ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI
Facultatea de Cibernetică, Statistică și Informatică Economică
Specializarea Informatică Economică

Lucrare de licență

Coordonator științific:
prof. univ. dr. Bâra Adela

Absolvent:
Mireag Mădălina-Andreea

București

2021

ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI
Facultatea de Cibernetică, Statistică și Informatică Economică
Specializarea Informatică Economică

Lucrare de licență

Aplicație pentru gestiunea unei biblioteci

Coordonator științific:
prof. univ. dr. Bâra Adela

Absolvent:
Mireag Mădălina-Andreea

București

2021

CUPRINS

INTRODUCERE

Trăim într-o eră a tehnologiei în care marea majoritate a activităților, de recreere sau nu, sunt desfășurate în mediul online.

Am ales să studiez această temă de licență deoarece, pe de o parte fiind pasionată de lectură este un domeniu ce prezintă interes pentru mine. Pe de altă parte, în special în contextul actual al pandemiei de COVID-19 de anul acesta am considerat că este importantă realizarea unei soluții care să le permită oamenilor împrumutarea cărților într-o modalitate sigură ce nu necesită deplasarea. Digitalizarea este un trend în această perioadă critică, devenind o necesitate în anumite domenii pentru adaptarea la schimbări.

Aplicația realizată se adresează tuturor iubitorilor de lectură ce doresc să împrumute cărți din confortul propriei locuințe. Cu o interfață prietenoasă și ușor de utilizat, aceasta oferă un beneficiu atât cititorilor cât și bibliotecii, asigurând siguranța procesului de împrumut al cărților dorite.

Capitolul 1 al lucrării își propune să studieze domeniul abordat de prezenta lucrare, activitatea ce urmează a fi informatizată și realizarea unei scurte analize a soluțiilor deja existente pe piață.

Capitolul 2 își propune analiza sistemului informatic, iar capitolul 3 prezintă proiectarea sistemului informatic și realizarea schemei bazei de date.

Capitolul 4 își propune descrierea tehnologiilor utilizate, a pașilor parcurși pentru implementarea aplicației și realizarea unei scurte prezentări a funcționalităților acesteia. La final vor fi prezentate concluziile rezultate din realizarea lucrării.

CAPITOLUL 1: DESCRIEREA PROBLEMEI ECONOMICE

1.1. PREZENTAREA DOMENIULUI ABORDAT

Lectura stă la baza culturii și a educației în orice societate modernă. La momentul actual, internetul și lectura sunt stâlpii educației și informării populației.

Posibilitățile populației pentru a avea acces la această activitate sunt numeroase, de la târguri de carte până la librării și biblioteci, fiecare cu propriile avantaje și dezavantaje. Desigur că în contextul actual al pandemiei primele două variante au devenit mai greu, chiar imposibil de realizat fără a supune populația la riscuri.

În ceea ce privește utilitatea cărților în format fizic în detrimentul internetului este de menționat nesiguranța corectitudinii informațiilor găsite pe internet. Din moment ce oricine are acces la acesta poate posta pe internet, există o șansă foarte mare să fie postate informații false sau greșite. Din acest punct de vedere cartea este o sursă mai sigură de informații, întrucât informațiile scrise în cărți sunt în general documentate și verificate în prealabil, iar unele cărți au nevoie chiar de aprobări pentru a fi publicate. Desigur, există și perspectiva cărților în format electronic, dar acestea au și ele un dezavantaj. Petrecând atât de mult timp concentrat în fața unui ecran citind o carte, omul riscă să aibă probleme de sănătate în viitor, de exemplu ochi obosiți sau diverse alte afecțiuni ce duc în general la portul ochelarilor de vedere.

Deși lectura este un factor atât de important, statisticile din România nu sunt foarte încurajatoare când vine vorba de acest lucru. Conform unui articol din anul 2019, în care este citată ca sursă Eurostat, se spune că aproximativ 6.5% din români cumpără cel puțin o carte pe an. În ceea ce privește cititul, România este printre ultimele țări din Europa la acest capitol. [1]

În acest sens au fost demarate numeroase campanii ce încurajează cititul, precum timpulsacitim.ro organizată de asociația Curtea Veche prin care o serie de oameni publici precum politicieni, sportivi și antreprenori să posteze pe rețelele de socializare materiale video cu privire la propria experiență legate de lectură și multe altele. [2]

De asemenea, conform Barometrului de consum cultural din 2019 se observă faptul că cititul este predominant în rândul persoanelor cu studii superioare. Tot aici se observă procente mici în ceea ce privește utilizarea internetului pentru a cumpăra cărți, procent ce

nu depășește 20%, dar și faptul că procentul persoanelor care declară că nu au citit niciodată o carte pare să fie mai semnificativ în rândul persoanelor cu venituri mai mici. [3]

Având la dispoziție atâtea statistici negative ne-am putea întreba ce îi împiedică pe oameni să citească. Deși majoritatea acuză lipsa timpului ca fiind principalul motiv, consider că acesta nu este întotdeauna un motiv valid. Lectura este o activitate ce ar trebui cultivată din copilărie, încă din anii de școală copiii ar trebui încurajați să citească. Rețelele de socializare pot fi de asemenea un motiv pentru care oamenii nu își fac timp sa citească, acaparându-le puținul timp liber într-un mod nu tocmai benefic.

Pe lângă informare și educație, lectura poate fi un hobby pentru cei ce simt nevoia să evadeze din realitatea cotidiană. O carte bună te poate scoate la lumină în cele mai întunecate momente, ajutându-te să te regăsești și să înțelegi mai bine persoanele din jurul tău.

Biblioteca vine în sprijinul oamenilor cu posibilități financiare reduse, oferind accesul la lectură într-un mod gratuit și facil. Întrucât librăriile au prețuri relativ mari în cele mai multe cazuri aceasta devine o alternativă preferată de cititori.

În timpul pandemiei însă principalele instituții de acest tip, precum Biblioteca Națională a României, Biblioteca Centrală Universitară și altele nu au oferit în general posibilitatea împrumuturilor online. Deși acestea oferă acces la catalogul propriu online, pentru a împrumuta cărți este necesară deplasarea până la sediul instituției, devenind un inconvenient în contextul actual.

Statisticile sunt totuși destul de îngrijorătoare și în acest sens. Conform Institutului Național de Statistică, numărul de biblioteci din România a fost în continuă scădere în ultimii 10 ani, după cum se observă și în graficul următor. [4]

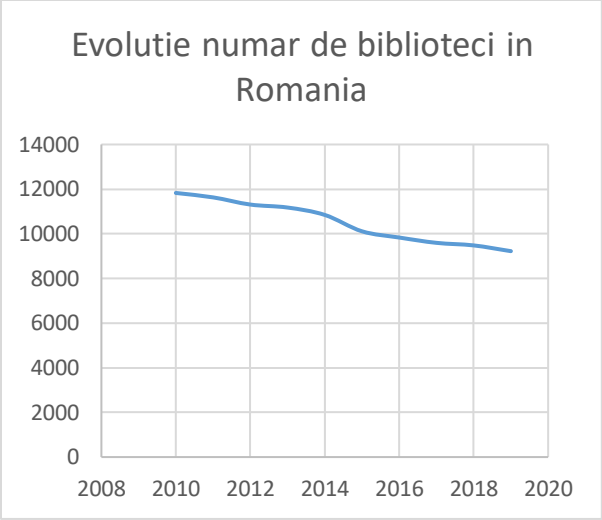


Figura 1: Graficul evoluției în perioada 2010-2019 a numărului de biblioteci

1.2. PREZENTAREA ACTIVITĂȚII CARE VA FI INFORMATIZATĂ

Aplicația își propune să faciliteze procesul de împrumutare a unei cărți de la bibliotecă, informatizând activitățile principale impuse de acest proces.

La momentul actual, în vederea împrumutării de cărți de la o bibliotecă este necesară parcurgerea unor anumiți pași de către persoana care dorește să împrumute.

În primul rând, este necesară deplasarea până la clădirea bibliotecii respective. Astfel, persoana trebuie să se documenteze și să găsească o bibliotecă aflată în proximitatea propriei locuințe.

În al doilea rând, în cazul în care persoana respectivă nu a mai avut nicio interacțiune cu acea instituție, este necesară comunicarea cu personalul responsabil în vederea înregistrării datelor personale în sistemul instituției și obținerii unei legitimații. În cazul în care persoana posedă legitimația, este necesară verificarea acesteia de personalul bibliotecii.

Regăsirea în baza de date a bibliotecii poate fi un proces care durează mult, în special dacă personalul prezent al bibliotecii nu este chiar la punct cu sistemul și cu tehnologia. Se pot crea astfel cozi și aglomerații nedorite.

Nu în ultimul rând, este necesară comunicarea cu personalul în vederea obținerii cărților dorite. În acest punct intervine un impediment pentru persoanele care nu sunt hotărâte în privința cărților pe care le doresc, ele fiind nevoite să solicite sfaturile personalului care pot corespunde sau nu nevoilor lor. Spre exemplu, personalul ar putea

să nu aibă cunoștințe suficiente cu privire la un anumit domeniu, astfel neștiind ce cărți anume să recomande.

Un alt impediment poate interveni în cazul în care instituția nu are disponibilă cartea cerută, caz în care deplasarea a fost făcută fără a obține rezultatul dorit în urma acesteia.

Desigur, toate aceste interacțiuni și chiar deplasarea de la domiciliu sunt un risc în sine în contextul pandemiei. De asemenea, în anumite biblioteci în care persoanele își pot alege cărțile direct de pe raft personalul nu are un real control asupra situației, întrucât fluxul de vizitatori poate fi necontrolabil.

De asemenea, bibliotecile din centrele universitare dedicate studenților devin mai greu accesibile în special celor care vin din provincie, aceștia neavând acces la materialele pe care le-ar fi găsit în biblioteca facultății.

1.3. ANALIZA SOLUȚIILOR EXISTENTE PE PIAȚĂ

1.3.1. Bookster

Una dintre soluțiile principale deja existente pe piață este site-ul celor de la Bookster, care permite împrumutul cărților de diferite genuri. Cu o colecție de peste 100.000 de cărți, Bookster oferă servicii de împrumut de cărți online. Deși au o platformă online pentru selecția cărților pe bază de categorie, în perioada crizei generate de COVID-19 aceștia și-au închis sediul și au permis livrarea cărților acasă sau la birou doar companiilor care au abonament la serviciile lor de livrare.

Deși soluția celor de la Bookster este foarte utilizată și populară, aceasta nu are o modalitate de utilizare eficientă în contextul pandemiei, persoanele fizice care nu sunt înscrise prin companiile la care lucrează nu beneficiază de livrare, fiind totuși nevoiți să se deplaseze pentru a-și prelua cărțile împrumutate. Un alt detaliu important este faptul că sediul acestora este situat în București, serviciile acestora devenind accesibile doar persoanelor care locuiesc în acest oraș.

Totodată, cei de la Bookster au fost acuzați de numeroase edituri din România de concurență neloială. Editurile ar fi fost nemulțumite de faptul că biblioteca ce are drept client țintă corporatiștii le-ar afecta vânzările, dar și de faptul că aceștia nu își cumpără cărțile direct de la edituri. [5]

1.3.2. Biblio-eXpert

Biblio-eXpert este o soluție contra cost propusă de compania Case Software. Aceasta le permite cititorilor să vizioneze cărțile disponibile în bibliotecă și să rezerve anumite cărți pe care le doresc pentru împrumut.

Totodată, aceasta dispune și de funcționalități pentru bibliotecari, care au dreptul de a adăuga, modifica, șterge cărți, categorii, și de a administra împrumuturile. Aplicația este disponibilă la momentul actual la un preț de aproximativ 49 de lei pe lună, pentru plata a cel puțin 12 luni. [6]

1.3.3. Aplicatia Bookup

Aplicația Bookup este o aplicație pentru telefoane disponibilă în PlayStore ce permite utilizatorilor să facă schimb (contra cost sau nu) de cărți utilizând geolocația pentru a găsi cărțile puse la dispoziție de utilizatori din zonă. Fiind o aplicație mobilă, aceasta este disponibilă doar pentru dispozitivele cu versiuni de android cu versiuni ulterioare celei de 4.1. Aplicația este disponibilă și pe Iphone. Fiind o aplicație pentru telefon este mult mai la îndemâna consumatorului, întrucât în ziua de astăzi majoritatea oamenilor dețin un smartphone care le este întotdeauna la dispoziție. [7]

Din păcate, aceasta nu pare a fi disponibilă în România în acest moment, utilizatorii din țară nu se pot astfel bucura de serviciile acesteia.

1.3.4. WizBooks

WizBooks este o soluție creată de compania Wiz Soft din Făgăraș pentru gestiunea activităților unei biblioteci. Aceasta oferă funcționalități precum adăugarea unui utilizator, adăugarea unei cărți în baza de date, oferind opțiunea de a adăuga cărți care se pot citi doar în sală sau cărți care se pot împrumuta și acasă, modificarea unei cărți existente, ștergerea unei cărți. Ștergerea nu este o ștergere efectivă, ci doar o scoatere din gestiune. O altă funcționalitate permisă de aceasta este căutarea în baza de date a unei cărți pe bază de diferite criterii. [8]

CAPITOLUL 2: ANALIZA SISTEMULUI INFORMATIC

2.1. SPECIFICAREA CERINȚELOR SISTEMULUI INFORMATIC

Principalul utilizator al sistemului propus este cititorul care efectuează împrumutul, iar pentru satisfacerea cerințelor acestuia este necesară cunoașterea și înțelegerea lor de către dezvoltator.

Sistemul realizat trebuie să îndeplinească o serie de cerințe privind datele de intrare, cât și datele de ieșire ce vor rezulta în urma prelucrării. Datele principale de intrare necesare procesului de împrumut sunt datele personale ale cititorului, dar și datele referitoare la disponibilitatea cărților

O primă cerință de menționat este corectitudinea datelor, atât a datelor de intrare cât și rapoartelor rezultate. Întrucât sistemul folosește ca informații de intrare datele personale ale utilizatorului pentru a realiza procesul de împrumut, acestea trebuie să fie introduse corect. Totodată, dacă datele privind disponibilitatea pentru împrumut a unei cărți sunt greșite pot apărea diverse probleme în buna funcționare a sistemului.

O altă cerință a sistemului este cea legată de relevanța datelor, conform căreia datele utilizate de sistem trebuie să fie relevante contextului activității ce se informatizează. Se vor utiliza astfel doar datele personale ale utilizatorului relevante activității de împrumut al unor cărți, ci nu date precum starea civilă sau nivelul de școlarizare, care nu au nicio importanță pentru această activitate.

În ceea ce privește modalitatea de funcționare a sistemului, există o serie de cerințe ce trebuie îndeplinite și din acest punct de vedere.

Sistemul va trebui să îndeplinească anumite specificații descriptive ce se referă la caracteristicile pe care acesta este nevoit să le realizeze, indiferent de modul de funcționare. Aceste tipuri de specificații sunt reprezentate de elemente de logică.

