

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

ExploreBooks

propusă de

Alexandru Corfu

Sesiunea: Iulie, 2018

Coordonator științific

Drd. Colab. Florin Olariu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ

ExploreBooks

Alexandru Corfu

Sesiunea: *Iulie, 2018*

Coordonator științific

Drd. Colab. Florin Olariu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,

absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de

..... specializarea,

promoția, declar pe propria răspundere, cunoscând consecințele

falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației

Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu

titlul: _____ elaborată sub

îndrumarea dl. / d-na _____, pe

care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său

în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „ExploreBooks”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent Alexandru Corfu

Cuprins

Introducere	6
1. Motivație	6
1.1. Alegerea temei	6
1.2. Alegerea tehnologiilor	7
2. Context	7
2.1. Aplicații similare	8
3. Cerințe funcționale	9
Contribuții.....	10
Capitolul I – Proiectarea aplicației	11
1. Structurarea aplicației.....	11
2. Modelarea bazei de date	13
Capitolul II - Detaliile de implementare	16
1. Server	16
1.1. Realizarea conexiunii cu baza de date.....	16
1.2. Popularea bazei de date	17
1.3. Service layer	18
1.4. Business Layer.....	20
2. Client	25
2.1. Controller	25
2.2. Views	27
Capitolul III – Manual de utilizare	33
Concluzii	44
1. Direcții de viitor	45
2. Concluzie finală.....	45
Bibliografie.....	46
Anexa 1 – Tehnologiile utilizate	48

Introducere

Literatura, ca orice altă formă de artă, a fost afectată odată cu dezvoltarea internetului.

Astfel, a apărut o alternativă a cărților tipărite – și anume, cele în format digital, fiind o soluție mai eficientă de stocare și acces a acestora.

1. Motivație

1.1. Alegerea temei

De multe ori simțeam nevoia de a citi o carte, fie pentru trecerea timpului sau pentru dezvoltarea personală, iar datorită accesibilității oferite de internet, nefiind nevoit să caut respectivele cărți la librării/biblioteci, acest lucru părea ușor de realizat. Din nefericire, pot apărea și obstacole în acest sens, cele mai frecvent întâlnite au fost:

- nu exista o variantă completă a cărții în format digital, astfel, singura opțiune viabilă era să achiziționez respectiva carte
- nu aveam la dispoziție semne de carte, astfel, riscam să pierd progresul realizat până atunci (sau să fiu nevoit să parcurg pagini bune până să-mi amintesc unde am rămas)

Prin aplicația „ExploreBooks” mi-am propus să ofer posibilitatea oricărui cititor cu acces la internet un mediu plăcut și facil pentru a gestiona, citi și exprima opinia în legătură cu cărțile citite.

„ExploreBooks” se focusează pe partea de comunicare între cititori. Activitatea comunității formate se desfășoară în două planuri: unul direct (prin postări de recomandări ale cărților pe diverse criterii, review-uri ale acestora, mecanismul de following/followers etc.) și celălalt indirect (aici fiind vorba de activitatea recentă a fiecărui user, influențând astfel recomandările automate – generate de aplicație).

Aplicația, prin intermediul sistemelor de statistici și review-uri, oferă cititorului posibilitatea de a dobândi o experiență cât mai completă cu privire la cartea citită, acestea fiind realizate ca urmare a activităților directe sau indirecte ale comunității.

1.2. Alegerea tehnologiilor

Motivația pe partea tehnică se datorează curiozității pe care o am când vine vorba de framework-ul .NET. Aplicația a fost realizată în **.NET Core 2.0**, având la dispoziție o serie de posibilități când vine vorba de comunicarea cu baza de date (prin intermediul **Entity Framework-ului**) sau a modului de organizare a aplicației. Pe partea de aplicație web, am ales o aplicație de tip **ASP.NET Core 2.0 MVC** deoarece oferă o gamă largă de servicii, printre acestea fiind și un sistem de autentificare – **Identity**, iar comunicarea între părți se realizează ușor datorită pattern-ului **MVC**. Pe partea de client-side, pe lângă **HTML**, **CSS** & **JavaScript**, am optat pentru a oferi o experiență cât mai completă cu ajutorul framework-ului Bootstrap 4, iar legat de bazele de date – am ales **SQL Server-ul** (variantea **Express**).

2. Context

Posibilitățile oferite de internet sunt nenumărate când vine vorba de cărți, de la gestiunea lor până la posibilitatea de a le citi online. Cu toate acestea, pot apărea și probleme:

O problemă de actualitate este cea legată de drepturile de autor asupra cărților respective - nu toate cele disponibile pe internet sunt și legale.

Din fericire pentru utilizatori, majoritatea țărilor păstrează drepturile de autor asupra unei cărți pe o perioadă de aproximativ 70 de ani de la moartea autorului. Odată expirată această perioadă, cartea poate fi distribuită în mod gratuit – cu alte cuvinte, poate fi citită în mod legal, fiind inițiate multe proiecte pentru a colecta aceste tipuri de cărți (InternetArchive, Project Gutenberg etc.)

Aplicațiile importante, când vine vorba de cărți, oferă: fie posibilitatea de a gestiona cărțile citite, fie de a le citi online sau pe orice dispozitiv (de exemplu: eReader-ul). Acest lucru îngreunează activitatea unui cititor, el fiind nevoit să asigure sincronizarea dintre cele două tipuri de aplicații (citirea și păstrarea progresului făcut de acesta), semnificând o pierdere a timpului prin acest tip de sincronizare.

Tot legat de aplicațiile de gestiune a cărților, acestea oferă o monitorizare vagă a progresului pentru cititor, acesta de obicei având posibilitatea doar de a muta cartea de la o stare la alta (de exemplu, de la *În curs de citire* la *Citită*).

O altă problemă o reprezintă o lipsă de recomandări/direcționări pentru cititorii mai puțin inițiați în tainele cititului. Aceștia sunt nevoiți să caute multe review-uri înainte de a își face o părere despre eventuala carte pe care ar putea să o citească.

Aplicația „ExploreBooks” își propune să rezolve toate aceste probleme, oferind posibilitatea unui cititor de: a își face o opinie despre o carte, a o citi în aplicație sau a urmări progresul legat de cartea respectivă. Mai mult, are posibilitatea de a recomanda liste cu cărți, selectate pe baza unor criterii personale.

Folosind API-ul celor de la Project Gutenberg, cărțile puse la dispoziție de aplicație sunt exemplare a căror drepturi de autor au expirat, deci cititorul are posibilitatea de a le citi în mod legal.

2.1. Aplicații similare

Un exemplu de aplicație similară ar reprezenta **GoodReads**, aplicație pusă la dispoziție gratuit de **amazon**. Scopul acesteia este de a permite utilizatorilor să poată ține o evidență a cărților citite (evidență sugerată doar de starea cărții – *Want to read* și *Currently Reading*).

Alte funcționalități importante sunt: permite autorilor să își promoveze și să facă management asupra cărților scrise, oferă posibilitatea de a cumpăra cărți și permite distribuirea, pe diferite rețele de socializare, a opiniei legate de o anumită carte.

Cu toate acestea, există și multe dezavantaje.

În primul rând, modul de gestionare a cărților citite este unul relativ limitat. Un utilizator fiind capabil doar să schimbe starea cărții, nu poate avea o evidență cu adevărat folositoare legate de progresul acestuia, ci doar una superficială.

În al doilea rând, cantitatea de informații disponibilă este în exces, astfel, navigarea în aplicație poate fi dificilă în multe cazuri.

3. Cerințe funcționale

Aplicația oferă următoarele funcționalități:

1. User's profile:

- a. **Account:** utilizatorul va putea modifica datele afișate pe site (cum ar fi numele, prenumele, țara sau scurta descriere), parolele, poza de profil sau în conformitate cu GDPR-ul, își va putea șterge profilul și implicit toate datele salvate despre acesta.
 - b. **Profile:** reprezintă principala parte de socializare, utilizatorul având posibilitatea să dea *Follow/Unfollow* anumitor persoane (acestea vor fi notificate în funcție de acțiune), să vadă istoricul acestuia sau să descopere alți utilizatori (fie din toată lumea sau din aceeași țară).
 - c. **Login & Register:** se realizează prin două posibilități: crearea unui cont direct în aplicație și utilizarea acestuia sau prin intermediul unor rețele de socializare: *Facebook*, *Twitter* sau *Google+*. În ambele situații, va primi un mail din partea aplicației pentru înștiințare în legătură cu contul creat.
2. **Explore:** utilizatorul va putea vizualiza și căuta cărți (căutarea se poate realiza în funcție de gen, autori sau prin intermediul search bar-ului).
 3. **List of books & Find me a book:** reprezintă zona de recomandări oferite de comunitate sau aplicație.
 4. **Announcements:** administratorul, pe pagina principală, va putea face anunțuri cu diverse activități pe site, fiecare utilizator fiind notificat în legătură cu anunțul făcut.

Orice utilizator, pentru a putea folosi aplicația, va avea nevoie de minim: o conexiune stabilă la internet, un dispozitiv pentru a putea naviga în aplicație (de preferat laptop sau desktop pentru o experiență cât mai completă).

Contribuții

O primă contribuție a reprezentat o etapă dedicată cercetării și descoperirii celor mai potrivite tehnologii în vederea realizării acestei aplicații.

Apoi, în urma stabilirii acestor tehnologii, am avut o etapă de structurare a proiectului, schițare și implementare a funcționalităților acestuia.

Principala contribuție a constat în implementarea și abordarea originală când vine vorba de aplicație.

Faptul că, pe lângă soluțiile clasice de citire și monitorizare a progresului, am îmbunătățit experiența unui utilizator și când vine vorba de modul de alegere a cărților sau de interacționare cu ceilalți utilizatori.

Astfel, principalele puncte inovative ale aplicației sunt:

1. **Posibilitatea de gestionare a cărților unui utilizator:** fiecare utilizator va avea un istoric al activității acestuia în aplicație, o librărie cu starea fiecărei cărți – prin stare mă refer, pe lângă progresul utilizatorului cu privire la aceasta, și la impactul cărții asupra utilizatorului, ce capitole i-au plăcut sau dacă este una din cărțile sale favorite.
2. **Posibilitatea de a citi online direct din aplicație:** astfel, va avea posibilitatea de a urmări în timp real progresul acestuia, modalitatea de citire și salvare a progresului fiind una ușor de realizat.
3. **Posibilitatea de a primi diverse recomandări:** aceste recomandări se împart la rândul lor în două categorii:
 - a. **recomandări oferite de aplicație:** aici este vorba de cele două opțiuni puse la dispoziție cititorului: fie de a primi o recomandare de carte la întâmplare, fie una pe baza unui istoric al acestuia.
 - b. **recomandări oferite de comunitate:** orice utilizator are posibilitatea de a crea o listă cu cărți, în funcție de dorința acestuia, dar și de a recomanda sau lăsa feedback pentru o anumită carte.
4. **Posibilitatea de a interacționa cu ceilalți utilizatori:** prin adăugarea de comentarii pe profilele lor, pe anumite secțiuni din aplicație sau posibilitatea de a da *follow/unfollow*.

Capitolul I – Proiectarea aplicației

În acest capitol voi acoperi tot ce ține de modul de organizare pe care l-am abordat în aplicație, de la structurarea acesteia până la modelarea bazei de date.

1. Structurarea aplicației

Întotdeauna un cod organizat va fi un cod mai ușor de gestionat și menținut, de aceea modul de organizare a aplicației a fost o prioritate.

Ca și pattern arhitectural principal – pentru întreg proiectul, am folosit *Onion Architecture* – structurând astfel componentele aplicației, în funcție de scopul fiecăreia. Motivul alegerii acestui pattern ar fi: o bună separare a conceptelor utilizate în aplicație, iar spre deosebire de pattern-ul arhitectural *3-tier*, se renunță și la cuplarea tare dintre componente.

Ideea de bază pentru această arhitectură ar fi că detaliile legate de dependența dintre componente să fie la un nivel cât mai înalt, astfel componentele de pe un layer superior să nu depindă de modul ales pentru implementarea celor de pe un layer inferior.

Astfel, aplicația după cum se observă și în **Figura 1**, a fost structurată în:

Domain layer: se ocupă cu tot ce ține de comunicarea directă cu baza de date, fiind împărțit în două proiecte principale: *Data* și *Persistence*. În *Data* – am adăugat entitățile cu care lucrează aplicația la nivel de back-end, fiecare semnificând o tabelă a bazei de date. În *Persistence* am adăugat contextul pus la dispoziție de către *Entity Framework* – fiind folosit la maparea entităților de pe back-end cu cele din baza de date.

