

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Library Management

propusă de

Marcu Pălii

Sesiunea: *iulie, 2019*

Coordonator științific

Lect.dr. Alex Moruz

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Library Management

Marcu Pălii

Sesiunea: *iulie, 2019*

Coordonator științific

Lect.dr. Alex Moruz

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 „*Library Management*”, 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 *Marcu Pălii*

(semnătura în original)

Cuprins

Introducere	7
Motivație	7
Soluție existentă.....	8
Soluție propusă	9
Funcționalități	9
Structura lucrării	12
Contribuții	13
1 Fundamentele teoretice ale aplicației	15
1.1 Algoritmul „stable matching” clasic al lui „Gale–Shapley” (2)	15
1.2 Variații ale algoritmului de „stable matching” :	17
1.2.1 „Stable matching” cu liste incomplete de preferințe (3)	17
1.2.2 „Stable matching” cu liste de preferințe de tip „ties” (3).....	17
1.3 Algoritmul de „stable matching” utilizat de aplicație	18
2 Integrarea algoritmului de „stable matching” in aplicația web.....	20
2.1 Tehnologii folosite	20
2.2 Comunicarea tehnologiilor si rolul îndeplinit de acestea	21
2.3 Dificultăți întâlnite	23
3 Structura aplicației web	26
3.1 Prezentare.....	26
3.2 Folosirea „design pattern”-ul „MVC”	27
4 „Back-end”-ul aplicației web	29
4.1 Tehnologii folosite	29
4.2 „Design”-ul bazei de date	29
4.3 Procesarea „request”-urilor și returnarea răspunsului corespunzător.....	34
4.4 Dificultăți întâlnite	35
5 „Front-end”-ul aplicației web.....	36
5.1 Tehnologii folosite	36
6 Manual de utilizare	37
6.1 Navigarea prin aplicație.....	37
6.2 Căutarea de cărți, rezervarea acestora sau adăugarea în „wishlist”	38
6.3 Interpretarea statisticilor oferite	38
6.4 Acceptarea cărților asignate din lista de preferințe	39
7 Concluzii	41
Bibliografie.....	42
Index imagini.....	43

Introducere

Motivație

În aceasta lucrare am abordat una dintre cele mai frecvente probleme existente, din perspectiva unui cititor, la o bibliotecă. Inconveniența este reprezentată printr-o administrare inefficientă a împrumutului de cărți. Acest lucru este mai accentuat în cazul acelor cărți ce sunt într-un număr relativ redus în comparație cu cei ce doresc să aibă acces la o anumită carte.

Un raport publicat de „Biblioteca Centrala Universitară” din Iași în anul 2018 evidențiază o scădere dramatică a numărului de cititori *„În ultimii zece ani, numărul utilizatorilor prezenți în bibliotecile universitare a scăzut constant și dramatic. Pe fondul declinului demografic care afectează România, scăderea efectivelor de studenți a însemnat și diminuarea cu 45% a numărului de utilizatori ai BCU „Mihai Eminescu”, de la 18.917, în anul 2009, la 10.444 utilizatori, în 2018”* (1). În urma acestui raport ar trebui să se încerce o diminuare al acestui efect, prin identificarea cauzelor care au dus la scăderea numărului de cititori.

Resursele tipărite ajung să fie din ce în ce mai neglijate, astfel încât cele de tip electronic sunt preferate datorită accesului facil la acestea. Există o concurență intensă între cărțile tipărite și cele virtuale, acestea din urmă ajungând să câștige teren pe măsură ce tot mai multe resurse existente ajung să aibă echivalent electronic. Cu toate acestea, colecțiile digitale nu sunt întotdeauna surse de încredere, informațiile transmise pot să difere de realitate, ajungând să dezinformeze.

Statul la rând, poate fi incomod pentru cititori, uneori chiar este inefficient în situațiile în care după perioada de așteptare ajung să afle că resursa dorită de persoanele în cauză, nu se află în colecția bibliotecii. Cu toate că resursa se află în catalog ea poate să nu fie disponibilă, fiind în acel moment împrumutată de alți cititori.

Unele dintre acestea încearcă să diminueze inconveniența prin existența unui catalog online, prin care se poate vedea dacă biblioteca dispune de carte respectivă, ba mai mult se poate realiza o rezervare on-line pentru cărțile disponibile în acel moment. Cu toate aceste funcționalități problema nu este rezolvată în totalitate, deoarece cartea poate fi indisponibilă neputând fi rezervată.