Un exemplu de astfel de specificație este faptul că același exemplar dintr-o carte nu poate fi împrumutat în același timp de mai multe persoane, întrucât acest fapt nu este fizic posibil. Astfel, atunci când un exemplar este deja împrumutat de un cititor, acesta devine indisponibil pentru ceilalți. În cazul în care mai există alte exemplare din cartea respectivă disponibile se pot oferi acelea, altfel va trebui ca cititorul să aștepte revenirea disponibilității unui exemplar.

Un alt exemplu de astfel de specificație este faptul că data returnării unei cărți nu poate precede data efectuării împrumutului, deoarece din punct de vedere logic este imposibil.

În ceea ce privește cerințele funcționale, legate de funcționalitățile pe care va trebui să le îndeplinească sistemul, menționez și în acest caz o serie de cerințe relevante din punct de vedere al temei propuse.

Pentru început, sistemul trebuie să ofere utilizatorului o modalitate de a-și introduce datele personale, cu scopul înregistrării în sistem, fapt care îi va permite acestuia ulterior accesul la restul funcționalităților sistemului.

Printre aceste cerințe se numără faptul că sistemul va trebui să pună la dispoziția utilizatorului în permanență o listă a cărților disponibile pentru împrumut, împreună cu informații precum autorii acestora. Utilizatorul va trebui să aibă posibilitatea de a parcurge această listă de cărți disponibile și alege o anumită carte cu scopul de a o împrumuta.

De asemenea, sistemul trebuie să fie capabil să proceseze datele utilizatorului, precum și cărțile selectate de acesta cu ajutorul cărora să realizeze un raport rezultat, în forma unei fișe electronice ce va reprezenta dovada efectuării împrumutului de către persoana respectivă. În realizarea acestuia, sistemul trebuie să verifice în prealabil disponibilitatea cărților cerute și încadrarea cititorului în condițiile necesare procesului de împrumut.

Pentru a reprezenta într-o formă grafică funcționalitățile pe care sistemul va trebui să le îndeplinească au fost create folosind limbajul UML diagramele cazurilor de utilizare ce urmează a fi prezentate în continuare.

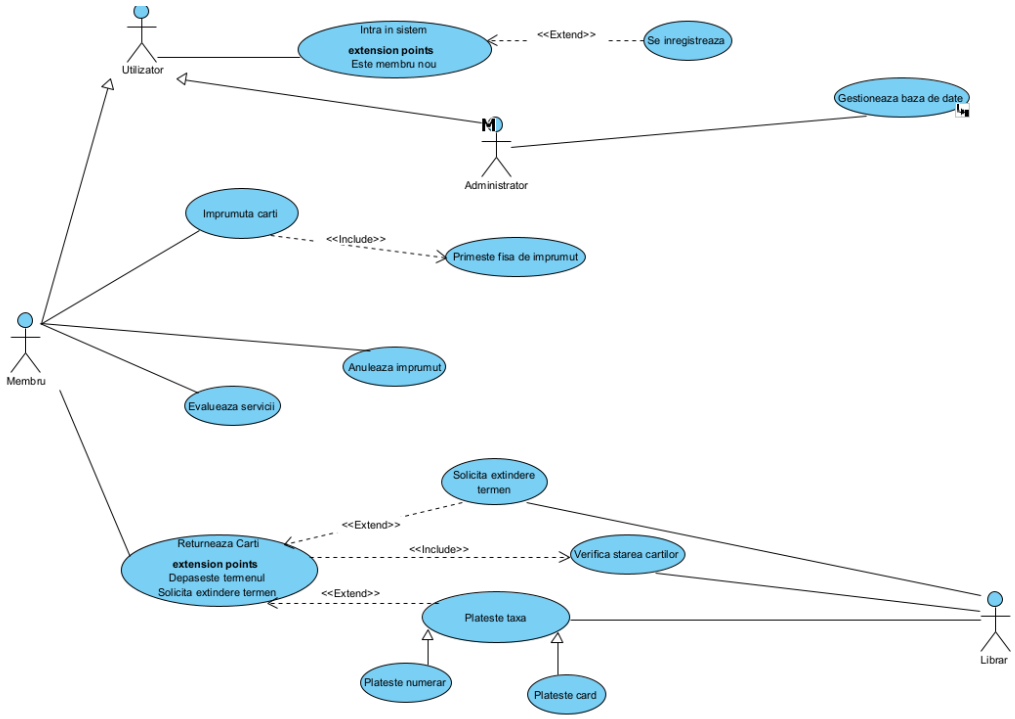


Figura 2: Diagrama generală a cazurilor de utilizare

În vederea efectuării unui împrumut, este necesar ca cititorul să își introducă datele personale în vederea înregistrării în sistem. Odată înregistrat, acesta dispune de un cont personal prin intermediul căruia poate interacționa cu sistemul. Acesta are la dispoziție o listă de cărți disponibile împreună cu autorii acestora, din care poate alege o carte în vederea împrumutării. Odată selectată cartea pe care dorește s-o împrumute, acesta poate continua procesul de împrumut căutând alte cărți, sau îl poate încheia dacă nu mai dorește alte cărți.

În momentul efectuării împrumutului, acesta primește o fișă de împrumut în care este menționată data la care a fost efectuat împrumutul și data la care trebuie returnate cărțile. Dacă acesta depășește data de returnare, el trebuie să plătească o taxă de întârziere pentru a putea continua interacțiunea cu biblioteca. Totodată, acesta are posibilitatea de a evalua serviciile oferite de aplicație și de a anula împrumutul dacă se răzgândește și nu dorește să îl finalizeze.

Tabelul cu descrierea textuală a cazului de utilizare pentru procesul de împrumut este următorul:

Element al cazului de utilizare	Descriere
Cod	CU01
Stare	Schiță
Scop	Monitorizare sistem împrumut cărți
Nume	Monitorizarea gestiunii operațiunilor utilizatorilor de împrumut de cărți
Actor principal	Membru, Sistem, Administrator
Descriere	Presupune realizarea unui împrumut cu una sau mai multe cărți
Precondiții	Utilizatorul inregistrat in sistem
Postcondiții	Utilizatorul finalizează împrumutul
Declanșator	Membrul se înregistrează în sistem
Flux de bază	1. Clientul își introduce datele de conectare [CA1: E deja înregistrat în sistem] 2. Membrul alege o carte spre împrumut 3. Dacă membrul dorește să continue cautand alte cărți o face, dacă nu finalizează împrumutul [CA2: Membrul nu finalizează împrumutul] 4. Sistemul îi generează o fișă de împrumut la finalizare, continand detaliile împrumutului
Fluxuri alternative	CA1: Datele sunt deja înregistrate în sistem CA2: Scenariul se încheie, împrumutul este anulat
Relații	Se extinde cu CU02 Gestionează baza de date
Frecvența utilizării	Medie
Reguli ale afacerii	-

Tabel 1: Descrierea textuală a cazului de utilizare general

Activitatea de gestiune a bazei de date îi aparține administratorului și va fi descrisă ulterior. Pentru simplificare, se va realiza o diagramă separată ce descrie activitatea de gestiune a bazei de date realizată de administrator:

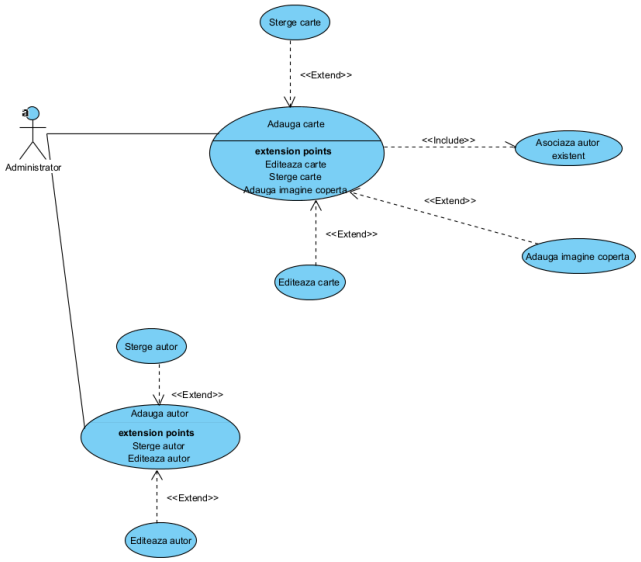


Figura 3: Diagrama pentru cazul de utilizare gestionează baza de date

Administratorul se ocupă cu adăugarea de noi autori în sistem, adăugarea de cărți noi, actualizarea acestora, dar și ștergerea atunci când ele nu mai sunt disponibile pentru împrumut. În momentul adăugării unei cărți acesta are posibilitatea de a selecta unul sau mai mulți autori din cei existenți în baza de date pentru a-i asocia cărții respective, după caz. Doar administratorul poate efectua modificări asupra bazei de date.

Element al cazului de utilizare	**Descriere**
Cod	CU02
Stare	Schiță
Scop	Activitate gestiune baze de date
Nume	Monitorizarea gestiunii bazei de date pentru împrumut cărți
Actor principal	Administrator
Descriere	Presupune realizarea operațiilor CRUD (Create, Read, Update, Delete)
Precondiții	Administrator înregistrat în sistem
Postcondiții	Baza de date reflectă realitatea
Declanșator	Administratorul se înregistrează în sistem
Flux de bază	1. Administratorul introduce un autor 2. Administratorul introduce o carte 3. Pentru acea carte, poate asocia autori existenți deja în baza de date 4. Dacă se dorește să aibă și o copertă, asociază o imagine cărții 5. Odată adăugată cartea sau autorul, poate opta să editeze anumite aspecte
Fluxuri alternative	
Relații	-
Frecvența utilizării	Medie
Reguli ale afacerii	Doar administratorul poate efectua modificări asupra bazei de date

Tabel 2: Descriere textuală a cazului de utilizare gestionează baza de date

2.2. ANALIZA SISTEMULUI EXISTENT

În vederea realizării etapei de analiză a sistemului au fost construite o serie de diagrame utilizând limbajele standardizate UML (Unified Modeling Language) și BPMN (Business Process Model and Notation). Rolul acestora este asigurarea unei bune înțelegeri a modului în care sistemul va trebui să funcționeze, dar și functionalitățile de bază ale acestuia, obiectele care vor fi prezente la nivelul acestuia și relațiile existente între ele.

Cu scopul prezentării principalelor secvențe de acțiuni îndeplinite de sistem au fost construite diagramele de activitate ce urmează a fi prezentate.

Principala activitate realizată de sistem este managementul împrumutului de cărți pentru un membru. În scopul prezentării acestei activități am realizat următoarea diagramă:

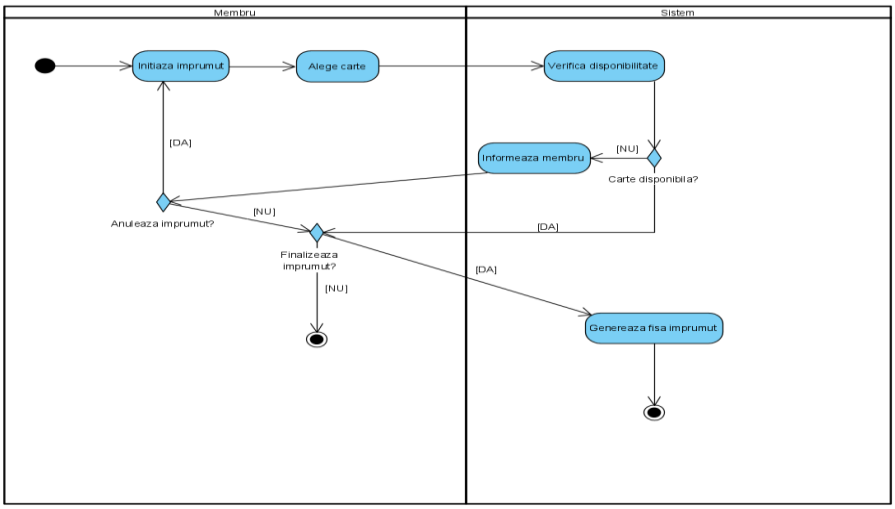


Figura 4: Diagrama de activitate pentru împrumutul unor cărți

Atunci când utilizatorul dorește să împrumute o carte mai întâi inițiază un împrumut, îi este pusă la dispoziție o listă de cărți disponibile din care alege, moment în care sistemul verifică disponibilitatea cărții alese. În cazul în care aceasta nu este disponibilă sistemul va informa utilizatorul, acesta având posibilitatea de a anula sau de a relua procesul alegând altă carte. În cazul în care cartea aleasă este disponibilă utilizatorul are posibilitatea de a finaliza procesul de împrumut, moment în care sistemul va genera fișa de împrumut aferentă.

O altă activitate importantă este cea a returnării de cărți, iar în scopul descrierii acesteia voi prezenta următoarea diagramă:

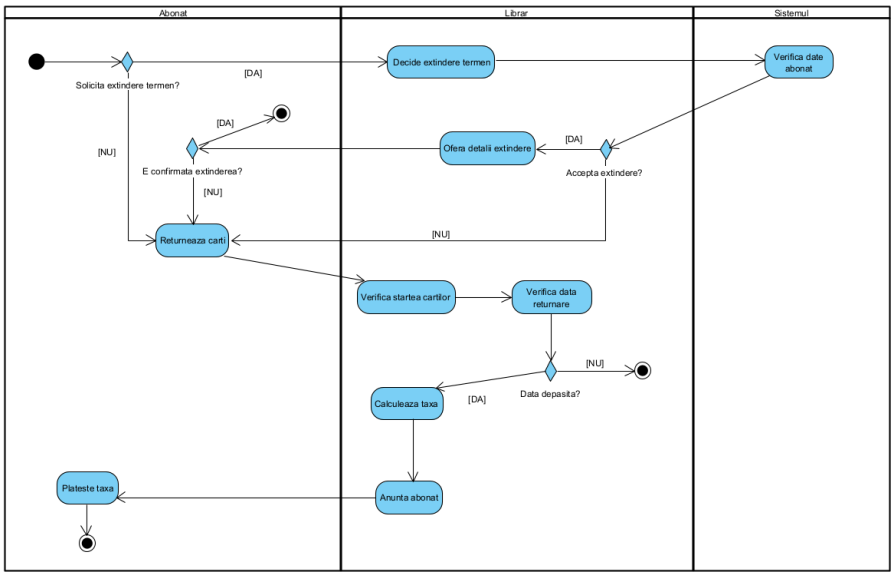


Figura 5: Diagrama de activitate pentru returnarea unei cărți împrumutate

Atunci când trebuie să returneze o carte, utilizatorul poate solicita o extindere a termenului. În acest caz, acesta este verificat de sistem, iar librarul decide dacă i se acordă sau nu extinderea. Dacă da, acesta îi propune o ofertă de extindere, iar dacă utilizatorul acceptă activitatea se încheie. În cazul în care acesta nu este de acord, va fi nevoit să continue activitatea de returnare. Librarul va verifica starea în care se află cărțile și dacă a fost depășit termenul sau nu. Dacă termenul a fost depășit, acesta va calcula o taxă ce va fi plătită de utilizator.