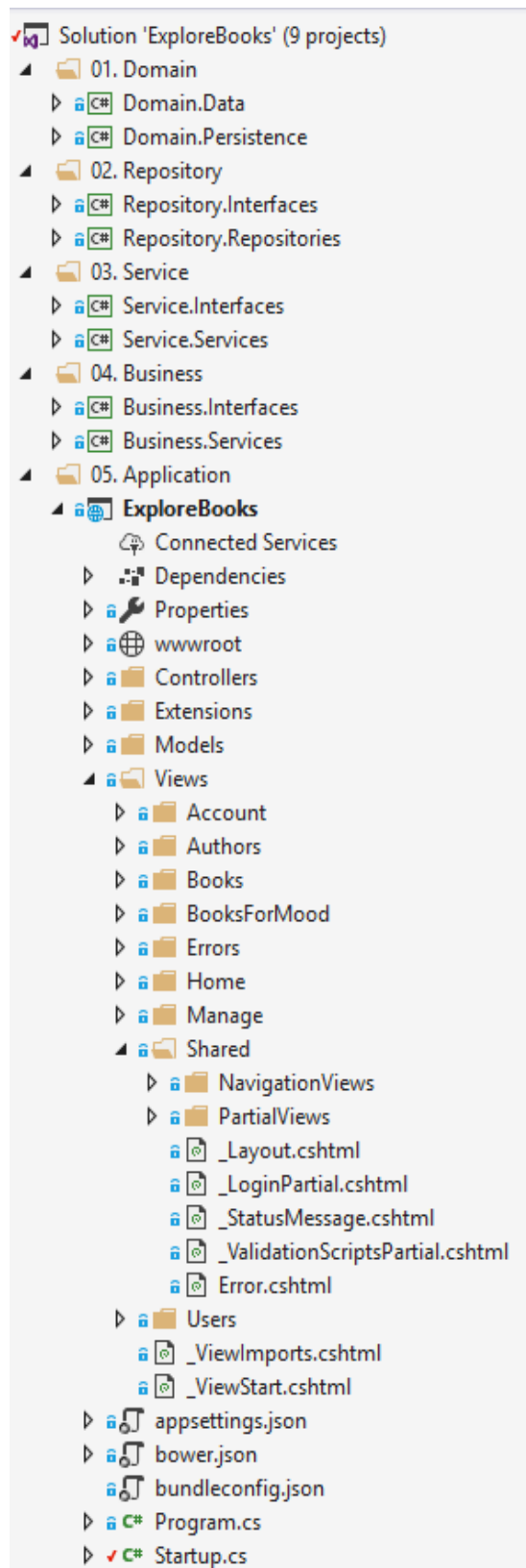


Figura 1: Modul de structurare a aplicației

Repository layer: structurată în interfețe și clase ce le implementează. În acest layer am apelat la *Repository Pattern*, ocupându-se doar cu operațiile CRUD pentru fiecare entitate. În plus, acest pattern ne oferă o separare a business logicului față de baza de date, ajutându-ne să creăm cod mai ușor de întreținut și citit.

Service layer: același tip de structurare: interfețe și clase ce le implementează. Scopul acestui layer a constat în oferirea unui set de servicii intermediare, în plus față de cele din repository, dar care să realizeze doar operații de bază.

Business layer: layer folosit mai departe de *Presentation layer* și care conține toate posibilitățile puse la dispoziție utilizatorului. Din acest motiv, structurarea a fost realizată astfel încât fiecare controller să poată avea un serviciu echivalent, dar să existe și servicii generale, folosite pentru un mai bun management al acestora la nivel de *Presentation layer*.

Presentation layer: organizat pe baza patternului arhitectural de tip *MVC (Model – View – Controller)*. Astfel, View-ul reprezintă interfața cu care utilizatorul interacționează, toate cererile acestuia fiind transmise la Controller, iar dacă este vorba de transfer de date – Modelul.

În funcție de cantitatea de informații aflată în View-uri, unele dintre aceste View-uri au fost structurate în mai multe PartialView-uri.

2. Modelarea bazei de date

Organizarea bazei de date se poate observa în **Figura 2** de mai jos:

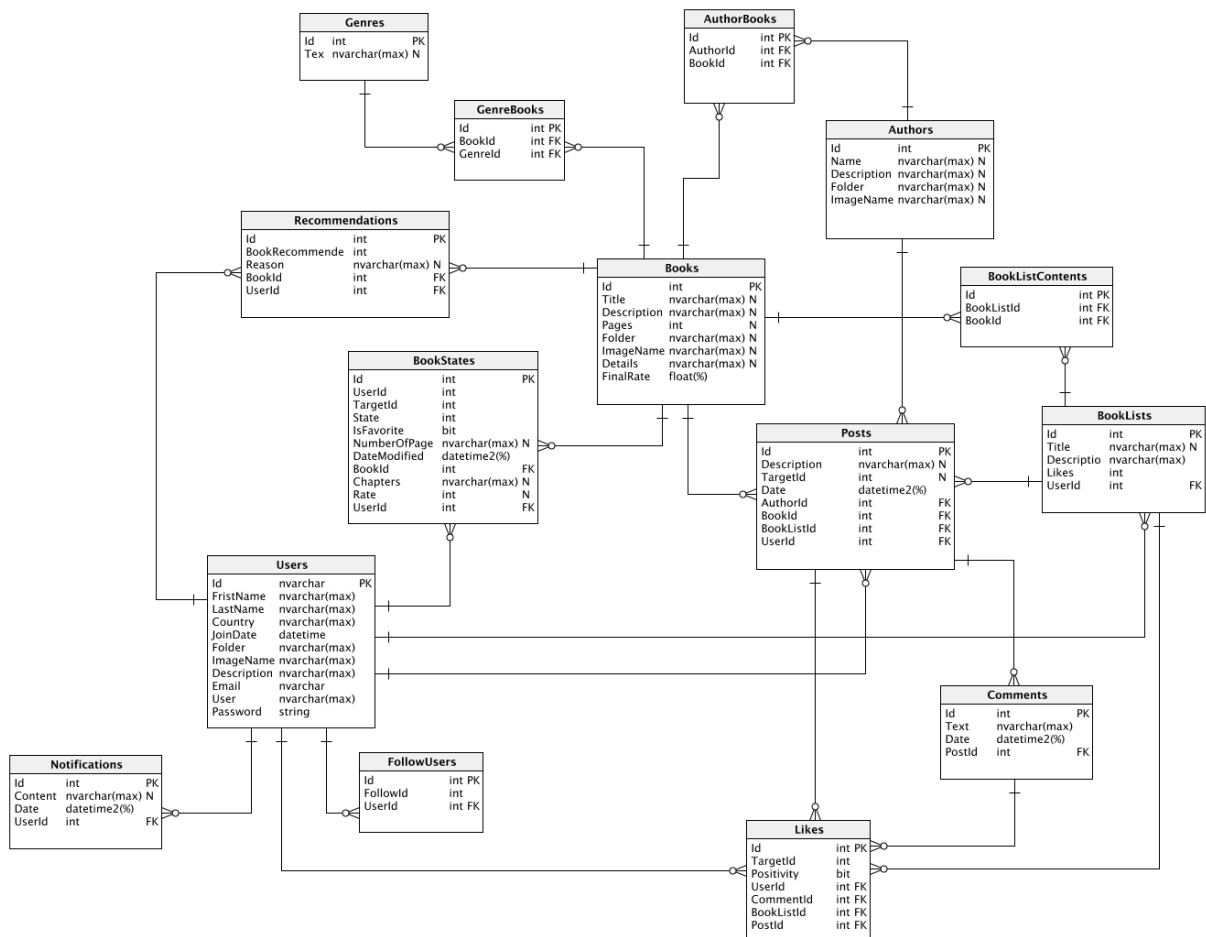


Figura 2: Modul de organizare a bazei de date

Principalul motiv al acestei organizări a constat în faptul că printr-o structură mai aerisită a unui tabel, gradul de reutilizare a acestuia este mai ridicat. Astfel, prin generalizarea și împărțirea curentă, multe componente au putut fi refolosite în diverse situații – cel mai evident exemplu fiind componentele de **Post & Comments**.

O altă observație ar fi faptul că există multe tabele de legătura. Scopul principal al acestei decizii se datorează faptului că *Entity Framework Core* tratează relațiile many-to-many prin

intermediul unui tabel intermediar (mai multe detalii în capitolul legat de implementări, secțiunea de **Populare a bazei de date**, iar pentru Entity Framework – **Anexa 1 – Server Side**).

O descriere a tabelelor ar fi:

Tabela **Books** – conține toate detaliile legate de o carte, majoritatea dintre acestea fiind extrase pe baza informațiilor oferite de *Project Gutenberg*. Câmpul **FinalRate** conține media pe baza notelor acordate de utilizatori, în timp ce câmpurile **Folder** și **ImageName** conțin path-ul către folderul cu datele respectivei cărți – imaginea și textul acesteia.

Tabela **Genres** – conține genurile disponibile în aplicație. Am ales să fac relaționarea ca fiind una many-to-many deoarece cărțile pot avea mai multe genuri, iar prin genuri să existe posibilitatea de a vizualiza toate cărțile specifice acelui gen.

Tabela **Authors** – pe lângă informațiile de bază – specifice autorului respectiv, se mai regăsesc și câmpurile **Folder** și **ImageName**, din același motiv de localizare locală a imaginii acestuia.

Tabela **BookLists** – tabelă ce stochează toate informațiile legate de listele de cărți recomandate de utilizatori și disponibile acestora în secțiunea **List of books**. Am adăugat și o evidență a numărului de voturi/like-uri acordate pentru fiecare listă, pentru a ușura modul de alegere a listelor potrivite. Tabela stochează pentru fiecare listă nou creată toate cărțile aflate în respectiva listă.

Tabela **Recommendations** – în această tabelă se regăsesc cărțile recomandate de utilizatori pentru o anumită carte.

Tabelele **Posts**, **Comments & Likes** – reprezintă **Community feedback-ul** dat de utilizatori. Relaționarea dintre acestea fiind: fiecare post și comment conțin o evidență a numărului de voturi/like-uri acordate, iar fiecare post este alcătuit din mai multe comment-uri.

Relaționarea dintre **Authors**, **BookLists**, **Books**, **Users** și **Posts** se datorează faptului că fiecare din aceste componente permit adăugarea de postări, astfel această relaționare ar putea fi văzută ca fiind una one-to-many (**Authors / BookLists / Books / Users – Posts**).

Tabela **Users** – a fost generată inițial cu ajutorul *ASP.NET Core Identity* (mai multe detalii despre acest sistem se regăsesc la **Anexa 1 – Server Side**), având multe din câmpuri generate

automat. Astfel, pe lângă acestea, am adăugat câmpuri suplimentare, cele mai importante fiind **Folder** și **ImageName** – identificarea locală a acestuia și câmpul **User** – fiind unic și reprezentând modul de identificarea a utilizatorului în cadrul aplicației (mai multe detalii în capitolul de implementări, secțiunea de **Profile**).

Tabela **BookStates** – este folosită pentru a gestiona legătura dintre utilizatori și cărți. Astfel, orice interacțiune a utilizatorului cu cartea (fie adăugarea cărții la una dintre listele acestuia, fie lăsarea unei postări aferente cărții respective etc.) este urmărită prin cadrul acestei tabele.

Statisticile asupra unei cărți se bazează tot pe această tabelă. Astfel, pentru a afișa cele mai adorate capitole sau media notelor acordate de utilizatori cărții respective, putem să extragem din tabela **BookStates** toate intrările a căror **BookId** reprezintă id-ul cărții căutate.

Tabela **FollowUsers** – proiectată pentru a ține evidența relațiilor de tip *follow/following* dintre utilizatori. Motivul pentru care am optat să adaug o tabelă separată pentru a ține această evidență se datorează logicii de relaționare pentru *follow/following*, astfel: următorul este dat de **UserId**, iar persoana urmărită este dată de id-ul **FollowId**, selectarea fiecărui tip fiind mai ușor de gestionat.

Tabela **Notifications** – păstrează toate notificările apărute utilizatorilor. Această tabelă este folosită și în cadrul anunțurilor făcute de administratori, astfel, fiecare utilizator urmează a fi notificat în legătură cu conținutul anunțului respectiv.

Capitolul II - Detaliile de implementare

În acest capitol voi prezenta modul de implementare a aplicației, pe baza proiectării detaliate din capitolul precedent. Structura aleasă pentru prezentare se focusează pe două mari componente: cea de server și cea de client.

1. Server

1.1. Realizarea conexiunii cu baza de date

Aplicația fiind una centrată pe model, modul de programare ales pentru crearea entităților din baza de date a fost *Code-First*. Pe scurt, acest concept permite programatorului să se focuseze pe domeniul aplicației, având posibilitatea de a crea entități (clase *POCO*), iar prin intermediul *Entity Framework*-ului, se vor crea echivalente ale acestora în baza de date. Am ales această abordare deoarece, consider eu, îți oferă o flexibilitate și abordare mai generală când vine vorba de entitățile utilizate. Fiind organizate în acest mod, se va putea face cu ușurință orice formă de modificare a entităților direct de pe server, fără a fi constrânși de felul în care sunt salvate acestea.

Astfel, pentru fiecare clasă creată, cu ajutorul contextului pus la dispoziție de către *Entity Framework*, s-a generat o entitate echivalentă cu aceasta pe partea bazei de date.

Orice relație de tip one-to-many sau many-to-many se realizează prin intermediul unei colecții cu entitățile implicate (**Figura 3**). În cazul many-to-many, pe lângă colecția respectivă, se va crea și o clasă de legătura, ce va conține cheile primare specifice celor două entități și se va specifica comportamentul acesteia în *OnModelCreating*.

```
public ICollection<AuthorBook> Authors { get; set; }  
  
public ICollection<GenreBook> Genres { get; set; }  
  
public ICollection<Post> Posts { get; set; }
```

Figura 3: Salvarea relațiilor one-to-many și many-to-many prin metoda **Code-First**

În unele cazuri, pentru a acorda o anumită comportare unor clase intermediare, cu alte cuvinte, modificarea comportamentului de bază a claselor, am recurs la *Fluent API*.