Astfel, bibliotecile dorind să rezolve aceste dificultăți ale cititorilor utilizează platformele on-line, prin intermediul cărora se facilitează accesul la cataloagele de cărți aflate în colecția acestora. În detrimentul caracteristicilor prezentate, platformele pot dispune de o optimizare aducând o eficientizare procesului de închiriere al cărților. Aceste schimbări pot fi reprezentate de accesul la statusul resurselor. Dacă sunt disponibile utilizatorul ar putea să le rezerve, pentru o perioadă de timp, funcționalitate deja existentă, dar ar trebui automatizată, în sensul că după o perioadă de timp predefinită, cunoscută de cititor, ea sa devină invalidă. Astfel cititorul având siguranța că atunci când vine la bibliotecă în limita de timp specificată, cărțile mai sunt disponibile. Există de asemenea cazul în care resursa este indisponibilă din faza inițială, iar utilizatorul ar trebui să poată preciza dacă o dorește și să o primească atunci când devine accesibilă.

Noutățile prezentate mai sus aduc o experiența mai plăcută unui cititor în procesul de închiriere al unei cărți, determinând și o creștere al numărului de persoane ce doresc să utilizeze resursele tipărite. Optimizările specificate reprezintă soluția mea găsită la această problemă și prezentate în 0.

Soluție existentă

După cum am specificat mai sus, bibliotecile s-au gândit la o soluție pentru a îmbunătăți eficiența procesului de închiriere al unei cărți. În continuare voi prezenta una dintre aceste platforme online. Descrierea va fi una succintă axată mai mult pe funcționalitățile la care are acces un utilizator obișnuit, astfel prezentarea va fi din perspectiva unui cititor.

➤ Utilizator:

- Conectare ca și vizitator – Nu necesită existența unui cont cu o parolă validă. Permite doar căutarea unei cărți și vizualizarea existenței acesteia.
- Conectare – Utilizatorul se poate conecta cu un nume de cont valid și cu parola corespunzătoare acestuia. Astfel, utilizatorul are acces la fișa personală, poate rezerva on-line o carte etc.
- Căutare carte – Există funcționalitatea de căutare a unei cărți folosind mai multe criterii (titlul cărții, autorul, seria, ediția, etc), peste care se pot aplica filtre (căutare în interiorul unui cuvânt, alegerea bazei de date în care se face căutarea).
- Rezervare on-line – Rezultatele căutării pot fi vizualizate sub formă de tabel, ele fiind afișate detaliat, existând și informații referitoare la rezervare.

- Vizualizare fișă personală – Utilizatorul poate avea acces la profilul acestuia, ce conține informații despre nume, adresă, numărul de permis etc.

Soluție propusă

Soluția propusă de mine pentru a optimiza procesul de închiriere al unei cărți la o bibliotecă, reducând inconveniența cititorilor, constă în realizarea unei platforme on-line ce conține cataloagele de cărți ale acesteia. Aplicația web în funcție de fiecare tip de utilizator în parte va permite realizarea de acțiuni specifice acestora.

Funcționalități

În aplicația mea există două tipuri de utilizatori: primul este de tip bibliotecar având rol de administrator, dar are posibilitatea de a utiliza aplicația ca un utilizator obișnuit.

I. Bibliotecar :

- Conectare – Pentru această acțiune este necesară existența unui cont valid reprezentat prin adresă de email validă și parola corespunzătoare acestuia. La conectare bibliotecarul este redirecționat pe o pagină web, unde are acces la funcționalitățile unui utilizator cu rol obișnuit, dar are în pagina de navigație buton către partea de administrator.
- Vizualizare listă utilizatori – Toți utilizatorii sunt prezentați în detaliu (nume, adresă, număr permis bibliotecă, etc) sub formă de tabel, iar în dreptul fiecărui utilizator există posibilitatea de actualizare informație sau ștergere.
- Actualizare utilizator – Se poate actualiza toate informațiile specifice unui utilizator dorit, referitoare la profilul acestuia reprezentat prin nume, prenume, adresă, etc.
- Ștergere utilizator – Prin această acțiune se șterge definitiv utilizator în cauză.
- Căutare utilizator – Se căuta un utilizator după nume sau prenume, cu posibilitatea de căutare în interiorul cuvântului.
- Căutare avansată al unui utilizator – Funcționalitatea este reprezentată de posibilitatea de a căuta un utilizator după mai multe criterii dorite: nume, prenume, adresă de email, numărul permisului de la bibliotecă.
- Vizualizare cărți – Toate cărțile sunt afișate detaliat (nume, tip, nume autor, disponibilitate etc), într-un format tabelar ce facilitează posibilitatea de administrare a unei cărți.