Una dintre activitățile principale ale administratorului este adăugarea de noi cărți în baza de date. Pentru a descrie modul de realizare al acestei activități a fost realizată următoarea diagramă de activitate:

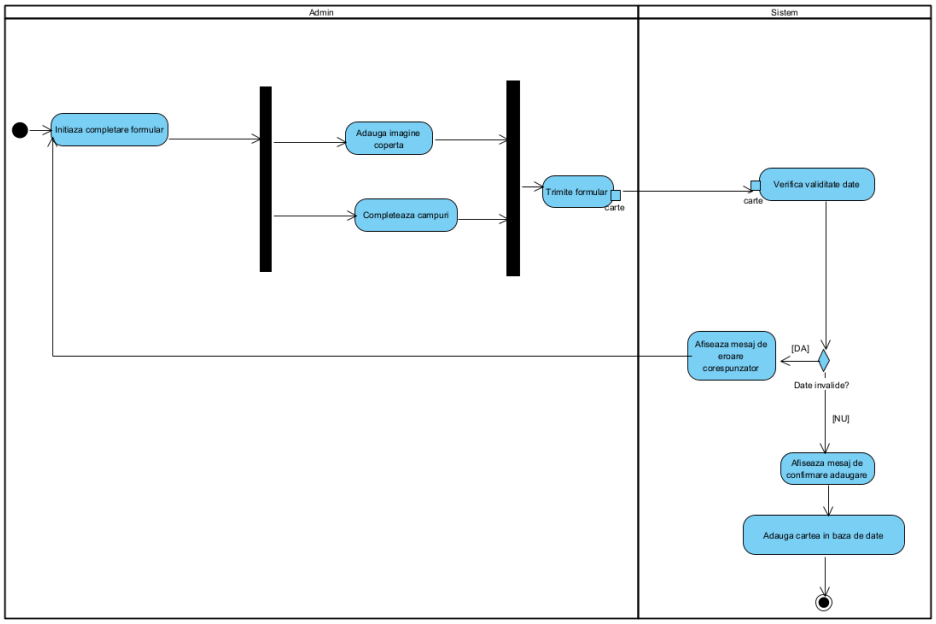


Figura 6: Diagrama de activitate pentru adaugarea unei cărți noi în sistem

Pentru realizarea activității de adăugare a unei cărți noi în sistem trebuie completat un formular de adăugare. Dacă se dorește adăugarea unei imagini a copertei pentru a fi afișată ulterior utilizatorului se va proceda spre adăugarea acesteia. Abia după finalizarea completării câmpurilor și a alegerii imaginii de copertă se poate proceda spre trimiterea formularului, care va fi validat de sistem. În funcție de rezultatul validării, sistemul va trimite fie un mesaj de eroare, fie un mesaj de confirmare a adăugării cu succes în baza de date a cărții.

În continuare voi prezenta diagrama de clase construită în etapa de analiză, cu scopul de a arăta principalele clase identificate și relațiile dintre acestea.

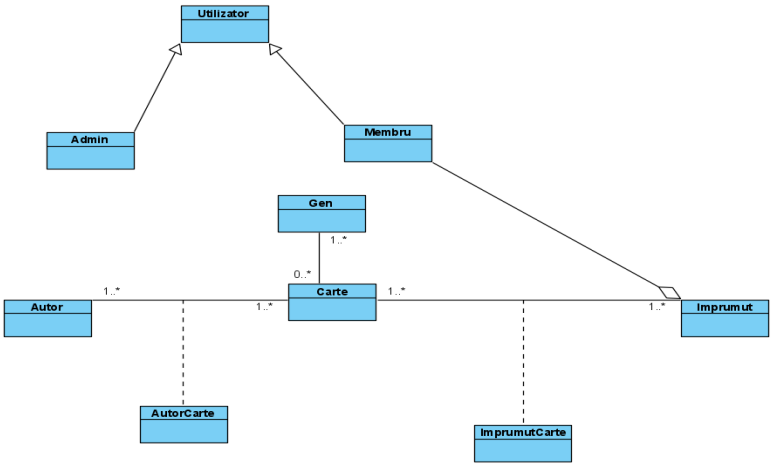


Figura 7: Diagrama de clase

Întrucât o carte poate fi scrisă de unul sau mai mulți autori și un autor poate participa la scrierea unei sau mai multor cărți, există o relație de mulți la mulți care a fost modelată printr-o clasă modelată ca o asociere. Același lucru se observă și în cazul claselor Carte și Împrumut, întrucât clasa carte nu se referă la un singur exemplar dintr-o carte, deci poate fi asociată mai multor împrumuturi dacă există mai multe exemplare disponibile din această carte. Astfel, relația dintre cele două clase este tot o relație mulți la multi și va fi modelată tot printr-o clasă modelată ca o asociere.

Pentru a putea descrie stările prin care trec principalele obiecte ale sistemului de-a lungul existenței lor s-a realizat câte o diagramă de mașini cu stare pentru fiecare dintre acestea.

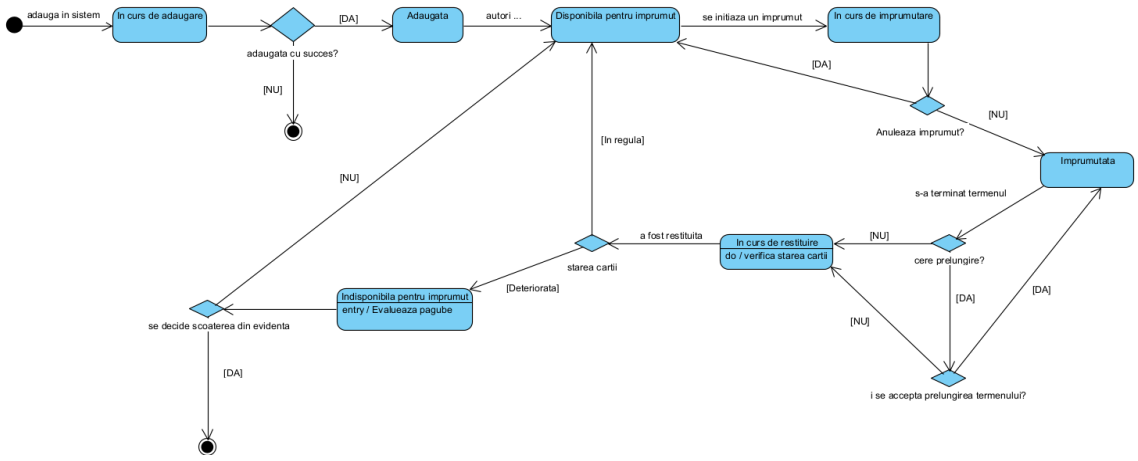


Figura 8: Diagrama de stare a obiectului carte

Obiectul carte devine disponibil pentru împrumut abia după ce are și autorul sau autorii corespunzători asociați. Atunci când se inițiază un împrumut pentru obiectul respectiv acesta intră în starea în curs de împrumutare. Dacă se anulează împrumutul devine din nou disponibil pentru împrumut, dacă se finalizează împrumutul aceasta devine împrumutată. Atunci când expiră termenul se poate cere prelungirea acestuia care, dacă este acceptată determină revenirea cărții la starea de împrumutată.

Dacă nu se cere o prelungire, sau aceasta nu este acceptată atunci cartea tranziționează spre starea în curs de restituire, în timpul căreia este verificată starea acesteia. Dacă se constată că starea cărții este deteriorată se poate decide scoaterea din evidență a acesteia. În caz contrar, aceasta va redeveni disponibilă pentru împrumut.

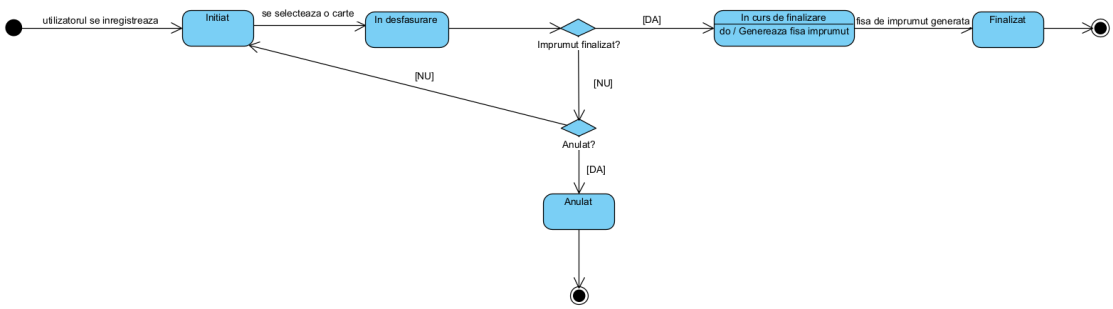


Figura 9: Diagrama de stare a obiectului Împrumut

Împrumutul este inițiat de către utilizator, în momentul în care acesta selectează o carte împrumutul este în desfășurare. Dacă utilizatorul dorește să finalizeze împrumutul, acesta devine în curs de finalizare, iar fișa de împrumut este generată de sistem. Când fișa a fost generată împrumutul este finalizat. Dacă totuși utilizatorul dorește să anuleze împrumutul, acesta devine anulat și nu mai este finalizat. Dacă utilizatorul nu dorește să finalizeze împrumutul, dar nici să-l anuleze înseamnă că mai dorește și alte cărți, moment în care împrumutul revine la starea de inițiat unde se așteaptă alegerea unei alte cărți.

Pentru a modela principalele interacțiuni dintre anumite elemente ale sistemului am realizat diagrame de secvențe pentru două dintre activitățile principale ale sistemului.

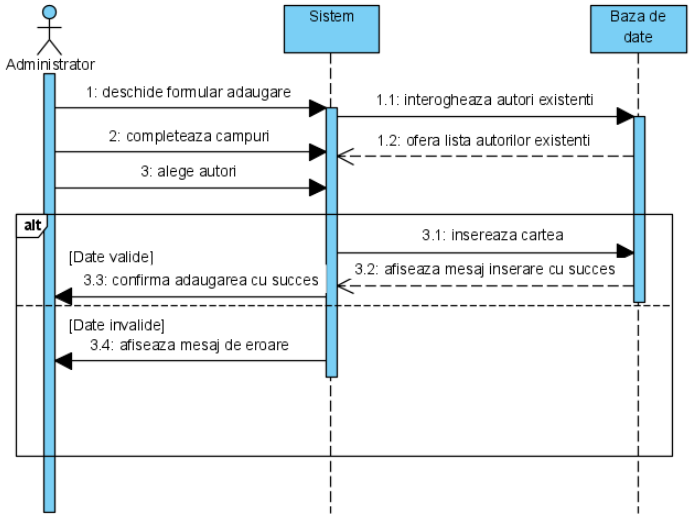


Figura 10: Diagrama de secvență pentru adăugarea unei cărți

Administratorul inițiază adăugarea unei cărți prin deschiderea formularului și completarea câmpurilor. Sistemul va interoga baza de date pentru a furniza autorii existenți din care acesta poate alege. Dacă datele introduse sunt valide sistemul va insera

nouă carte în baza de date și va afișa un mesaj de confirmare, altfel va trimite un mesaj de eroare.

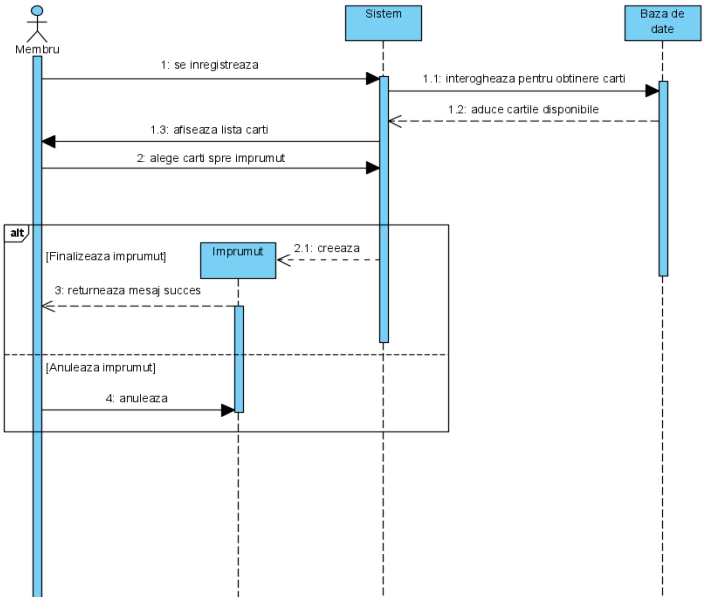


Figura 11: Diagrama de secvență pentru imprumutul de cărți

În momentul înregistrării utilizatorului, sistemul va interoga baza de date pentru a aduce cărțile existente, urmând ca utilizatorul să aleagă dintre acestea. Dacă utilizatorul finalizează împrumutul sistemul va crea un obiect de tip împrumut și va returna un mesaj de succes.

Diagrama de procese aferentă procesului de împrumut al unei cărți este următoarea:

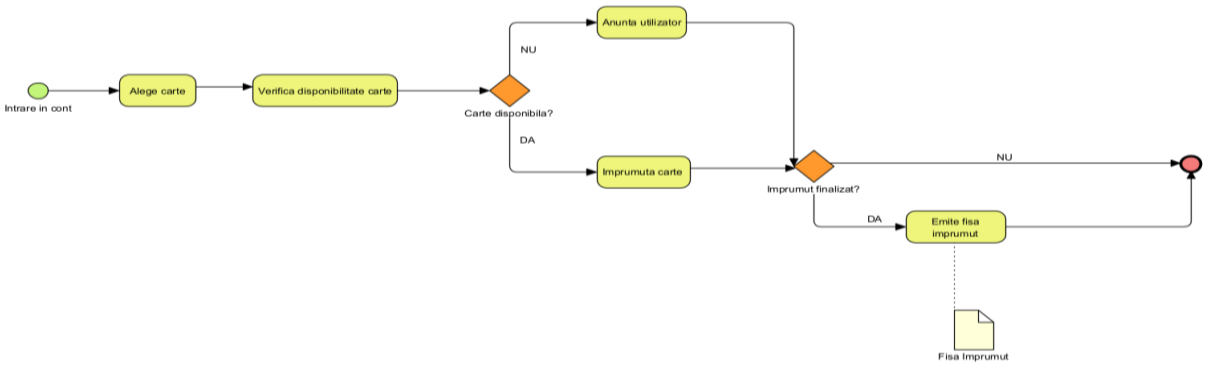


Figura 12: Diagrama de procese pentru împrumut

Pentru a ilustra modul în care interacționează librarul cu membrul în procesul de returnare a unei cărți am creat următoarea diagramă de colaborare:

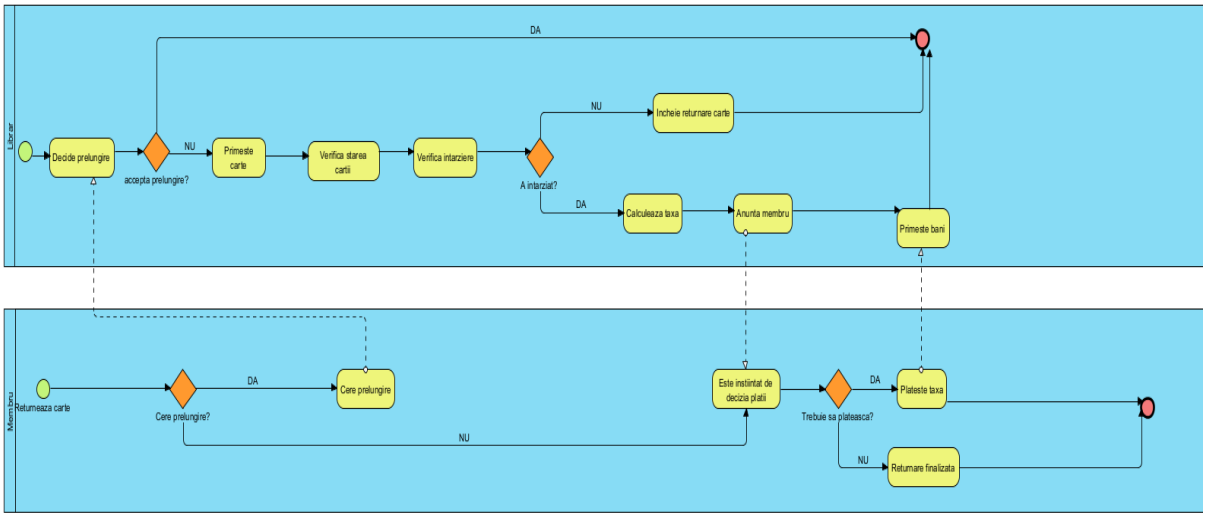


Figura 13 Diagrama de colaborare pentru returnarea unei cărți

CAPITOLUL 3: PROIECTAREA SISTEMULUI INFORMATIC

3.1. PROIECTAREA NOULUI SISTEM

În această etapă se va realiza proiectarea noului sistem la nivel de interfață și funcționalități, introducându-se și detalii specifice tehnologiei ce urmează a fi utilizată. Întrucât aplicația realizată va fi o aplicație mobilă Android, se vor adăuga în continuare detalii specifice acestui sistem de operare.