Principiul este unul relativ simplu, fiind nevoie doar de suprascrierea metodei `OnModelCreating` din context, după cum se poate observa în **Figura 4**.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<AuthorBook>()
        .HasKey(authorBook => new { authorBook.AuthorId, authorBook.BookId });

    modelBuilder.Entity<GenreBook>()
        .HasKey(genreBook => new { genreBook.GenreId, genreBook.BookId });

    base.OnModelCreating(modelBuilder);
}
```

Figura 4: Aplicarea **Fluent API-ului**, prin suprascrierea metodei **OnModelCreating**

Am recurs la această abordare deoarece specificarea comportamentului, în acest caz, semnifică explicitarea modului de relaționare dintre entități, cu alte cuvinte, specifică fiecare cheie străină la ce tabelă face referință.

1.2. Popularea bazei de date

Popularea bazei de date, în acest caz, constă în adăugarea cărților și modul cum a afectat această adăugare și restul tabelelor. Pentru a realiza popularea, am apelat la API-ul pus la dispoziție de către **Project Gutenberg**.

Conform indicațiilor prezente pe pagina lor, una din modalitățile de descărcare a cărților a fost prin utilizarea **wget-ului** (mai multe detalii despre acest software sunt disponibile la **Anexa 1**). API-ul îți oferă posibilitatea de a alege formatul în care dorești să salvezi cărțile, iar în acest caz, am ales să fie în format *.txt*, extragerea informațiilor din acest tip de fișiere fiind destul de flexibilă. Apoi, prin rularea următoarei comenzi¹ am populat folder-ul local unde a fost instalat **wget-ul** - **Figura 6**.

```
wget -w 2 -m -H "http://www.gutenberg.org/robot/harvest?filetypes[]=txt"
```

Figura 6: Comanda rulată pentru extragerea arhivelor cu cărți în format *.txt*

De menționat faptul că o structură a unui fișier text descărcat conține: partea de introducere – unde se precizează condiția folosirii textului respectiv și câteva date legate de carte, partea de conținut a cărții și termenii în care s-a salvat acest fișier. În următoarea etapă, m-am folosit de un

¹ **Sursă:** https://www.gutenberg.org/wiki/Gutenberg:Information_About_Robot_Access_to_our_Pages

serviciu la nivelul de *Services Layer* pentru a modifica/extrage informațiile din fișiere. Astfel, toate informațiile cu privire la cărți și autori au fost păstrate, iar toate celelalte informații din cadrul introducerii au fost șterse.

Motivul pentru care am păstrat termenii de la sfârșitul fișierului s-ar datora condiționării impuse de **Project Gutenberg**, dar și pentru a oferi posibilitatea utilizatorului de a verifica cartea direct pe site-ul acestora. Pe baza informațiilor păstrate s-au populat tabelele respective, iar pentru susținerea relației many-to-many, entitatea ce joacă rol de intermediar va păstra cheile primare ale acestora.

1.3. Service layer

Este structurat în *Interfaces* și *Services*. Scopul acestui layer este de a pune la dispoziție metode intermediare pentru a facilita lucrul din *Business Layer*. Spre deosebire de *Repository layer*, acest serviciu pune la dispoziție o serie de metode suplimentare, având o complexitate medie cu privire la operațiile realizate de acestea.

În acest layer, cele mai importante servicii, care pun la dispoziție o serie numeroasă de metode sunt:

WorkingWithFiles – se ocupă de lucrul cu fișiere, având posibilitatea de a crea, șterge sau modifica foldere/fișiere, pe diverse criterii. Cu ajutorul acestui serviciu se salvează tot ce ține de informația în plus, ce nu se regăsește în tabele – imagini, conținuturile cărților etc.

Modul de salvare al acestor date suplimentare se realizează astfel: în funcție de tipul entității, se salvează în folderul specific acelui tip (*Books* – în folderul *books/*), într-un folder specific acelei înregistrări, pe baza id-ului pe care îl are în baza de date (de exemplu: cartea cu id-ul 3 va fi găsită la locația: *books/3/date*).

WorkingWithDatabase – se ocupă de modul în care sunt salvate anumite date, dar și de obținerea informațiilor cu privire la conținutul unei cărți. Acest serviciu a fost folosit și la popularea bazei de date cu cărți, autori și genuri, prin folosirea mai multor repository-uri și a serviciului *WorkingWithFiles* – pentru salvarea datelor respective în fișierele corespunzătoare.

Modul de prelucrare a fișierelor .txt a fost: au fost selectate toate liniile care au conținut și am considerat că o pagină din cartea respectivă conține 35 astfel de linii. Așadar, numărul de pagini

ale cărții respective este dat de raportul dintre numărul de linii nevide ale fișierului și 35 (număr_linii / 35).

Tot la acest nivel am adăugat și rolurile posibile pentru utilizatori (*owner*, *administrator* sau *user*), dar și serviciul de creare a rolurilor respective. Acest serviciu urmează a fi apelat explicit odată ce un nou utilizator a fost creat, fiindu-i asignat un rol “by default” – cel de *user*.

```
public Book CheckBook(string bookTitle)
{
    var check = _repository.GetBookBasedOnTitle(bookTitle);
    if (check == null)
    {
        var value = Guid.NewGuid();
        var path = _folder + "\\\" + value;
        var imageName = _folder + ".jpg";

        _fileManagement.CopyFile(_folder, value);
        var book = Book.CreateBook(
            bookTitle,
            "No description for this book at the moment..",
            path,
            imageName,
            "There are mysteries surrounding this book.."
        );

        _repository.CreateBook(book);

        return book;
    }

    return check;
}

public List<Book> GetBooks(string bookTitles)
{
    var books = bookTitles.Split(",");
    var bookList = new List<Book>();

    foreach (var book in books)
    {
        bookList.Add(CheckBook(book));
    }
    return bookList;
}
```

Figura 7: Implementarea metodelor descrise în serviciul **BookMiddleware**

Alte servicii cu o implementare mai interesantă sunt cele ce se ocupă cu managementul autorilor și cărților. Voi prezenta în cazul serviciului specific cărților, fiind același tip de abordare

și în cazul serviciului dedicat autorilor. Astfel, două din metodele prezentate în **Figura 7** au drept scop:

- verificarea dacă deja există o înregistrare cu acel nume, iar în caz contrar o va crea
- pe baza unui string ce conține toate titlurile de cărți, îl divizează și pentru fiecare carte, verifică prin metoda precedentă dacă există sau nu, caz în care o va crea.

1.4. Business Layer

Acest layer interacționează în mod direct cu *Presentation Layer-ul*. Am optat ca pentru fiecare Controller existent în *Presentation Layer*, să existe un echivalent al acestuia în *Business Layer*. Am ales această abordare pentru a păstra logica din Controller la un nivel minim, astfel, în Controller se vor apela doar metodele din servicii, verificările și modificările realizându-se la nivelul acestora.

De adăugat faptul că un Controller, din pricina tipurilor de acțiuni ce le are, poate folosi mai multe servicii.

Un caz aparte îl reprezintă cel legat de controller-ul *Users*. Pentru o mai bună separare a conceptelor, am împărțit task-urile ce se realizează în mai multe servicii. Astfel, există un serviciu ce se ocupă strict de interacțiunea user-ului cu cărțile, un serviciu ce se ocupă de interacțiunea dintre mai mulți useri, unul ce se ocupă de modificările făcute pe tabela *Users*, iar ultimul ce se ocupă cu relaționarea dintre imaginile salvate aferente userului și modul de utilizare ale acestora (imaginile iarăși fiind salvate local).

Modul de proiectare a serviciului pentru cărți iarăși este unul unic. Pe lângă modul de modificare a cărților (*edit* și *create*), aici se regăsesc și implementări cum ar fi:

- modalitatea de căutare a unei cărți pe baza unui text introdus – aplicată pe bara de search a aplicației, implementarea fiind făcută în **Figura 8**.

```
public List<Book> SearchBooks(string text)
{
    var books = _bookRepository.GetAllBooksContainingTheTitle(text);

    foreach (var book in books)
    {
        book.Authors = _authorBookService.
            GetAllAuthorBooksBasedOnBookId(book.Id);
        book.Genres = _genreBookService.
            GetAllGenreBooksBasedOnBookId(book.Id);
        book.Posts = _postService.
            GetAllPostsForTargetId();
    }

    return books;
}
```

Figura 8: Implementarea metodei de căutare a unor cărți în funcție de textul introdus

- modalitatea de a obține următoarele 12 cărți, fiind utilizată la paginare - **Figura 9**

```
public IReadOnlyList<Book> GetFirstNBooks(int count)
{
    var books = _bookRepository.GetFirstNBooks(count, 12);

    return books;
}
```

Figura 9: Implementarea metodei de obținere a cărților în intervalul $[count * 12 + 1, (count + 1) * 12]$

- modalitatea de a obține un *SelectListItem* cu toate cărțile disponibile în aplicație, lucru folosit la metoda de recomandări, unde utilizatorul poate selecta dintr-un dropdown ce cărți dorește, dropdown-ul fiind populat cu această metodă - **Figura 10**

```
public List<SelectListItem> GetAllBooksForRecommendation(Guid bookId)
{
    var books = _bookRepository.GetAllBooksExcept(bookId);
    var bookList = new List<SelectListItem>();

    foreach (var book in books)
    {
        bookList.Add(new SelectListItem
        {
            Text = book.Title,
            Value = book.Id.ToString()
        });
    }

    return bookList;
}
```

Figura 10: Implementarea metodei de populare a SelectListItem-ului

Metodele de *Create* și *Edit* au următoarea abordare: în *Create*, pe baza genurilor și autorilor primiți ca parametru de tip string, apelez echivalentul lor în layer-ul de *Services* pentru a diviza și obține autorii și genurile respective. Modalitatea de obținere este similară cu cea prezentată la *CheckBook* și *GetBooks* (**Figura 7**).

WorkingWithFiles este folosit în salvarea imaginilor pentru cartea respectivă. Dacă s-a introdus o imagine în parametru, atunci se va copia imaginea respectivă, iar dacă nu există– se va crea o imagine *by default* la locația specifică. Apoi, în funcție de scopul fiecărei metode, se va crea/edita cartea respectivă cu informațiile obținute.

Serviciul de *EmailSender*: acest serviciu vine apelat odată ce un utilizator și-a creat un cont cu succes, scopul acestuia fiind de a îl notifica pe acesta printr-un email de faptul că contul lui este funcțional. Modul de implementare constă în utilizarea unui client SMTP pentru a trimite mail, pe baza credențialelor salvate. Acest serviciu va fi apelat în cadrul unei metode de la nivelul *Presentation Layer*-ului. Acest serviciu a fost generat prin intermediul *Identity*-ului², fiind implementat în **Figura 11**.

```
public Task SendEmailAsync(string email, string subject, string message)
{
    var mail = new MailMessage(Username, email, subject, message);

    var client = new SmtpClient(Server)
    {
        Port = 587,
        Credentials = new NetworkCredential(Username, Password),
        EnableSsl = true,
    };

    return client.SendMailAsync(mail);
}
```

Figura 11: Implementarea metodei de trimitere a email-ului de informare

O categorie aparte de servicii ce se regăsesc în acest layer este cea a serviciilor de utilitate, fiind reutilizate în diverse servicii, pentru un mai bun management.

Un astfel de exemplu îl constituie *LikeService*-ul, unde este apelat în cadrul fiecărui serviciu ce se folosește de componenta *Like*. În interiorul acestuia, se verifică dacă un anumit utilizator a dat deja like unui anumit target (postare, comment sau listă cu cărți), caz în care îi va anula like-ul, iar în caz contrar îl va salva. O altă metodă expusă de acesta este cea în care calculează numărul de like-uri pentru o anumită postare, pe baza diferenței dintre like-urile pozitive și cele negative.

Cel mai folosit serviciu utilitar este *UtilityService*, acesta punând la dispoziție o serie de metode, cum ar fi: obținerea unor entități pe bază de id sau modificarea modului de afișare pentru anumite informații (aici mă refer la metoda ce pe baza unui *DateTime* primit ca și parametru va returna un string cu un mod mai plăcut de afișare a acestuia – folosit la data afișată utilizatorilor).

² **Sursă:** <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x>

Alte metode notabile din acest serviciu sunt cele de recomandare a unei cărți, în funcție de criterii. Așadar, opțiunea pusă la dispoziție de aplicație când vine vorba de recomandările automate se regăsește în acest serviciu.

Pentru recomandarea la întâmplare se realizează o simplă generare din toate id-urile cărților disponibile. Pentru recomandarea în funcție de utilizator ar veni: mai întâi se verifică dacă utilizatorul respectiv are un istoric al cărților disponibil în aplicație. În caz pozitiv, atunci va căuta pentru fiecare carte aflată în istoric dacă conține o recomandare făcută de cineva (pe pagina cărții respective), iar dacă există, va întoarce cartea recomandată. În cazul în care nu s-a găsit nicio recomandare, atunci va căuta în funcție de genurile cărților din istoric și va recomanda o carte cu același gen. În cazul în care utilizatorul este nou în aplicație, atunci se va căuta cartea cu cel mai mare rating disponibil³ și se va returna - **Figura 12**.