- Actualizare cărți – Caracteristicile specifice unei cărți(nume, nume autor, tipul unei cărți, seria etc) pot fi actualizate prin noi caracteristici furnizate.
- Căutare carte – Funcționalitatea permite căutarea unei cărți după nume, totodată există și posibilitatea căutării în interiorul unui cuvânt. Rezultatele sunt căutării sunt prezentate sub formă de tabel.
- Căutare avansată a unei cărți – Utilizatorul poate căuta o carte după mai multe criterii dorite: după nume sau tipul cărții, numele autorului sau prenumele acestuia. Se pot adăuga filtre la căutare: căutare în interiorul cuvântului, exclude cărți ce nu sunt returnate, afișează doar rezultatele ce au statusul disponibil.
- Ștergere cărți – Oferă posibilitatea de ștergere a unei cărți, în același timp se șterg și caracteristicile acesteia, precum autorul dacă el nu mai are în colecție și alte cărți.
- Deconectare – Utilizatorul are posibilitatea de deconectare, acesta fiind redirecționat către pagina de conectare.

II. Utilizator:

- Conectare – Utilizatorul pentru a se conecta la aplicație are nevoie de o adresă de email validă cu rol de utilizator obișnuit și parola corespunzătoare acesteia.
- Deconectare – Orice utilizator se poate deconecta atunci când dorește, iar ca rezultat el va fi redirecționat de către platformă spre pagina de conectare.
- Vizualizare profil: Utilizatorul poate accesa profilul său, unde sunt specificate informații referitoare la numele și prenumele acestuia, adresă, poză de profil, adresă de email.
- Schimbare parolă de conectare cont: Permite opțiunea de schimbare a parolei, dacă se dorește acest lucru, totodată trebuie cunoscută vechea parolă pentru validarea datelor.
- Căutare carte – Funcționalitatea permite căutarea unei cărți după nume, totodată există și posibilitatea căutării în interiorul unui cuvânt. Rezultatele sunt căutării sunt prezentate sub formă de tabel.
- Căutare avansată carte – Utilizatorul poate căuta o carte după mai multe criterii dorite: după nume sau tipul cărții, numele autorului sau prenumele acestuia. Se pot adăuga filtre la căutare: căutare în interiorul cuvântului, excludere cărți din lista de dorințe, exclude cărți ce nu sunt returnate, afișează doar rezultatele ce au statusul disponibil. Rezultatele sunt afișate detaliat sub formă de tabel, fiecare având opțiunea de rezervare sau adăugare în lista de dorințe.

- Rezervare carte disponibilă – Cărțile ce au statusul de disponibil pot fi rezervate, pentru o perioadă de 24 de ore, timp în care pot fi ridicate de la bibliotecă. După perioada specificată, rezervarea expiră.
- Adaugă carte indisponibilă la lista de dorințe – Acele cărți ce au statusul indisponibil pot fi adăugate în lista de dorințe, astfel utilizatorul specificând că o dorește atunci când cartea devine disponibilă.
- Vizualizare istoric cărți împrumutate – Funcționalitatea permite vizualizarea istoricului de cărți împrumutate până acum, dar și posibilitatea observării cărților ce au fost rezervate și nu au fost ridicate la timp.
- Vizualizare listă de dorințe curentă – Utilizatorul poate vizualiza lista de dorințe în care sunt prezentate cărțile în ordine crescătoare a importanței pentru acesta. Lista este prezentată sub formă de tabel în care fiecare carte are informații specifice acesteia, dar și perioada pentru care se dorește să fie împrumutată.
- Actualizare listă de dorințe curentă – Utilizatorul poate actualiza informația referitoare la clasa de importanță a unei cărți și totodată poate schimba și perioada pentru care dorește să o închirieze.
- Vizualizare notificări – Notificările noi și necitite sunt disponibile în bara de meniu, dar utilizatorul are și posibilitatea vizualizării tuturor notificărilor în pagina de notificări. Notificările sunt generate în urma expirării unei rezervări, în urma asignării unei cărți din lista de dorințe sau ca efect al neacceptării cărții din „wishlist”, asignată în intervalul de timp specificat.
- Vizualizare statistici despre :
 - tipul de carte preferat – Utilizatorul poate vedea un top trei al tipurilor de cărți preferate de acesta.
 - coeficientul mediu de returnare al cărților – Este reprezentat printr-un procent și evidențiază timpul mediu de returnare al unei cărți închiriate. Dacă procentul are culoarea roșie înseamnă că utilizatorul obișnuiește să întârzie, iar în caz contrar are culoarea verde.
 - numărul de cărți returnate la timp sau întârziate pe lună – Statisticile acestea au un rol informativ cu privire la istoricul cărților închiriate, totodată ele validează corectitudinea procentului de returnare.

