrea sa o împrumute.

Pentru această aplicație am folosit limbajul *Java* si IDE-ul *Android* *Studio*.

Am ales limbajul java deoarece sunt mult mai familiar cu el fata de alternativa *Kotlin* si mi-a oferit mult mai mult control asupra implementarii.

*IDE*-ul *Android* *Studio* a fost o alegere foarte usoara datorită facilitătiilor oferite pentru dezvoltarea aplicațiilor Android si in special pentru funcționalitatea de emulare care este ușoara de folosit si ofera un mediu de testare optim.

Pentru partea de baza de date am folosit *Firebase* oferit de *Google*. Această baza de date este foarte ușoara de folosit si integrat si ofera suport cloud dupa integrarea aplicației astfel încat nu mai este nevoie de un setup adițional atunci cand rulăm aplicația de pe alt dispozitiv.

Principalele funcționalităti ale utilizatorului sunt de a isi înregistra un cont nou si opțiunea de logare cu acel cont, logare directă cu contul de *Google* folosind *API*-ul *Google,* adăugarea de carți noi împreuna cu detaliile aferente carțiilor ( imagine ,autor ,titlu) in lista de carți valabile pentru ceilalti utilizatori, optiunea de a împrumuta o carte valabila, opțiunea de a îsi schimba parola sau a îsi șterge contul din baza de date comform regulilor *GDPR*.

În concluzie, acest proiect m-a învatat cum sa gândesc o arhitectura, cum pot apărea probleme pe care inițial nu le-am conștientizat si cum să le gestionez si depedența strânsa între funcționalitățiile prezente într-o aplicație. Aceste aspecte m-au ajutat sa devin un programator mai bun si sa îmi dezvolt gândirea critică in gestionarea unei aplicații înainte de a scrie prima linie de cod.

***Termenii cheie***: carti , Java , Android Studio , Firebase

CUPRINS

[1 Introducere 1](#_Toc143947024)

[1.1 Scopul 1](#_Toc143947025)

[1.2 Motivația 1](#_Toc143947026)

[2 Concepte teoretice 2](#_Toc143947027)

[2.1 Tehnologii folosite 2](#_Toc143947028)

[2.1.1 Java 2](#_Toc143947029)

[2.1.2 Android Studio 3](#_Toc143947030)

[2.1.3 Firebase 4](#_Toc143947031)

[2.1.4 XML 5](#_Toc143947032)

[2.2 Motivație pentru alegeri 5](#_Toc143947033)

[3 despre book sharing service 6](#_Toc143947034)

[3.1 Design-ul sistemului 6](#_Toc143947035)

[3.1.1 Cazuri de utilizare 6](#_Toc143947036)

[3.1.2 Diagrame de activitate 7](#_Toc143947037)

[3.1.3 Diagrame de secvență 16](#_Toc143947038)

[3.1.4 Arhitectura de sistem 18](#_Toc143947039)

[3.1.5 Structura bazei de date 19](#_Toc143947040)

[3.2 Funcționalități 21](#_Toc143947041)

[3.2.1 Înregistrare 22](#_Toc143947042)

[3.2.2 Logare 22](#_Toc143947043)

[3.2.3 Afișarea cărțiilor postate de utilizatori 23](#_Toc143947044)

[3.2.4 Postarea unei cărți 24](#_Toc143947045)

[3.2.5 Închirierea unei cărți 25](#_Toc143947046)

[3.2.6 Căutarea unei cărți 27](#_Toc143947047)

[3.2.7 Ștergerea contului 27](#_Toc143947048)

[3.2.8 Resetarea parolei 28](#_Toc143947049)

[3.2.9 Gestionarea requesturilor 28](#_Toc143947050)

[3.2.10 Delogare 30](#_Toc143947051)

[3.2.11 Chat 31](#_Toc143947052)

[3.3 Manualul utilizatorului. 31](#_Toc143947053)

[4 Concluzii 41](#_Toc143947054)

[4.1 Dezvoltarea aplicației in viitor 41](#_Toc143947055)

[5 Bibliografie 41](#_Toc143947056)

[6 Referințe web 41](#_Toc143947057)

[A. Codul sursă 42](#_Toc143947058)

[A. website-ul proiectului 54](#_Toc143947059)

[A. CD / DVD 55](#_Toc143947060)

[Index 55](#_Toc143947061)

**LISTA FIGURILOR**

FIGURA 1. DIAGRAMA CAZURILOR DE UTILIZARE 6

FIGURA 2. DIAGRAMA DE ACTIVITATE PENTRU ÎNREGISTRARE……………………………7

FIGURA 3. DIAGRAMA DE ACTIVITATE PENTRU LOGARE 8

FIGURA 4. DIAGRAMA DE ACTIVITATE PENTRU CĂUTAREA UNEI CĂRȚI ..9

FIGURA 5. ACTIVITY DIAGRAM FOR REGISTER 12

FIGURA 6. ACTIVITY DIAGRAM FOR LOG IN 12

FIGURA 7. ACTIVITY DIAGRAM FOR CREATE A NEW ALBUM 13

FIGURA 8. ACTIVITY DIAGRAM FOR UPDATE ALBUM 14

FIGURA 9. ACTIVITY DIAGRAM FOR DELETE ALBUM 14

FIGURA 10. ACTIVITY DIAGRAM FOR VIEW ALBUM’S DETAILS 15

FIGURA 11. ACTIVITY DIAGRAM FOR SEARCH ALBUM 15

FIGURA 12. ACTIVITY DIAGRAM FOR ADD COMMENT 16

FIGURA 13. ACTIVITY DIAGRAM FOR ADD FRIEND 17

FIGURA 14. ACTIVITY DIAGRAM FOR REMOVE FRIENDS 17

FIGURA 15. ACTIVITY DIAGRAM FOR ADD ALBUM TO FAVOURITES 18

FIGURA 16. ACTIVITY DIAGRAM FOR ADD ALBUM TO WISHLIST 18

FIGURA 17. ACTIVITY DIAGRAM FOR ADD ALBUM TO LISTENING 19

FIGURA 18. ACTIVITY DIAGRAM FOR REMOVE ALBUM FROM FAVOURITES 20

FIGURA 19. SEQUENCE DIAGRAM FOR ADDING AN ALBUM 20

FIGURA 20. SEQUENCE DIAGRAM FOR DELETING AN ALBUM 21

FIGURA 21. SEQUENCE DIAGRAM FOR UPDATING AN ALBUM 21

FIGURA 22. SEQUENCE DIAGRAM FOR ADDING AN ALBUM TO FAVOURITES 22

FIGURA 23. SEQUENCE DIAGRAM FOR REMOVING AN ALBUM FROM FAVOURITES 22

FIGURA 24. SEQUENCE DIAGRAM FOR SEARCH AN ALBUM 23

FIGURA 25. SYSTEM ARCHITECTURE SCHEMA 24

FIGURA 26. RELATIONAL MODEL DIAGRAM OF RECORDCLIQUE DB 25

FIGURA 27. ENTITY RELATIONSHIP DIAGRAM OF RECORDCLIQUE DB 26

FIGURA 28. ADD ALBUM CODE 28

FIGURA 29. DELETE ALBUM CODE 29

FIGURA 30. UPDATE ALBUM CODE 30

FIGURA 31. VIEW ALBUM’S DETAILS CODE 31

FIGURA 32. SEARCH ALBUM CODE 31

FIGURA 33. ADD FRIEND CODE 32

FIGURA 34. REMOVE FRIEND CODE 3

FIGURA 35. VIEW FRIENDS CODE 33

FIGURA 36. ADD COMMENT CODE 34

FIGURA 37. VIEW ALBUM’S COMMENTS CODE 34

FIGURA 38. VIEW NEWSFEED INDEX CODE 35

FIGURA 39. NEWSFEEDLOGEXTENSIONS 36

FIGURA 40. VIEW ADD ALBUM TO FAVOURITES CODE 36

FIGURA 41. VIEW ADD ALBUM TO LISTENING CODE 37

FIGURA 42. VIEW ADD ALBUM TO WISHLIST CODE 38

# Introducere

## Scopul

Scopul aplicației de *Book* *Sharing* *Service* este de a ii oferi utilizatorului o alternativă librariilor clasice. Utilizatorii isi pot împrumuta carti intre ei si le pot returna conform hotărarii dintre cei 2 utilizatori. Aceasta aplicație isi are ca scop adunarea de cât mai mulți utilizatori pentru a avea o librarie vastă de carți disponibile. Cu cât numarul de utilizatori este mai mare cu atât aplicația este mai folositoare pentru utilizatorii de rând.

In acest scop aplicația nu va conține tranzacții monetare pentru imprumutarea de carti. Aceasta decizie cântărește foarte mult in atragerea utilizatoriilor.

Deoarece librăriile cer un preț pentru cumpărarea carțiilor, un utilizator va fii mult mai atras de opțiunea citirii cărtiilor respective pe gratis. Modelul este asemănător cu cel al bibliotecilor. Aducerea acestui model in format *online* si mobil reprezintă o îmbunătațire pentru utilizator deoarece nu mai este nevoie sa se deplaseze fizic la bibliotecă pentru a viziona librăria oferită sau a împrumuta carți.

Aplicația ofera utilizatorului o interfața ușor de folosit si un proces de înregistrare foarte ușor de urmat.

## Motivația

Într-o lume în care resursele naturale sunt limitate, promovarea sustenabilității devine vitală. Prin intermediul acestei aplicatii , poți contribui la reducerea consumului de hârtie și la crearea unei mentalități de consum responsabil. Fiecare carte împărtășită înseamnă o reducere a amprentei ecologice și o investiție într-un viitor mai verde.

Pe langa acest aspect , aplicația *Book* *Sharing* *Service* reprezintă o îmbunătațire a calității vieții minimizând efortul si banii necesari pentru a te bucura de o carte.

# Concepte teoretice

## Tehnologii folosite

### Java

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor ʼ90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care Oracle, IBM, Bea, FSF. În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.[Wik02]

Avantaje ale programării pe obiect în Java:

Reutilizarea codului: OOP încurajează crearea de clase și obiecte care pot fi reutilizate în diferite părți ale unei aplicații sau chiar în diferite aplicații. Acest lucru duce la un cod mai eficient și mai ușor de întreținut.

Modularitate: OOP încurajează împărțirea codului în module (clase), fiecare gestionând o funcționalitate specifică. Această modularitate face dezvoltarea și întreținerea mai ușoară, deoarece modificările pot fi limitate la modulele relevante.

Încapsulare: OOP permite ascunderea detaliilor interne ale unei clase și furnizează doar o interfață publică pentru interacțiune. Aceasta îmbunătățește securitatea, deoarece accesul la date sensibile este controlat și oferă o mai mare flexibilitate pentru a schimba implementările interne fără a afecta utilizatorii clasei.

Abstractizare: OOP permite definirea de clase abstracte și interfețe care furnizează schelete pentru alte clase derivate. Aceasta facilitează concentrarea asupra conceptelor cheie ale unei aplicații și ajută la reducerea complexității.

Polimorfism: OOP permite utilizarea unui obiect în mai multe forme. Asta înseamnă că puteți trata diverse obiecte derivate ca obiecte ale aceleiași clase de bază. Acest lucru conduce la flexibilitate în implementare și extensibilitate.

Dezavantaje ale programării pe obiect în Java:

Complexitate: Programarea pe obiect poate aduce o complexitate crescută în comparație cu abordări mai procedurale. Definirea claselor, relațiilor dintre ele și gestionarea interacțiunilor poate necesita o înțelegere mai profundă și poate fi mai dificil de implementat în anumite situații.

Memorie: Utilizarea obiectelor poate consuma mai multă memorie în comparație cu abordări mai strânse la nivel de memorie, deoarece fiecare obiect poartă cu sine informații suplimentare legate de clasă și de gestionarea memoriei.

Curba de învățare: Trecerea de la programarea procedurală la cea pe obiect poate necesita o curba de învățare, deoarece implică concepte noi precum clase, obiecte, moștenire, polimorfism etc.

### Android Studio

Android Studio este un mediu de dezvoltare (engl. software development environment, sau integrated development environment - "mediu integrat de dezvoltare) pentru colaborarea cu platforma Android, anunțată pe data de 16 mai 2013 în cadrul conferinței I / O Google.

IDE este disponibil gratuit începând cu versiunea 0.1, publicată în mai 2013, apoi a trecut la testarea beta, începând cu versiunea 0.8, care a fost lansată în iunie 2014. Prima versiune stabilă 1.0 a fost lansată în decembrie 2014, apoi suportul pentru pluginul Android Development Tools (ADT) pentru Eclipse a încetat.

Android Studio este bazat pe software-ul IntelliJ IDEA de la JetBrains, este instrumentul oficial de dezvoltare a aplicațiilor Android. Acest mediu de dezvoltare este disponibil pentru Windows, OS X și Linux. Pe 17 mai 2017, la conferința anuală Google I / O, Google a anunțat asistență pentru limbajul Kotlin utilizat de Android Studio ca limbaj de programare oficial pentru platforma Android, pe lângă Java și C ++.[Wik01]

### Firebase

Firebase este un set cuprinzător de instrumente pentru crearea și implementarea de aplicații și jocuri mobile, bazate pe serviciile cloud Google Cloud Platform (GCP). Utilizând potențialului serviciilor GCP, extinderea proiectului nu necesită investiții costisitoare în infrastructură. Firebase nu oferă numai soluții pentru backend. De asemenea, oferă soluții în domeniul testării, promovării și dezvoltării angajamentului, precum și analiză avansată a datelor.

Serviciul acceptă crearea de produse (aplicații și jocuri) pentru iOS, Android și versiuni web direct pregătite de rulare într-un browser web PWA (Progressive Web Apps).

Firebase este compus dintr-un total de 19 produse împărțite în trei module mai mari: Build (instrumente pentru dezvoltarea aplicațiilor), Release & monitor (instrumente pentru îmbunătățirea calității) și Engage (instrumente pentru creșterea afacerii). Mediul poate fi extins cu funcții suplimentare datorită API-ului deschis.

Firebase este un instrument cheie, parte din kit-ul de dezvoltare pus la dispoziție de Google cloud. Câteva dintre funcțiile Firebase Google sunt:

* Firebase Analytics. Google Analytics îți prezintă date corelate cu comportamentul utilizatorilor în aplicații iOS sau Android. Astfel, îți va susține luarea de decizii bazate exclusiv pe cifre, menite să îmbunătățească performanța companiei.
* Firebase Authentication. Cu Firebase Authentication, dezvolatorii pot construi sisteme sigure de autentificare și pot facilita experiența înscrierii și logării pentru utilizatori.
* Cloud Messaging. Acest serviciu intermediază livrarea și primitea de mesaje pentru companii pe toate platformele, fără niciun cost adițional.
* Firebase Realtime database. Serviciul de baze de date Firebase este un database NoSQL bazat pe cloud care permite stocarea și sincronzarea datelor între utiliatori în timp real. De asemenea, cu Firebase, datele sunt disponibile pentru clienți chiar și când aplicația este offline.