```
public Guid GetBestRatedBooks()
{
    var books = _bookRepository.GetAllBooks();
    var savedData = new Dictionary<Book, double>();
    if (books.Count > 0)
    {
        foreach (var book in books)
        {
            savedData.Add(book, book.FinalRate);
        }

        return savedData.OrderByDescending(value => value.Value)
            .FirstOrDefault().Key.Id;
    }

    return Guid.Empty;
}
```

Figura 12: Implementarea căutării în funcție de rating și returnarea id-ului corespunzător cărții cu rating-ul cel mai mare

Serviciul *ApplicationBookLogic* - conține toate metodele ce tratează modul cum un user interacționează cu entitatea *Book*. Câteva din aceste metode sunt:

- *ModifyBookPages*: pe baza unui număr primit ca și parametru, va modifica starea cărții (dacă acel număr este mai mare sau egal cu numărul total de pagini ale cărții atunci va trece cartea la *Read*, iar pentru orice alt număr, va trece cartea la *Reading*.
- *AddToFavorites* – tratează cazul în care un utilizator a selectat cartea ca fiind una din favoritele acestuia. Pentru cazul în care utilizatorul n-a avut nicio stare față de cartea

³ Rating-ul reprezintă media tuturor notelor acordate de utilizatori cărții respective.

respectivă (n-a selectat sau modificat numărul de pagini), atunci se va trece automat cartea la *Reading*.

- *GetRatingList*: unde se returnează un *SelectedItem* care conține toate posibilitățile de rating pe care un utilizator le poate acorda.
- *GetRatingsAverageForBook* – se calculează media tuturor notelor existente pentru cartea respectivă.

2. Client

Am ales să includ în acest capitol și partea de *Presentation Layer*, pentru o mai bună evidențiere a modului cum se realizează comunicarea dintre componente.

2.1. Controller

Fiecare Controller are un set de acțiuni specifice, după modulul pe care îl tratează.

AccountController: Se ocupă de logica legată de modul de autentificare disponibilă clientului.

BooksController: reprezintă toate acțiunile legate de utilizarea unei cărți. În acest controller sunt apelate metodele necesare pentru a face căutarea în funcție de diverse criterii (în funcție de genuri, autori sau recomandări), dar și metode care fac posibilă recomandarea unei cărți, pe lângă acțiunile standard specifice unui controller – Create, Edit, Details și Delete.

Am ales această abordare deoarece reprezintă toate metodele ce au în prim plan cartea.

CommunityFeedbackController: conține toate metodele ce țin de tratarea cazurilor de postare/comentare la un anumit material. Multe din acțiunile regăsite aici nu sunt apelate în mod direct de către un request al user-ului, ci prin intermediul unui apel de tip AJAX din jQuery, fiind folosit pentru a îmbogăți experiența utilizatorului, nefiind nevoie să dea refresh la pagină pentru a verifica update-urile legate de postări sau comentarii.

ErrorsController: tratează tipuri de erori care au apărut în cazul unei acțiuni ilegale/invalide din partea clientului, acțiunile fiind apelate, după caz, în alte controllere.

ManageController: se ocupă de partea de setări specifice utilizatorului.

UsersControllers: tratează tot ce ține de acțiunile utilizatorului în cadrul aplicației: interacțiunea utilizatorilor între ei, acțiunile ce le pot executa sau interacțiunea dintre utilizator și carte, cu opțiunile respective.

De menționat faptul că am modificat modul de rutare spre acțiunile din controllere. Astfel, pentru fiecare controller am adăugat ruta unde poate fi acționat - **Figura 14**, iar în acțiuni am adnotat modul cum vor apărea în aplicație - **Figura 13**.

<pre>[Route("books/[action]")] public class BooksController : Controller {</pre>	<pre>[HttpGet, ActionName("genres")] public IActionResult Genres(string genre) {</pre>
--	--

Figura 14: Adnotarea astfel încât acțiunile din Controller vor fi apelate la endpoint-ul: **/books/**

Figura 13: Adnotarea astfel încât acțiunea va fi folosită prin intermediul: **/books/genres?genre=**

Un caz aparte al rutării este cel din cadrul *UsersController*-ului. Pentru a face posibilă căutarea unui alt user în aplicație, a trebuit să adaug posibilitatea găsirii acestuia la o anumită adresă, după cum se observă și în **Figura 15**.

<pre>[HttpGet("{username}")] public IActionResult Index(string username) [HttpGet("{username}/library")] public IActionResult Library(string username)</pre>

Figura 15: Modul de acces a acțiunilor, astfel pentru index va fi: **/users/nume_user** iar pentru library: **/users/nume_user/library**

2.2. Views

Majoritatea View-urilor au fost organizate în folderul specific Controller-ului din care fac parte.

Excepție la această regulă o reprezintă *PartialViews-urile* din folderul *Shared*, unde partial view-urile sunt legate de opțiunile disponibile pentru community feedback (postări sau comentarii). Au fost plasate în acest mod pentru reuzabilitate, opțiunea de feedback fiind prezentă pe mai multe pagini.

Opțiunea de editare și ștergere a unui comentariu/postări se realizează prin intermediul unui modal ce apelează în controller acțiunea selectată. În schimb, acțiunea de create, fie pentru postare sau comentariu, dar și acțiunea de upvote/downvote se realizează prin intermediul apelurilor AJAX, evitând astfel reîncărcarea întregii pagini, această abordare fiind implementată în **Figura 16**.

```
$( ".submitComment" ).submit(function (e) {
    e.preventDefault();
    var user = this.userId.value;
    var target = this.postId.value;
    var comment = $(this).find("input[name='commentText']").val();
    $.ajax({
        url: "/CommunityFeedback/CreateComment",
        data: { userId: user, postId: target, commentText: comment },
        type: "POST",
        success: function () {
            var result = "@Url.Action('GetAllCommentsForPostId',
"CommunityFeedback")?postId=" + target;
            $("#" + target).load(result);
            $(".submitComment").trigger("reset");
            $(".focus").blur();
        }
    });
});

$( ".submitPostEdit" ).submit(function (e) {
    e.preventDefault();
    var target = this.postId.value;
    var post = $(this).find("textarea[name='postText']").val();
    $.ajax({
        url: "/CommunityFeedback/EditPost",
        data: { postId: target, postEdited: post },
        type: "POST",
        success: function () {
            location.reload();
        }
    });
});
```

Figura 16: Modurile de apelare pentru cele două abordări

Bara de căutare face posibilă accesarea mai rapidă a unei cărți specifice sau afișarea cărților respective pe baza text-ului introdus, afișarea fiind realizată printr-un dropdown ce se modifică printr-o funcție javascript ce face filtrarea conținutului afișat.

În cele ce urmează, voi acoperi cele mai notabile abordări din View-uri, astfel voi avea:

View-urile specifice opțiunilor de *Manage* și *Users* se folosesc de o bară de navigație specifică paginilor respective, care pot fi accesate fără a fi reîncărcată pagina. Acest lucru se datorează folosirii unei clase ce încarcă doar conținutul disponibil la acea pagină, fără a afecta restul container-ului unde se găsește aceasta.

View-urile specifice secțiunii de *Books* au următoarele caracteristici:

Pentru View-ul specific acțiunii *Index*, cea mai importantă abordare (**Figura 17**) a fost:

Modul de afișare a cărților/autorilor în indexul lor a constat prin folosirea grid system-ului pus la dispoziție de către *Bootstrap*. Astfel, populez cu câte 4 cărți/autori pe pagină, iar odată ce s-a atins limita de 12 coloane, trec pe următoarea linie, prin intermediul unui divizor.

```
@{
    var books = Model.ToList();
    <div class="row">
        @for (var index = 0; index < books.Count; ++index)
        {
            var checkIndex = index + 1;
            var displayInfo = "<h4>" + books[index].Title + "</h4><p>" +
            books[index].Description + "</p>";

            <div class="col-md-3 col-3">
                <a asp-action="Details" asp-route-id="@books[index].Id" data-
                toggle="tooltip" data-placement="right" data-html="true" title="@displayInfo">
                    
                </a>
            </div>

            if (checkIndex % 4 == 0)
            {
                <div class="w-100"></div>
                <br />
            }
        }
    </div>
}
```

Figura 17: Afișarea cărților pe pagina principală/Index a controller-ului Books

Pentru View-ul specific acțiunii *Details*, cele mai interesante abordări au fost:

La secțiunea de *Recomandări* - **Figura 18**, m-am folosit de modali. Astfel, am introdus un form care apelează acțiunea din controller, iar la opțiuni are posibilitatea să aleagă o anumită carte pe baza unui selector de tip *Razor*, populat cu ajutorul metodei descrise în **Figura 10**.

```
@{
    var books = BookService.GetAllBooksForRecommendation(Model);
}
<form asp-controller="Books" asp-action="RecommendABook"
class="submitRecommendation" role="form">
    @Html.Hidden("bookId", Model)
    @Html.Hidden("userId", Guid.Parse(user))
    <div class="form-group">
        <div class="col-11 col-md-11">
            @Html.DropDownList("recommendedBookId", books, "Select a book
            to recommend", new
                {
                    @class = "form-control selectpicker",
                    data_live_search = "true"
                })
        </div>
    </div>
</div>
```

Figura 18: Modul de recomandare și felul cum sunt salvate/selectate datele

Pe pagina de *Details* a cărților, există secțiunea ce conține cărțile recomandate de utilizatori și motivele aferente alegerii lor. Această secțiune reprezintă, de fapt, un carusel populat cu cărți, unde fiecare slide este alcătuit din patru cărți.

Modul de afișare/alegere este următorul: verific numărul total de cărți recomandate și le împart în grupuri de câte 4 (3 coloane pentru fiecare carte) astfel, voi avea evidența numărului de slide-uri posibile. Apoi, pentru fiecare grupare de câte 4 le afișez datele respective. Ultima grupare s-ar putea să conțină cel puțin o carte, astfel, voi parcurge restul cărților și le voi afișa în funcție de numărul acestora (astfel, nu toate cărțile vor mai ocupa 3 coloane, ci numărul de coloane este influențat de câte cărți sunt, după formula: $12 / \text{număr_cărți_rămase}$), implementarea fiind realizată în **Figura 19**.

Același procedeu l-am folosit și la caruselul cu cărțile favorite – *Favorites*, disponibil la secțiunea *Library* din pagina cu profilul utilizatorului.

```

<div class="carousel-inner">
  @for (var divNumber = 0; divNumber <= max; ++divNumber)
  {
    var active = "";
    if (divNumber == 0)
    {
      active += " active";
    }

    if (divNumber == max)
    {
      var difference = count - 4 * divNumber;
      <div class="carousel-item @active">
        <div class="row justify-content-center">
          @for (var index = 0; index < difference; ++index)
          {
            var currentIndex = index + 4 * divNumber;
            var book = UtilityService.GetBookById(recommendedBooks[currentIndex].
              BookRecommended);
            var recommendedDetails = "recommendedDetails"
              + recommendedBooks[currentIndex].Id;

            <div class="col-md col recommendedBooks">
              <a data-toggle="modal" data-target="#@recommendedDetails">
                
              </a>
            </div>
          }
        </div>
      </div>
    }
    else
    {
      <div class="carousel-item @active">
        <div class="row justify-content-center">
          @for (var index = 0; index < 4; ++index)
          {
            var currentIndex = index + 4 * divNumber;
            var book = UtilityService.GetBookById(recommendedBooks[currentIndex].
              BookRecommended);
            var recommendedDetails = "recommendedDetails"
              + recommendedBooks[currentIndex].Id;

            <div class="col-md col recommendedBooks">
              <a data-toggle="modal" data-target="#@recommendedDetails">
                
              </a>
            </div>
          }
        </div>
      </div>
    }
  }
</div>

```

Figura 19: Popularea caruselului cu slide-urile specifice, numărul itemelor/slide fiind variabil

Modalitatea de ascundere/afișare a unor content-uri a fost folosită în mai multe rânduri, ca de exemplu: trecerile dintre afișările de statistici pentru rating și chapters, alegerea utilizatorilor în funcție de regiune, alegerea cărților din *Library* în funcție de starea lor sau verificarea *followers/following*. Voi exemplifica acest concept folosindu-mă de secțiunea *followers/following*: fiecare opțiune are o anumită funcție scrisă în JavaScript care ascunde/afișează elemente în funcție de id-ul lor, apoi pentru fiecare componentă am populat cu utilizatorii specifici.

Paginarea disponibilă la *Authors* și *Cărți* a fost realizată prin apelul acțiunii Index care primește ca parametru numărul cărții la care s-a ajuns. Apoi, pe baza acestui parametru, se va da *Skip* la acel număr de cărți și va lua următoarele 12 de cărți, modalitatea de selecție fiind prezentată în **Figura 9**. Codul de selectare, pe View fiind prezentat în **Figura 20**:

```
<div class="row justify-content-center">
  <form asp-action="Index">
    <ul class="pagination">
      @for (var index = 0; index < BookService.GetAllBooks().Count; ++index)
      {
        if (index % 12 == 0)
        {
          var firstValue = index / 10 * 12 + 1;
          if (index == 0)
          {
            firstValue = 1;
          }

          var secondValue = firstValue + 11;

          <li class="page-item">
            <button type="submit" name="pageNumber" class="btn
actionButton" value="@index">@firstValue - @secondValue</button>
          </li>
        }
      }
    </ul>
  </form>
</div>
```

Figura 20: Paginarea realizată pe View

Pe partea de organizare a View-urilor, am optat să le divizez informația oferită în mai multe *Partial Views*, pentru o mai bună gestionare a acestora. Un astfel de exemplu ar fi **Figura 21**.