Pornind de la diagrama de clase din etapa de analiză a sistemului, a fost realizată diagrama de clase detaliată folosind limbajul UML. Au fost adăugate detalii ce țin de implementare, precum și atributele și metodele principale pe care le va avea fiecare clasă.

Diagrama de clase detaliată realizată este următoarea:

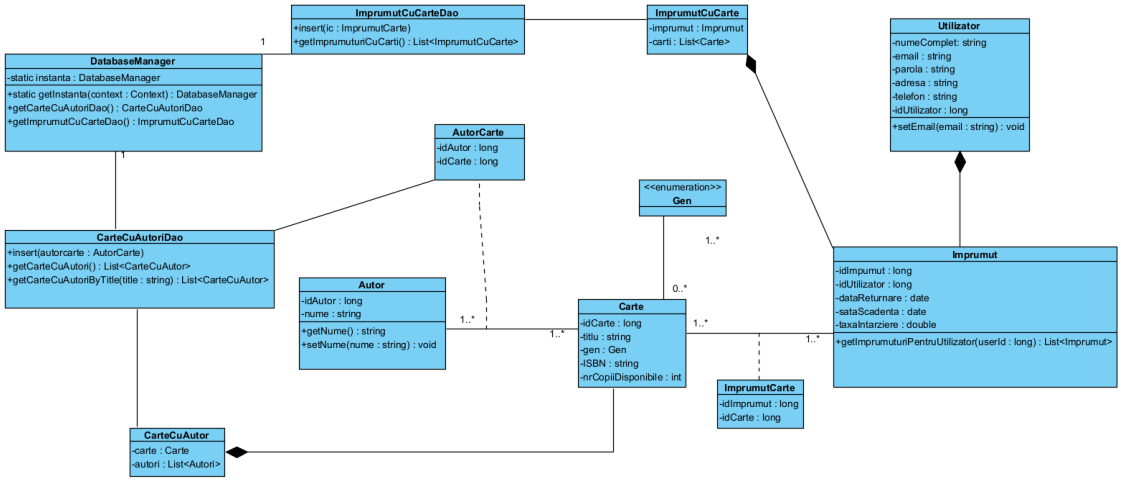


Figura 14: Diagrama de clase detaliată

Managerul bazei de date (DatabaseManager) va fi cel care va gestiona crearea conexiunii la baza de date și gestiunea tabelelor prin biblioteca Room disponibilă în Android. Aceasta va avea nevoie obiecte de tip DaO (Database Object), motiv pentru care este necesară introducerea claselor CarteCuAutoriDao și ImprumutCuCarteDao cu ajutorul cărora Room va realiza operațiile specifice SQL, precum inserare, ștergere și interogare pentru obiectele respective.

Pentru a evidenția componentele software ale sistemului și modul în care acestea depind unele de altele am realizat o diagramă de componente, utilizând limbajul UML.

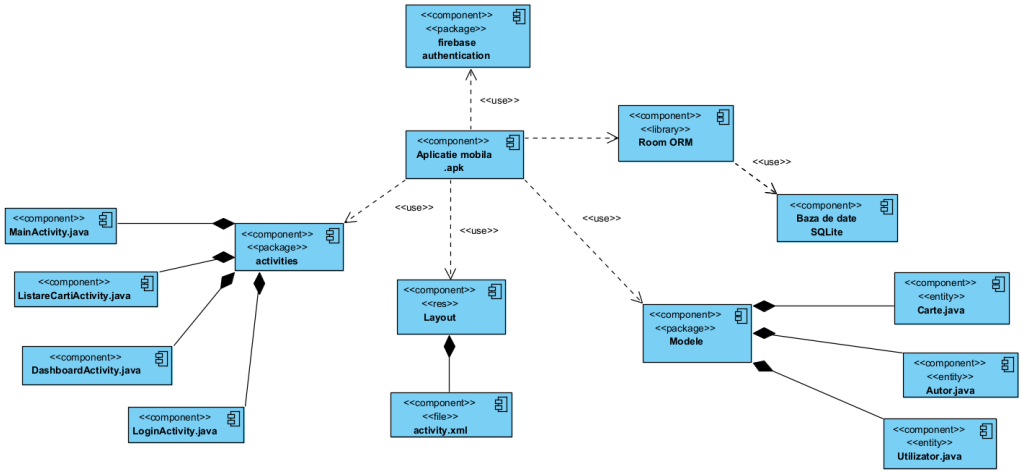


Figura 15: Diagrama de componente

Pentru a funcționa corect aplicația mobilă are nevoie de modelele de date, reprezentate de clasele ce implementează logica aplicației precum Carte, Autor, Utilizator etc. Aceasta mai are nevoie să utilizeze și activități, precum MainActivity.java, ListareCartiActivity.java etc, concept care este specific Android. Acestea sunt cele care stabilesc ce ferestre vor fi afișate, cum vor arăta acestea și ce funcționalități vor avea.

De asemenea, aplicația mai are nevoie de o dependință către librăria Room pentru a abstractiza accesul la baza de date SQLite și a efectua operațiile aferente, dar și de o dependință către partea de autentificare a platformei Firebase pentru a putea accesa clasele și funcționalitățile acesteia în implementarea autentificării utilizatorilor.

Diagrama de desfășurare ce prezintă relațiile între dispozitivul mobil și componentele necesare funcționării aplicației este următoarea:

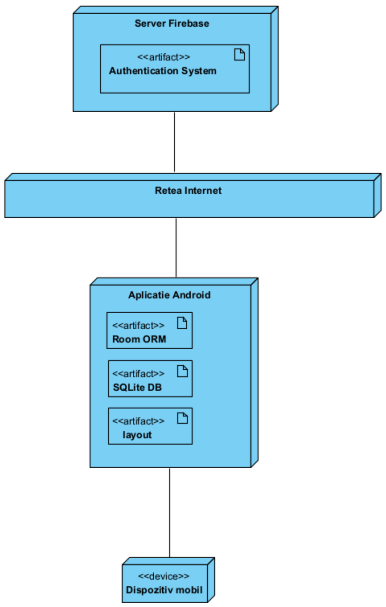


Figura 16: Diagrama de desfășurare

Pentru a funcționa este nevoie de un dispozitiv mobil pe care să fie instalată aplicația și cu conexiune la internet. Aplicația se va conecta la serverul Firebase pentru a verifica informațiile utilizatorului și a realiza autentificarea, apoi va putea fi utilizată.

Pentru a proiecta interfețele utilizator principale pe care le va avea aplicația am folosit programul Balsamiq, acestea fiind împărțite între categoria utilizatorului obișnuit și administratorului. Atât administratorul cât și utilizatorul obișnuit vor avea fereastra de login la dispoziție, utilizatorul având și opțiunea de a-și crea un nou cont.

Machetele interfețelor principale ce vor fi utilizate de un utilizator obișnuit sunt următoarele:

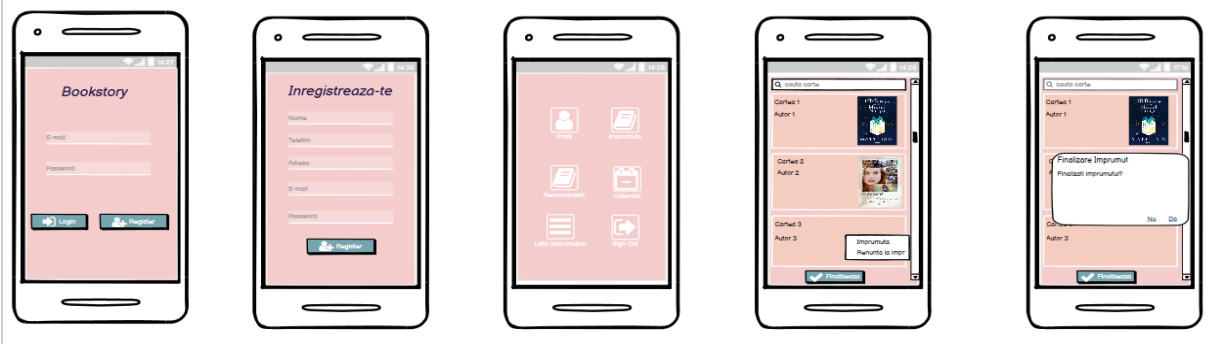


Figura 17: Macheta interfețe membru obișnuit

Utilizatorii vor avea posibilitatea de a intra în cont sau a-și crea unul nou, ajungând apoi la un meniu din care pot alege mai multe funcționalități. Aceștia își pot

vedea și modifica profilul, pot vedea lista de cărți, de împrumuturi sau se pot deconecta. Odată ce ajung în fereastra cu lista cărților disponibile aceștia pot împrumuta cărțile dorite.

Macheta cu cele mai importante interfețe pentru administrator este următoarea:

Figura 18: Macheta interfete administrator

Administratorul va avea și el la dispoziție un meniu în care va avea opțiunea de a șterge cărțile și autorii din baza de date, de a adăuga o nouă carte, de a vizualiza cărțile sau de a ieși din cont.

3.2. PROIECTAREA SCHEMEI BAZEI DE DATE

Pe baza cerințelor și a funcționalităților pe care va trebui să le îndeplinească aplicația a fost creată schema bazei de date folosind modelul entitate-asociere. Baza de date utilizată este una relațională întrucât între entitățile sistemului se stabilesc relații cu ajutorul cărora acestea comunică între ele, iar datele sunt structurate. Astfel, relațiile dintre entități vor fi modelate prin intermediul conceptelor de chei primare și chei externe.

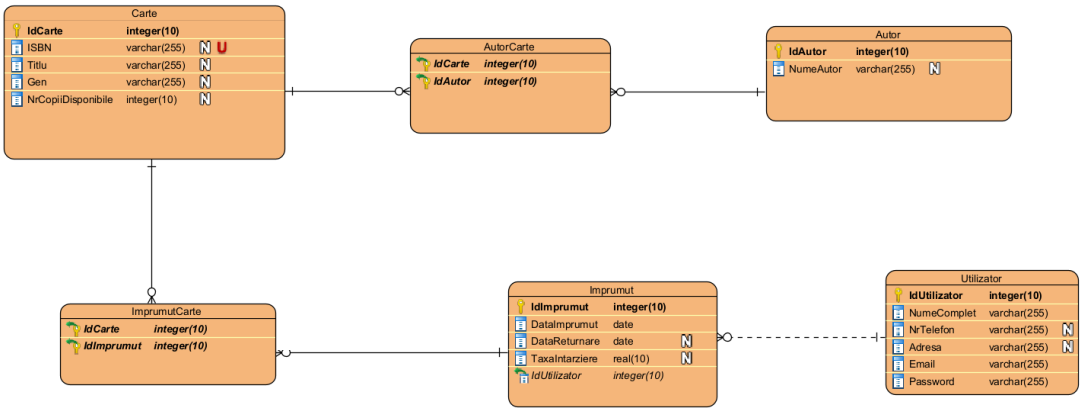


Figura 19: Schema bazei de date

Entitățile importante necesare aplicației ce au fost identificate sunt Carte, Autor, Utilizator și Împrumut cu atributele aferente.

Tabela Carte va avea câmpul idCarte (integer) ca și cheie primară, ISBN (varchar) ca și cheie unică, titlu (varchar), gen (varchar) și un număr de copii disponibile (integer). Tabela Autor va stoca doar cheia primară idAutor (integer) și numele autorului (varchar), fiind singurele informații necesare aplicației.

Tabela Utilizator va stoca informații precum numele complet (varchar), numărul de telefon (varchar), adresa (varchar), adresa de e-mail (varchar) și parola (varchar) și va avea ca și cheie primară câmpul idUtilizator(integer).

Tabela Împrumut va stoca informații precum data împrumutului (date), data returnării (date), taxa de întârziere (real), cheia primară idImprumut (integer) și va avea în plus o cheie externă, idUtilizator, către tabela Utilizator.

Între tabela Carte și tabela Autor se stabilește o relație de mulți-mulți, întrucât o carte poate fi scrisă de unul sau mai mulți autori și un autor poate scrie una sau mai multe cărți. Acest tip de relație este implementat utilizând tabela de legătură AutorCarte care va conține ca și chei primare cele două chei primare ale tabelelor referite.

Același lucru se observă și în cazul tabelelor Carte și Împrumut, între care se stabilește tot o relație de tipul mulți-mulți. Acest lucru se întâmplă deoarece un împrumut poate conține mai multe cărți, iar o carte poate apărea în mai multe împrumuturi întrucât are mai multe exemplare disponibile. Relația este implementată utilizând tabela de legătură ImprumutCarte care va conține ca și chei primare cele două chei primare ale tabelelor referite.

CAPITOLUL 4: IMPLEMENTAREA SISTEMULUI INFORMATIC

4.1. TEHNOLOGII INFORMATICE UTILIZATE

Pentru realizarea acestei aplicații au fost folosite tehnologii precum Android, limbajul de programare Java, Firebase, baza de date SQLite.