Structura lucrării

Lucrarea realizată este împărțită în următoarele capitole:

I. Fundamentele teoretice ale aplicației

Aplicația mea folosește pentru administrarea resurselor limitate un algoritm de „stable matching” și de aceea consider că mai întâi este firesc să vorbesc despre acesta. Astfel capitolul descrie noțiuni referitoare la acest algoritm, cât și algoritmul propriu-zis. Sunt evidențiate motivele pentru care algoritmul clasic nu este utilizat în practică, de asemenea sunt prezentate derivări ale acestuia, și tipul de algoritm ales de mine.

II. Integrarea algoritmului de „stable matching” în aplicația web

Capitolul acesta are rolul de a evidenția modul în care algoritmul este folosit într-o aplicație web. Sunt descrise tehnologiile folosite, modul în care este integrat astfel încât să nu împiedice funcționarea aplicației ce este asincronă, păstrând eficiența cu care răspunde la „request”-uri.

III. Structura aplicației web

Acest capitol are rolul de a prezenta arhitectura aplicației utilizate, bazate pe „design pattern”-ul „,MVC”. Astfel voi prezenta fiecare modul în parte, voi detalia rolul pe care îl îndeplinește în „flow”-ul aplicației.

Totodată voi prezenta comunicarea între module, adică interacțiunea dintre acestea, în special pentru cele ce fac parte din arhitectura „,MVC”-ului.

Voi argumenta succint, alegerea acestui tip de structură și voi evidenția beneficiile aduse aplicației.

IV. „Back-end”-ul aplicației web

Ca în orice aplicație web și în aceasta se face o distincție între modulul de „front-end” și cel de „back-end”. În prima parte a capitol voi prezenta tehnologiilor folosite.

Tot în cadrul acestui capitol va fi prezentată structura bazei de date, sub formă de diagramă „UML”. Se va putea observa relațiile dintre acestea(„one to many” , „one to one” sau „many to many”).

V. „Front-end”-ul aplicației web

La începutul acestui modul voi vorbi pe scurt despre tehnologiile alese, și motivul alegerii acestora în realizarea aplicației.

Contribuții

Utilizatorul are acces la funcționalitatea de căutare a unei cărți dorite, iar în cazul existenței unui rezultat corespunzător acestuia îi sunt afișate detalii despre carte, totodată și statusul acesteia ce poate fi disponibil sau indisponibil.

Adițional aplicațiilor web actuale, oferite de biblioteci, platforma realizată de mine oferă o automatizare a procesului de închiriere al unei cărți oferind noi funcționalități cum ar fi rezervarea pentru o perioadă de timp a unei resurse disponibile sau posibilitatea de adăugare în „wishlist” a celor ce sunt indisponibile, reprezentate prin resurse momentan reduse.

Rezervarea unei cărți – După căutarea rezultatului dorit, utilizatorul poate opta pentru rezervarea acesteia, dacă are statusul de disponibil. Această acțiune este realizată automat de către platformă, iar în cazul în care resursa nu este revendicată într-un interval de timp de 24 de ore de la momentul rezervării, ea devine invalidă. Odată devenită invalidă, utilizatorul o poate rezerva din nou, dacă aceasta mai este disponibilă, altfel el poate opta pentru opțiunea de adăugare la lista proprie de dorințe. Prin intermediul funcționalității de rezervare se dorește evitarea situației întâlnite la platformele actuale, ce nu garantează că existența cărții în colecția bibliotecii este echivalent cu disponibilitatea acesteia.