Avantajele utilizării Firebase în dezvoltarea de aplicații:

* Un produs gratuit pentru începători. Platforma nu te taxează pentru marea majoritate a serviciilor. Variantele plătite intervin abia după un anumit prag de memorie folosită în bazele de date. Astfel, este un produs accesibil începătorilor care au nevoie de o platformă pe care să testeze fezabilitatea produsului.
* Un produs popular. Peste 1,5 milioane de aplicații sunt construite cu Firebase. Platforma se bucură de o comunitate mare și de numeroase resurse puse astfel la dispoziție programatorilor care o folosesc.
* Firestore și Realtime sunt opțiuni Firebase de baze de date pe care le poți alege pentru construirea unui produs. Ambele sunt database în cloud NoSQL, flexibile și scalabile, care îți oferă acces offline și în timp real pentru un răspuns rapid și competitiv.
* Integrare ușoară. Nu în ultimul rând, API-urile oferite de Firebase îți permit să adaugi noi funcții aplicațiilor tale cu doar câteva click-uri.[Fot01]

### XML

Extensible Markup Language (XML) este un meta-limbaj de marcare recomandat de Consorțiul Web pentru crearea de alte limbaje de marcare, cum ar fi XHTML, RDF, RSS, MathML, SVG, OWL etc. Aceste limbaje formează familia de limbaje XML.

Meta-limbajul XML este o simplificare a limbajului SGML (din care se trage și HTML) și a fost proiectat în scopul transferului de date între aplicații pe internet, descriere structură date.

XML este acum și un model de stocare a datelor nestructurate și semi-structurate în cadrul bazelor de date native XML.

Datele XML pot fi utilizate în limbajul HTML, permit o identificare rapidă a documentelor cu ajutorul motoarelor de căutare. Cu ajutorul codurilor javascript, php etc. fișierele XML pot fi înglobate în paginile de internet, cel mai elocvent exemplu este sitemul RSS care folosește un fișier XML pentru a transporta informațiile dintr-o pagină web către mai multe pagini web.

Avantaje:

* extensibilitate (se pot defini noi indicatori dacă este nevoie)
* validitate (se verifică corectitudinea structurală a datelor )
* oferă utilizatorilor posibilitatea de a-și reprezenta datele într-un mod independent de aplicație
* XML este simplu și accesibil (sunt fișiere text create pentru a structura, stoca și a transporta informația)
* poate fi editat, modificat foarte ușor (necesită doar un editor de text simplu precum notepad, wordpad etc.) [Wik03]

## Motivație pentru alegeri

Am ales aceste tehnologii deoarece deși nu am fost familiar cu ele la început, au fost accesibile de ințeles si folosit. Folosirea acestor tehnologii noi a fost o provocare noua pentru mine dar din care am avut foarte multe de învațat si m-au ajutat in a-îmi dezvolta gandirea de programator.

# despre book sharing service

## Design-ul sistemului

Aplicația iși are ca scop facilitarea unor nevoi a iubitorilor de carți. Aceste nevoi implica o usurința in a obține carțiile favorite și formarea unei comunițați a iubitorilor de carți.

### Cazuri de utilizare

Aplicația ofera un mediu indepedent utilizatorului , oferindu-i o mare autonomie in modul de utilizare.

Cazurile de utilizare pentru un utilizator sunt descrise mai jos.

1.Înregistrarea unui cont nou.

2.Logarea in aplicație folosind un cont nou / un cont deja existent.

3.Opțiunea de logare folosind contul de Google.

4.Opțiunea de a adăuga o carte pentru a fii vizualizată si preluată de alți utilizatori.

5.Opțiunea de a căuta o carte in lista de cărți valabile.

6.Opțiunea de a trimite un *request* către un utilizator pentru a prelua cartea respectiva postată.

7.Opțiunea de a vizualiza *requesturile* primite de la alți utilizatori.

8.Opțiunea de a vizualiza libraria impreună cu *status*-ul carțiilor.

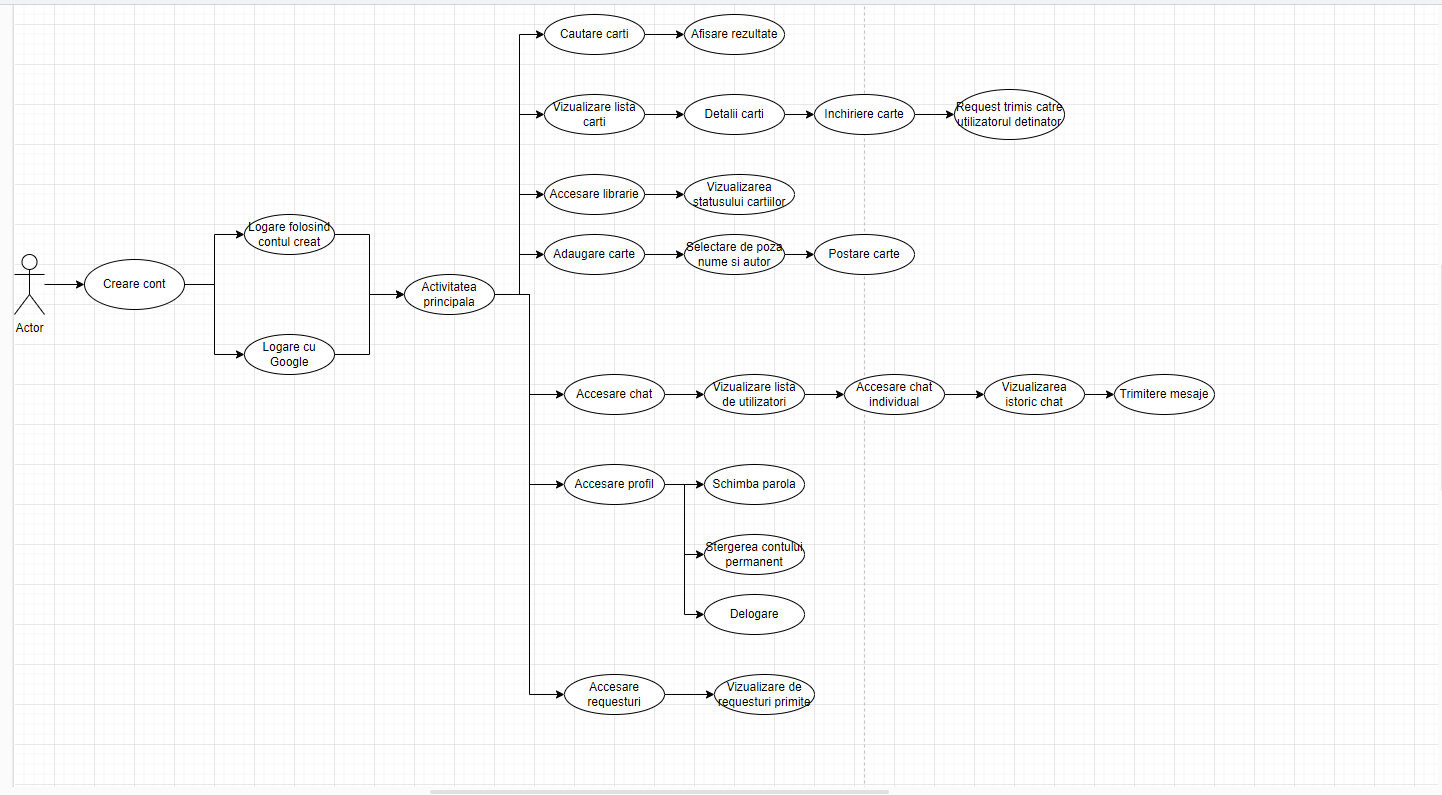
9.Opțiunea de delogare.

10.Opțiunea de a schimba parola.

11.Opțiunea de a șterge contul permanent.

12.Opțiunea de a comunica cu alți utilizatori ai aplicației folosind *chat*-ul integrat.

13.Opțiunea de a vizualiza detalii despre carțiile valabile.

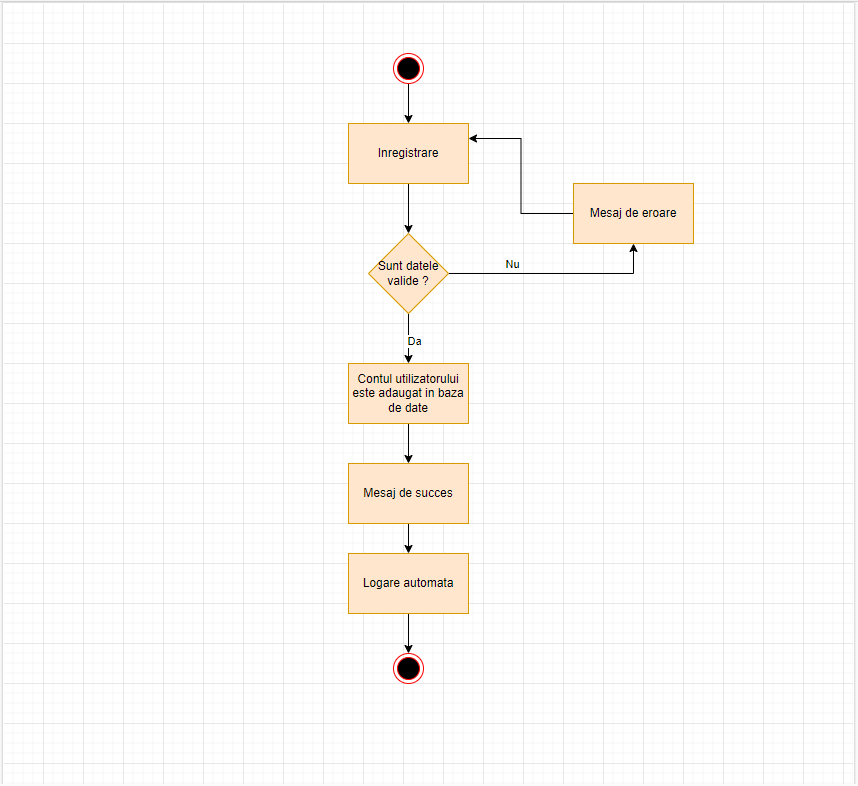


**Figura 1. Diagrama cazurilor de utilizare**

### Diagrame de activitate

**Înregistrare**

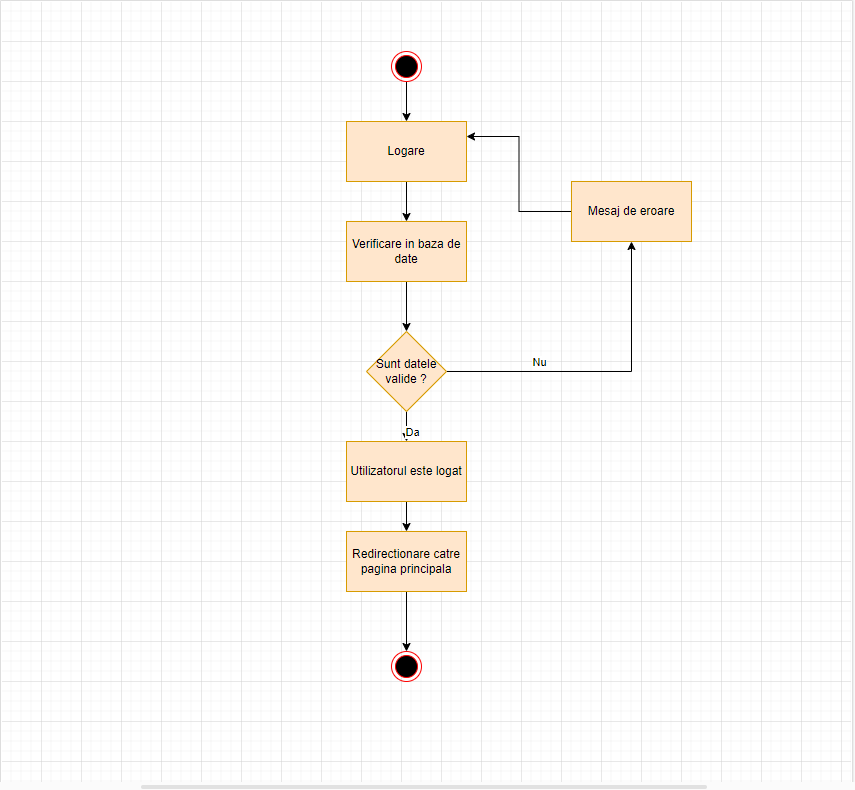
Diagrama de activitate pentru înregistrare este următoarea:



**Figura 2. Diagrama de activitate pentru înregistrare**

**Logare**

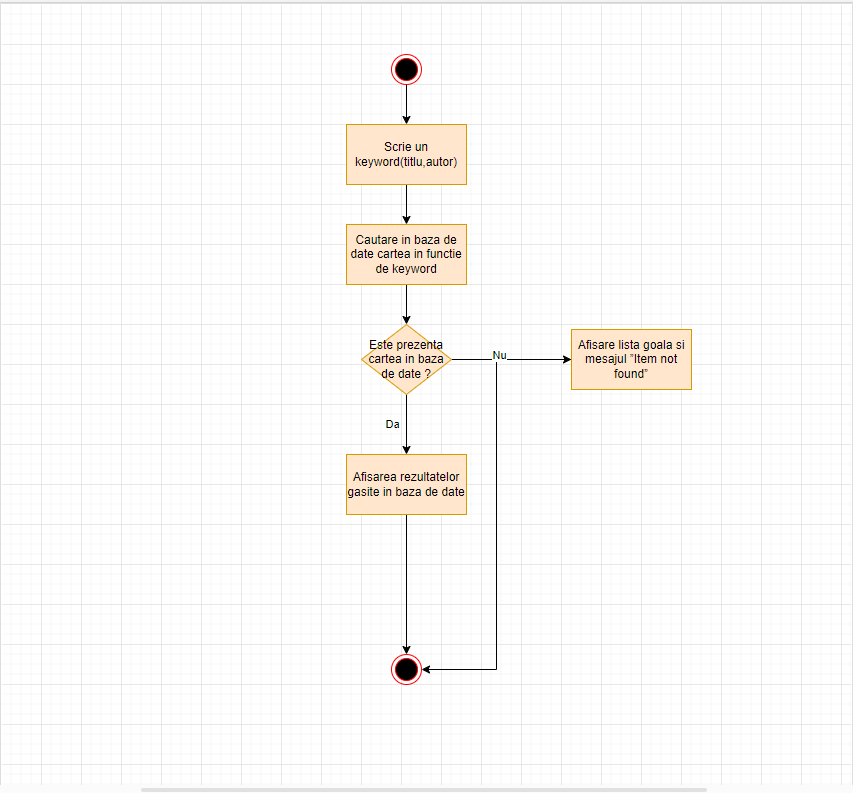
Diagrama de activitate pentru logare este următoarea:



**Figura 3. Diagrama de activitate pentru logare**

**Căutare cărți**

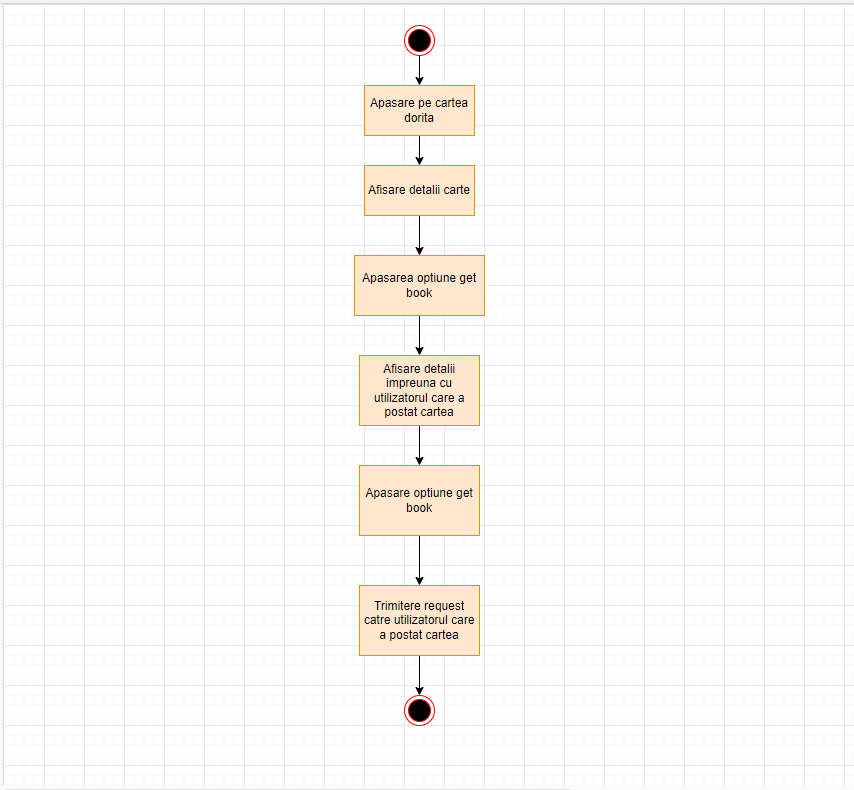
Diagrama de activitate pentru căutarea unei cărți este următoarea:



**Figura 4. Diagrama de activitate pentru căutarea unei cărți**

**Închiriere cârți:**

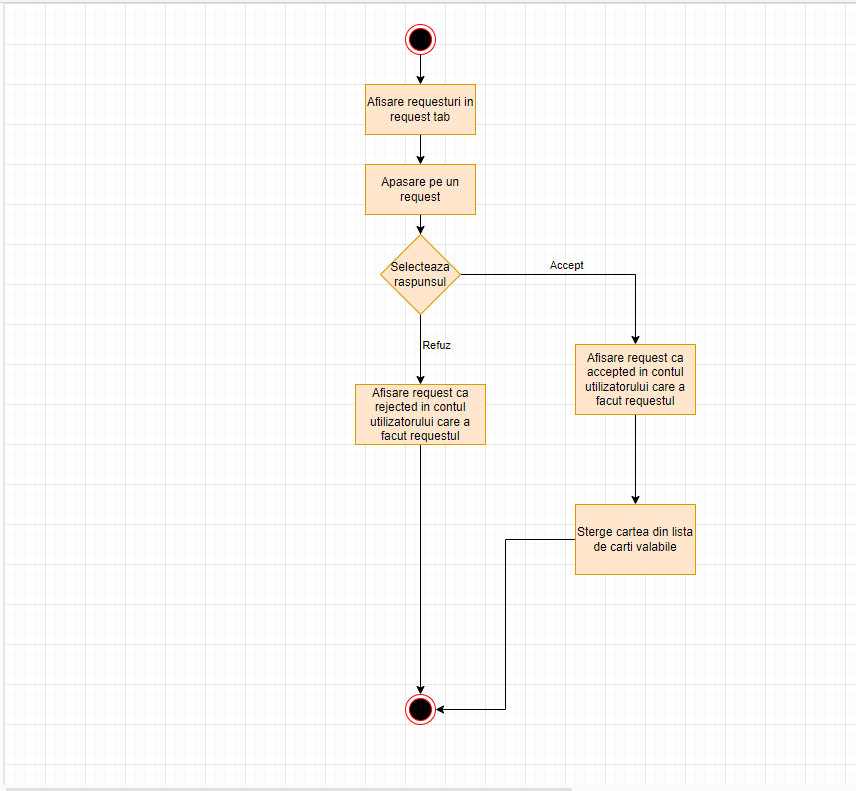
Diagrama de activitate pentru închirierea unei cărți este următoarea:



**Figura 5. Diagrama de activitate pentru închirierea unei cărți**

**Gestionarea Requesturilor:**

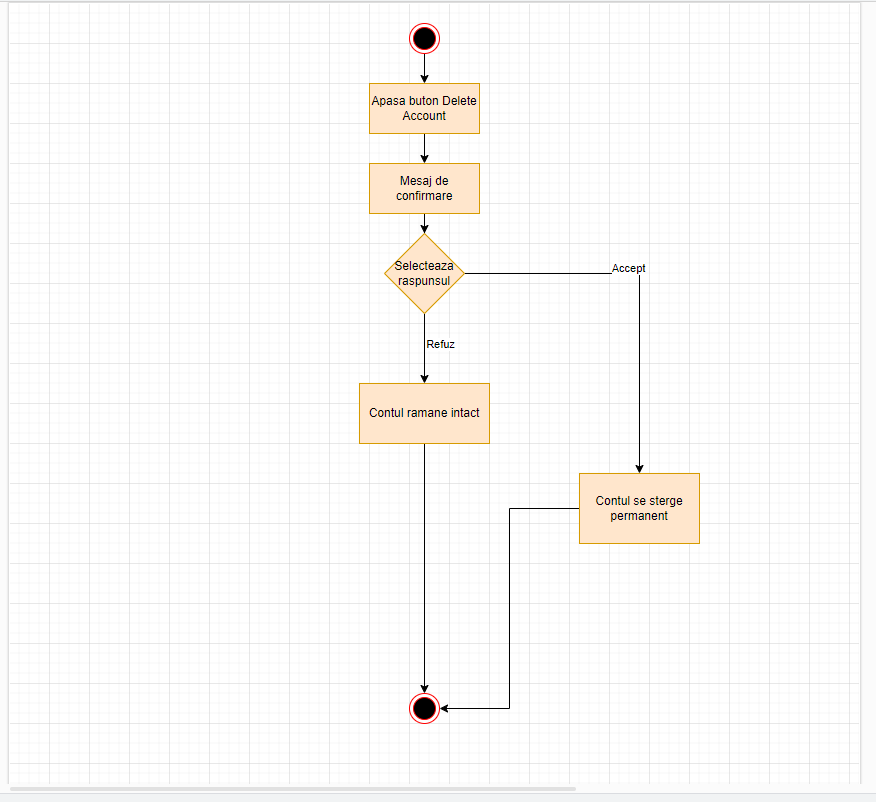
Diagrama de activitate pentru gestionarea requesturilor este următoarea:



**Figura 6. Diagrama de activitate pentru gestionarea requesturilor**

**Ștergerea contului**

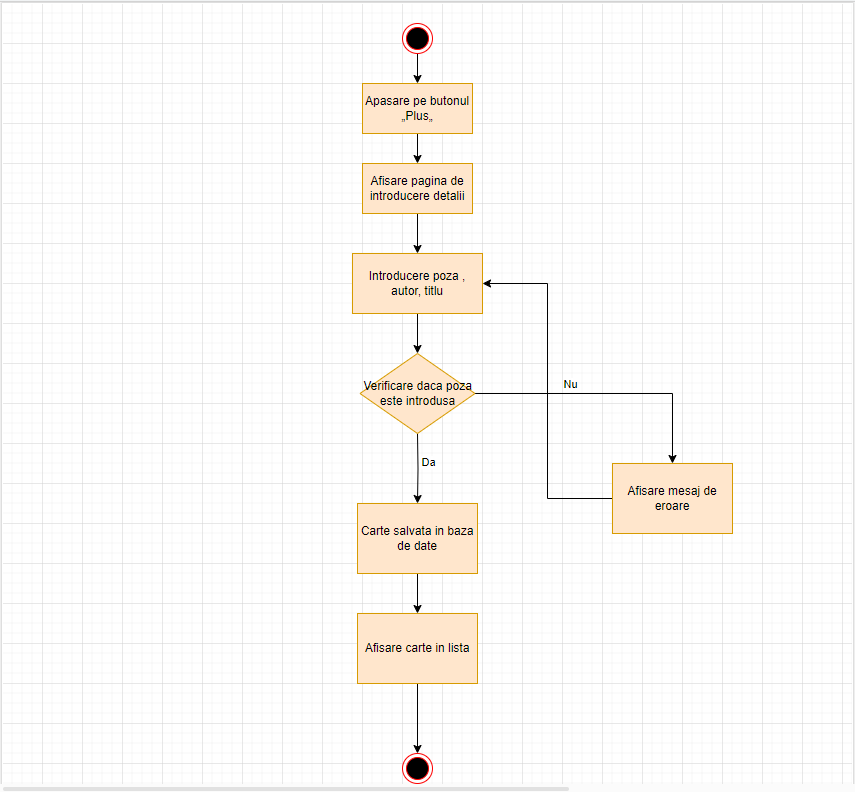
Diagrama de activitate pentru stergerea contului este următoarea:



**Figura 7. Diagrama de activitate pentru stergerea contului**

**Postarea unei cărți**

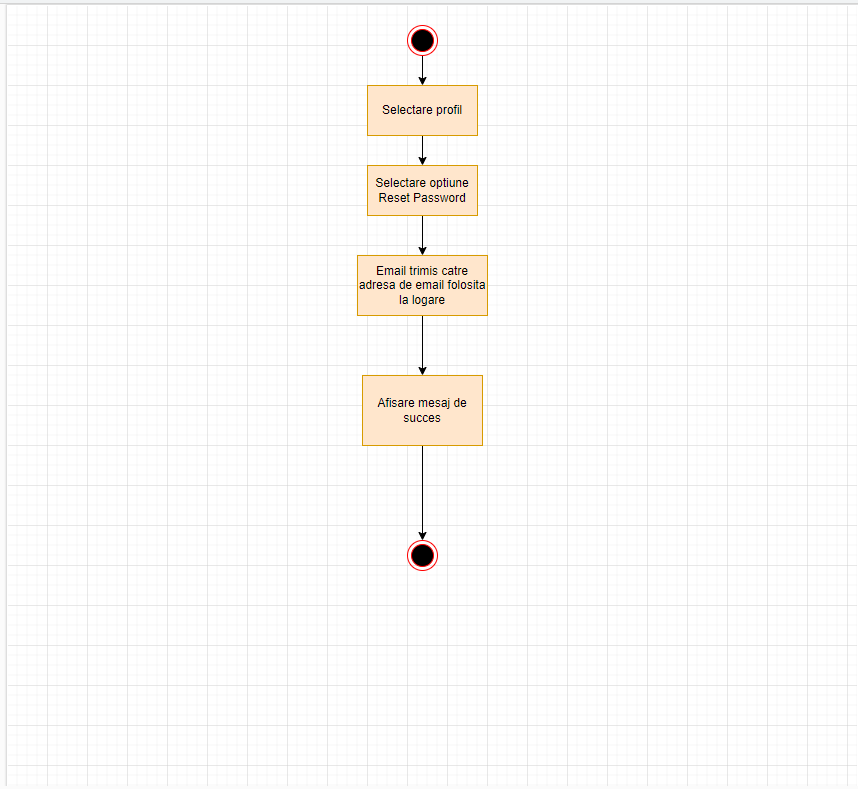
Diagrama de activitate pentru postarea unei cărți este următoarea:



**Figura 8. Diagrama de activitate pentru postarea unei cărți**

**Schimbarea parolei:**

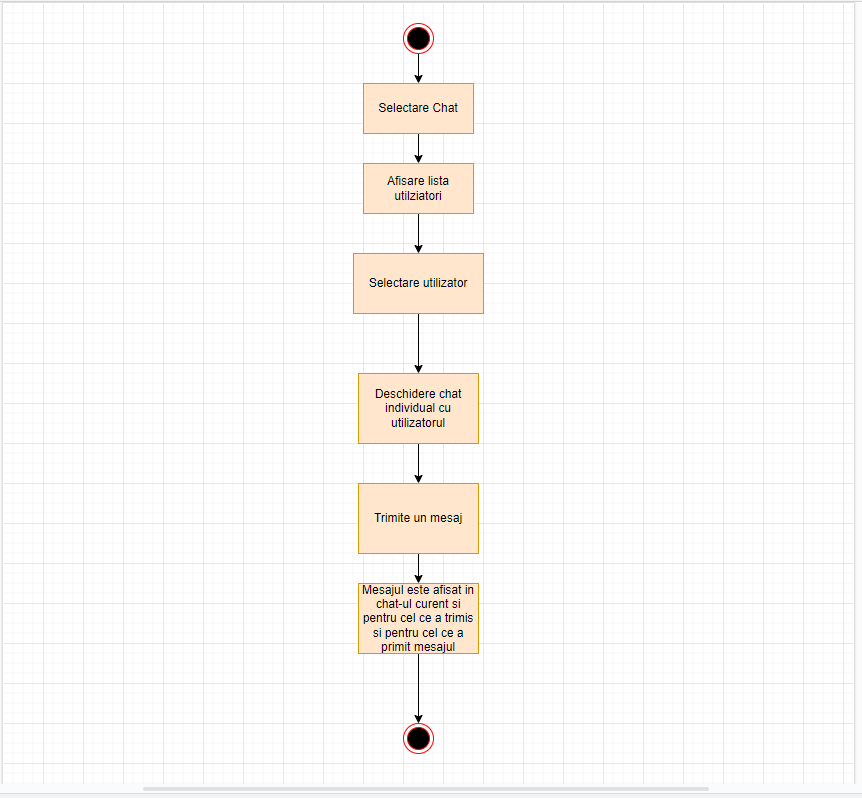
Diagrama de activitate pentru schimbarea parolei este următoarea:



**Figura 9. Diagrama de activitate pentru schimbarea parolei**

**Folosirea funcționalitații de chat:**

Diagrama de activitate pentru folosirea funcționalitații de chat este următoarea:

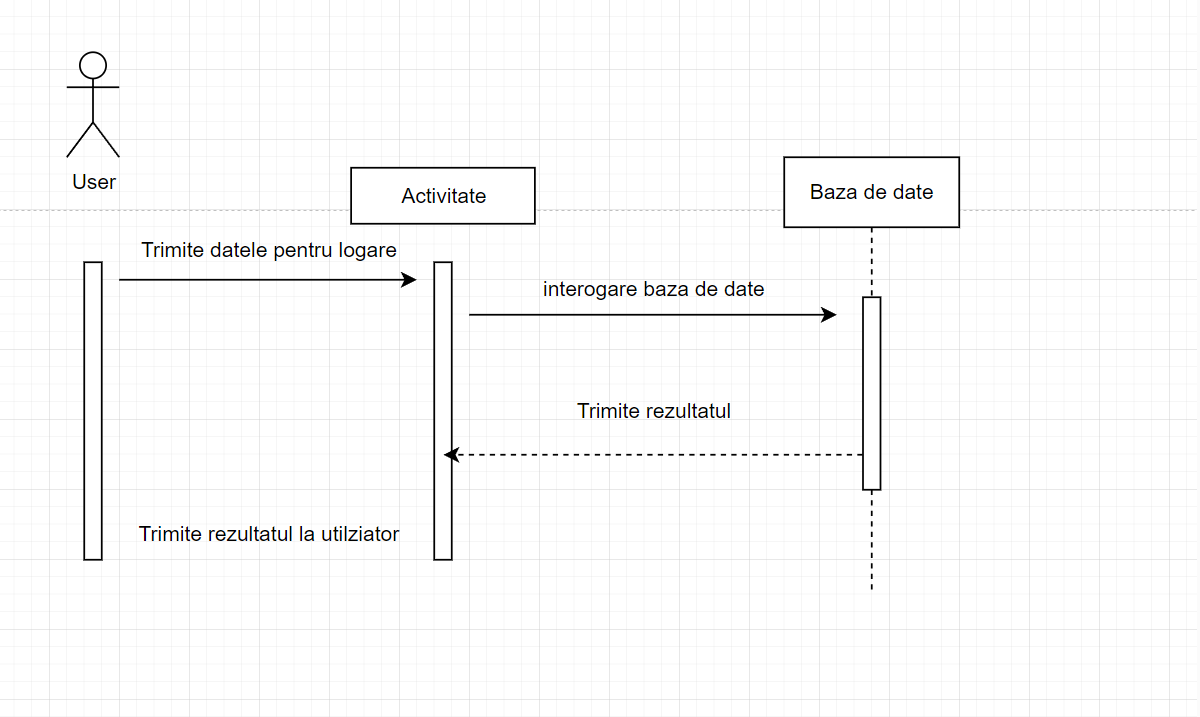
****

**Figura 10. Diagrama de activitate pentru folosirea funcționalitații de chat**

### Diagrame de secvență

**Logare**

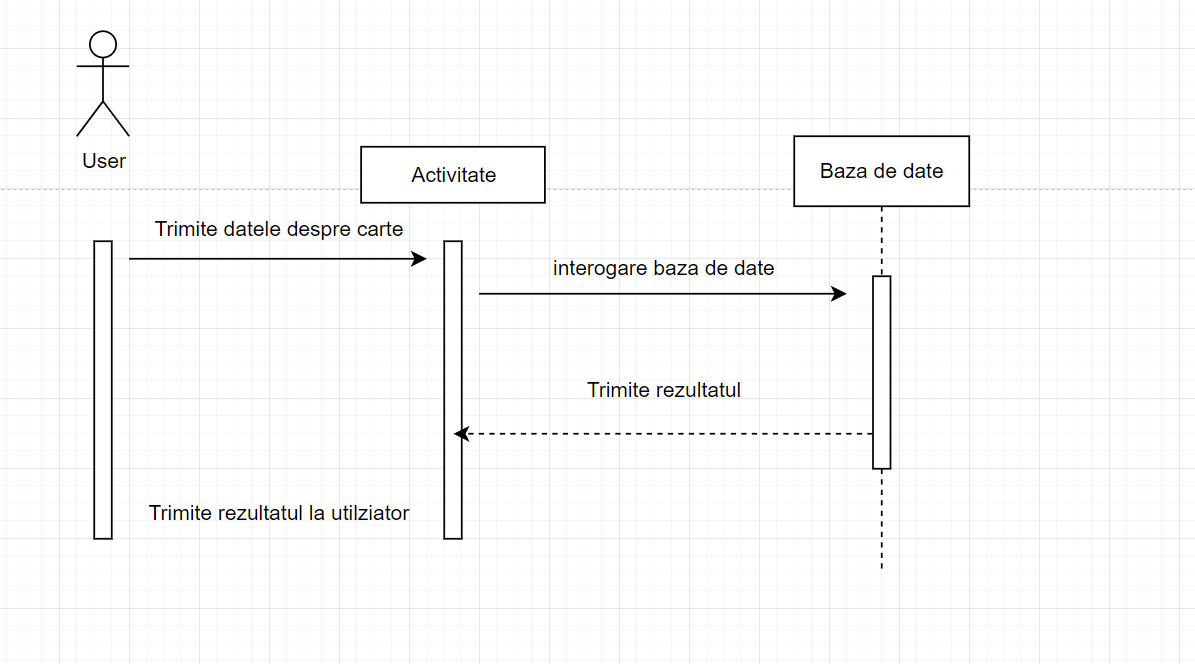
Diagrama de secență pentru logare este urmatoarea:



**Figura 11. Diagrama de secență pentru logare**

**Căutarea unei cărți:**

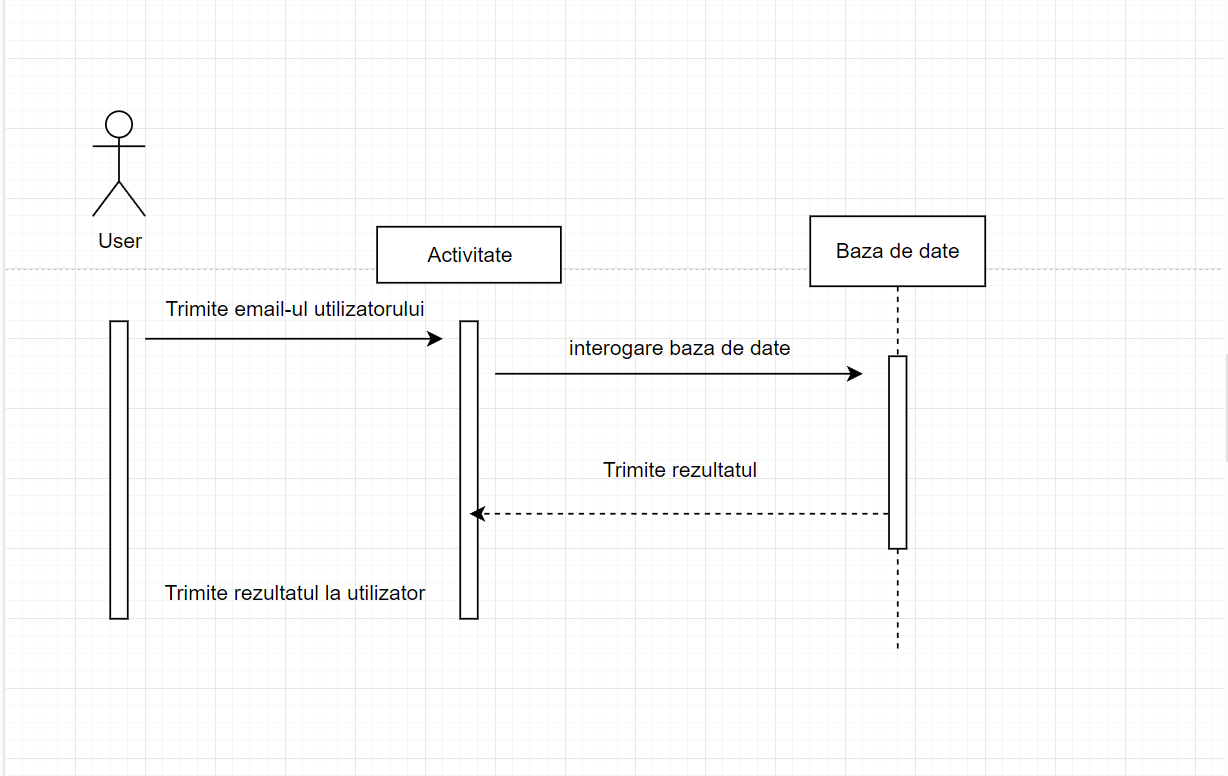
Diagrama de secvență pentru Căutarea unei cărți este urmatoarea:



**Figura 12. Diagrama de secvență pentru Căutarea unei cărți**

**Schimbarea parolei:**

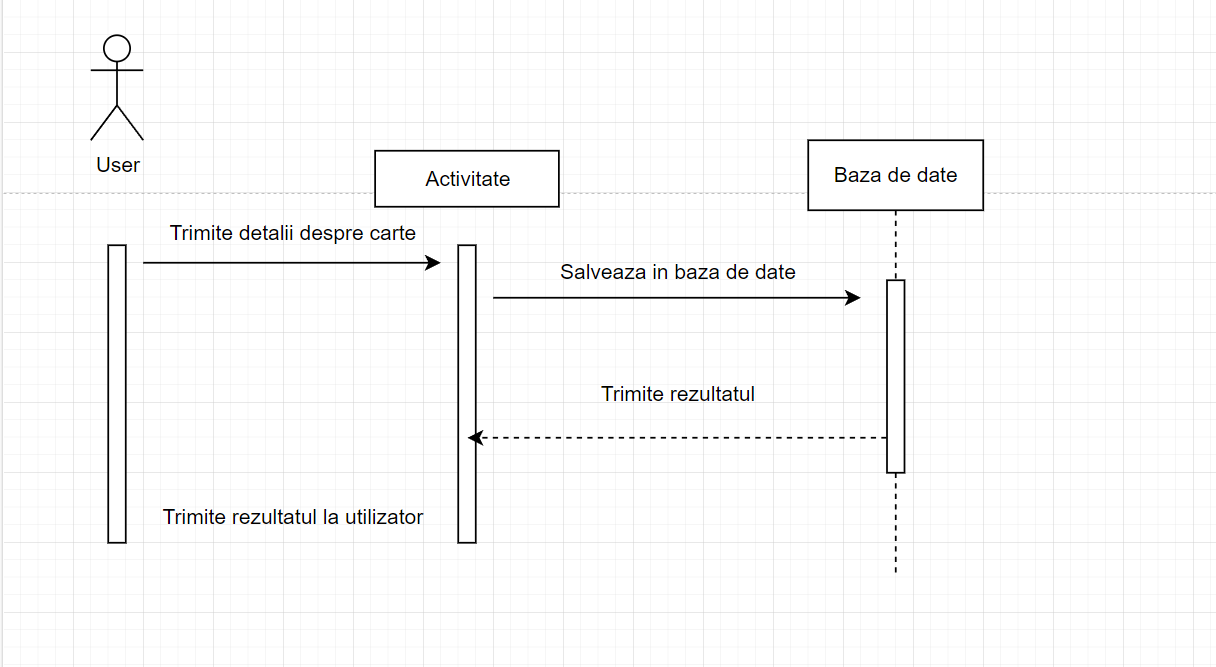
Diagrama de secvență pentru schimbarea parolei este următoarea:



**Figura 13. Diagrama de secvență pentru schimbarea parolei**

**Postarea unei cărți:**

Diagrama de secvență pentru Postarea unei cărți este următoarea:



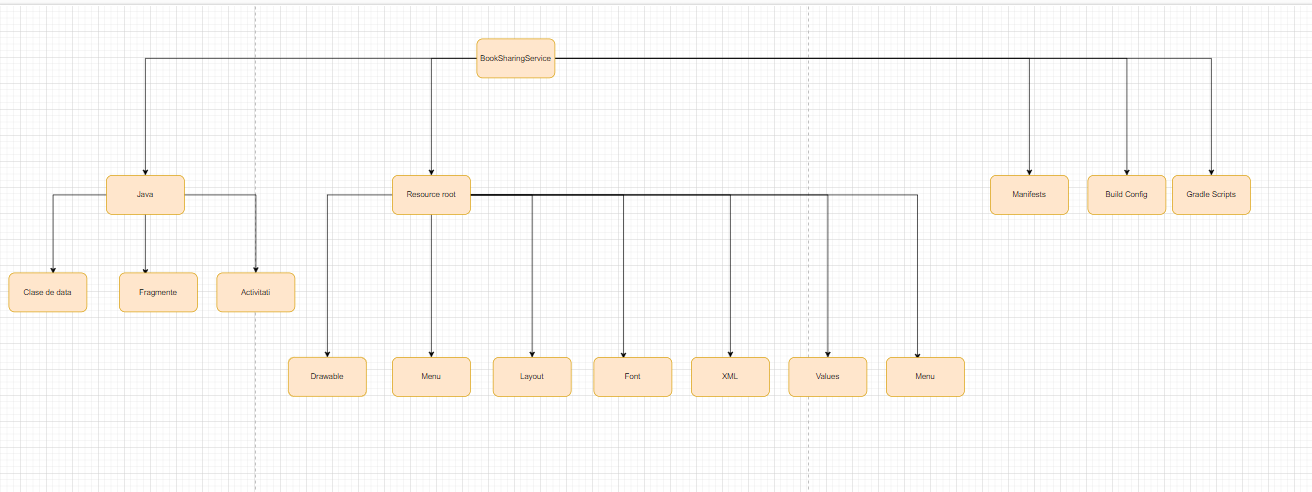
**Figura 14. Diagrama de secvență pentru Postarea unei cărți**

### Arhitectura de sistem

Proiectul este creat folosind limbajul Java si ide-ul Android Studio. Structura este urmatoarea:

* Manifests: aici se află fisier AndroidManifest.xml care reprezintă configurația necesară ca aplicația sa ruleze împreună cu permisiunile necesare aplicației
* Java: aici se află fișierele java care conțin codul pentru partea de back-end a aplicației. Aici putem găsi Fragmente, Activități si clasele de dată.
* Build Config: acest fișier este o clasă generată automat in urma compilării care conține informații legate de configurarea proiectului.
* Drawable: aici găsim toate iconițele/imaginiile folosite in interfața utilizatorului a aplicației
* Font: aici găsim informațiile despre fontul folosit.
* Layout: aici se află toate fisierele care țin de partea de front-end a aplicației. Reprezintă interfața cu utilizatorul.
* Menu: aici se află *layout*-ul de meniu care este folosit in toată aplicația.
* Values: acest folder este utilizat pentru a organiza și stoca diverse tipuri de resurse, cum ar fi șiruri de text, culori, dimensiuni, stiluri și atribute.
* XML: acest folder este folosit pentru a separa diferite aspecte ale aplicației în fișiere distincte, ceea ce face mai ușor gestionarea și organizarea proiectului.
* Gradle Scripts: "Gradle scripts" se referă la fișierele de configurare și scripturi care sunt utilizate pentru gestionarea dependențelor, compilarea și construirea proiectelor Android folosind sistemul de construire Gradle.

Gradle este o platformă open-source de automatizare a construirii și gestionării proiectelor software. În cadrul dezvoltării Android, Gradle este utilizat pentru a gestiona dependențele de librării, pentru a compila codul sursă și pentru a genera fișierele APK necesare pentru rularea aplicațiilor pe dispozitive Android

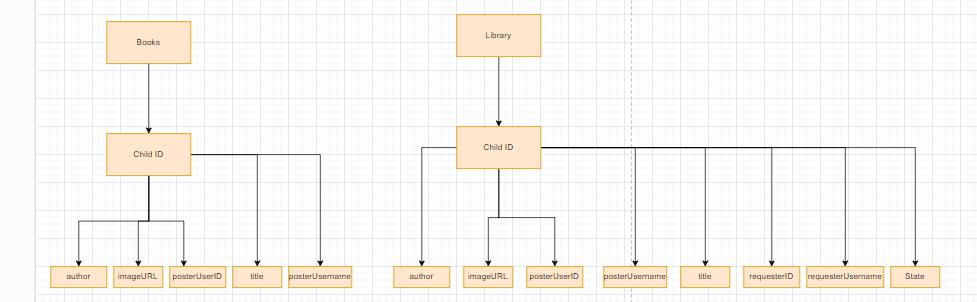
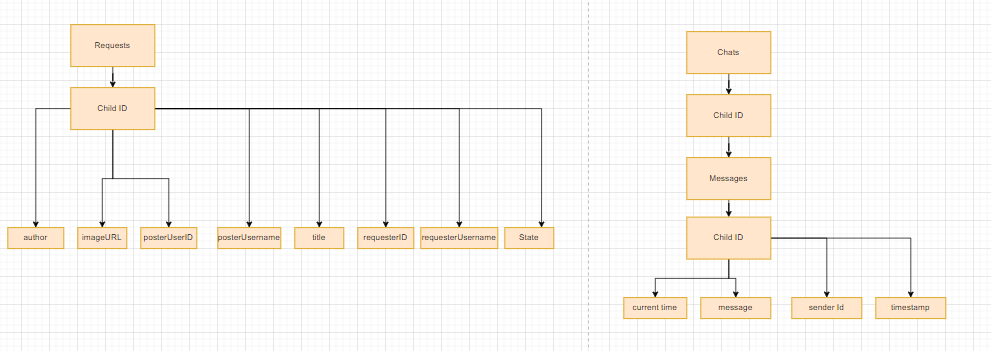


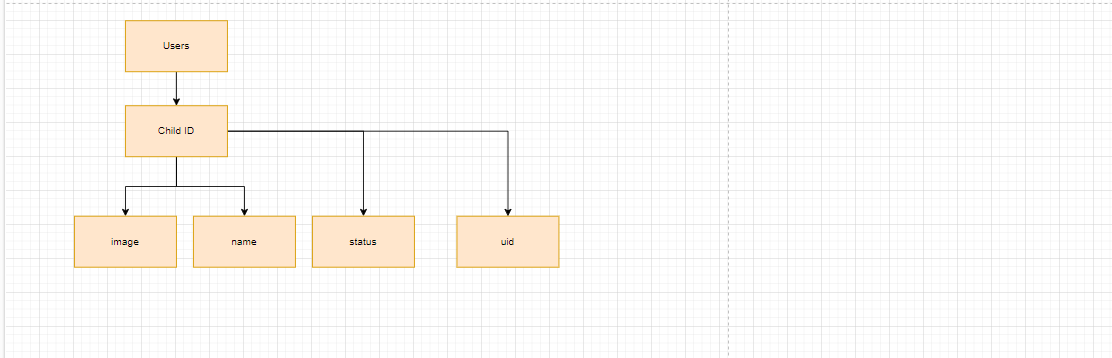
**Figura 15. Schema arhitecturii de sistem**

### Structura bazei de date

Deoarece folosesc baza de date Firebase care folosește un model noSQL ,diagramele UML asemânătoare celor clasice pentru bazele de date SQL nu sunt posibile.