Am ales să utilizez Android întrucât este unul dintre cele mai utilizate sisteme de operare dezvoltat de Google pentru dispozitivele mobile în prezent. În continuare voi prezenta o serie de statistici cu privire la utilizarea acestui sistem de operare atât la nivel național cât și la nivel mondial.

Conform StatCounter, în perioada Mai 2020 - Aprilie 2021, Android a avut o cotă de piață de aproximativ 80.67% în România, find astfel lider în acest domeniu. [9]

Tot conform StatCounter se poate observa evoluția continuă a cotei de piață a sistemului de operare la nivel mondial, ajungând la o valoare de 72,19% în Aprilie 2021. [10]

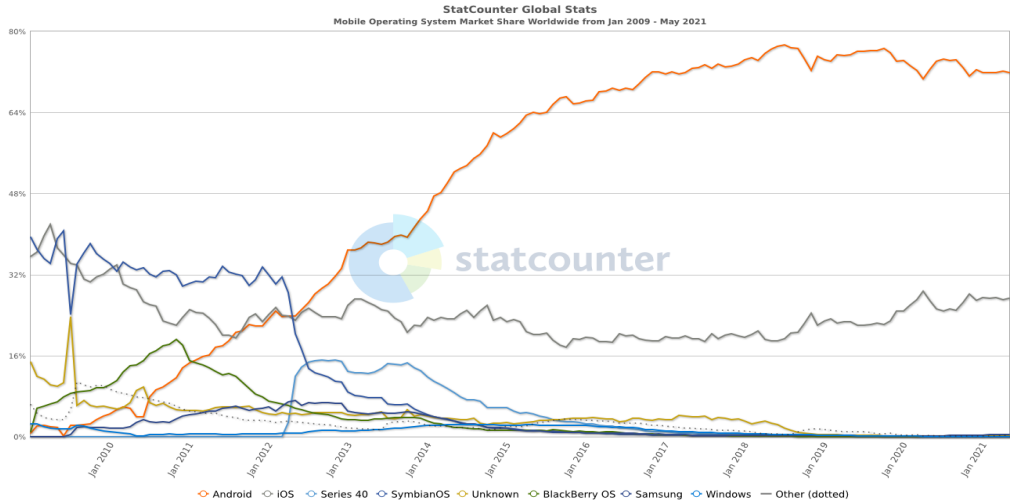


Figura 20: Cota de piață a sistemelor de operare mobile în perioada Ianuarie 2009–Mai 2021

Sursa: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202105

Arhitectura implicită a unei aplicații Android este aceea de MVC (Model-View-Controller), dar, după cum dorește dezvoltatorul, se poate folosi și MVVM (Model-View-ViewModel) sau MVP (Model-View-Presenter). Am ales să utilizez arhitectura MVC, întrucât este cea implicită existentă în Android.

Arhitectura MVC presupune separarea aplicației în trei părți logice mari, pentru creșterea calității acesteia și a timpului necesar dezvoltării.

Prima parte este cea de model ce conține toată logica de business a aplicației și este reprezentată în Android Studio de clasele java aferente modelării, precum Carte, Autor, Împrumut, Utilizator.

Următoarea parte, cea de View este implementată în Android prin fișierele de resurse în format XML utilizate pentru afișarea pe ecran, precum activity_dashboard.xml, activity_adauga_carte.xml, activity_login.xml etc. Aceasta este combinată totuși cu partea de activități, întrucât fișierele de resurse nu pot genera apariția pe ecran a interfeței specifice fără ajutorul clasei java asociate numită activitate, prin care se face legătura. [11]

Partea de Controller este reprezentată în Android tot de către activități, care au rolul de a face legătura dintre model și ce se afișează utilizatorului pe ecran. În Android Studio acestea sunt reprezentate prin clase Java. De exemplu, o parte din activitățile utilizate sunt AdaugăCarteActivity.java, ListareCartiActivity.java, RecomandariActivity.java etc.

Sistemul de operare a fost într-o evoluție continuă, apărând numeroase versiuni de-a lungul timpului. Prima versiune a fost Android 1.0 apărută în anul 2008 și suporta functionalitățile de bază precum camera, accesarea browserelor web, Wi-Fi etc. Între timp, evoluția a continuat apărând pe piața numeroase versiuni la care s-au adăugat funcționalități și s-a optimizat sistemul, cea mai recentă fiind Android 11.

Am ales să utilizez versiunea Android 8.0 (Oreo), întrucât este una relativ nouă și o mare parte din telefoane au preinstalată o versiune recentă a sistemului de operare. Ca și caracteristici aduse de această versiune dezvoltatorilor se poate menționa posibilitatea de a utiliza diversele fonturi disponibile ca și resurse, în cadrul directorului drawable.

Printre funcționalitățile introduse de această versiune se numără introducerea de noi permisiuni pentru aplicație, introducerea modului Picture-in-Picture, stiluri noi pentru emoji, introducerea canalelor de notificații pentru a permite organizarea acestora în funcție de importanța lor. [12]

Instrumentul de dezvoltare utilizat pentru realizarea aplicației este Android Studio, fiind bazat pe IntelliJ IDEA de la JetBrains, care este un instrument de dezvoltare pentru programarea în limbajul Java. Acesta pune la dispoziția dezvoltatorilor o colecție de librării și instrumente de dezvoltare precum depanatorul și editorul de cod java ce

oferă sugestii automate, cu ajutorul căruia aceștia pot crea aplicații pentru platforma Android.

O altă funcționalitate foarte utilă adusă dezvoltatorilor de aplicații Android este posibilitatea testării aplicației în orice moment folosind un dispozitiv mobil virtual numit emulator, ce simulează un dispozitiv real cu toate funcționalitățile acestuia. Se pot crea diferite configurații de emulatoare utilizând instrumentul AVD Manager prezent în Android Studio, alegându-se, printre altele, versiunea de Android pentru acestea și tipul de dispozitiv. Dispozitivul pe care l-am utilizat pentru testare este un emulator Nexus 5X cu un nivel API 27, adică cu versiunea Android Oreo. Totodată, Android Studio permite integrarea aplicațiilor folosind soluții de versionare precum Git.

Întrucât în Android Studio se poate dezvolta o aplicație fie folosind limbajul Java, fie Kotlin, am ales pentru implementare limbajul Java.

Principalele avantaje ale limbajului Java sunt faptul că este orientat-obiect, relativ ușor de învățat și este independent de sistemul de operare pe care este rulat. În prezent își menține poziția printre cele mai utilizate limbaje de programare, fiind clasat pe locul 3 în indicatorul de popularitatea limbajelor de programare TIOBE din Iunie 2021. [13]

SQLite este baza de date relațională implicită a dispozitivelor Android, iar SDK-ul de Android oferă numeroase clase și metode pentru gestionarea acesteia. Deoarece implementarea folosind aceste seturi de clase și metode poate deveni dificilă și costisitoare, a fost introdus un nivel de abstractizare oferit de biblioteca Room. Principalul avantaj al SQLite este faptul că este o bază de date compactă, iar utilizarea acesteia împreună cu sistemul de operare Android nu implică instalări suplimentare.

Astfel, am folosit biblioteca Room ce funcționează ca un ORM (Object Relational Mapping) pentru a beneficia de un nivel de abstractizare peste baza de date SQLite. Baza de date Room interacționează cu restul aplicației cu ajutorul obiectelor de tip DaO (Database Object) pe care aplicația le folosește pentru a realiza interogări specifice bazei de date pe baza entităților. În figura următoare este prezentat modul în care funcționează biblioteca.

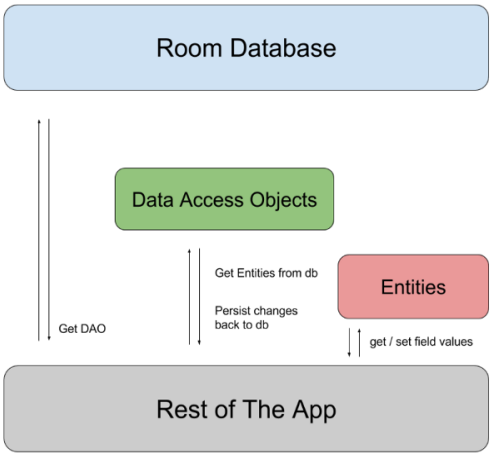


Figura 21: Arhitectura Librăriei Room

Sursa: https://developer.android.com/training/data-storage/room

Astfel, partea dificilă va fi gestionată de către librăria Room, iar sarcina dezvoltatorului este doar să definească metode pentru aceste interogări și să utilizeze adnotările specifice. Când una dintre aceste metode este apelată, se va genera tot codul SQL necesar de către biblioteca Room.

De exemplu, pentru a insera în tabela de legătură AutorCarte se folosește interfața CarteCuAutoriDao, în care există semnătura metodei insert cu adnotarea "@Insert". Atunci când se dorește inserarea unui obiect se va apela această metodă prin intermediul managerului bazei de date, urmând ca Room să genereze codul SQL necesar și să realizeze inserarea.

O altă tehnologie utilizată este platforma Firebase, ce oferă numeroase servicii pentru aplicațiile mobile și web. Partea de authentication a platformei oferă suport utilizatorului pentru crearea și gestionarea de conturi la nivelul aplicației, atât folosind e-mailul și parola cât și diverși furnizori. În dezvoltarea aplicației am utilizat această facilitate pentru a implementa autentificarea pe bază de e-mail și parolă.

Ușurința implementării este unul dintre motivele pentru care aceasta este utilizată de dezvoltatori, întrucât în cazul aplicațiilor Android oferă clase, interfețe și metode specifice pentru crearea și gestiunea utilizatorilor.

4.2. IMPLEMENTAREA APLICATIEI

Primul pas parcurs în implementarea aplicației este crearea proiectului în Android Studio și definirea claselor necesare pentru a determina logica aplicației. În acest sens au fost create clasele Utilizator, Carte, Autor, Împrumut. Pe baza acestora și a schemei bazei de date determinată în etapa de proiectare a sistemului se procedează pentru construirea bazei de date SQLite folosind biblioteca Room. În acest scop a fost necesară crearea claselor AutorCarte și ÎmprumutCarte, care reprezintă tabelele de legătură menționate anterior. Pentru a putea implementa relațiile mulți-mulți între tabelele Cărți și Autori tabelele Împrumuturi și Cărți este necesară crearea unor clase ce vor determina cum arată obiectele și relațiile dintre ele. Astfel, clasa CarteCuAutor va conține o carte ce are asociată o listă de autori aferenți. Același lucru se întâmplă și pentru clasa ImprumutCuCarte, ce conține un împrumut și o listă de cărți asociate acestuia. Alte clase utilitare sunt cele aferente activităților și adaptoarelor. Structura claselor este prezentată în figura din Anexa 3.

Pentru a defini toate resursele necesare afișării pe ecran a componentelor aplicației au fost adăugate în directorul res fișierele XML corespunzătoare activităților, meniurilor, fonturilor și pictogramelor, imaginile și culorile.

Pentru personalizarea modului în care sunt afișate elementele de tip Carte, respectiv Împrumut în cadrul listelor au fost folosite obiectele de tip adaptor, ce permit afișarea pe baza unui layout definit de dezvoltator, în care acesta specifică controalele prin care va fi reprezentat un element din tipul respectiv și modul de așezare al acestora. Afișarea elementelor se face într-un element de tip ListView, iar controalele utilizate pentru afișarea atributelor sunt de tip ImageView, EditText și TextView. Pentru crearea unor adaptoare personalizate a fost necesară crearea unor clase noi ce vor extinde clasa ArrayAdapter<Carte> respectiv ArrayAdapter<Împrumut> în cadrul cărora se suprascrie metoda getView() în care se specifică ce date vor fi afișate. Codul aferent acestora este disponibil în figurile din Anexa 4.

Pentru autentificarea și înregistrarea utilizatorului au fost folosite metode precum signInWithEmailAndPassword() și createUserWithEmailAndPassword() accesate prin intermediul clasei FirebaseAuth, pusă la dispoziție de platforma Firebase. Datele

introduse de utilizator sunt validate cu ajutorul unor expresii regulate înainte de a se face autentificarea conform figurilor din Anexa 5.

Pentru formularul de adăugare a unei cărți a fost folosit un obiect de tip Intent cu metoda startActivityForResult() pentru a deschide galeria cu imaginile de pe dispozitiv, conform figurii din Anexa 6.

Pentru a permite selectarea autorilor disponibili ce vor fi adăugați la carte a fost creată o fereastră de tip dialog AlertDialog folosindu-se metode precum setTitle(), setPositiveButton(), setMultiChoiceItems() de la nivelul clasei interne AlertDialog.Builder conform figurii din Anexa 7. Se preiau autorii selectați și se adaugă în tabela de legătură din baza de date, creându-se legătura între carte și autorii selectați.

Atunci când se finalizează un împrumut se creează un obiect de tip Calendar și cu ajutorul metodei add() se adaugă datei curente 14 zile, reprezentând 2 săptămâni adică termenul unui împrumut. În continuare se salvează obiectul de tip Împrumut creat în baza de date. Înainte de a se apela metoda de generare a fișei de împrumut este necesară cererea permisiunii pentru accesarea și scrierea în spațiul de stocare a dispozitivului. Codul aferent acestor funcționalități este prezent în figurile din Anexa 8.

Utilizatorul are la dispoziție un control de tip SearchView pentru a căuta o carte pe baza titlului. În cadrul acestuia se va apela metoda getCarteCuAutoriByName() prezentă în interfața CarteCuAutoriDao ce va primi ca parametru textul introdus în cadrul controlului. Se va executa interogarea aferentă pentru a aduce toate cărțile cu titlul acela. În figurile din Anexa 9 se observă codul aferent căutării și cum arată interogarea ce va fi realizată de către Room.

Algoritmul de recomandare de cărți pentru utilizator pe baza împrumuturilor anterioare este o funcționalitate de bază a aplicației, deoarece ține cont de istoricul împrumuturilor acestuia și pe baza lui oferă recomandări personalizate de cărți.

Se începe prin interogarea bazei de date pentru a obține toate împrumuturile efectuate de utilizatorul curent. Acestea se grupează în funcție de genul cărților împrumutate și se determină genul de cărți pentru care acesta are cele mai multe împrumuturi efectuate. În cazul în care utilizatorul este un utilizator nou și nu are alte împrumuturi efectuate anterior se va selecta categoria cea mai populară la nivel global al aplicației, de către toți utilizatorii.