Adăugare la lista proprie de dorințe a unei resurse indisponibile – O carte existentă în catalogul aplicației mele web, ce are la status indisponibil poate fi adăugată la „wishlist”. Statusul de indisponibil evidențiază că acea carte se încadrează în categoria de resurse limitate (aceasta limitare se poate referi la una momentană, în sensul că resursa este tipărită într-o mulțime de exemplare, dar majoritatea sunt închiriate sau este echivalent cu un număr redus de exemplare ce trebuie administrate eficient). Decizia de asignare a unei cărți din lista de dorințe se realizează pe baza unui algoritm de „Stable matching”.

Odată la un interval de timp de 2 minute un „task scheduler” generează un „task” care este memorat într-un „message broker”, împreună cu momentul în care trebuie să înceapă execuția acestuia. Un „worker” verifică „task”-urile memorate și le execută în „background”

în funcție de timpul specificat. Astfel rutina este formată din execuția consecutivă a următoarelor etape:

- Ștergere rezervări ce nu au fost revendicate la timp
- Actualizează starea – Se interoghează baza de date și se face „update” la informațiile necesare algoritmului de „Stable matching”
- Executarea algoritmului
- Trecerea rezultatului în baza de date – Existând un „delay” între momentul actualizării cărții și cel de scriere al rezultatului, se face totodată și o verificare al acestuia, existând posibilitatea ca o carte asignată să nu mai fie dorită de utilizator. Utilizatorii ce nu au primit o carte în urma acestui algoritm și lista de dorințe nu este vidă, au posibilitatea de a le fi asignată o carte la următoare iterație a rutinei.

Algoritmul de „Stable matching” asignează unui utilizator una dintre cărțile dorite dacă este posibil. De menționat este că spre deosebire de algoritmul clasic, cele două mulțimi curente (cărțile dintr-un „wishlist” și numărul total de cărți) au un număr de elemente diferite ce poate conduce la o incompatibilitate între preferințe. De aceea se consideră configurație stabilă, dacă în urma acestui algoritm un utilizator nu primește o carte, chiar dacă are o lista de preferințe, dar cărțile preferate de acesta sunt mult mai eficient asignate altui utilizator.

Criteriile de asignare ale algoritmului sunt: „rank”-ul unei cărți definit în „wishlist” perioada pentru care este dorită cartea, dar cel mai important pentru optimizarea împrumutului de cărți este coeficientul de încredere al unui utilizator. Acest coeficient este de fapt o medie între procentul din perioada rămasă de închiriere (sau întârziere) atunci când a fost predată cartea, împărțit la numărul de cărți.

1 Fundamentele teoretice ale aplicației

1.1 Algoritmul „stable matching” clasic al lui „Gale–Shapley” (2)

O problemă de tip „stable matching” presupune găsirea unui „stable matching” între două mulțimi ce conțin același număr de elemente. Această potrivire se face ținându-se cont de o listă ordonată de preferințe pentru fiecare element în parte. Astfel se încearcă o potrivire menită cu scopul de a satisface preferințele date.

O potrivire între două mulțimi distincte, A și B , fiecare având n elemente nu este stabilă dacă se întâmplă următoarea situație:

- există perechea formată din (a, b') , cu $a \in A, b' \in B$
- există perechea formată din (b, a') , cu $b \in B, a' \in A$
- $a \neq a', b \neq b'$
- a preferă mai mult pe b decât b'
- b preferă mai mult pe a decât a'

Astfel situația prezentată mai sus în care nu este formată perechea (a, b) , deși aceasta ar fi trebuit să se realizeze reprezintă o stare instabilă a rezultatului.

- Formularea problemei : Dându-se un număr de n bărbați și n femei, fiecare dintre aceștia având o listă de preferințe de la 1 la n , cu o ordine totală peste persoanele de sex opus, să se cupleze un bărbat și o femeie astfel încât să nu existe 2 persoane de sex opus ce se preferă reciproc în detrimentul persoanei actuale din cuplu. Dacă nu există astfel de pereche precizată anterior, atunci există un „matching” stabil.
- Soluția acestei probleme este descrisă prin algoritmul din Figură 1.

În prima iterație a algoritmului, se cuplează primul bărbat cu una dintre femeile aflate prima pe lista lui de preferințe. Această cuplare nu este neapărat finală, ci este momentană, astfel încât până la final poate să se schimbe.