```

for (var index = 0; index < booksForMood.Count; ++index)
{
    var books = booksForMood[index].Books;

    <div class="row border-top border-bottom">
        <div class="col-md-6 col-6">
            <br />
            @await Html.PartialAsync("IndexPages/_CreatorContent",
booksForMood[index])
        </div>

        <div class="col-md-6 col-6 justify-content-center">
            <br/>
            @await Html.PartialAsync("IndexPages/_DisplayBooksContent",
books)
        <br/>
        </div>
    </div>
    <br />
}

```

Figura 21: Modul de organizare a paginii principale a controller-ului **ListOfBooks**

De menționat faptul că în fișierul *Startup.cs*, unde se regăsesc configurațiile realizate în aplicație, cum ar fi: string-ul de conectare către baza de date, adăugarea de servicii în funcție de layer sau conectarea la API-urile puse la dispoziție de rețelele de socializare, datele folosite pentru conectare fiind salvate într-un fișier disponibil local – *UserSecrets.json*.

Un alt subiect important pentru acest capitol ar fi *Securitatea*. În cadrul aplicației, aceasta este realizată prin intermediul sistemului de autentificare pus la dispoziție de către *Identity*. Astfel, modul de salvare a utilizatorilor în baza de date le asigură siguranța și confidențialitatea datelor, deoarece parola acestora este hashuită.

O altă posibilitatea oferită de *Identity* ar reprezenta și posibilitatea de adăugarea a rolurilor pentru utilizatori. Astfel, putem restricționa accesul în funcție de aceste roluri, oferind o siguranță în plus când vine vorba de integritatea datelor salvate, un exemplu fiind ilustrat în **Figura 22**.