În continuare, pentru categoria selectată se aleg doar cărțile care nu au mai fost împrumutate anterior de către utilizatorul respectiv, deoarece nu ar avea sens ca acestuia să îi fie recomandate cărți pe care le-a împrumutat deja. Pentru a face această verificare se parcurge lista cărților selectate din genul cel mai popular determinat anterior și cu ajutorul unei variabile de tip boolean se determină dacă aceasta apare și în împrumuturile anterioare ale utilizatorului sau nu. În cazul în care aceasta nu apare va fi adăugată într-o listă pentru a fi afișată pe ecran. Codul aferent verificării este disponibil în figura din Anexa 10.

```
private void recomandaCartiDupaGen() {
    Map<Gen, List<Carte>> cartiByGenre = new HashMap<>();
    Map<Gen, List<Carte>> cartiByGenreFinal = new HashMap<>();
    for (ImprumutCuCarte i : listaImprumuturiCuCarti) {
        cartiByGenre.putAll(i.listaCartiImprumut.stream()
                .collect(groupingBy(Carte::getGenCarte)));
        cartiByGenre.forEach((k, v) -> {
            if (cartiByGenreFinal.containsKey(k)) {
                List<Carte> cartiExistenteInMap = cartiByGenreFinal.get(k);
                if (cartiExistenteInMap != null) {
                    cartiExistenteInMap.addAll(cartiByGenre.get(k));
                }
                cartiByGenreFinal.remove(k);
                cartiByGenreFinal.put(k, cartiExistenteInMap);
            } else {
                cartiByGenreFinal.put(k, cartiByGenre.get(k));
            }
        });
    }
    AtomicInteger max = new AtomicInteger();
    cartiByGenreFinal.forEach((k, v) -> {
        if (v.size() > max.get()) {
            max.set(v.size());
        }
        if (v.size() == Integer.parseInt(String.valueOf(max))) {
            listaCartiCuAutori = dbInstance.getCarteCuAutoriDao().getCartiByGenre(k);
        }

    });
}
```

Figura 22: Metoda de recomandare de cărți dupa gen

Dacă se dorește efectuarea de recomandări pe baza autorilor care apar cel mai frecvent în împrumuturile utilizatorului, se va itera prin lista de împrumuturi contorizându-se numărul de apariții pentru fiecare autor, salvându-se într-o colecție de tip cheie-valoare de tip LinkedHashMap unde cheia reprezintă numele autorului, iar valoarea reprezintă frecvența de apariție. Pentru a putea fi mai ușor de utilizat, colecția rezultată este sortată descrescător și sunt selectați primii 3 autori, reprezentând cei 3 autori care

apar cel mai frecvent în împrumuturi. În funcție de aceștia se vor selecta din baza de date cărțile care apar pe ecran, realizându-se o interogare pe baza numelui autorului.

```
private void selectareAutoriFavoriti() {
    curataListe();
    preluareCartiImprumutate();
    int contor = 0;
    Map<String, Integer> autoriCuFrecventaAparitiei = new HashMap<>();
    for (CarteCuAutor ca : listaCartiCuAutori) {
        for (Autor a : ca.autori) {
            if (!autoriCuFrecventaAparitiei.containsKey(a.getNume())) {
                contor = 0;
                contor++;
            } else {
                contor++;
                autoriCuFrecventaAparitiei.remove(a.getNume());
            }
            autoriCuFrecventaAparitiei.put(a.getNume(), contor);
        }
    }
    LinkedHashMap<String, Integer> autoriCuFrecventaAparitieiDescrescator = new LinkedHashMap<>();
    autoriCuFrecventaAparitiei.entrySet() Set<Map<K, V>.Entry<String, Integer>>
            .stream() Stream<Map<K, V>.Entry<String, Integer>>
            .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
            .forEachOrdered(x -> autoriCuFrecventaAparitieiDescrescator.put(x.getKey(), x.getValue()));

    LinkedHashMap<String, Integer> finalReverseSortedMap = new LinkedHashMap<>();
    AtomicInteger n = new AtomicInteger();
    autoriCuFrecventaAparitieiDescrescator.forEach((k, v) -> {
                if (n.get() != 2) {
                    finalReverseSortedMap.put(k, v);
                }
                n.getAndIncrement();
            }
    );
    finalReverseSortedMap.forEach((k, v) -> autorCuCarti.addAll(dbInstance.getAutorCuCartiDao().getAutoriCuCartiByName(k)));
}
```

Figura 23: Metoda de selectare a autorilor favoriți

O altă parte importantă a aplicației este partea de generare a fișei de împrumut, funcționalitate implementată folosind clasa PdfDocument, existentă în SDK-ul de Android, clasă ce permite crearea și salvarea de fișiere în format PDF, utilizând un obiect de tip Canvas pentru a scrie conținutul în acestea.

Odată ce utilizatorul apasă butonul de finalizare al împrumutului se va crea documentul de tip PDF, în care se vor scrie numele membrului, data împrumutului și data de returnare, dar și titlurile cărților pe care acesta le-a împrumutat. Cu ajutorul unor obiecte de tip Paint, cărora le sunt setate caracteristici precum dimensiunea, culoarea, alinierea și coordonatele sunt scrise în cadrul obiectului de tip Canvas datele ce apar ulterior în document.

Pentru a formata data împrumutului și data returnării într-un format mai intuitiv de citit se folosesc obiecte de tip LocalDate cu metodele getYear(), getMonthValue() și getDayOfMonth() și se scrie data în noul format pe canvas-ul aferent documentului.

Documentul se va salva în cadrul mediului extern de stocare al dispozitivului cu ajutorul metodei writeTo() aferentă clasei PdfDocument care primește ca parametru calea către directorul de stocare externă și i se va asigna un nume ce conține data curentă și numele membrului. Codul aferent creării fișierului PDF este disponibil în figura din Anexa 11.

În cadrul profilului utilizatorul își poate schimba datele personale utilizând atât funcționalitățile aduse de Firebase cât și realizând o interogare de tip update în baza de date relațională.

În ceea ce privește poza de profil, aceasta este afișată într-un control de tip CircleImageView. Apăsarea pe acesta duce la deschiderea galeriei folosind un obiect de tip Intent cu ACTION_OPEN_DOCUMENT. După alegerea unei fotografii aceasta este schimbată utilizând metoda updateProfile() din cadrul obiectului FirebaseUser

Totodată acesta își poate schimba numele, e-mailul, parola, numărul de telefon sau adresa. Numele, e-mailul și parola vor fi schimbate și în cadrul platformei Firebase prin metodele updateEmail() și updatePassword(). Toate datele utilizatorului vor fi actualizate și în cadrul bazei de date prin metoda update().

Ștergerea contului se realizează apelând metodele de tip delete() atât în cadrul Firebase cât și în cadrul Room. Codul aferent actualizării datelor utilizatorului este disponibil în figurile din Anexa 12.

4.3. PREZENTAREA APLICAȚIEI

Aplicația începe cu pagina principală de introducere, care are un buton ce duce spre pagina de login unde utilizatorul poate intra în cont sau își poate crea unul nou. În figurile următoare este prezentat comportamentul aplicației atunci când se încearcă accesarea fără a introduce e-mailul și parola, precum și fereastra de înregistrare a unui utilizator nou.

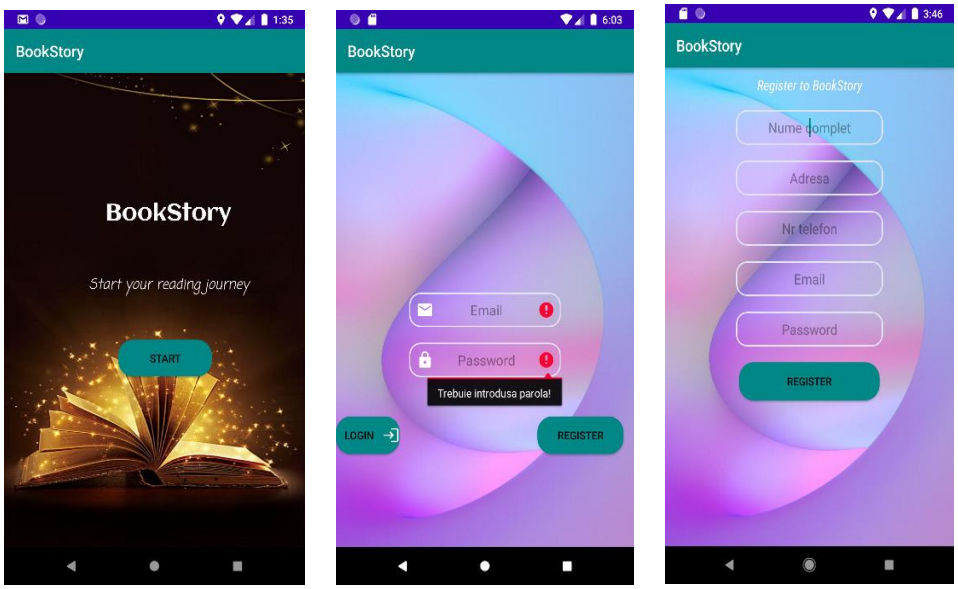


Figura 24: Interfețe autentificare și înregistrare utilizator

Dacă datele utilizatorului sunt introduse corect se va trece la următoarea fereastră. În acest moment, un membru obișnuit are la dispoziție un meniu principal din care poate alege ce dorește să facă în continuare.

Atunci când dorește să-și vizualizeze profilul de utilizator sau să schimbe anumite detalii din acesta, acesta va apăsa în continuare pe opțiunea “Profil “. Se va deschide o fereastră ce afișează poza de profil, numele, telefonul, adresa, e-mailul și parola utilizatorului oferindu-i posibilitatea să le schimbe. Tot aici utilizatorul are opțiunea să-și șteargă contul definitiv.

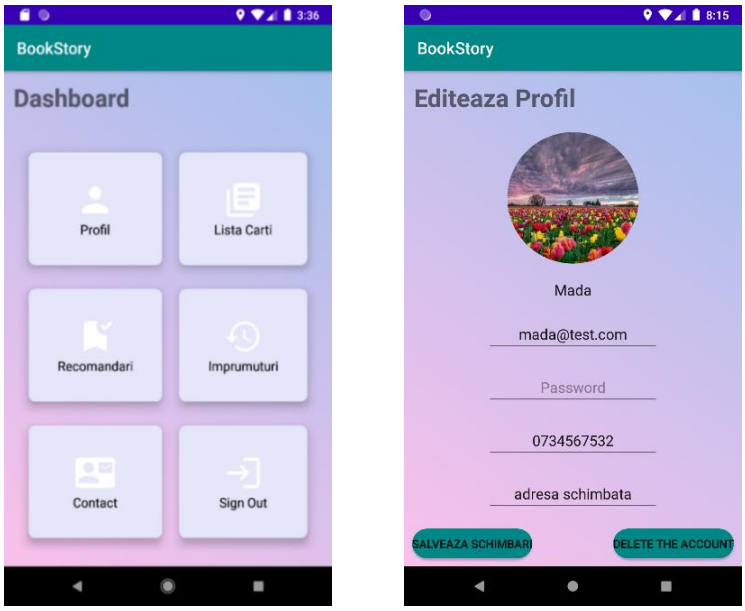


Figura 25: Interfețe meniu principal si profil utilizator

Atunci când utilizatorul dorește să vizualizeze lista cu toate cărțile disponibile pentru împrumut, acesta va accesa opțiunea “Listă Cărți”. Aceasta va deschide o fereastră în care sunt prezentate cărțile, cu opțiunea de căutare după titlul cărții. Acesta poate selecta oricâte cărți dorește pentru împrumut și apoi, când dorește să-l finalizeze va apăsa butonul din colțul din dreapta jos pentru a finaliza împrumutul. Aplicația va genera un document în format PDF, reprezentând fișa împrumutului.

Atunci când utilizatorul dorește totuși să vizualizeze recomandările oferite de aplicație în materie de cărți și bazate pe împrumuturile anterioare, acesta va accesa opțiunea “Recomandări”. Aceasta îi va deschide o fereastră cu cărțile recomandate de aplicație, pe baza algoritmului descris anterior, pe care poate alege să le împrumute în continuare.

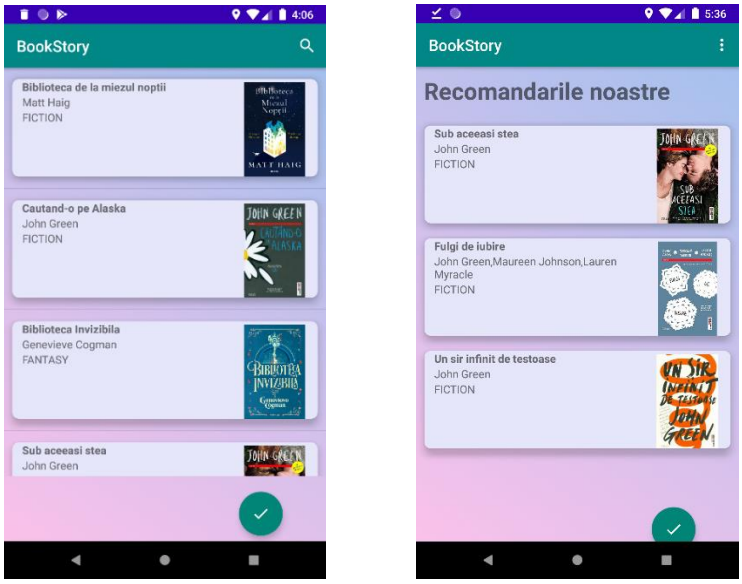


Figura 26: Interfețe listare cărți și recomandare de cărți

Pentru a vedea istoricul împrumuturilor efectuate, utilizatorul are la dispoziție opțiunea "Împrumuturi" din cadrul meniului principal. În urma accesării acesteia va apărea o fereastră cu toate împrumuturile efectuate de utilizator împreună cu titlurile cărților împrumutate.

Pentru a putea contacta dezvoltatorul aplicației prin intermediul poștei electronice există opțiunea "Contact" din cadrul meniului principal, ce va deschide o fereastră în care utilizatorul poate introduce subiectul și mesajul pe care dorește să-l transmită. La apăsarea butonului de trimitere, i se va cere să aleagă aplicația pentru poștă electronică pe care dorește să o utilizeze, urmând că aceasta să se deschidă cu datele completate anterior.

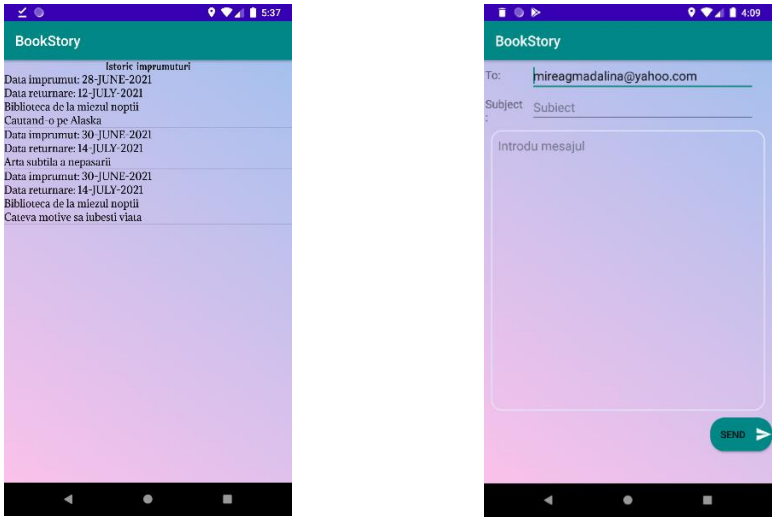


Figura 27: Interfețe vizualizare istoric împrumuturi și contact

Dacă atunci când se realizează autentificarea utilizatorul nu este un membru obișnuit, ci administratorul acesta are la dispoziție propriul meniu principal pentru a interacționa cu aplicația.