În figuriile următoare o să prezint o schema in mare a bazei de date



**Figuriile 16,17,18. Schema bazei de date**

În această bază de date avem urmatoarele ”tabele”:

* Books: reprezintă tabelul în care sunt prezente toate cărțiile postate de către utilizatori. Ca atribute avem:

-*Author* ,*title,ImageURL* care reprezintă titlu,autorul si poza cărții.

-*PosterID* si *posterUsername* care reprezintă uid-ul si username-ul utilizatorului care a postat cartea

* Library: reprezintă tabelul în care sunt prezente cărțiile închiriate de către utilizatori. Toate cărțiile închiriate vor fii stocate in acest tabel si se vor diferenția de la utilizator la utilizator.Ca atribute avem:

-*Author* ,*title,ImageURL* care reprezintă titlu,autorul si poza cărții.

-*PosterID* si *posterUsername* care reprezintă uid-ul si username-ul utilizatorului care a postat cartea

-*RequesterId* si *requesterUsername* care reprezintăuid-ul si username-ul utilizatorului care a cerut cartea spre închiriere.

-*State* care reprezintă starea cărții , daca request-ul a fost sau nu acceptat.

* Requests: reprezinta tabelul în care sunt prezente toate requesturile pentru cărți. Ca atribute avem:

-*Author* ,*title,ImageURL* care reprezintă titlu,autorul si poza cărții.

-*PosterID* si *posterUsername* care reprezintă uid-ul si username-ul utilizatorului care a postat cartea

-*RequesterId* si *requesterUsername* care reprezintăuid-ul si username-ul utilizatorului care a cerut cartea spre închiriere.

-*State* care reprezintă starea cărții , daca request-ul a fost sau nu acceptat.

* Chats: reprezinta tabelul în care sunt stocate mesajele dintre utilizatori. Ca atribute avem:

-*CurrentTime* care reprezintă timpul curent.

-*Message* si *senderId* care reprezintă mesajul si id-ul utilizatorului care a transmis mesajul.

-*Timestamp* amprenta de timp preluată la momentul transmiterii mesajului.

* Users: reprezintă tabelul în care sunt stocați utilizatorii chat-ului aplicației. Ca atribute avem:

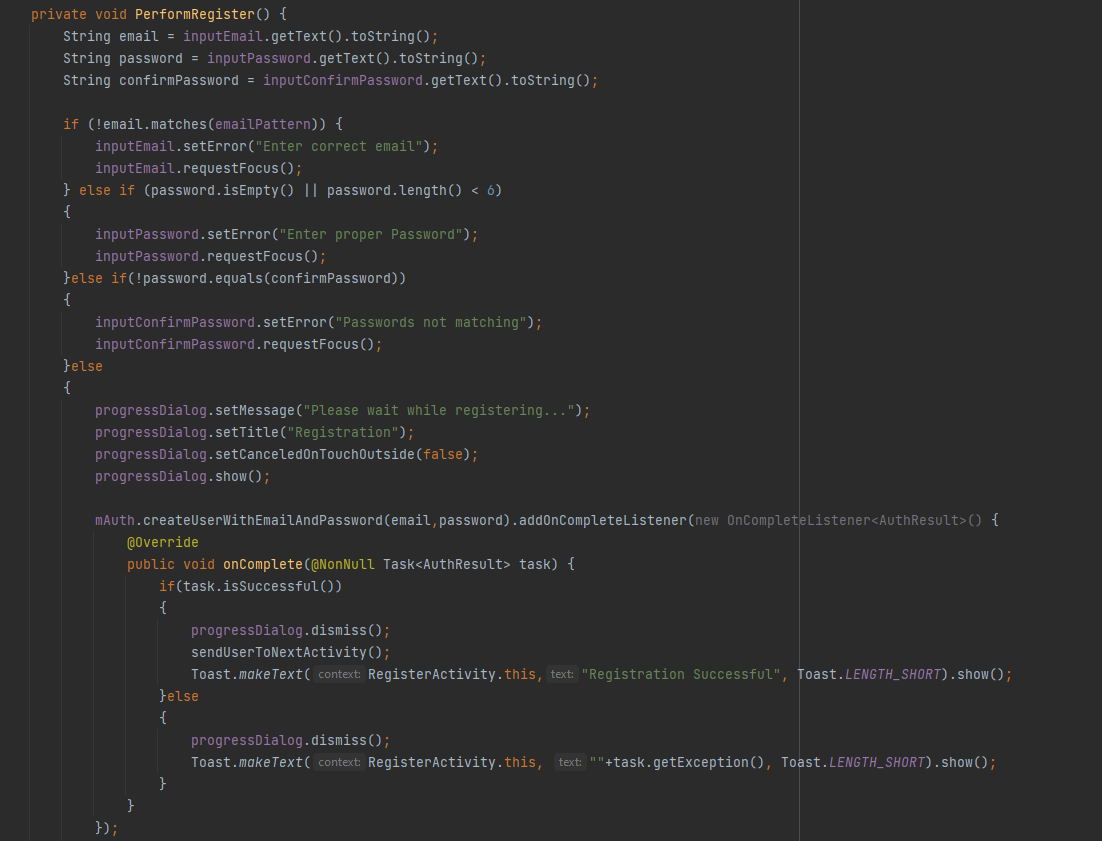
-*Image*,*name*,*status* si *uid* care reprezintă informațiile despre utilizator.

## Funcționalități

În acest capitol prezint funcționalitățiile principale ale aplicației.

### Înregistrare

Această funcționalitate îi permite utilizatorului sa își creze un cont pentru a putea folosi aplicația. În această funcționalitate sunt incluse si verificări ale datelor ca de exemplu valoarea câmpului ”Confirm Password” să fie egala cu valoarea câmpului ”Password” .



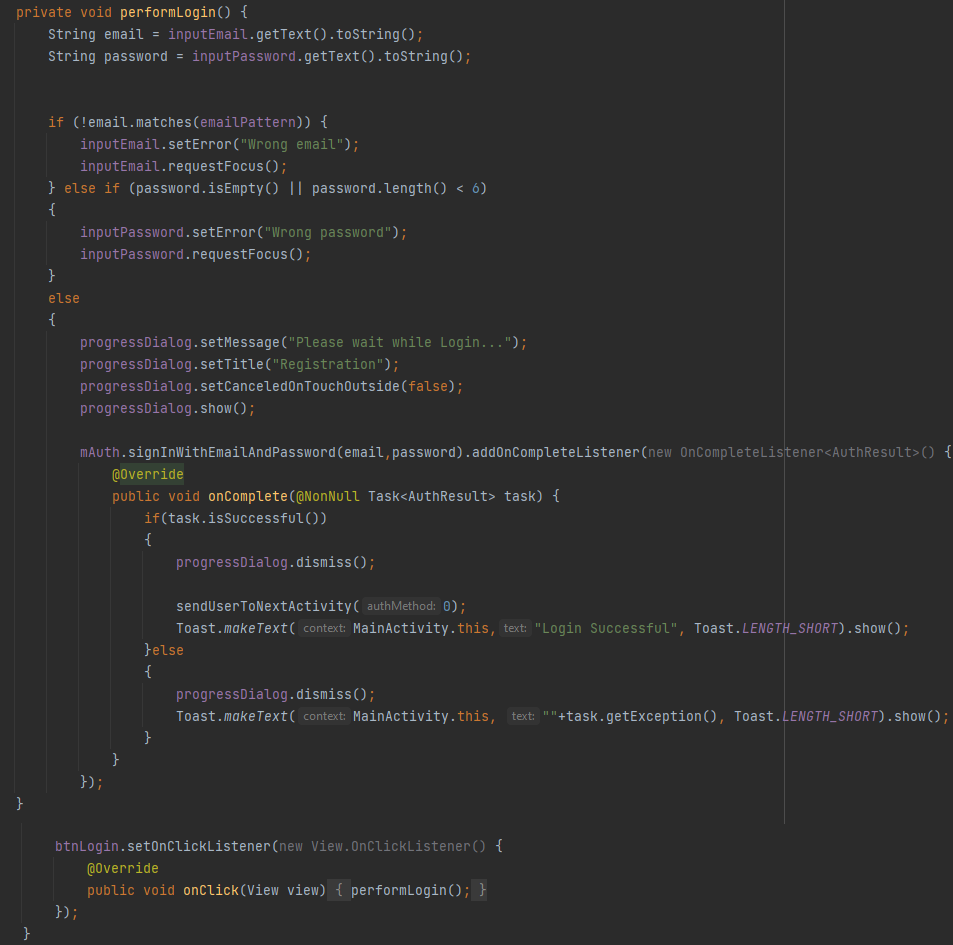
**Figura 19. Codul pentru funcționalitatea de înregistrare.**

### Logare

Această funcționalitate îi permite utilizatorului sa se logheze cu un cont pentru a avea acces in aplicație. Această funcționalitate se împarte in login-ul folosind contul creat si stocat in firebase și login-ul folosind contul de gmail.



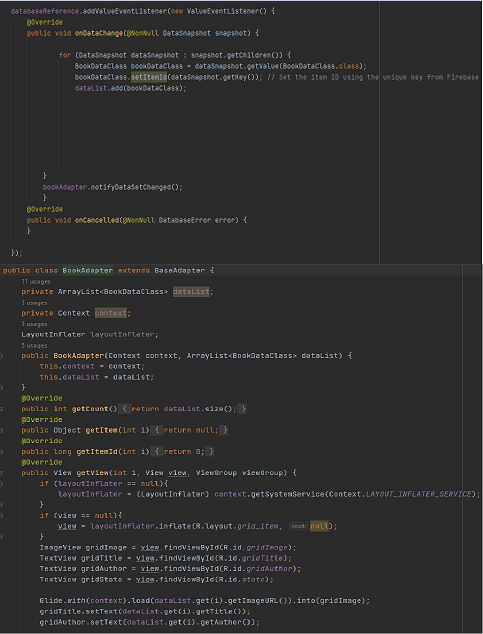
**Figura 20. Codul pentru login folosind google api**

****

**Figura 21. Codul pentru login folosind contul creat de utiliziator**

### Afișarea cărțiilor postate de utilizatori

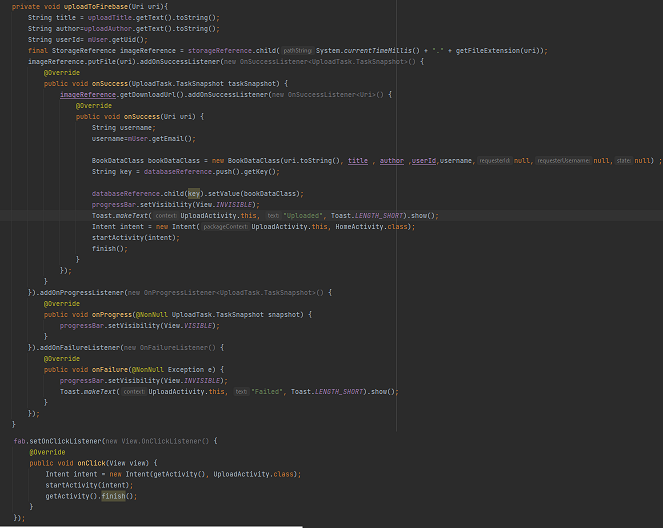
Această funcționalitate permite afișarea cărțiilor postate de către toți utilizatorii preluând un *snapshot* al tabelului ”Books” din baza de date.

****

**Figura 22. Codul pentru afișarea cărțiilor postate de utilizatori.**

### Postarea unei cărți

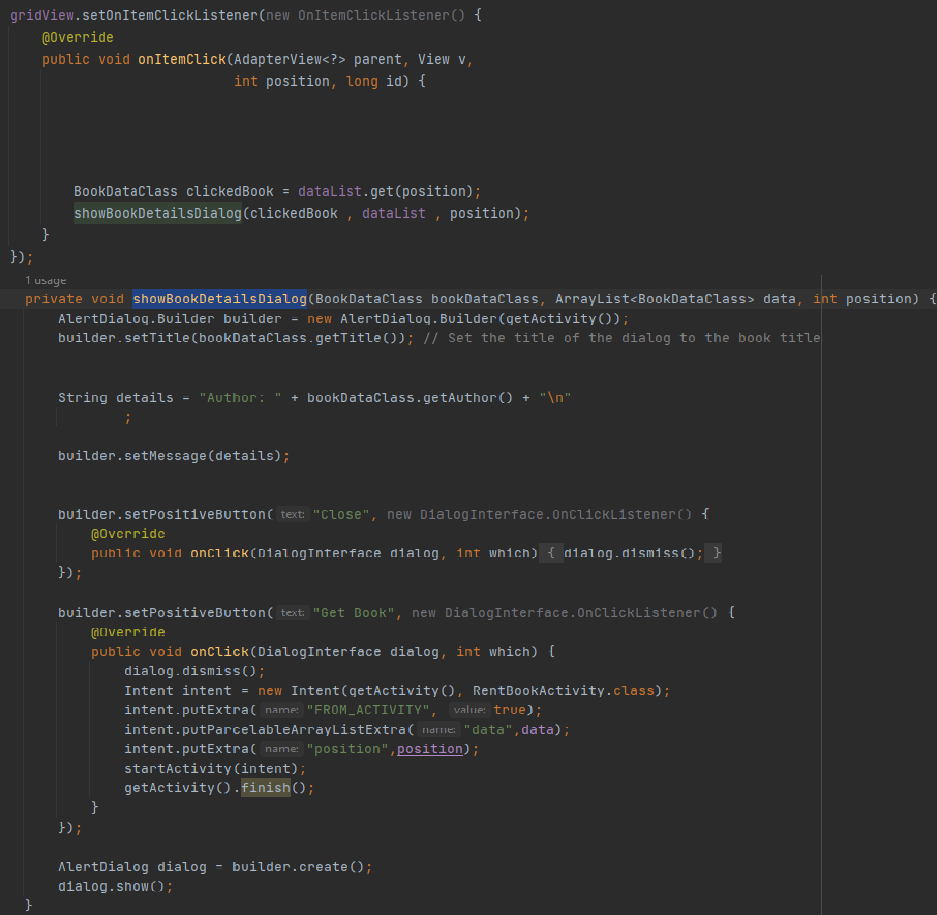
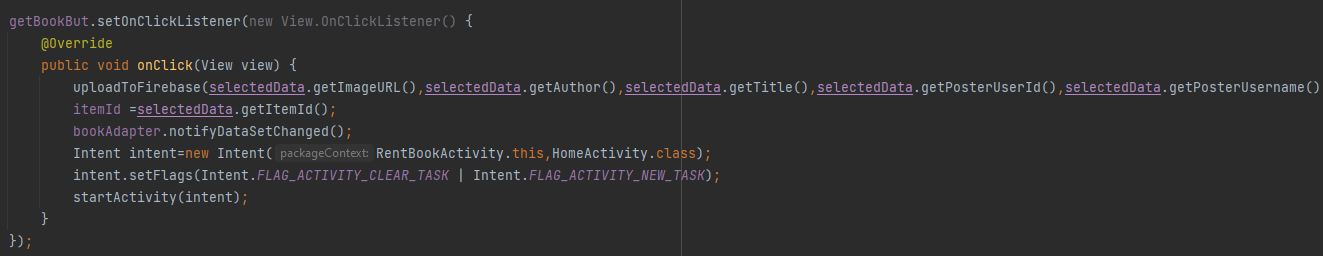
Aceasta funcționalitate îi permite utilizatorului sa posteze cartea dorită, acesta trebuie sa introduca detalii despre carte cum ar fi autor, titlu , imagine.