```

[HttpPost]
[Authorize(Roles = "Owner, Administrator")]
public IActionResult CreateNews(string content)

```

Figura 22: Restricționarea posibilității de creare a unui anunț, fiind posibilă doar utilizatorilor cu rolurile indicate mai sus

Capitolul III – Manual de utilizare

În acest capitol voi face o prezentare generală a aplicației, sub forma unui manual de utilizare.

De observat faptul că pentru utilizatorii neconectați, majoritatea paginilor sunt disponibile pentru vizualizare, dar cantitatea de informații/opțiuni este limitată.

1. Autentificare:

Autentificarea se împarte în trei posibilități: prin login, register sau logare externă – pe baza unor rețele de socializare (opțiunile disponibile fiind: *Facebook*, *Twitter* și *Google+*).

De menționat faptul că autentificarea nu este influențată de rolurile disponibile în aplicație, fiind o logare generală pentru ambele tipuri de conturi. Modul de afișare constă în apariția unui modal ce prezintă opțiunile disponibile, cum reiese din **Figura 23**.

Login:

Se realizează odată ce utilizatorul a completat cu succes cele două câmpuri: *email* și *password*.

Register: necesită introducerea unor date suplimentare, cele mai importante fiind *email-ul* și *username-ul*. Pe baza acestora se va face identificarea contului, așadar sunt unice.

The image displays two side-by-side user interface modals for account management.

Left Modal: Create an account

- Fields: Username, Email, Password, Confirm password.
- Buttons: Register, and three social login buttons (facebook, google, twitter) under the heading "Or consider using..".

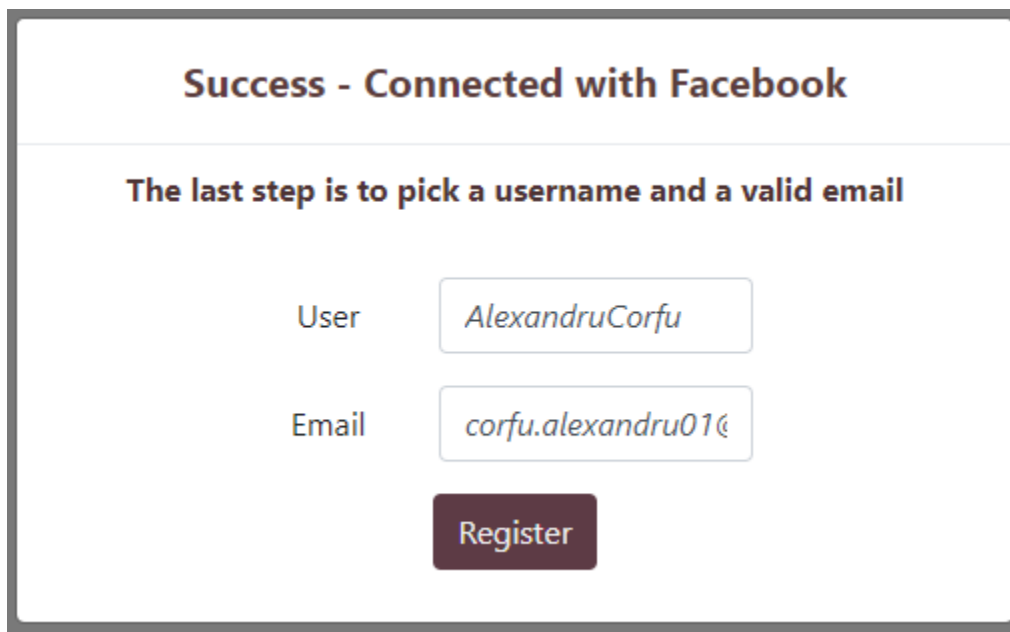
Right Modal: Login using

- Fields: Email (pre-filled with *explore@books.com*), Password.
- Buttons: Login, and three social login buttons (facebook, google, twitter) under the heading "Or consider using..".
- Footer: "Don't have an account? Register here".

Figura 23: Modalul afișate pentru opțiunile de Register & Login

External register/login:

Odată accesată una din opțiuni cu privire la logarea externă, utilizatorul va fi redirecționat spre un modal ce îi va cere să confirme faptul că informațiile disponibile pe respectiva rețea să fie folosite și în intermediul aplicației, după cum se observă în **Figura 24**.



The screenshot shows a registration confirmation interface. At the top, it says "Success - Connected with Facebook". Below that, it states "The last step is to pick a username and a valid email". There are two input fields: "User" with the text "AlexandruCorfu" and "Email" with the text "corfu.alexandru01@". A dark red "Register" button is positioned below the email field.

Figura 24: Display-ul pentru opțiunea de External register/login

Odată realizată înregistrarea, utilizatorul va primi un mail de notificare din partea aplicației.

La autentificarea cu succes - utilizatorul va fi redirecționat pe pagina principală, iar în caz contrar – va primi mesaje de avertisment corespunzătoare.

2. Pagina principală:

Structurată în trei componente principale, modul de vizualizare regăsindu-se în **Figura 25**:

General announcements: unde administratorii pot face anunțuri, fiecare utilizator fiind notificat pe baza acestora.

Discover books: componenta este structurată în două părți: un carusel alcătuit din cele mai populare cărți (în funcție de numărul utilizatorilor ce citesc cartea respectivă) și o a doua componentă ce permite căutarea în funcție de genurile disponibile în aplicație. Cele două părți sunt divizate de către *Application description*.

Application description: conține o scurtă descriere a aplicației.

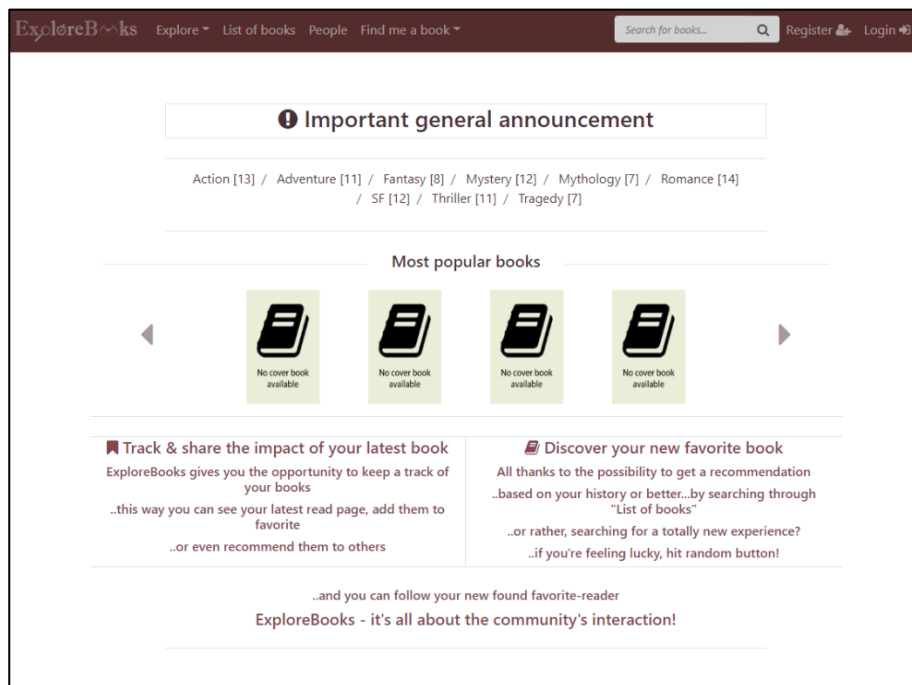


Figura 25: Display-ul pentru pagina principală

3. User's:

Reprezintă meniul afișat utilizatorului odată ce s-a autentificat, iar opțiunile disponibile sunt afișate odată ce acesta va accesa numele sau iconița din bara de navigație, conform cu **Figura 26**.

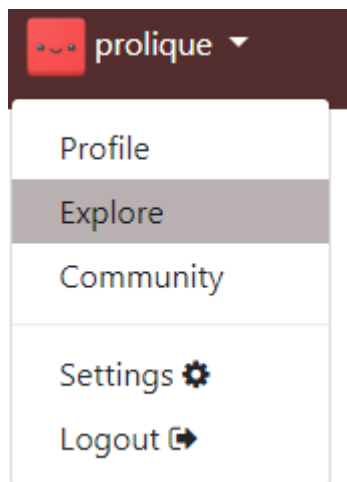


Figura 26: Opțiunile disponibile odată ce s-a accesat numele/iconița

3.1. Profile:

De menționat faptul că profilul fiecărui utilizator este public .

Activity: prezintă un scurt istoric al activității utilizatorului legate de ultimele cărți adăugate de acesta, observându-se în **Figura 27**.

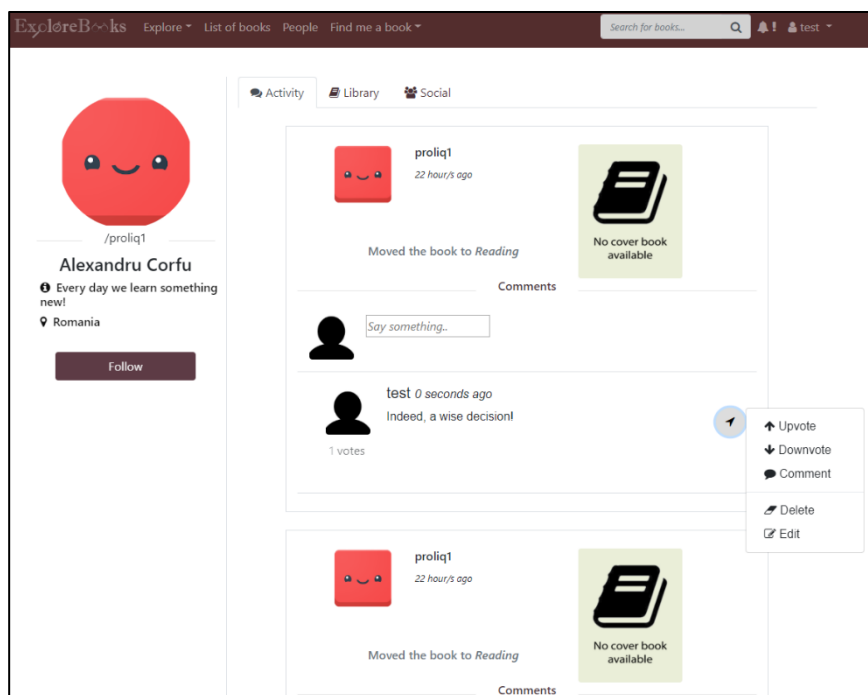


Figura 27: Display-ul pentru pagina de activitate

Library: conține o împărțire a cărților pe diverse categorii, utilizatorul având posibilitatea de a naviga printre acestea, conform cu **Figura 28**. La secțiunea **Favorites** sunt afișate toate cărțile preferate de utilizator, modul de afișare fiind printr-un carusel care conține 4 cărți/slide.

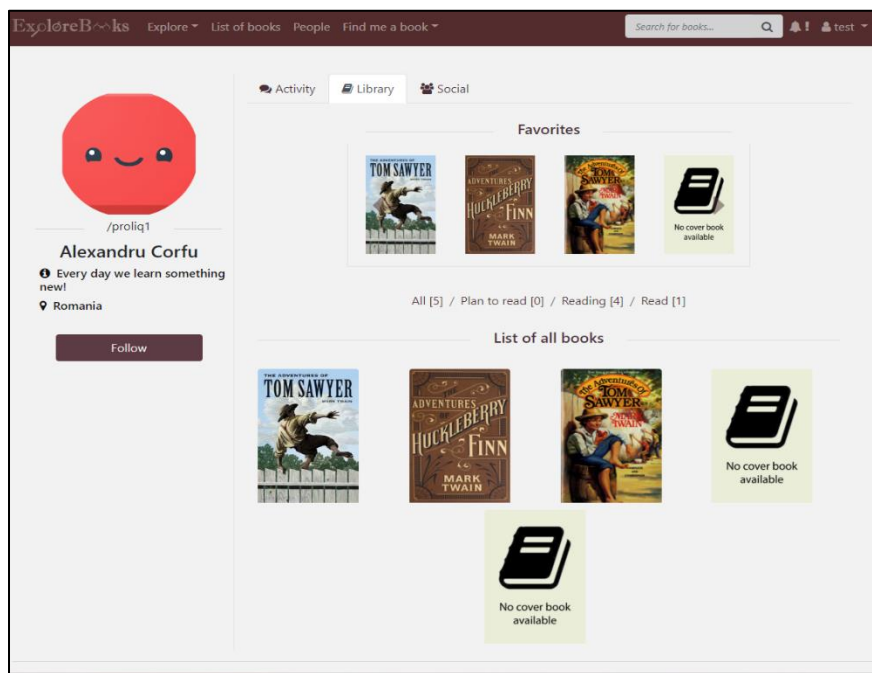


Figura 28: Display-ul pentru pagina de Library

Social: reprezintă o cale de acces și vizualizare a tuturor utilizatorilor care sunt interesați de utilizatorul curent, modul de afișare fiind prezentat în **Figura 29**.

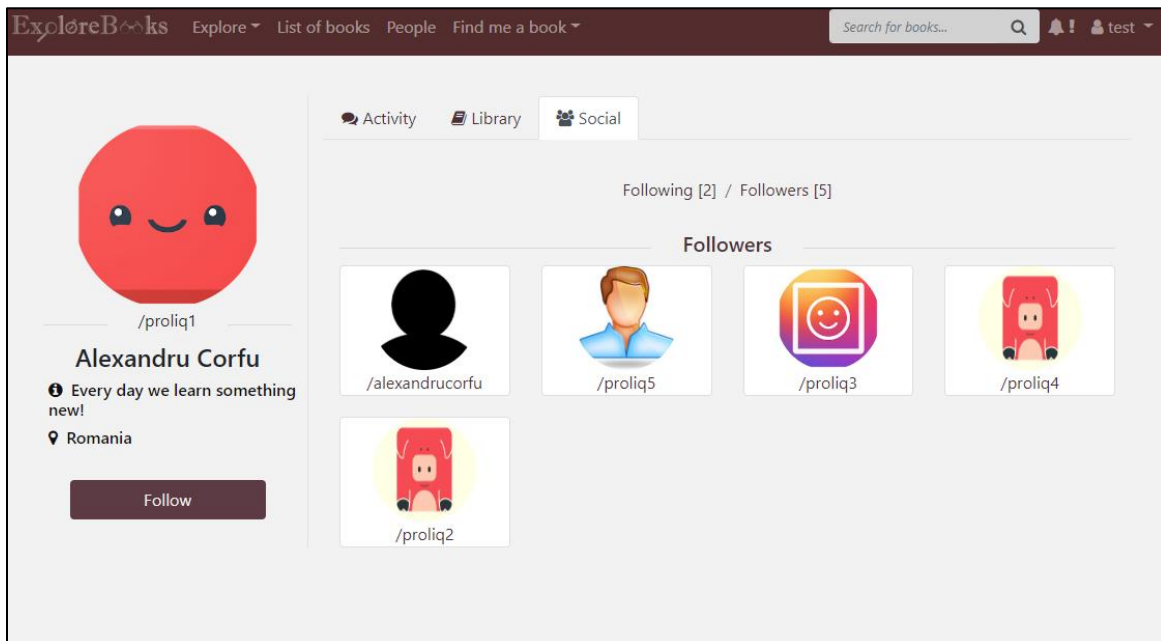


Figura 29: Display-ul pentru pagina de profil

Settings:

The image shows a 'Settings' form for updating a user profile. It contains several input fields: 'FirstName' with the value 'prolique', 'LastName' (empty), 'Country' with the value 'Somewhere on the internet', 'Username' with the value 'prolique', and 'Email' with the value 'test@test.com'. Below these is a text area for 'Describe yourself in few words..' containing the text 'It's a mystery..'. At the bottom of the form is a dark 'Update profile' button.

Figura 30: Display-ul pentru modificarea tab-ului de Profile

Această secțiune acoperă toate setările private ale utilizatorului, având posibilitatea să modifice datele afișate pe pagina de profil (în afară de *username* & *email*, după cum se poate observa în **Figura 30**), să modifice parola/să o seteze (în cazul autentificării externe) sau să își schimbe poza de profil.

Delete account: reprezintă o opțiune aparte care, conform GDPR-ului, îi permite oricărui utilizator să șteargă orice tip de informație legat de acesta, după cum reiese din **Figura 31**.

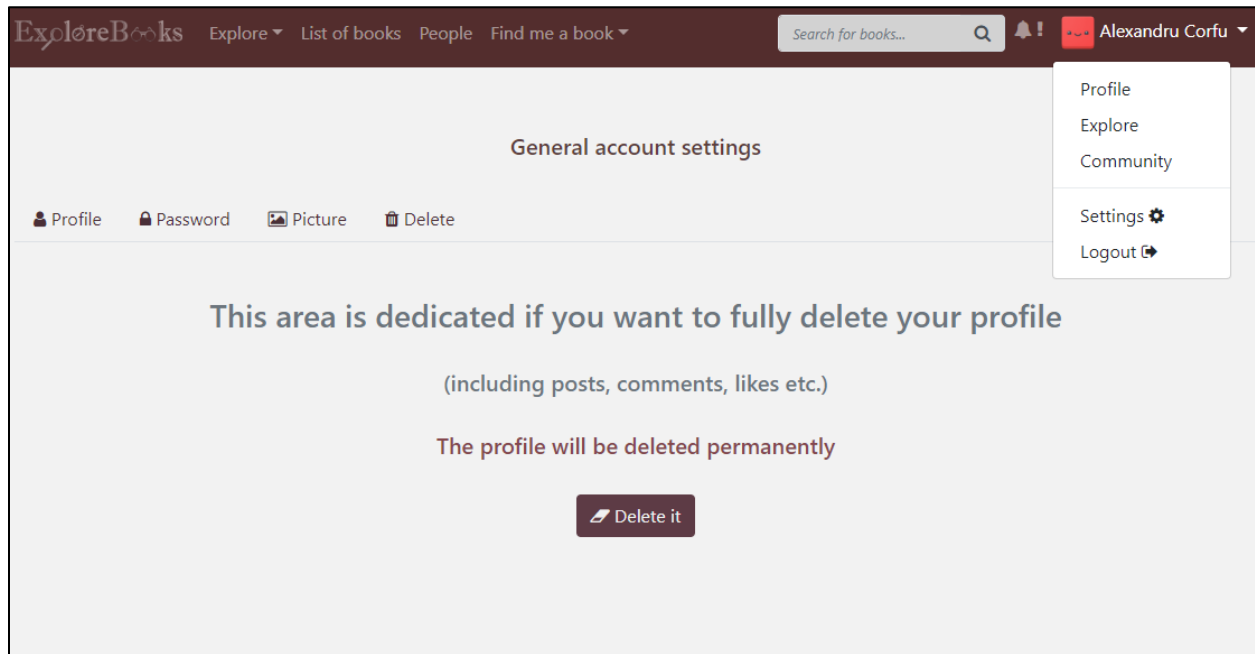


Figura 31: Display-ul cu toate opțiunile disponibile din cadrul componentei **Settings**, fiind selectată opțiunea pentru ștergerea contului

4. Find a book:

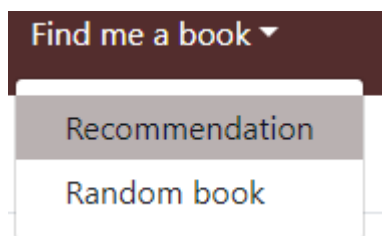


Figura 32: Display-ul opțiunilor disponibile pentru **Find me a book**

Reprezintă secțiunea de recomandări realizate de aplicație, afișate în **Figura 32**. Astfel, în funcție de opțiunea aleasă de către utilizator, aplicația va returna o carte aleasă complet la întâmplare (*Random book*) sau pe baza unor criterii (dacă are un istoric, atunci va căuta o carte asemănătoare, altfel, va primi una din cele mai apreciate cărți⁴).

⁴ Cea mai apreciată carte este determinată de media notelor acordate de utilizatori cărții respective

5. List of books:

Dacă secțiunea precedentă acoperea recomandările făcute de aplicație, în această secțiune este vorba de una din posibilele recomandări pe care un utilizator le-ar putea face. Astfel, va avea posibilitatea de a crea o listă cu cărți, selectate pe diverse criterii, utilizatorii având dreptul de a le vizualiza, a își exprima opinia sau vota cărțile respective, un exemplu fiind în **Figura 33**.

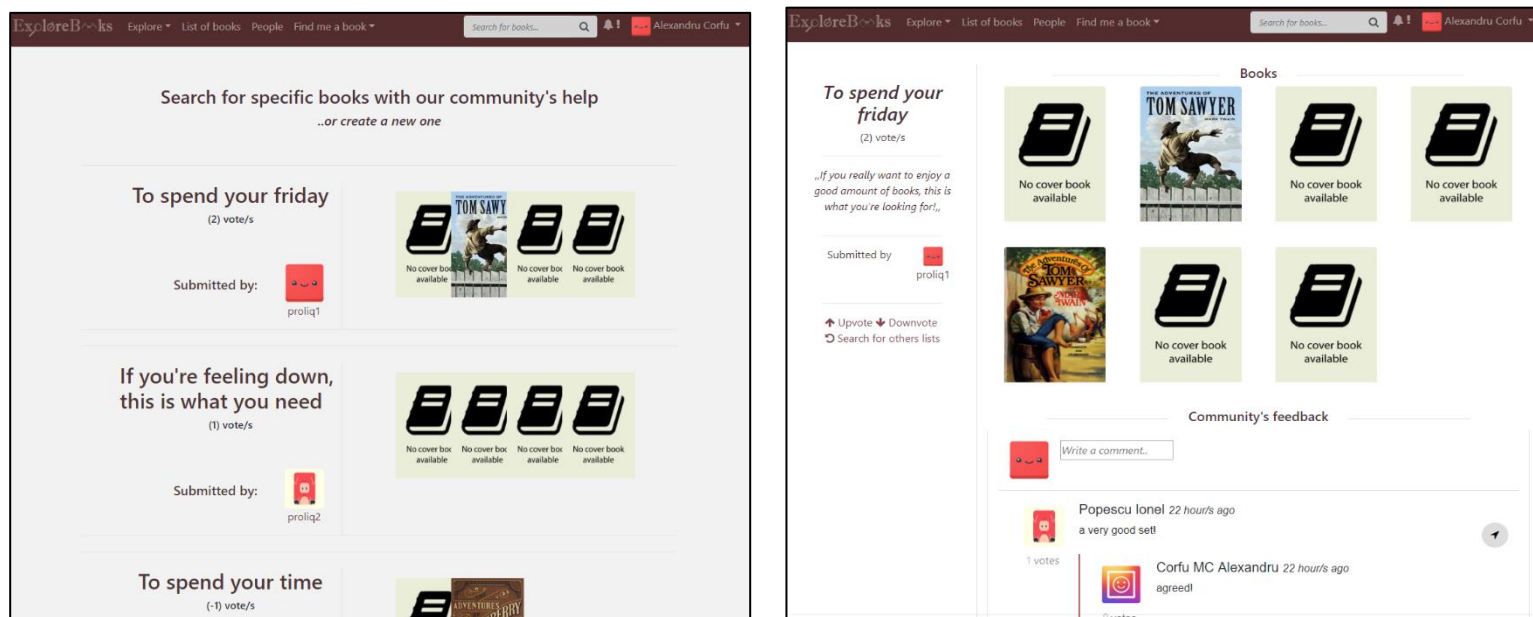


Figura 33: Display-ul pentru pagina principală (în stânga) și detaliile aferente selecției făcute (în dreapta)