Atunci când dorește să acceseze lista de cărți disponibile în baza de date, va accesa opțiunea "Listă Cărți", ce îi va deschide o fereastră cu toate cărțile disponibile.

Tot aici are posibilitatea de a adăuga o carte nouă prin butonul disponibil în cadrul ferestrei. Apăsarea acestuia va deschide formularul de adăugare pentru o carte nouă. După introducerea datelor are posibilitatea adăugării unei imagini de copertă și posibilitatea de a salva cartea.

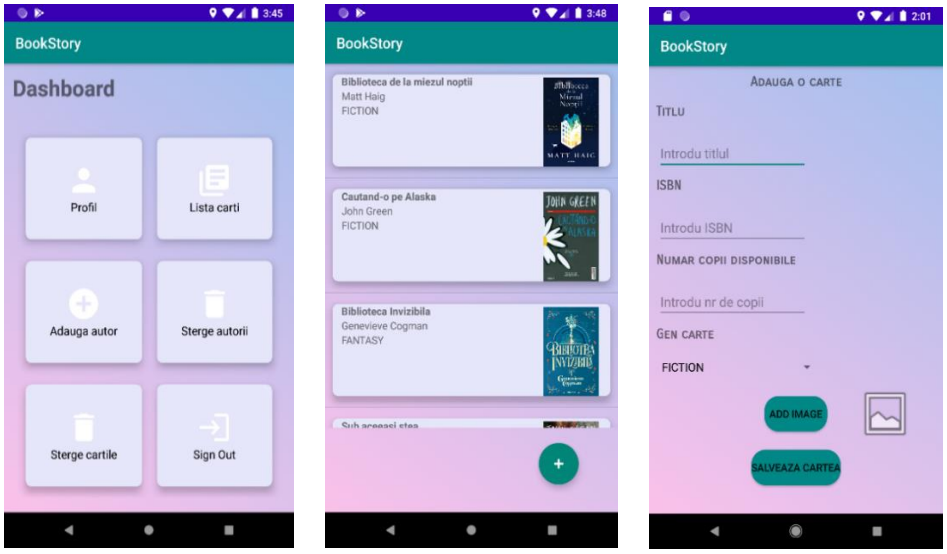


Figura 28: Interfețe meniu administrator, listare cărți și adăugare carte

Odată întors în fereastra cu lista de cărți, administratorul are posibilitatea de a selecta cartea nou introdusă și a alege să îi adauge autori disponibili în baza de date. Apăsarea opțiunii va duce la deschiderea unei ferestre de dialog din care acesta poate selecta mai mulți autori sau poate adăuga unul nou, deschizând formularul de adăugare a unui autor nou.

Pe lângă adăugarea de autori, acesta mai are opțiunea de a edita sau șterge cartea selectată.

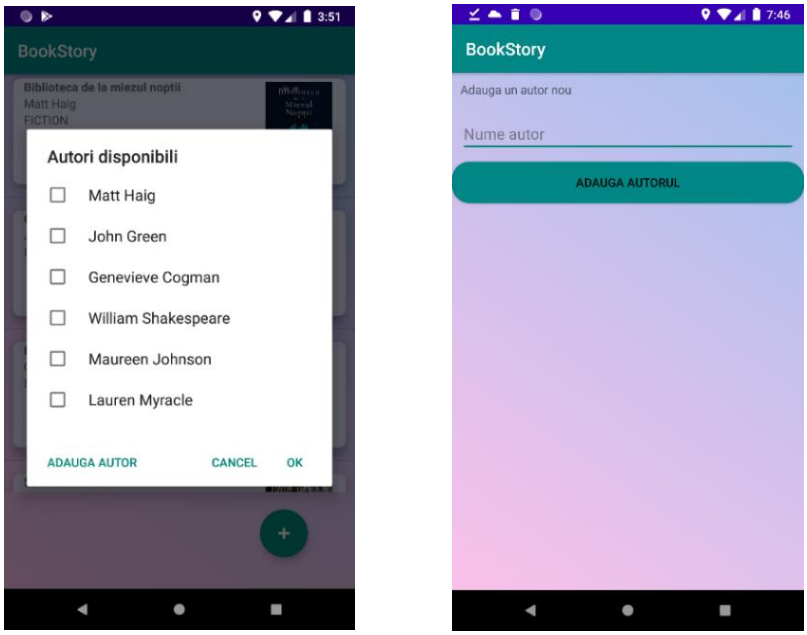


Figura 29: Interfețe adăugare autori

Deși aplicația are disponibile funcționalitățile de bază, menționez o serie de îmbunătățiri ce ar putea fi aduse pe viitor pentru a o face mai atractivă. O astfel de îmbunătățire ce ar putea fi adusă ar fi crearea de mai multe rapoarte pentru utilizator pentru a-și putea vedea obiceiurile, de exemplu grafice cu numărul cărților împrumutate în funcție de gen, de autori și diverse alte statistici referitoare la împrumuturile anterioare.

Alte îmbunătățiri posibile ar fi introducerea posibilității de a-și personaliza experiența adăugând notițe proprii legate de cărțile citite, primirea de notificări atunci când se apropie termenul de restituire al unui împrumut, opțiunea de a cere o prelungire a termenului direct din aplicație, posibilitatea acordării unui rating fiecărei cărți, adăugarea unei secțiuni în care utilizatorii își pot comunica opiniile cu privire la cartea respectivă, afișarea cărților sortate după un anumit criteriu, posibilitatea adăugării cărții într-o listă de dorințe în cazul în care nu este disponibilă.

CONCLUZIE

Aplicația rezultată cu numele Bookstory îndeplinește funcționalitățile de bază descrise la începutul lucrării, oferind utilizatorului posibilitatea de a accesa lista de cărți disponibilă și de a împrumuta cărți și generând o fișă de împrumut în format PDF ca dovadă a efectuării acestuia. Utilizatorul are și posibilitatea de a căuta cartea dorită pe baza titlului sau a unor cuvinte cheie din titlu

De asemenea, aplicația prelucrează rapoartele obținute în urma interacțiunii cu utilizatorul, oferindu-i și recomandări personalizate pe baza împrumuturilor anterioare sau, dacă acestea nu există, pe baza popularității cărților în rândul celorlalți utilizatori.

Opțiunea de contact din cadrul aplicației permite transmiterea de feedback de către utilizatori către dezvoltator, astfel putând fi identificate mai bine eventualele probleme dar și diverse posibilități de îmbunătățire a acesteia.

În prezent aplicația este disponibilă doar pe telefon și este adresată utilizatorului ce dorește să efectueze împrumutul, partea de administrare fiind implementată ca să faciliteze testarea funcționalităților. O posibilă extindere a acestui sistem ar putea fi realizarea unei aplicații separate pentru funcționalitățile realizate de către administrator astfel încât să poată fi cu adevărat utilă și acestora, întrucât realizarea lor pe telefonul mobil nu este o soluție tocmai practică. Astfel, partea de administrare a bibliotecii ar putea fi implementată complet și utilizată ca o aplicație web, iar cea de împrumut ar fi utilizată pe telefon, cum este și în prezent.

În concluzie, aplicația Bookstory atinge scopul principal al acestei lucrări, oferind utilizatorului o interfață intuitivă și prietenoasă pentru a împrumuta cărți din confortul propriei locuințe.

BIBLIOGRAFIE

[1] - https://www.romania-insider.com/eurostat-reading-2019, 2019, Articol statistici lectură
[Accesat la 11 Ianuarie 2021]
[2] - https://culturaladuba.ro/1-din-5-romani-nu-a-citit-niciodata-o-carte-timpulsacitim-o-campanie-a-asociatiei-curtea-veche/, 2019, Campanie încurajare lectură
[Accesat la 13 Ianuarie 2021]
[3] - https://www.culturadata.ro/barometrul-de-consum-cultural-2019-experienta-si-practicile-culturale-de-timp-liber/, Barometrul de consum cultural 2019
[Accesat la 11 Ianuarie 2021]
[4] - http://statistici.insse.ro:8077/tempo-online/#/pages/tables/insse-table, Numărul de biblioteci în România
[Accesat la 13 Ianuarie 2021]
[5] - https://www.zf.ro/companii/zece-edituri-din-romania-s-au-aliat-si-au-chemat-in-judecata-biblioteca-pentru-corporatisti-bookster-pentru-concurenta-neloiala-18824728, 2020, Articol acuzații aduse Bookster
[Accesat la 12 Ianuarie 2021]
[6] - https://www.casesoftware.ro/index.php/aplicatii/biblio-expert-online#caracteristici, Fără an, Aplicația Biblio-Expert
[Accesat la 12 Ianuarie 2021]
[7] - http://www.vitainternational.media/en/article/2015/11/18/booxup-the-app-for-printed-book-sharing/60/, 2015, Articol aplicația Booxup
[Accesat la 12 Ianuarie 2021]
[8] - https://www.wiz-soft.ro/wizbooks.htm. Fără an, Aplicația WizBooks
[Accesat la 12 Ianuarie 2021]
[9] - https://gs.statcounter.com/os-market-share/mobile-tablet/romania/#monthly-202005-202104-bar, Fără an, Cota de piață a sistemelor de operare mobile Romania
[Accesat la 13 Iunie 2021]
[10] - https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202105, Fără an, Cota de piață a sistemelor de operare mobile la nivel mondial
[Accesat la 13 Iunie 2021]
[11] - Tian Lou - A comparison of Android Native App Architecture: MVC, MVP and MVVM, 2016, Disponibil la:
https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf
[Accesat la 14 Iunie 2021]
[12] - https://developer.android.com/about/versions/oreo/android-8.0, Fără an, Caracteristici Android Oreo
[Accesat la 15 Iunie 2021]
[13] - https://www.tiobe.com/tiobe-index/, 2021, Indicatorul de popularitate al limbajelor de programare
[Accesat la 13 Iunie 2021]

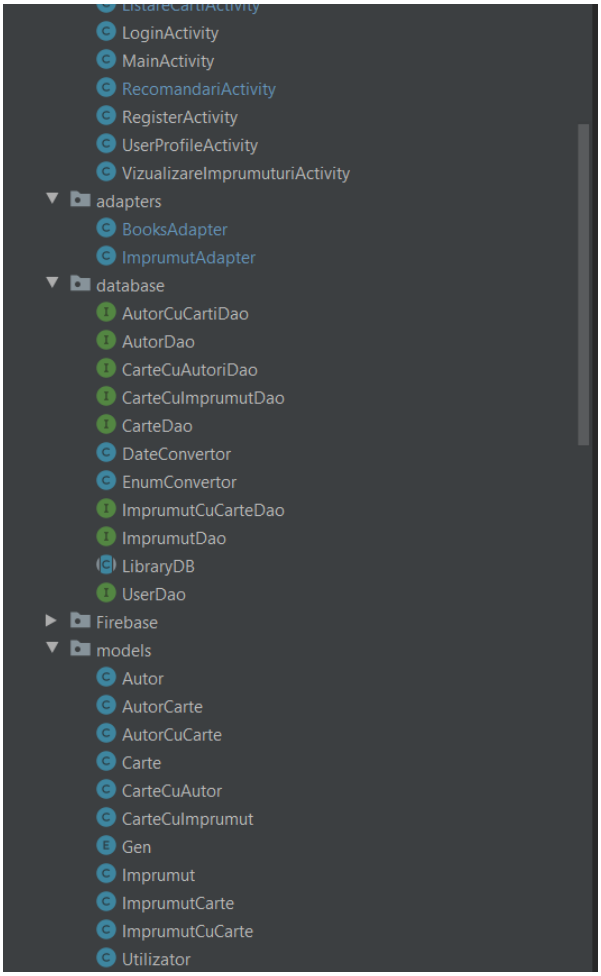
ANEXE

ANEXA 1 – LISTA FIGURILOR

Lista Figurilor

ANEXA 2 – LISTA TABELELOR

ANEXA 3 - CLASELE AUXILIARE CREATE PENTRU BAZA DE DATE ȘI ADAPTOARELE



ANEXA 4 – CODUL SURSĂ AFERENT ADAPTOARELOR PENTRU CĂRȚI RESPECTIV ÎMPRUMUTURI

```java
public class BooksAdapter extends ArrayAdapter<Carte> {

    private LayoutInflater layoutInflater;
    private Context context;
    private int resource;
    private List<Carte> booksList;

    public BooksAdapter(@NonNull Context context, int resource, List<Carte> booksList, LayoutInflater layoutInflater) {
        super(context, resource, booksList);
        this.context = context;
        this.resource = resource;
        this.layoutInflater = layoutInflater;
        this.booksList = booksList;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        View view = layoutInflater.inflate(resource, parent,  attachToRoot: false);
        Carte carte = booksList.get(position);
        if (carte != null) {
            TextView tvTitlu = view.findViewById(R.id.titlu);
            tvTitlu.setText(carte.getTitlu());
            TextView tvGen = view.findViewById(R.id.gen);
            tvGen.setText(String.valueOf(carte.getGenCarte()));
            ImageView cover = view.findViewById(R.id.ivCoperta);
            cover.setImageURI(Uri.parse(carte.getCopertaURI()));
        }
        return view;
    }
}
```

```java
public class ImprumutAdapter extends ArrayAdapter<Imprumut> {
    private List<Imprumut> imprumuturi;
    private LayoutInflater layoutInflater;
    private Context context;
    private int resource;

    public ImprumutAdapter(Context context, int resource, List<Imprumut> imprumuturi, LayoutInflater layoutInflater) {
        super(context, resource, imprumuturi);
        this.layoutInflater = layoutInflater;
        this.imprumuturi = imprumuturi;
        this.context = context;
        this.resource = resource;
    }

    @{...}
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        View view = layoutInflater.inflate(resource, parent,  attachToRoot: false);
        Imprumut imprumut = imprumuturi.get(position);
        if (imprumut != null) {
            TextView tvDataImprumut = view.findViewById(R.id.tvDataImprumut);
            TextView tvDataScadenta = view.findViewById(R.id.tvDataScadenta);
            LocalDate localDateBorrow;
            if (imprumut.getDataImprumut() != null) {
                localDateBorrow = imprumut.getDataImprumut().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
                int yearBorrow = localDateBorrow.getYear();
                String monthBorrow = localDateBorrow.getMonth().toString();
                int dayBorrow = localDateBorrow.getDayOfMonth();
                tvDataImprumut.setText(String.format("Data imprumut: %d-%s-%d", dayBorrow, monthBorrow, yearBorrow));
            }
            if (imprumut.getDataScadenta() != null) {
                LocalDate localDateReturn = imprumut.getDataScadenta().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
                int yearReturn = localDateReturn.getYear();
                String monthReturn = localDateReturn.getMonth().toString();
                int dayReturn = localDateReturn.getDayOfMonth();
                tvDataScadenta.setText(String.format("Data returnare: %d-%s-%d", dayReturn, monthReturn, yearReturn));
            }
        }
        return view;
    }
```