**Figura 23. Codul pentru postarea unei cărți.**

### Închirierea unei cărți

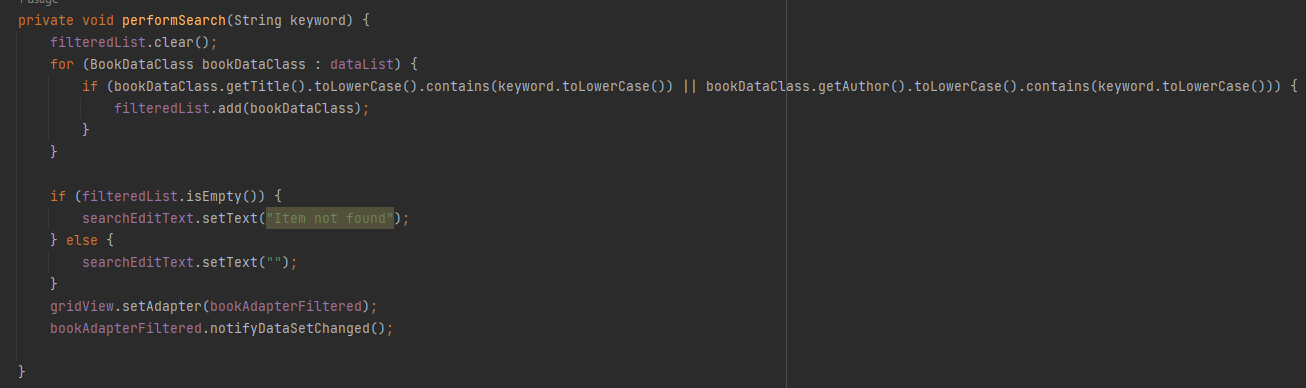
Cu aceasta funcționalitate un utilizator poate selecta o carte din lista cărțiilor valabile si o poate prelua trimitănd un request către utilizatorul care a postat cartea. In momentul in care un utilizatorul alege opțiunea ”get book” pe cartea aleasă un nou request este creat si trimis către utilizatorul care a postat cartea.



**Figuriile 24,25. Codul pentru închirierea unei cărți.**

### Căutarea unei cărți

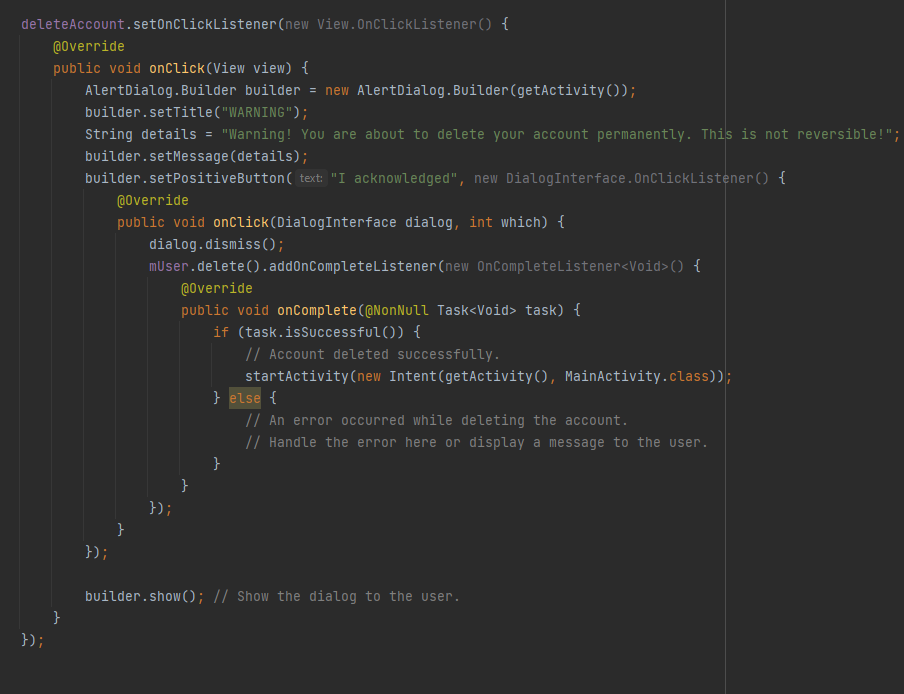
Aceasta funcționalitate îi permite utilizatorului sa caute o carte in lista de cărți postate si dacă este prezentă , este afisată ulterior in listă.



**Figura 26. Codul pentru căutarea unei cărți.**

### Ștergerea contului

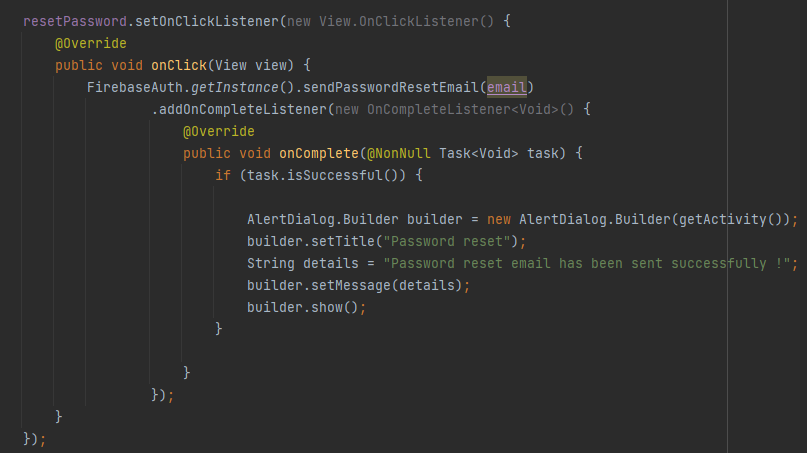
Aceasta funcționalitate îi permite utilizatorului sa își steargă contul permanent.



**Figura 27. Codul pentru ștergerea contului.**

### Resetarea parolei

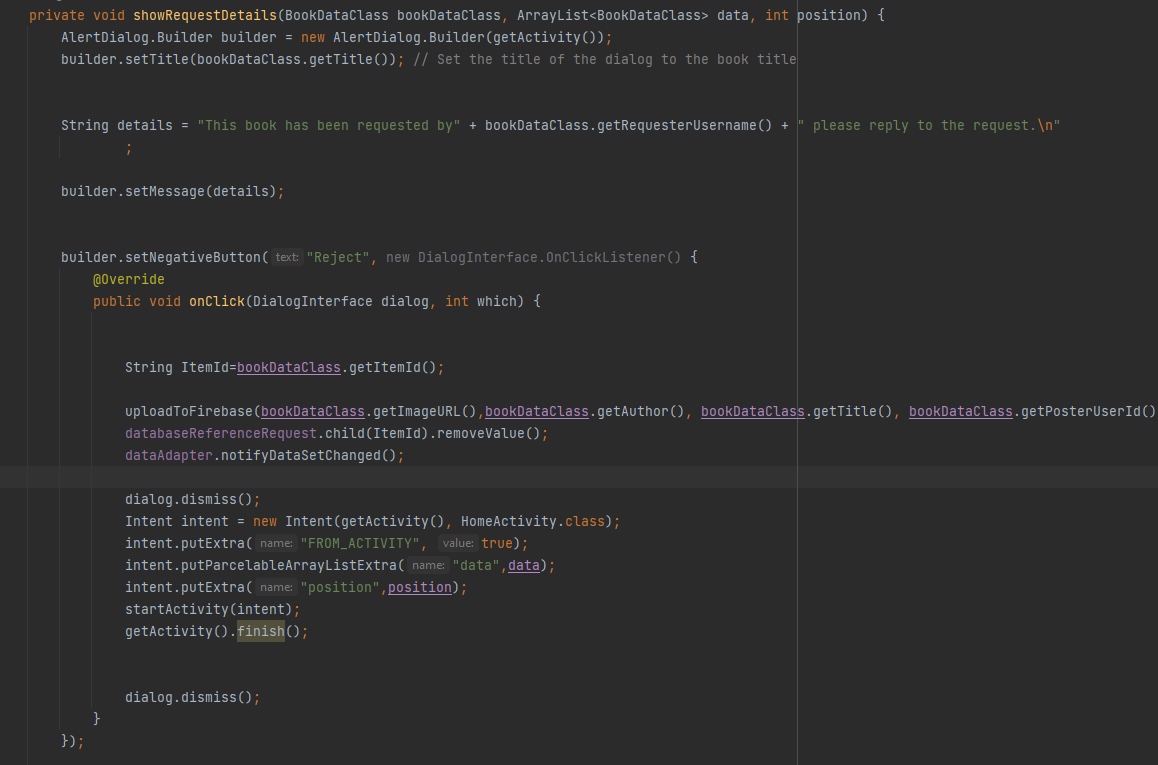
Aceasta funcționalitate îi permite utilizatorului sa își reseteze parola. Utilizatorul primește un mail pe mail-ul folosit la înregistrare.

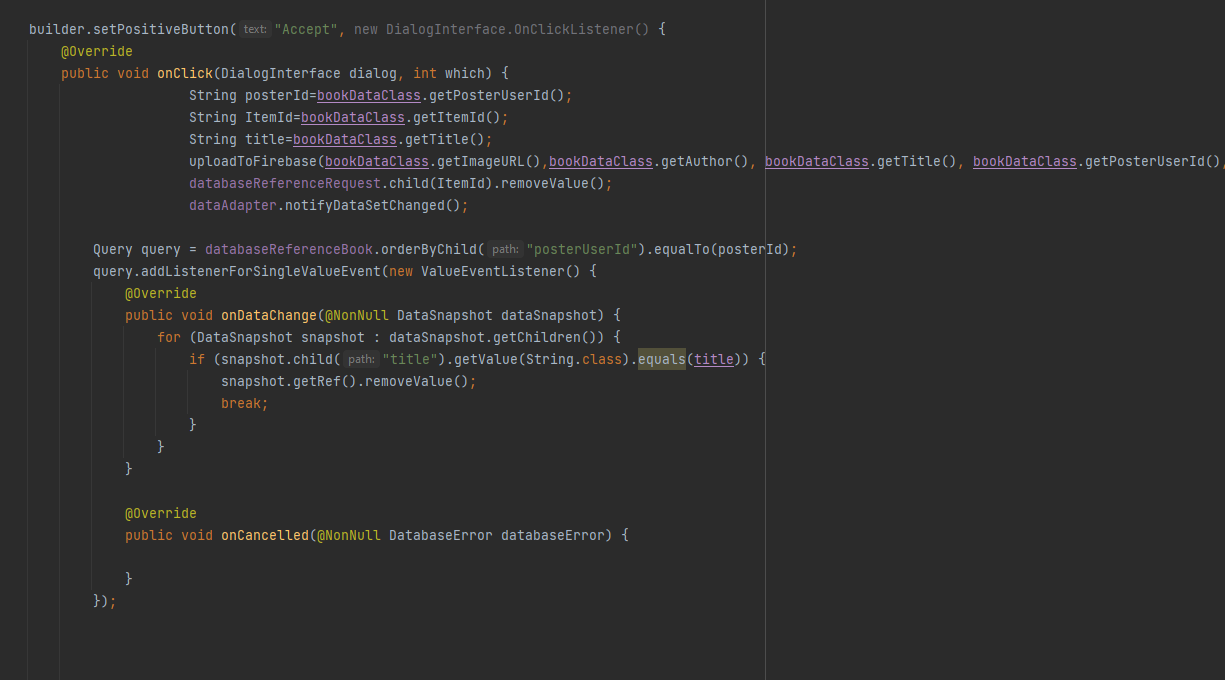
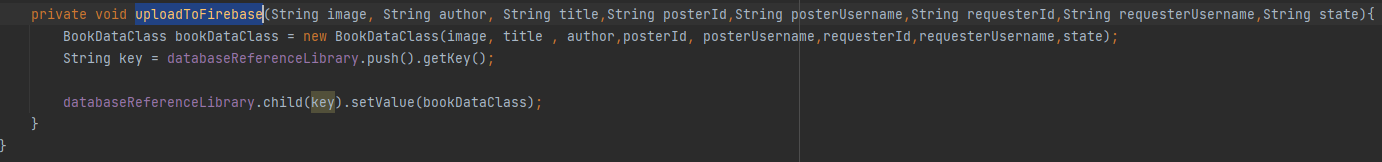
****

**Figura 28. Codul pentru resetarea parolei.**

### Gestionarea requesturilor

Aceasta funcționalitate îi permite utilizatorului sa gestioneze requesturile primite de la alți utilizatori. Acest lucru însemnănd daca accepta sau nu inchirierea cărții postate. Daca requestul este acceptat , cartea apare in librăria utilizatorului care a făcut requestul cu statusul ”Accepted”. Daca requestul este refuzat, cartea apare in librăria utilizatorului care a făcut requestul cu statusul ”Rejected”.