6. Explore:

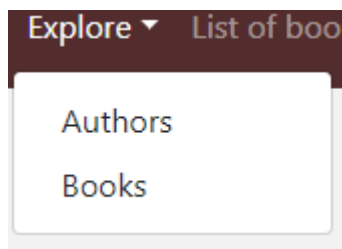


Figura 34: Display-ul pentru opțiunile disponibile în Explore

Secțiunea principală a aplicației, unde utilizatorii pot căuta anumite cărți sau autori - **Figura 34**.

Odată selectat tipul dorit, va apărea o listă cu obiectele disponibile selecției efectuate, având posibilitatea de a vizualiza mai multe detalii aferente unei entități de acel tip, după cum se observă în **Figura 35**.

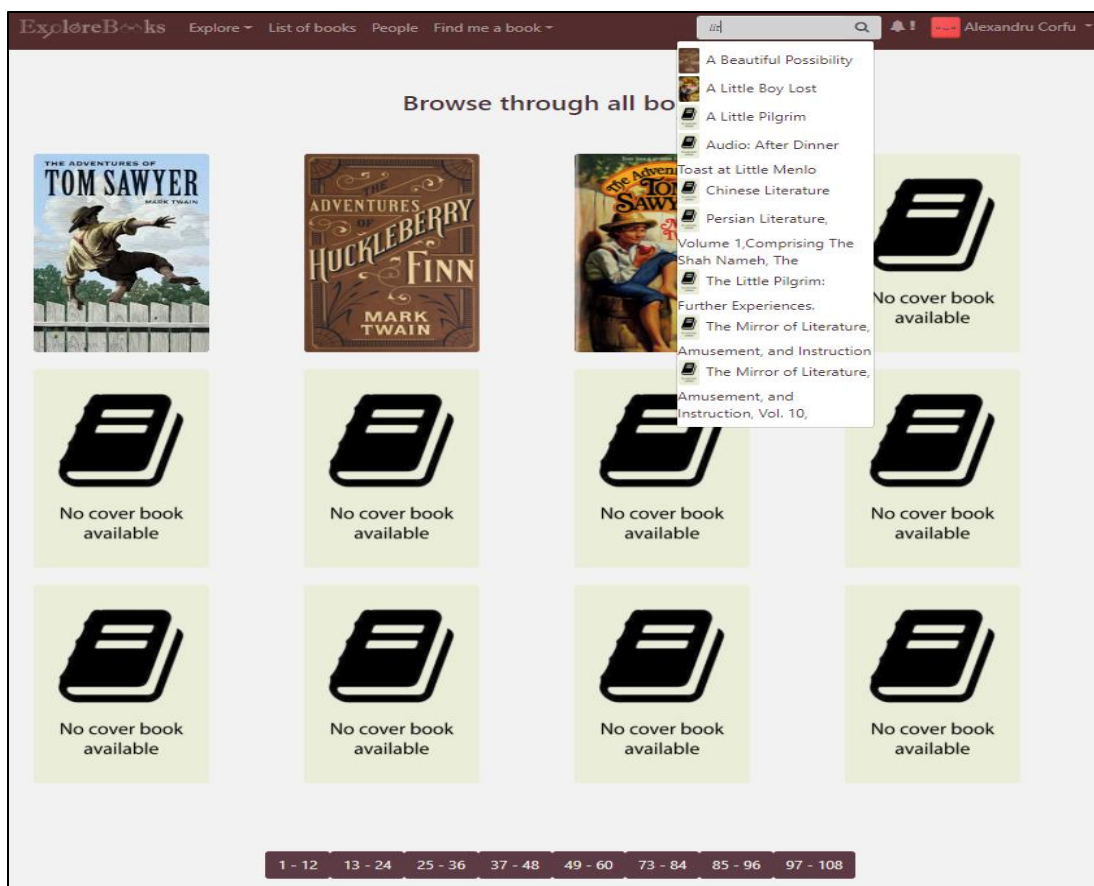


Figura 35: Display-ul paginii principale a cărților (același principiu fiind aplicat și la autori) și afișarea cărților disponibile prin intermediul search bar-ului

6.1. Books

Conține detalii legate de carte, de la review-urile și statisticile obținute de la utilizatori (**Figura 36**), până la posibilitatea de a o citi direct din aplicație sau lăsa un comentariu cărții respective (**Figura 37**).

Statisticile se realizează pe baza activității utilizatorilor legată de respectiva carte, prin notele acordate, capitolele preferate sau prin starea cărții: *Reading*, *Read* sau *Plan to read*.

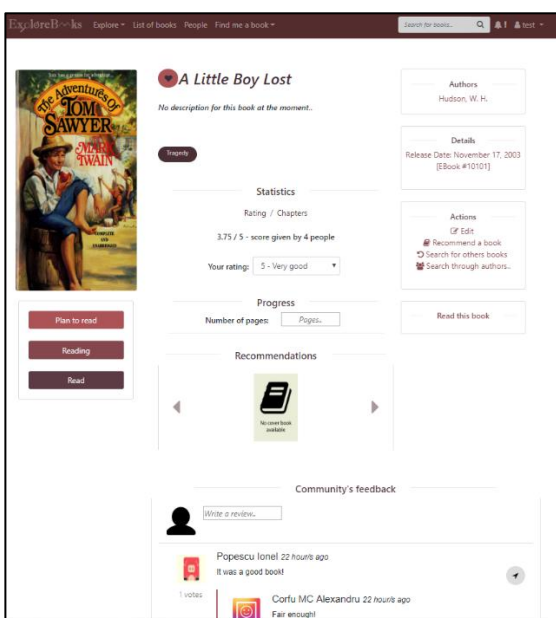


Figura 36: Afișarea paginii specifice unei cărți

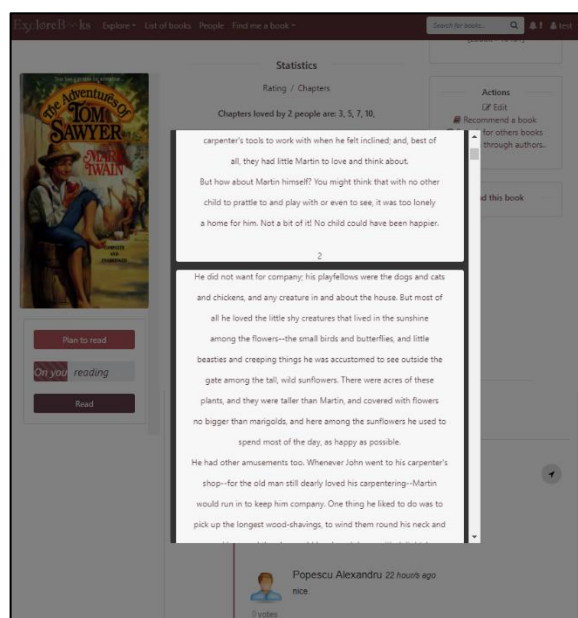


Figura 37: Afișarea textului cărții selectate

6.2. Authors

Pe pagina cu detalii specifice unui anumit autor apare o descriere (dacă există) pentru respectivul autor și cărțile redactate de acesta. Mai mult, fiecare utilizator va putea să-și lase opinia, după cum reiese din **Figura 38**.

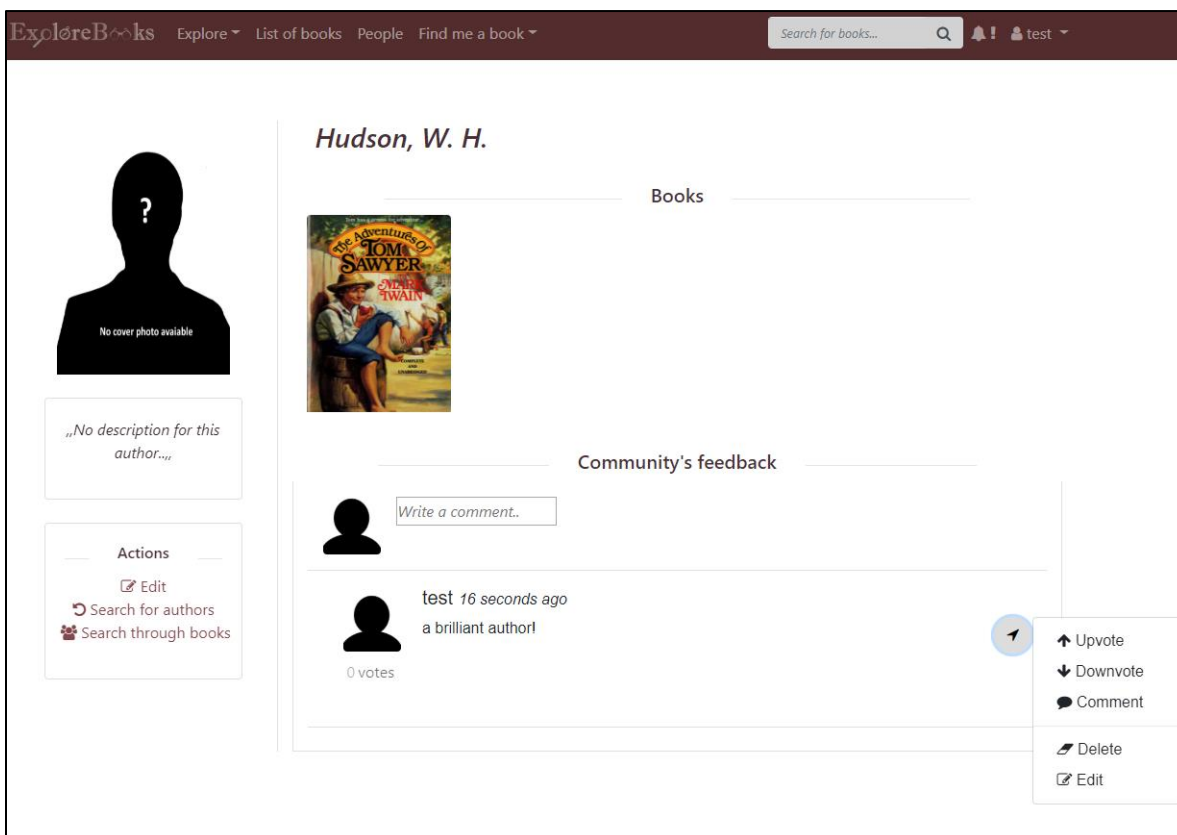


Figura 38: Display-ul pentru pagina cu detaliile unui autor

De menționat faptul că administratorul are posibilitatea de a modifica informațiile afișate.

O altă menționare ar fi faptul că pentru a vizualiza mai multe detalii despre o anumită carte sau un anumit autor/utilizator, se poate da hover peste imaginea acestora, afișându-le informațiile, după caz.

7. People:

Reprezintă secțiunea ce permite căutarea utilizatorilor înscriși în aplicație.

Astfel, utilizatorul conectat, pe baza țării introduse, va avea opțiunea de a vizualiza toți utilizatorii sau doar pe cei din țara acestuia – **Figura 39**.

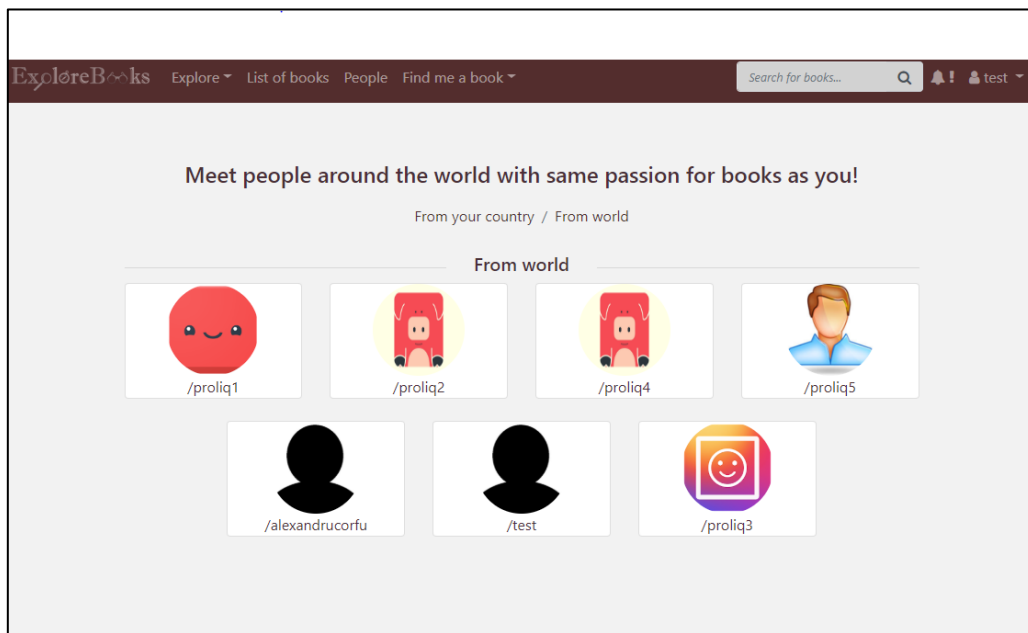


Figura 39: Display-ul pentru pagina **People**

8. Erori:

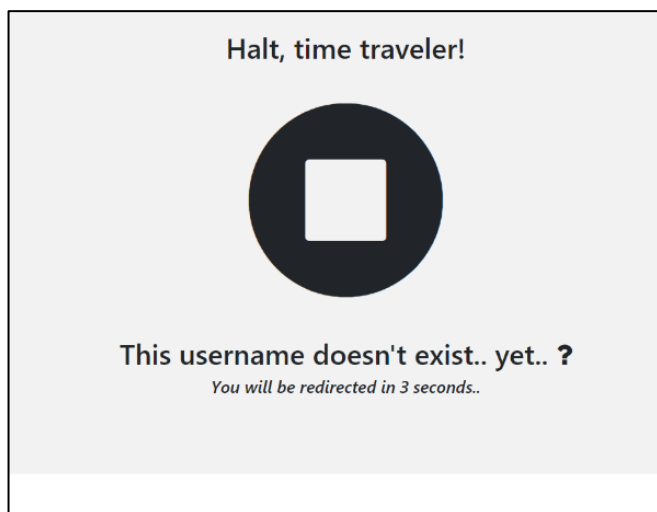


Figura 40: Mesajul de eroare în cazul introducerii unui username invalid

Pentru un mai bun mod de desfășurare a aplicației, am inclus și pagini de eroare în caz că un utilizator introduce ca și adresă o valoare invalidă (*username* sau id-ul unui obiect care momentan nu există). Pagina de eroare, după 3 secunde, va redirecționa utilizatorul către pagina principală (**Figura 40**).

Concluzii

Consider că aplicația „ExploreBooks” într-adevăr poate facilita viața oricărui utilizator ce o folosește, îmbunătățirea experienței acestuia fiind datorată unor funcționalități cum ar fi: posibilitatea citirii oricărei cărți direct din aplicație, sistemul de review-uri pentru cărți (fie că este vorba de cel automat – realizat de aplicație, fie cel bazat pe comunitate) sau posibilitatea și felul cum este urmărit progresul acestuia legat de o anumită carte.

Deși pe parcursul capitolelor anterioare am prezentat abordarea tehnică și logică în elaborarea aplicației, am întâmpinat de asemenea și multe dificultăți, cum ar fi:

Pe partea bazelor de date, am avut probleme cu importul cărților prin intermediul API-ului celor de la *Project Gutenberg*. Astfel, am fost nevoit să apelez la *wget* pentru a extrage cărțile respective, urmând să prelucrez fiecare arhivă cu conținutul acestora (descrierea modului de prelucrare se regăsește în secțiunea de *Populare a bazei de date*, din capitolul *Detalii de implementare*). Un alt impediment pe partea populării bazei de date s-a datorat lipsei detaliilor aferente cărților descărcate. *Project Gutenberg* pune la dispoziție un set limitat de detalii ale cărților, cele mai importante fiind: numele autorului și titlul cărții. Dar conform structurii plănuite, mai aveam nevoie de genul cărților, numărul de pagini sau o scurtă descriere a acestora, astfel am fost nevoit să le inițializez la creare cu valori „default”, iar editarea ulterioară a acestora se putea realiza oricând de administrator.

Pe partea de back-end, cel mai notabil impediment a apărut la adăugarea rolurilor pentru utilizatori. Din păcate, aproape toate tutorialele/explicațiile cu privire la adăugarea de roluri pentru *Identity* sunt realizate pentru versiunea de *ASP.NET Core 1.0*. Cum abordarea la *ASP.NET Core 2.0* este diferită, durata de testare a diverselor abordări a fost de câteva zile, într-un final realizându-se cu succes.

Iar pe partea de front-end – abordarea în sine. În general front-ul a reprezentat o slăbiciune pentru mine, mai ales la partea de aranjare corectă a elementelor într-o pagină. Astfel, am apelat la *Bootstrap* care pune la dispoziție un sistem de grid-view pentru o mai bună așezare a elementelor, dar și pentru diversele clase css care m-au scutit de multe dificultăți și mai ales a scalabilității pe diverse rezoluții oferite. O altă problemă a constatat în îmbinarea elementelor *HTML*

clasice cu cele puse la dispoziție de *Razor*. La început a fost grea adaptarea, dar pe parcurs multe lucruri au devenit mai clare și mai ușor de utilizat.

1. Direcții de viitor

Datorită structurării și modalității de implementare a aplicației, aceasta poate fi dezvoltată în viitor, câteva dintre îmbunătățiri fiind:

În primul rând, numărul cărților disponibile în aplicație. În 2019, *Project Gutenberg* va adăuga un număr relativ mare de noi cărți a căror drepturi de autor urmează să expire. Astfel, s-ar putea popula baza de date cu mai multe exemplare din acestea. Tot legat de populare, s-ar putea adăuga opțiunea de adăugare a cărților de către utilizatori, direct din aplicație. Desigur că noile cărți adăugate vor avea drepturile de autor expirate, urmând să existe posibilitatea de verificare dacă într-adevăr au expirat.

În al doilea rând, datorită API-urilor celor de la *Facebook*, *Twitter* sau *Google+*, s-ar putea oferi posibilitatea de a distribui o anumită carte/opinie legată de aceasta pe una din aceste rețele de socializare.

O altă îmbunătățire ar consta în adăugarea unui forum și grupuri pentru utilizatori. Astfel, pe acestea, un utilizator ar putea posta diverse subiecte sau întrebări legate de cărți. Tot pe această parte de comunicare, s-ar putea implementa și un messenger între utilizatori, oferind o modalitate de comunicare în aplicație într-un mod privat.

2. Concluzie finală

În urma realizării acestei aplicații, pot spune că sunt mulțumit de felul cum a ieșit, de la funcționalitate până la design-ul acesteia.

Deși la început am pornit cu un set de idei legat de funcționalitatea aplicației, pe măsură ce mă loveam de diverse obstacole sau alternative cu privire la acestea, am descoperit că funcționalitățile respective pot fi îmbunătățite, chiar se pot adăuga și altele. Astfel, s-a ajuns la acest produs final. Mai mult, am avut parte de o evoluție tehnică impresionantă pe parcursul acesteia, asimilând diverse concepte de programare (stăpânirea unor concepte legate de arhitectural patterns) sau diverse tehnologii (o mai bună înțelegere a JavaScript-ului, AJAX-ului), dar și modalități de îmbinare ale acestora.

Bibliografie