ANEXA 5 – VALIDAREA E-MAILULUI ȘI PAROLEI ȘI AUTENTIFICAREA UTILIZATORULUI

```
private void verificaCredentiale(String email, String parola) {
    String emailRegex = "^[A-Za-z0-9+_.-]+@(.+)$";

    if (email.isEmpty()) {
        etEmail.setError("Trebuie sa introduceti adresa de email!");
        isValidEmail = false;
    } else {
        isValidEmail = true;
    }
    if (!email.matches(emailRegex)) {
        etEmail.setError("Format email invalid!");
        isValidEmail = false;
    } else {
        isValidEmail = true;
    }
    if (parola.isEmpty()) {
        etPassword.setError("Trebuie introdusa parola!");
        isValidPassword = false;
    } else {
        isValidPassword = true;
    }
}
```

```
private void loginUser() {
    verificaCredentiale(etEmail.getText().toString(), etPassword.getText().toString());
    if (isValidEmail && isValidPassword) {
        auth.signInWithEmailAndPassword(etEmail.getText().toString(), etPassword.getText()
                .toString()).addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                Toast.makeText(getApplicationContext(), "Login successful!", Toast.LENGTH_LONG).show();
                if (ADMIN_EMAIL.equals(etEmail.getText().toString()) && ADMIN_PASS.equals(etPassword.getText().toString())) {
                    Intent intent = new Intent(getApplicationContext(), AdminDashboardActivity.class);
                    startActivity(intent);
                } else {
                    Intent intent = new Intent(getApplicationContext(), DashboardActivity.class);
                    startActivity(intent);
                }
            } else {
                Toast.makeText(getApplicationContext(), "Login failed!", Toast.LENGTH_LONG).show();
                Snackbar.make(layout, task.getException().getLocalizedMessage(), Snackbar.LENGTH_LONG).show();
            }
        }).addOnFailureListener(e -> Snackbar.make(layout, e.getMessage(), Snackbar.LENGTH_LONG).show());
    } else {
        Toast.makeText(getApplicationContext(), "Date de conectare invalide! Introduceti datele corecte", Toast.LENGTH_LONG).show();
    }

}
```

ANEXA 6 - DESCHIDERE GALERIE DE IMAGINI ȘI SETAREA IMAGINII DE COPERTĂ

```
    btnAddImagine.setOnClickListener(v -> {
        Intent gallery = new Intent(Intent.ACTION_OPEN_DOCUMENT, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(gallery, GALLERY_REQUEST_CODE);
    });
}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == GALLERY_REQUEST_CODE && resultCode == Activity.RESULT_OK) {
        if (data != null) {
            Uri contentUri = data.getData();
            this.bookCoverUri = contentUri;
            this.imageView.setImageURI(contentUri);
        }
    }
}
```

ANEXA 7 - CREARE FEREASTRĂ DIALOG CU AUTORII DISPONIBILI ÎN BAZA DE DATE

```
        break;
    case R.id.ctxaddautor:
        poz = info.position;
        AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
        builder.setTitle(R.string.autori_disponibili);
        builder.setMultiChoiceItems(authorNames, checkedAuthors, (dialog, which, isChecked) -> checkedAuthors[which] = isChecked);
        builder.setPositiveButton(R.string.ok, (dialog, which) -> {
            Toast.makeText(getApplicationContext(), R.string.mesaj_dialog_autori_ok, Toast.LENGTH_LONG).show();
            for (int i = 0; i < checkedAuthors.length; i++) {
                boolean checked = checkedAuthors[i];
                if (checked) {
                    AutorCarte ac = new AutorCarte(authorIds[i], adapter.getItem(info.position).getIdCarte());
                    db.getCarteCuAutoriDao().insert(ac);
                    updateUI();
                }
            }
        });
        builder.setNeutralButton(R.string.adauga_autor, (dialog, which) -> {
            Intent intent = new Intent(getApplicationContext(), AddAuthorActivity.class);
            startActivityForResult(intent, REQUEST_CODE_ADD_AUTOR);
        });
        builder.setNegativeButton(R.string.cancel, (dialog, which) -> Toast.makeText(getApplicationContext(),
                R.string.mesaj_dialog_cancel, Toast.LENGTH_LONG).show());
        AlertDialog newDialog = builder.create();
        newDialog.show();
        break;
```

ANEXA 8 - COD SURSĂ FINALIZARE ÎMPRUMUT ȘI CERERE PERMISIUNI DE ACCESARE A SPAȚIULUI DE STOCARE

```
btnFinalizeaza.setOnClickListener(v -> {
    AlertDialog dialog = new AlertDialog.Builder( context: ListaCartiUserActivity.this)
            .setTitle("Confirmare finalizare")
            .setMessage("Doriti sa finalizati imprumutul?")
            .setNegativeButton( text: "Nu", (dialogInterface, which) -> {
                anuleazaImprumut();
                dialogInterface.cancel();
            })
            .setPositiveButton( text: "Da", (dialogInterface, which) -> {
                if (auth.getCurrentUser() != null) {
                    Utilizator utilizator = db.getUserDao().getUserByUid(auth.getCurrentUser().getUid());
                    Calendar calendar = Calendar.getInstance();
                    calendar.setTime(new Date());
                    calendar.add(Calendar.DATE,  amount: 14);
                    Date data = new Date();
                    Imprumut imprumut = new Imprumut(utilizator.getId(), data, calendar.getTime(),  taxaIntarziere: 0);
                    imprumut.setIdImprumut(db.getImprumutDao().insert(imprumut));
                    for (Carte c : cartiImprumutate) {
                        ImprumutCarte imprumutCarte = new ImprumutCarte(imprumut.getIdImprumut(), c.getIdCarte());
                        db.getImprumutCuCarteDao().insert(imprumutCarte);
                    }
                    if (!checkPermission()) {
                        requestPermission();
                    }
                    ImprumutCuCarte ic = db.getImprumutCuCarteDao().getImprumutcuCartiByImprumutId(imprumut.getIdImprumut());
                    genereazaFisaImprumut(imprumut, ic);
                    Toast.makeText(getApplicationContext(), "Imprumut finalizat!", Toast.LENGTH_LONG).show();
                    dialogInterface.cancel();
                }
            }).create();
    dialog.show();
});
```

```
private boolean checkPermission() {
    int permission1 = ContextCompat.checkSelfPermission(getApplicationContext(), WRITE_EXTERNAL_STORAGE);
    int permission2 = ContextCompat.checkSelfPermission(getApplicationContext(), READ_EXTERNAL_STORAGE);
    return permission1 == PackageManager.PERMISSION_GRANTED && permission2 == PackageManager.PERMISSION_GRANTED;
}

private void requestPermission() {
    ActivityCompat.requestPermissions( activity: this, new String[]{WRITE_EXTERNAL_STORAGE, READ_EXTERNAL_STORAGE}, PERMISSION_REQUEST_CODE);
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    if (requestCode == PERMISSION_REQUEST_CODE) {
        if (grantResults.length > 0) {
            boolean writeStorage = grantResults[0] == PackageManager.PERMISSION_GRANTED;
            boolean readStorage = grantResults[1] == PackageManager.PERMISSION_GRANTED;
            if (writeStorage && readStorage) {
                Toast.makeText( context: this, R.string.permission_granted, Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText( context: this, R.string.permission_denied, Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }
}
```

ANEXA 9 - COD SURSĂ CĂUTARE CARTE DUPĂ TITLU ÎN BAZA DE DATE

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options_menu, menu);
    MenuItem item = menu.findItem(R.id.action_search);
    SearchView searchView = (SearchView) item.getActionView();
    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextSubmit(String query) {
            carteCuAutorList.clear();
            carti.clear();
            carteCuAutorList = db.getCarteCuAutoriDao().getCarteCuAutoriByName(searchView.getQuery().toString());
            listareCarti();
            return true;
        }

        @Override
        public boolean onQueryTextChange(String newText) { return false; }
    });
    return super.onCreateOptionsMenu(menu);
}
```

```java
public interface CarteCuAutoriDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    long insert(AutorCarte ca);

    @Transaction
    @Query("SELECT * FROM carti")
    List<CarteCuAutor> getCarteCuAutori();

    @Transaction
    @Query("SELECT * FROM carti where idCarte=:idCarte")
    List<CarteCuAutor> getCarteCuAutoriById(long idCarte);

    @Transaction
    @Query("SELECT * FROM carti where titlu like '%' || :title || '%'")
    List<CarteCuAutor> getCarteCuAutoriByName(String title);

    @Query("DELETE from AutorCarte where idCarte = :id")
    void deleteBookById(long id);

    @Query("select * from carti where genCarte=:gen")
    List<CarteCuAutor> getCartiByGenre(Gen gen);
```

ANEXA 10 - COD SURSĂ AFERENT VERIFICĂRII EXISTENȚEI UNUI ÎMPRUMUT ANTERIOR

```java
private void verificaImprumutAnteriorCarti() {
    boolean found;
    for (CarteCuAutor ca : listaCartiCuAutori) {
        found = false;
        for (ImprumutCuCarte ic : listaImprumuturiCuCarti) {
            for (Carte c : ic.listaCartiImprumut) {
                if (ca.carte.getIdCarte() == c.getIdCarte()) {
                    found = true;
                    break;
                }
            }
        }
        if (areImprumuturiAnterioare) {
            if (!found) {
                listaCartiDeAfisat.add(ca.carte);
                cartiDePastrat.add(ca);
            }
        } else {
            listaCartiDeAfisat.add(ca.carte);
            cartiDePastrat.add(ca);
        }
    }
}
```

ANEXA 11 - CODUL SURSĂ AFERENT CREĂRII FIȘEI DE ÎMPRUMUT CA DOCUMENT PDF

```
private void genereazaFisaImprumut(Imprumut imprumut, ImprumutCuCarte ic) {
    DisplayMetrics displayMetrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
    int height = displayMetrics.heightPixels;
    int width = displayMetrics.widthPixels;
    PdfDocument document = new PdfDocument();
    PdfDocument.PageInfo pageInfo = new PdfDocument.PageInfo.Builder(width, height,  pageNumber: 1).create();
    PdfDocument.Page page = document.startPage(pageInfo);
    Canvas canvas = page.getCanvas();

    Paint title = new Paint();
    title.setTypeface(Typeface.create(Typeface.DEFAULT, Typeface.NORMAL));
    title.setTextSize(31);
    title.setColor(ContextCompat.getColor( context: this, R.color.black));
    title.setTextAlign(Paint.Align.CENTER);
    Paint paint = new Paint();
    paint.setTextSize(28);
    paint.setTextAlign(Paint.Align.LEFT);

    int y = 100;
    int x = 460;
    LocalDate localDate = imprumut.getDataImprumut().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
    int year = localDate.getYear();
    int month = localDate.getMonthValue();
    int day = localDate.getDayOfMonth();
    canvas.drawText( text: "Fisa imprumut la data de " + day + "-" + month + "-" + year, x, y, title);
    x = 20;
    y += 100;
    String numeMembru = null;
    if (auth.getCurrentUser() != null) {
        numeMembru = auth.getCurrentUser().getDisplayName();
    }
    canvas.drawText( text: "Nume membru: " + numeMembru, x, y, paint);
    y += 50;
    canvas.drawText( text: "Au fost rezervate pentru imprumut urmatoarele listaCartiDeAfisat:" + System.lineSeparator(), x, y, paint);
    for (Carte c : ic.listaCartiImprumut) {
        y += 50;
        canvas.drawText( text: c.getTitlu() + System.lineSeparator(), x, y, paint);
    }
    y += 50;
    LocalDate localDate2 = imprumut.getDataScadenta().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
    int year2 = localDate2.getYear();
    int month2 = localDate2.getMonthValue();
    int day2 = localDate2.getDayOfMonth();
    canvas.drawText( text: "Data returnarii este: " + day2 + "-" + month2 + "-" + year2, x, y, paint);
    String mesajTaxa = "In cazul depasirii termenului se va percepe o taxa stabilita in momentul predarii";
    canvas.drawText( text: mesajTaxa + System.lineSeparator(), x,  y: y + 50, paint);
    document.finishPage(page);
    try {
        String numePDF = "Imprumut" + day + "-" + month + "-" + year + ".pdf";
        File f = new File(Environment.getExternalStorageDirectory(), numePDF);
        document.writeTo(new FileOutputStream(f));
        Toast.makeText( context: this,  text: "Done", Toast.LENGTH_LONG).show();
    } catch (IOException e) {
        Log.e( tag: "main",  msg: "error " + e.toString());
        Toast.makeText( context: this, e.toString(), Toast.LENGTH_LONG).show();
    }
    document.close();
```

ANEXA 12: CODUL SURSĂ AFERENT ACTUALIZĂRII DATELOR PERSONALE ALE UTILIZATORULUI

```
    btnSaveChanges.setOnClickListener(v -> {
        if (!etChangeEmail.getText().toString().isEmpty()) {
            firebaseUser.updateEmail(etChangeEmail.getText().toString()).addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    utilizator.setEmail(etChangeEmail.getText().toString());
                }
            });
        }
        if (!etChangePassword.getText().toString().isEmpty()) {
            firebaseUser.updatePassword(etChangePassword.getText().toString()).addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    utilizator.setPassword(etChangePassword.getText().toString());
                }
            });
        }
        if (!etChangePhone.getText().toString().isEmpty()) {
            utilizator.setNrTelefon(etChangePhone.getText().toString());
        }
        if (!etChangeAdresa.getText().toString().isEmpty()) {
            utilizator.setAdresa(etChangeAdresa.getText().toString());
        }
        db.getUserDao().update(utilizator);
        UserProfileChangeRequest profileChangeRequest = new UserProfileChangeRequest.Builder()
                .setDisplayName(etDisplayName.getText().toString())
                .build();
        firebaseUser.updateProfile(profileChangeRequest)
                .addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        etDisplayName.setText(firebaseUser.getDisplayName());
                        utilizator.setNumeComplet(etDisplayName.getText().toString());
                    }
                });
        Toast.makeText(getApplicationContext(),  text: "Profile updated", Toast.LENGTH_LONG).show();
```

```
    btnDeleteAccount.setOnClickListener(v -> {
        AlertDialog alertDialog = new AlertDialog.Builder( context: UserProfileActivity.this)
                .setTitle("Confirmare stergere")
                .setMessage("Doriti sa stergeti contul?")
                .setPositiveButton( text: "Da", (dialog, which) ->
                        stergeContUtilizator())
                .setNegativeButton( text: "Nu", (dialog, which) -> dialog.cancel()).create();
        alertDialog.show();
    });

    circleImageView.setOnClickListener(v -> {
        Intent gallery = new Intent(Intent.ACTION_OPEN_DOCUMENT, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(gallery, GALLERY_REQUEST_CODE_PROFILE);
    });
}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == GALLERY_REQUEST_CODE_PROFILE && resultCode == Activity.RESULT_OK) {
        if (data != null && data.getData() != null) {
            profilePic = data.getData();
        }
        UserProfileChangeRequest profileChangeRequest = new UserProfileChangeRequest.Builder()
                .setPhotoUri(profilePic)
                .build();
        firebaseUser.updateProfile(profileChangeRequest)
                .addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        circleImageView.setImageURI(profilePic);
                    }
                });
    }
```