**** ****

**Figuriile 29,30,31. Codul pentru Gestionarea requesturilor.**

### Delogare

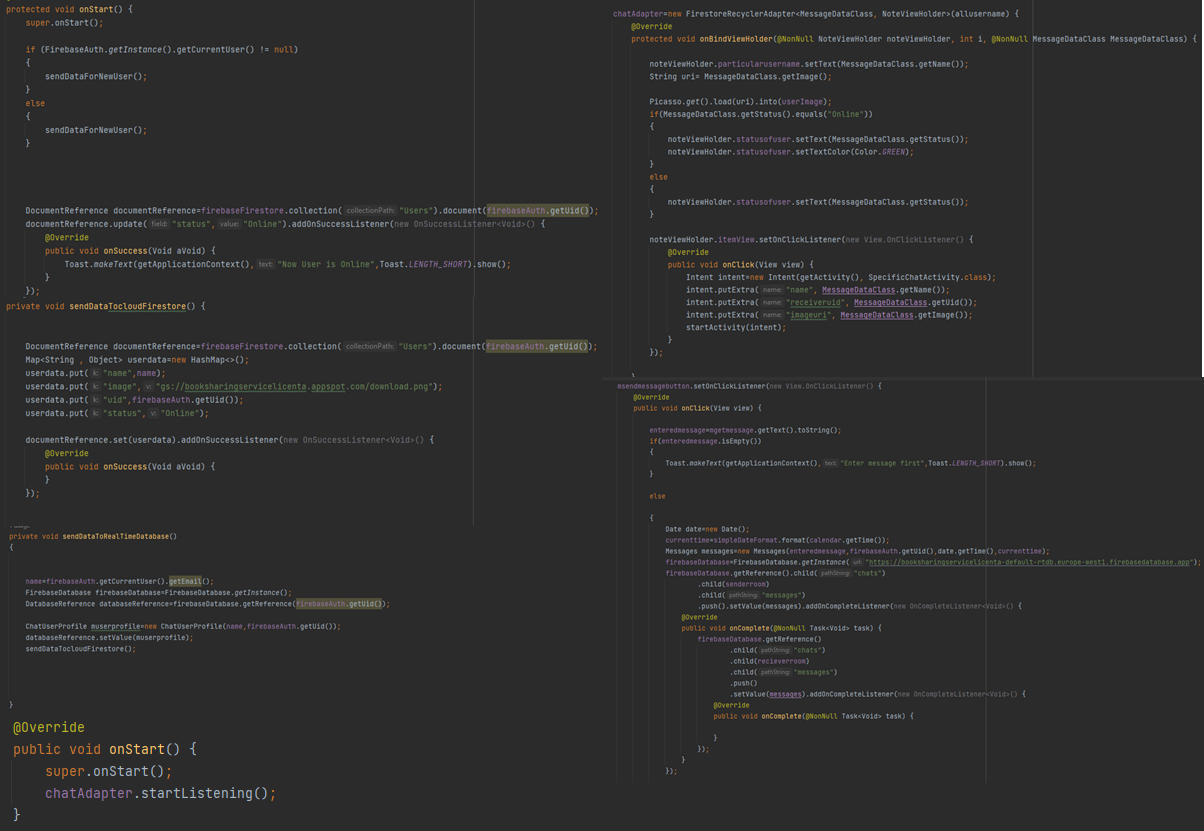
Aceasta opțiune îi permite utilizatorului sa se delogheze din aplicație in funcție de metoda de logare folosită și să fie redirecționat către pagina principală.

****

**Figura 32. Codul pentru Delogare.**

### Chat

Această funcționalitate permite utilizatoriilor sa comunice între ei.



**Figura 33. Codul pentru chat**

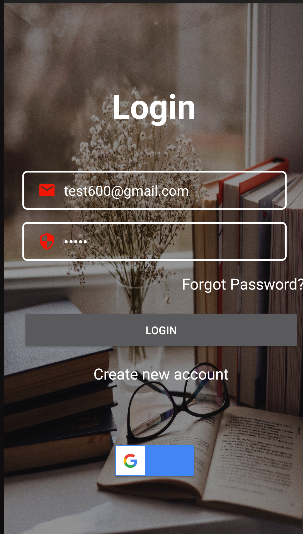
## Manualul utilizatorului.

În acest capitol prezint cele mai importante funcționalități si cum pot fi accesate de către utilizatori.

În momentul rulării aplicației utilizatorul este direcționat pe pagina de login. Aici utilizatorul poate completa cămpuriile de *email* si *password* , în cazul în care are un cont , si a continua prin a apăsa butonul Login

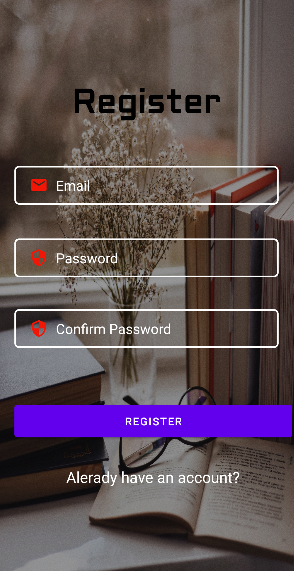
Dacă utilizatorul și-a uitat parola , poate apăsa pe butonul ”Forgot password” si un email se va trimite automat către adresa din campul Email. Dacă nu există niciun cont cu adresa de email specificată , o eroare este afișată.

Deasemenea, utilizatorul are opțiunea de apăsa pe butonul ”Google” pentru a se conecta cu contul de gmail.

****

**Figura 34. Pagina de logare**

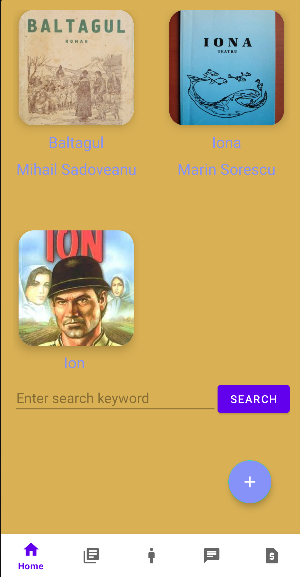
Dacă utilizatorul nu are niciun cont si vrea sa creeze unul , poate apăsa pe butonul ”Create New Account” urmând să fie redirecționat către pagina de înregistrare.



**Figura 35. Pagina de înregistrare**

Pentru înregistrare utilizatorul trebuie să completeze câmpurile ”Email”, ”Password”, ”Confirm Password”. Dacă adresa de email este validă si parola introdusă respectă ceriințele , utilizatorul poate apăsa pe butonul ”Register” și contul va fii creat și stocat in baza de date.Utilizatorul apoi este logat automat si redirecționat către pagina principală.

Dacă utilizatorul are deja un cont poate apăsa optiunea ”Alerady have an account?” si este redirecționat către pagina de login.



**Figura 36. Pagina de *Home***

Pe pagina principală avem lista cu toate cărțiile postate, bara de căutare, butonul de adăugare carte, butonul de căutare si meniul de navigație.

Dacă un utilizator dorește o carte poate apăsa pe cartea respectivă si i se va afișa un mesaj cu detalii despre carte si un buton ”Get Book” care îl va redirecționa către pagina de închiriere.

Dacă un utilizator dorește să posteze o carte poate apăsa pe butonul de ”Add” si va fi redirecționat către pagina de adăugare a unei cărți.

Utilizatorul poate scrie in câmpul ”Enter search keyword” un cuvănt cheie pentru a îl căuta in baza de date. De exemplu daca vrem sa căutam cartea ”Baltagul” scriem in cămp cuvăntul ”Baltagul” si apăsăm pe butonul ”Search” iar în listă va apărea cartea respectivă dacă există.

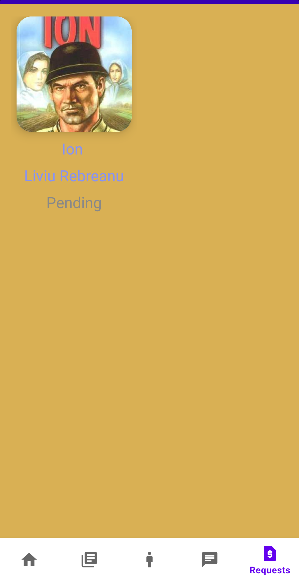
Utilizatorul poate naviga între paginiile aplicației folosind bara de navigație.

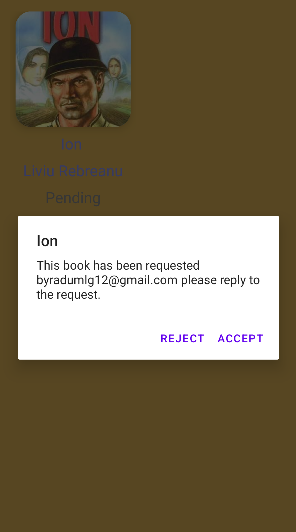


**Figura 36. Pagina de închiriere**

Dacă utilizatorul a selectat o carte cu scopul de a o închiria este redirecționat pe pagina cărții unde sunt prezente detalii despre carte si utilizatorul care a adăugat cartea.

Dacă utilizatorul selectează opțiunea ”Get Book” este redirecționat înapoi către pagina principală si request-ul este trimis către utilizatorul care a postat cartea.

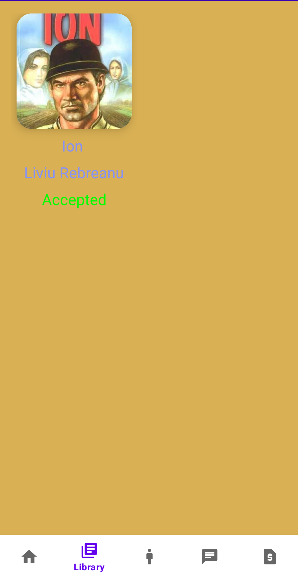




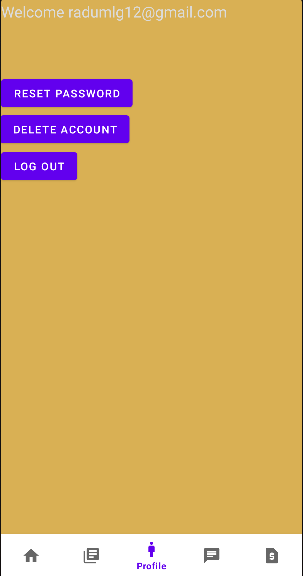
**Figuriile 37,38. Pagina de requesturi**

Pe această pagină sunt prezente requesturile făcute către utilizatorul conectat. Utilizatorul are opțiunea să apese pe un request și să accepte sau să refuze requestul respectiv.

Dacă utilizatorul accepta requestul , cartea este ștearsă din lista de cărți postate , si raspunsul este trimis către utilizatorul care a făcut requestul in pagina ”Library”.



**Figura 39. Pagina librăriei**

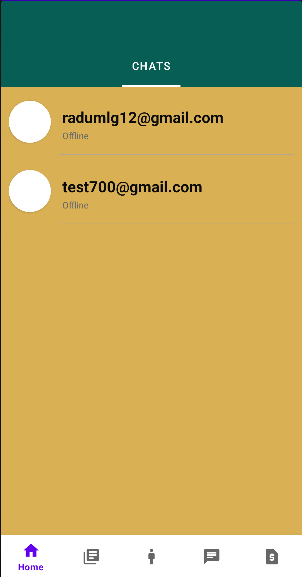


**Figura 40. Pagina profilului**

Utilizatorul are opțiunea sa își reseteze parola apăsând butonul ”Reset Password” , un email trimițându-se către adresa de email folosită la logare.

Dacă utilizatorul vrea să se deconecteze , poate apăsa butonul ”Log Out”.

Dacă utilizatorul vrea să își șteargă contul , poate apăsa pe butonul ”Delete Account”. Acesta va fi atenționat că această acțiune este ireversibilă



**Figura 41. Pagina de chat**

Utilizatorul are opțiunea de a comunica cu alți utilizatori folosind *chat*-ul. Dacă un utilizator apasă pe iconița unui *chat*, este redirecționat către conversația cu utilizatorul respectiv.



**Figura 42. Pagina de conversație cu un utilizator**

Aici utilizatorii pot conversa folosind bara ”Type a message” si apăsănd butonul de ”Send”.

# Concluzii

În concluzie, această aplicație a reprezentat o provocare foarte mare pentru mine , nefiind familiar cu tehnologiile folosite. În urma realizării proiectului am învățat foarte multe si găndirea mea logică a fost cu mult îmbunătățită. Deasemenea , deoarece tehnologiile folosite sunt de actualitate , sunt sigur ca mă vor ajuta în cariera de programator.

## Dezvoltarea aplicației in viitor

Vreau sa continui în a dezvolta aplicația pe viitor si unele din urmatoarele schimbări sau funcționalități noi sunt următoarele:

* O interfață grafică cu utilizatorul îmbunătățită si mai placută
* Posibilitatea de a oferi rating utilizatorilor.
* Posibilitatea de a primi recomandări bazate pe librăria existentă a utilizatorului.
* Organizarea cărțiilor pe categorii.
* Introducerea de ”*Wish* *List*” în care utilizatorul primește notificare atunci cănd o carte pe care o dorește este postată.

# Bibliografie

* [Wik01] <https://ro.wikipedia.org/wiki/Android_Studio>
* [Wik02] <https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)>
* [Fot01] <https://fotc.com/ro/blog/firebase/#:~:text=Firebase%20este%20un%20set%20cuprinz%C4%83tor,necesit%C4%83%20investi%C8%9Bii%20costisitoare%20%C3%AEn%20infrastructur%C4%83.>
* [Wik03] <https://ro.wikipedia.org/wiki/XML>

# Referințe web

* [Wik01] <https://ro.wikipedia.org/wiki/Android_Studio>
* [Wik02] <https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)>
* [Fot01] <https://fotc.com/ro/blog/firebase/#:~:text=Firebase%20este%20un%20set%20cuprinz%C4%83tor,necesit%C4%83%20investi%C8%9Bii%20costisitoare%20%C3%AEn%20infrastructur%C4%83.>
* [Wik03] <https://ro.wikipedia.org/wiki/XML>

# Codul sursă

În acest capitol atașez codul sursă pentru cele mai importante funcționalități

**Înregistrare**

private void PerformRegister() {  
 String email = inputEmail.getText().toString();  
 String password = inputPassword.getText().toString();  
 String confirmPassword = inputConfirmPassword.getText().toString();  
  
 if (!email.matches(emailPattern)) {  
 inputEmail.setError("Enter correct email");  
 inputEmail.requestFocus();  
 } else if (password.isEmpty() || password.length() < 6)  
 {  
 inputPassword.setError("Enter proper Password");  
 inputPassword.requestFocus();  
 }else if(!password.equals(confirmPassword))  
 {  
 inputConfirmPassword.setError("Passwords not matching");  
 inputConfirmPassword.requestFocus();  
 }else  
 {  
 progressDialog.setMessage("Please wait while registering...");  
 progressDialog.setTitle("Registration");  
 progressDialog.setCanceledOnTouchOutside(false);  
 progressDialog.show();  
  
 mAuth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if(task.isSuccessful())  
 {  
 progressDialog.dismiss();  
 sendUserToNextActivity();  
 Toast.*makeText*(RegisterActivity.this,"Registration Successful", Toast.*LENGTH\_SHORT*).show();  
 }else  
 {  
 progressDialog.dismiss();  
 Toast.*makeText*(RegisterActivity.this, ""+task.getException(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
  
 }  
  
  
  
}

private void sendUserToNextActivity() {  
 Intent intent=new Intent(RegisterActivity.this,HomeActivity.class);  
 intent.setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK* | Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 startActivity(intent);  
}

**Logare**

private void performLogin() {  
 String email = inputEmail.getText().toString();  
 String password = inputPassword.getText().toString();  
  
  
 if (!email.matches(emailPattern)) {  
 inputEmail.setError("Wrong email");  
 inputEmail.requestFocus();  
 } else if (password.isEmpty() || password.length() < 6)  
 {  
 inputPassword.setError("Wrong password");  
 inputPassword.requestFocus();  
 }  
 else  
 {  
 progressDialog.setMessage("Please wait while Login...");  
 progressDialog.setTitle("Registration");  
 progressDialog.setCanceledOnTouchOutside(false);  
 progressDialog.show();  
  
 mAuth.signInWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if(task.isSuccessful())  
 {  
 progressDialog.dismiss();  
  
 sendUserToNextActivity(0);  
 Toast.*makeText*(MainActivity.this,"Login Successful", Toast.*LENGTH\_SHORT*).show();  
 }else  
 {  
 progressDialog.dismiss();  
 Toast.*makeText*(MainActivity.this, ""+task.getException(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 }  
  