[1] Site-ul **Microsoft** pentru informații legate ASP.NET Core:

<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1>

[2] Site-ul **Microsoft** pentru informații legate C# & .NET framework:

<https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>

[3] Site-ul **Microsoft** pentru informații cu privire la folosirea Identity-ului pentru logarea cu rețelele de socializare:

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/?view=aspnetcore-2.1>

[4] Site-ul **Microsoft** pentru informații legate de ASP.NET Core MVC:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1>

[5] Site-ul **Microsoft** pentru informații legate de Entity Framework Core:

<https://docs.microsoft.com/en-us/ef/core/>

[6] Site-ul **Microsoft** pentru informații legate de rutarea în ASP.NET Core MVC:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-2.1>

[7] Site-ul **Microsoft** pentru modalități de implementare a pattern-ului Onion Architecture:

<https://social.technet.microsoft.com/wiki/contents/articles/36655.onion-architecture-in-asp-net-core-mvc.aspx>

[8] Site-ul **Bootstrap** pentru documentație:

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>

[9] Site-ul **Project Gutenberg** pentru popularea bazei de date cu cărți:

<https://www.gutenberg.org/>

[10] Site-ul **wget** – pentru extragerea unui număr mai mare de cărți de pe site-ul precedent:
<https://www.gnu.org/software/wget/>

[11] Site-ul **Vertabelo** – l-am utilizat în schițarea bazei de date cu privire la felul cum urmează să relaționeze entitățile: <https://www.vertabelo.com/>

[12] Site-ul **w3schools** – pentru documentarea legată de HTML:

https://www.w3schools.com/Html/html_intro.asp

[13] Site-ul **w3schools** – pentru documentarea legată de CSS:

https://www.w3schools.com/css/css_intro.asp

[14] Site-ul **w3schools** – pentru documentarea legată de AJAX:

https://www.w3schools.com/js/js_ajax_intro.asp

[15] Site-ul **w3schools** – pentru documentarea legată de jQuery:

https://www.w3schools.com/Jquery/jquery_intro.asp

[16] Site-ul **fontawesome-ului** – utilizată pentru a căuta diverse iconițe, în funcție de context:

<https://fontawesome.com/icons?from=io>

Anexa 1 – Tehnologiile utilizate

În elaborarea prezentei lucrări, am apelat la următoarele tehnologii:

1. Server-side:

C#: reprezintă un limbaj orientat obiect ce permite crearea unui număr mare și diversificat de aplicații.

Spre deosebire de limbajele clasice orientate obiect, C# oferă componente noi, ce facilitează lucrul, una din acestea fiind: *LINQ-ul – Language Integrated Query* ce permite crearea și executarea de query-uri asupra unor tipuri de date.

ASP.NET Core: este un framework folosit pentru crearea aplicațiilor moderne, acesta fiind o implementare a *.NET-ului* standard. Spre deosebire de framework-ul original, are următoarele avantaje: *Cross-platform* (are posibilitatea de a fi rulat pe mai multe platforme - MacOS, Linux etc.), *Open source* (oricine are acces la cod, astfel, comunitatea are un rol important în evoluția acestuia) și *Modular*.

ASP.NET Core Identity: adaugă posibilitatea de a crea membri, prin implementarea unui sistem de autentificare. În plus, oferă posibilitatea de a adăuga logarea externă, prin diferite aplicații cum ar fi: *Facebook, Microsoft* etc. Un alt avantaj îl reprezintă faptul că poți adăuga roluri utilizatorilor, pentru un mai bun management al aplicației.

ASP.NET Core MVC: reprezintă un framework folosit pentru a construi aplicații web folosind design pattern-ul de tip *MVC (Model – View – Controller)*. Acest framework pune la dispoziție folosirea paginilor *Razor*, ce ajută la adăugarea unor componente de tip c# direct în pagina HTML, aceste pagini ajungând să conțină extensia *.cshtml*.

Model-View-Controller: separă aplicația în trei componente principale: *Controllerul* – pentru management-ul de request-uri și modificarea corespunzătoare a view-urilor și modelelor, *View* – reprezintă ceea ce interacționează în mod direct cu utilizatorul, ceea ce vede acesta, iar *Modelul* – folosit pentru business logic-ul la nivel de aplicație, asigurând transferul de date dintre componente.

Entity Framework Core: este un framework O/RM (*Object-relational mapper*) ce permite lucrul cu baza de date folosind obiecte/entități .NET, modul de comunicare fiind evidențiat în **Figura 41** de mai jos⁵.

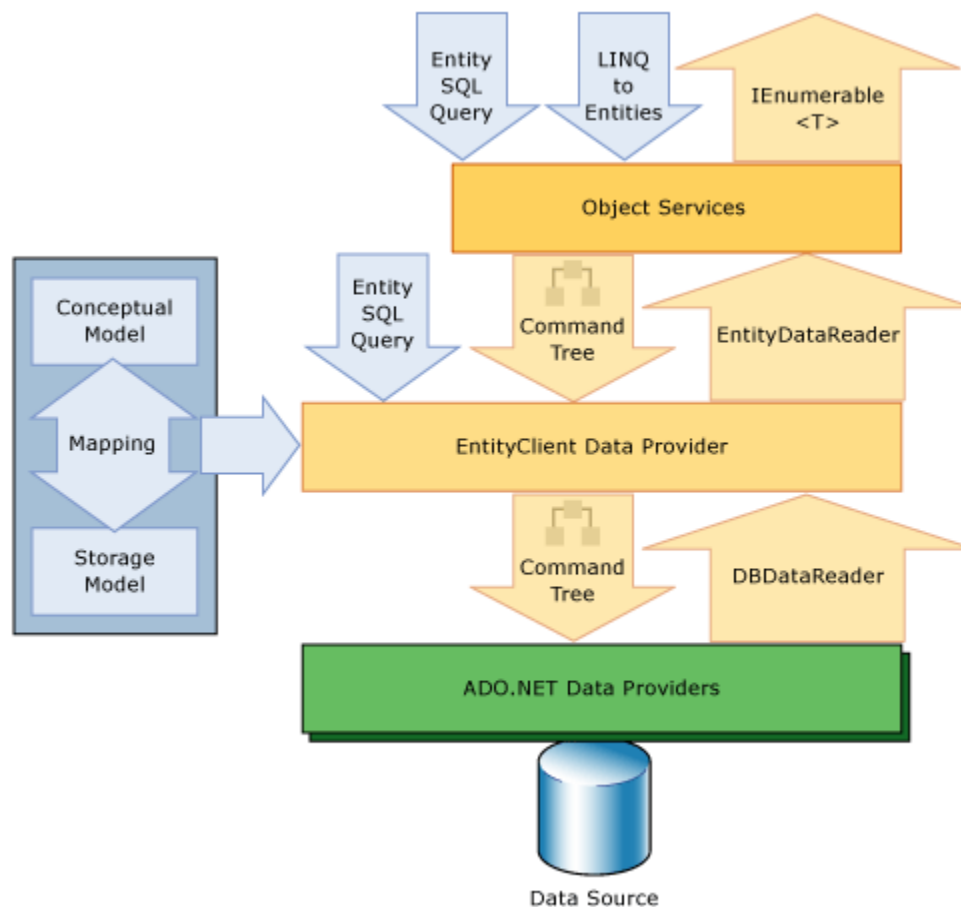


Figura 41: Arhitectura Entity Framework pentru accesul datelor

2. Baza de date:

SQL Server Express: permite o mai bună relaționare a bazelor de date, permițând stocarea și distribuirea datelor în funcție de solicitările efectuate asupra acestora.

⁵ **Sursă:** <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview>

3. Client-side:

HTML (Hyper Text Markup Language): limbaj de marcare folosit pentru a crea pagini web.

CSS (Cascading Style Sheets): folosit pentru modificarea felului cum sunt arătate în pagină elementele HTML.

JavaScript: reprezintă un limbaj de programare, fiind folosit atât pentru dezvoltarea unor pagini web mai interactive, dar este folosit și pentru anumite programe fie pe partea de server, cel mai cunoscut fiind *Node.js*, fie pentru anumite baze de date, cum ar fi *MongoDB*.

jQuery: este o librărie *JavaScript* folosită pentru a ușura și îmbunătăți anumite procese, câteva dintre ele fiind: o mai bună manipulare a *HTML* & *CSS*-urilor, dar și pentru manipularea evenimentelor sau apelurilor *AJAX*.

AJAX (Asynchronous JavaScript And XML): permite paginilor web să facă modificări asincrone de date, prin transmiterea/modificarea datelor de la un document HTML și o aplicație ce se află pe server – astfel, se pot modifica doar porțiuni din pagina web, fără a reîncărca întreaga pagină.

Bootstrap 4: framework pe front-end folosit pentru a crea aplicații responsive, oferind diverse clase *CSS* și *HTML* pentru o mai bună stilizare a paginilor. Cu ajutorul *jQuery*-ului și *Popper.js*-ului, Bootstrap-ul pune la dispoziție mai multe componente, cele mai cunoscute fiind: carousel-ul, modal-urile și tooltip-urile.

Acesta se folosește de un grid system, pe fiecare linie fiind un număr de 12 celule, unde permite aranjarea elementelor pe oricare din aceste celule.

4. APIs:

API-urile puse la dispoziție de către *Facebook*, *Twitter* și *Google+*, permit comunicarea cu diverse aplicații, de la posibilitatea de autentificare până la cea de distribuire a unor materiale de pe acea aplicație.

API-ul pus la dispoziție de către *Project Gutenberg* oferă posibilitatea de a descărca cărți din arhiva acestora, arhivă ce la momentul de față ajunge pe la aproximativ 57.000 de cărți. Cărțile pot fi descărcate în diverse formate: html, txt, pdf etc. Pentru a putea descărca automat, va fi nevoie fie de crearea unui script, fie de utilizarea diverselor aplicații – de exemplu *wget*, care permit descărcarea automată folosind *HTTP*, *HTTPS*, *FTP* sau *FTPS* (în funcție de caz).

5. Version control:

Github: este un serviciu ce permite management-ul codului sursă. Fiind un serviciu de tip version control, urmărește toate schimbările efectuate la nivelul fișierele distribuite cu alți oameni. Astfel, se pot modifica, salva sau crea proiecte de tip open-source sau private.

De menționat faptul că tot codul este hostat pe pagina acestora.

Ca și GUI pentru utilizarea git-ului, am optat pentru **SourceTree**⁶.

⁶ Sursă: <https://www.sourcetreeapp.com/>