}  
  
 private void sendUserToNextActivity(int authMethod) {  
 Intent intent=new Intent(MainActivity.this,HomeActivity.class);  
 intent.putExtra("authMethod",authMethod);  
 intent.setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK* | Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 startActivity(intent);  
 }  
 }

googleBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 signIn();  
 }  
});

void signIn(){  
 Intent signInIntent = gsc.getSignInIntent();  
 startActivityForResult(signInIntent,1000);  
}

protected void onActivityResult(int requestCode, int resultCode,Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
 if(requestCode == 1000){  
 Task<GoogleSignInAccount> task = GoogleSignIn.*getSignedInAccountFromIntent*(data);  
  
 try {  
 task.getResult(ApiException.class);  
   
 sendUserToNextActivity(1);  
 } catch (ApiException e) {  
 Toast.*makeText*(getApplicationContext(), "Something went wrong", Toast.*LENGTH\_SHORT*).show();  
 }  
 }

**Căutarea unei cărți**

searchButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 String keyword = searchEditText.getText().toString().trim();  
 if (!TextUtils.*isEmpty*(keyword)) {  
 performSearch(keyword);  
 }  
 }  
});

private void performSearch(String keyword) {  
 filteredList.clear();  
 for (BookDataClass bookDataClass : dataList) {  
 if (bookDataClass.getTitle().toLowerCase().contains(keyword.toLowerCase()) || bookDataClass.getAuthor().toLowerCase().contains(keyword.toLowerCase())) {  
 filteredList.add(bookDataClass);  
 }  
 }  
  
 if (filteredList.isEmpty()) {  
 searchEditText.setText("Item not found");  
 } else {  
 searchEditText.setText("");  
 }  
 gridView.setAdapter(bookAdapterFiltered);  
 bookAdapterFiltered.notifyDataSetChanged();  
  
}

**Adăugarea unei cărți**

uploadImage.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent photoPicker = new Intent();  
 photoPicker.setAction(Intent.*ACTION\_GET\_CONTENT*);  
 photoPicker.setType("image/\*");  
 activityResultLauncher.launch(photoPicker);  
 }  
 });  
 uploadButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 if (imageUri != null){  
 uploadToFirebase(imageUri);  
 } else {  
 Toast.*makeText*(UploadActivity.this, "Please select image", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 }  
 //Outside onCreate  
 private void uploadToFirebase(Uri uri){  
 String title = uploadTitle.getText().toString();  
 String author=uploadAuthor.getText().toString();  
 String userId= mUser.getUid();  
 final StorageReference imageReference = storageReference.child(System.*currentTimeMillis*() + "." + getFileExtension(uri));  
 imageReference.putFile(uri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {  
 @Override  
 public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {  
 imageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {  
 @Override  
 public void onSuccess(Uri uri) {  
 String username;  
 username=mUser.getEmail();  
  
 BookDataClass bookDataClass = new BookDataClass(uri.toString(), title , author ,userId,username,null,null,null) ;  
 String key = databaseReference.push().getKey();  
  
 databaseReference.child(key).setValue(bookDataClass);  
 progressBar.setVisibility(View.*INVISIBLE*);  
 Toast.*makeText*(UploadActivity.this, "Uploaded", Toast.*LENGTH\_SHORT*).show();  
 Intent intent = new Intent(UploadActivity.this, HomeActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 });  
 }  
 }).addOnProgressListener(new OnProgressListener<UploadTask.TaskSnapshot>() {  
 @Override  
 public void onProgress(@NonNull UploadTask.TaskSnapshot snapshot) {  
 progressBar.setVisibility(View.*VISIBLE*);  
 }  
 }).addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 progressBar.setVisibility(View.*INVISIBLE*);  
 Toast.*makeText*(UploadActivity.this, "Failed", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
 private String getFileExtension(Uri fileUri){  
 ContentResolver contentResolver = getContentResolver();  
 MimeTypeMap mime = MimeTypeMap.*getSingleton*();  
 return mime.getExtensionFromMimeType(contentResolver.getType(fileUri));  
 }  
}

**Vizualizarea si gestionarea requesturilor**

databaseReferenceRequest.addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
  
 for (DataSnapshot dataSnapshot : snapshot.getChildren()) {  
 BookDataClass bookDataClass = dataSnapshot.getValue(BookDataClass.class);  
  
 bookDataClass.setItemId(dataSnapshot.getKey()); // Set the item ID using the unique key from Firebase  
 String posterId=bookDataClass.getPosterUserId();  
 String state=bookDataClass.getState();  
 if(posterId.equals(mUser.getUid()) && state.equals("Pending")) {  
 dataList.add(bookDataClass);  
 }  
  
 }  
 dataAdapter.notifyDataSetChanged();  
  
 }  
  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
 }  
  
 });  
  
  
 gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView<?> parent, View v,  
 int position, long id) {  
  
  
  
  
 BookDataClass clickedBook = dataList.get(position);  
 showRequestDetails(clickedBook , dataList , position);  
 }  
 });  
  
  
  
  
  
  
 return v;  
 }  
  
 private void showRequestDetails(BookDataClass bookDataClass, ArrayList<BookDataClass> data, int position) {  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 builder.setTitle(bookDataClass.getTitle()); // Set the title of the dialog to the book title  
  
  
 String details = "This book has been requested by" + bookDataClass.getRequesterUsername() + " please reply to the request.\n"  
 ;  
  
 builder.setMessage(details);  
  
  
 builder.setNegativeButton("Reject", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
  
  
 String ItemId=bookDataClass.getItemId();  
  
 uploadToFirebase(bookDataClass.getImageURL(),bookDataClass.getAuthor(), bookDataClass.getTitle(), bookDataClass.getPosterUserId(),bookDataClass.getPosterUsername(),bookDataClass.getRequesterId(), bookDataClass.getRequesterUsername(), "Rejected");  
 databaseReferenceRequest.child(ItemId).removeValue();  
 dataAdapter.notifyDataSetChanged();  
  
 dialog.dismiss();  
 Intent intent = new Intent(getActivity(), HomeActivity.class);  
 intent.putExtra("FROM\_ACTIVITY", true);  
 intent.putParcelableArrayListExtra("data",data);  
 intent.putExtra("position",position);  
 startActivity(intent);  
 getActivity().finish();  
  
  
 dialog.dismiss();  
 }  
 });  
  
 builder.setPositiveButton("Accept", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 String posterId=bookDataClass.getPosterUserId();  
 String ItemId=bookDataClass.getItemId();  
 String title=bookDataClass.getTitle();  
 uploadToFirebase(bookDataClass.getImageURL(),bookDataClass.getAuthor(), bookDataClass.getTitle(), bookDataClass.getPosterUserId(),bookDataClass.getPosterUsername(),bookDataClass.getRequesterId(), bookDataClass.getRequesterUsername(), "Accepted");  
 databaseReferenceRequest.child(ItemId).removeValue();  
 dataAdapter.notifyDataSetChanged();  
  
 Query query = databaseReferenceBook.orderByChild("posterUserId").equalTo(posterId);  
 query.addListenerForSingleValueEvent(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 for (DataSnapshot snapshot : dataSnapshot.getChildren()) {  
 if (snapshot.child("title").getValue(String.class).equals(title)) {  
 snapshot.getRef().removeValue();  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
  
  
  
 dialog.dismiss();  
 Intent intent = new Intent(getActivity(), HomeActivity.class);  
 intent.putExtra("FROM\_ACTIVITY", true);  
 intent.putParcelableArrayListExtra("data",data);  
 intent.putExtra("position",position);  
 startActivity(intent);  
 getActivity().finish();  
 }  
 });  
  
 AlertDialog dialog = builder.create();  
 dialog.show();  
 }  
  
 private void uploadToFirebase(String image, String author, String title,String posterId,String posterUsername,String requesterId,String requesterUsername,String state){  
 BookDataClass bookDataClass = new BookDataClass(image, title , author,posterId, posterUsername,requesterId,requesterUsername,state);  
 String key = databaseReferenceLibrary.push().getKey();  
  
 databaseReferenceLibrary.child(key).setValue(bookDataClass);  
 }  
}

**Vizualizarea librariei**

databaseReferenceLibrary.addValueEventListener(new ValueEventListener() {  
  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
 String userId= mUser.getUid();  
 String userUID;  
 String userRUID;  
 String itemId;  
 dataList.clear();  
 for (DataSnapshot dataSnapshot : snapshot.getChildren()) {  
 BookDataClass bookDataClass = dataSnapshot.getValue(BookDataClass.class);  
 bookDataClass.setItemId(dataSnapshot.getKey()); // Set the item ID using the unique key from Firebase  
 itemId=bookDataClass.getItemId();  
 dataList.add(bookDataClass);  
  
  
 for(int i=0; i<dataList.size();i++)  
 {  
 userUID=dataList.get(i).getRequesterId();  
 userRUID=dataList.get(i).getRequesterId();  
 if(userId.equals(userUID))  
 {  
  
  
 }  
 else  
 {  
 dataList.remove(i);  
 }  
 }  
  
  
  
  
  
 }  
 bookAdapter.notifyDataSetChanged();  
  
  
 }

**Ștergerea contului**

deleteAccount.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 builder.setTitle("WARNING");  
 String details = "Warning! You are about to delete your account permanently. This is not reversible!";  
 builder.setMessage(details);  
 builder.setPositiveButton("I acknowledged", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 dialog.dismiss();  
 mUser.delete().addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 if (task.isSuccessful()) {  
 // Account deleted successfully.  
 startActivity(new Intent(getActivity(), MainActivity.class));  
 } else {  
 // An error occurred while deleting the account.  
 // Handle the error here or display a message to the user.  
 }  
 }  
 });  
 }  
 });  
  
 builder.show(); // Show the dialog to the user.  
 }  
 });  
  
}

**Delogare**

if(authMethod == 1) {  
 String email = gsa.getEmail();  
 userMail.setText("Welcome " + email);  
 signOutBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
  
 signOut();  
 }  
  
 });  
}else if(authMethod== 0){  
  
  
  
 String email = mUser.getEmail();  
 userMail.setText("Welcome " + email);  
 signOutBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 mAuth.signOut();  
 startActivity(new Intent(getActivity(),MainActivity.class));  
 }  
  
  
 });

void signOut(){  
 gsc.signOut().addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 startActivity(new Intent(getActivity(),MainActivity.class));  
 }  
 });  
}

**Schimbarea parolei**

resetPassword.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 FirebaseAuth.*getInstance*().sendPasswordResetEmail(email)  
 .addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 if (task.isSuccessful()) {  
  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 builder.setTitle("Password reset");  
 String details = "Password reset email has been sent successfully !";  
 builder.setMessage(details);  
 builder.show();  
 }  
  
 }  
 });  
 }  
});

**Folosirea chatului**

intent=getIntent();  
  
 setSupportActionBar(mtoolbarofspecificchat);  
 mtoolbarofspecificchat.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Toast.*makeText*(getApplicationContext(),"Toolbar is Clicked",Toast.*LENGTH\_SHORT*).show();  
  
  
 }  
 });  
  
 firebaseAuth=FirebaseAuth.*getInstance*();  
 firebaseDatabase=FirebaseDatabase.*getInstance*("https://booksharingservicelicenta-default-rtdb.europe-west1.firebasedatabase.app");  
 calendar=Calendar.*getInstance*();  
 simpleDateFormat=new SimpleDateFormat("hh:mm a");  
  
  
 msenderuid=firebaseAuth.getUid();  
 mrecieveruid=getIntent().getStringExtra("receiveruid");  
 mrecievername=getIntent().getStringExtra("name");  
  
  
  
 senderroom=msenderuid+mrecieveruid;  
 recieverroom=mrecieveruid+msenderuid;  
  
  
  
 DatabaseReference databaseReference=firebaseDatabase.getReference().child("chats").child(senderroom).child("messages");  
 messagesAdapter=new MessagesAdapter(SpecificChatActivity.this,messagesArrayList);  
 databaseReference.addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
 messagesArrayList.clear();  
 for(DataSnapshot snapshot1:snapshot.getChildren())  
 {  
 Messages messages=snapshot1.getValue(Messages.class);  
 messagesArrayList.add(messages);  
 }  
 messagesAdapter.notifyDataSetChanged();  
 }  
  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
  
 }  
 });  
  
  
  
  
 mbackbuttonofspecificchat.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent=new Intent(SpecificChatActivity.this, ChatActivity.class);  
 intent.setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK* | Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 startActivity(intent);  
 }  
 });  
  
  
 mnameofspecificuser.setText(mrecievername);  
 String uri=intent.getStringExtra("imageuri");  
 if(uri.isEmpty())  
 {  
 Toast.*makeText*(getApplicationContext(),"null is recieved",Toast.*LENGTH\_SHORT*).show();  
 }  
 else  
 {  
 Picasso.*get*().load(uri).into(mimageviewofspecificuser);  
 }  
  
  
 msendmessagebutton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
  
 enteredmessage=mgetmessage.getText().toString();  
 if(enteredmessage.isEmpty())  
 {  
 Toast.*makeText*(getApplicationContext(),"Enter message first",Toast.*LENGTH\_SHORT*).show();  
 }  
  
 else  
  
 {  
 Date date=new Date();  
 currenttime=simpleDateFormat.format(calendar.getTime());  
 Messages messages=new Messages(enteredmessage,firebaseAuth.getUid(),date.getTime(),currenttime);  
 firebaseDatabase=FirebaseDatabase.*getInstance*("https://booksharingservicelicenta-default-rtdb.europe-west1.firebasedatabase.app");  
 firebaseDatabase.getReference().child("chats")  
 .child(senderroom)  
 .child("messages")  
 .push().setValue(messages).addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 firebaseDatabase.getReference()  
 .child("chats")  
 .child(recieverroom)  
 .child("messages")  
 .push()  
 .setValue(messages).addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
  
 }  
 });  
 }  
 });  
  
 mgetmessage.setText(null);  
  
  
  
  
 }  
  
  
  
  
 }  
 });  
  
  
  
  
}  
  
  
@Override  
public void onStart() {  
 super.onStart();  
 messagesAdapter.notifyDataSetChanged();  
}  
  
@Override  
public void onStop() {  
 super.onStop();  
 if(messagesAdapter!=null)  
 {  
 messagesAdapter.notifyDataSetChanged();  
 }  
}

# website-ul proiectului

Repositoriu GitHub pentru proiect este:

<https://github.com/RomanescuRaduAndrei/BookSharingServiceLICENTA>

# CD / DVD

Autorul atașează în această anexă obligatorie, versiunea electronică a aplicației, a acestei lucrări, precum și prezentarea finală a tezei.



# Index

B

Bibliografie 9

C

CUPRINSUL xi

D

Dimensiuni 3

F

Figuri 4

Formulele matematice 4

I

Ilustrațiile 4

L

Legenda 6

LISTA FIGURILOR xii

LISTA TABELELOR xiii

R

Referințe web 10

S

Structura documentului 2

T

Tabele 5