În fiecare iterație a algoritmului de „stable matching”, se alege primul bărbat ce nu are încă o pereche și se parcurge pe rând preferințele acestuia. Dacă preferința curentă face parte din altă pereche, se verifică dacă nu cumva este mai convenabil atât pentru femeie cât și bărbatul în cauză să se cupleze. În

cazul în care femeia îl preferă pe mai mult pe bărbatul pentru care se face cuplarea decât perechea ei, cei doi vor forma un nou cuplu, iar bărbatul din fostul cuplu va rămâne din nou fără femeie.

```
Initialize all  $m \in M$  and  $w \in W$  to free
while  $\exists$  free man  $m$  who still has a woman  $w$  to propose to {
     $w$  = first woman on  $m$ 's list to whom  $m$  has not yet proposed
    if  $w$  is free
        ( $m$ ,  $w$ ) become engaged
    else some pair ( $m'$ ,  $w$ ) already exists
        if  $w$  prefers  $m$  to  $m'$ 
             $m'$  becomes free
            ( $m$ ,  $w$ ) become engaged
        else
            ( $m'$ ,  $w$ ) remain engaged
}
```

Figură 1 – Algoritmul de „stable matching” clasic

Algoritmul de clasic de „stable matching” are o complexitate de execuție polinomială, terminându-se în cel mai ineficient caz în $O(n^2)$, unde n reprezintă numărul de femei sau de bărbați, acesta fiind egal.

La sfârșitul execuției acestuia, algoritmul garantează că nu poate exista un bărbat ce a rămas fără pereche, deoarece dacă un bărbat nu a găsit în o femeie, la acesta se va reveni, până când toate opțiunile sale vor fi verificate. Astfel toate cele „ n ” opțiuni ale sale vor fi verificate și ținând cont că există un număr „ n ” bărbați și femei, fiecare bărbat / femeie va face parte dintr-o pereche.

În practică acest algoritm în forma lui clasică, care presupune existența a două mulțimi cu același număr de element, listele de preferință să fie complete, având numărul de elemente din acestea egal cu numărul de elemente din una dintre cele două mulțimi, nu este folosit. Aceste condiții reprezintă caracteristici ideale existând o șansă extrem de mică să fie îndeplinite, astfel s-a recurs la o relaxare a acestor condiții apărând unele variații ce sunt folosite în practică și enumerate mai jos.

1.2 Variații ale algoritmului de „stable matching” :

1.2.1 „Stable matching” cu liste incomplete de preferințe (3)

În această derivație al algoritmului se permite ca lista de preferințe, a unui bărbat sau a unei femei, să conțină un număr mai mic decât numărul de elemente al primei mulțimii, reprezentată de bărbați sau mai mic decât cea a femeilor.

Din cauza acestei relaxări, algoritmul nu garantează că la finalizarea acestuia toate persoanele sunt cuplate, în schimb ideea de bază al acestuia este de a realiza perechi convenabile pentru câte mai multe persoane. Astfel situația descrisă mai jos după finalizarea „stable matching”-ului reprezintă o stare validă:

- 2 mulțimi: B a bărbaților, și F a femeilor, fiecare având n elemente
- există perechea formată din (b, f) , cu $b \in B, f \in F$
- b' nu are pereche, dar preferă pe f
- f nu preferă pe b' mai mult decât pe b

Algoritmul de „stable matching” cu liste de preferințe incomplete are o complexitate de execuție polinomială, terminându-se în cel mai ineficient caz în $O(n^2)$, unde n reprezintă numărul de femei sau de bărbați, acesta fiind egal.

1.2.2 „Stable matching” cu liste de preferințe de tip „ties” (3)

Această formă a algoritmului presupune ca peste lista de preferințe să nu se aplice o ordonare totală, ea fiind înlocuită cu o ordonare parțială. Uneori poate lipsi de tot, fiind îndeajuns cuplarea cu o persoană indiferent de poziția acesteia în listă.

În acest tip de extensie cu liste de preferințe de tip „ties”, o stare instabilă este descrisă în următoare situație:

- 2 mulțimi: B a bărbaților, și F a femeilor, fiecare având n elemente
- nu există perechea formată din (b, f) , cu $b \in B, f \in F$
- b preferă în mod strict pe f' (f și f' nu au același nivel de preferință)
- f preferă în mod strict pe b' (b și b' nu au același nivel de preferință)

Dacă nu există o stare blocantă echivalentă cu cea descrisă mai sus, atunci rezultatul în urma executării algoritmului este de tip „weakly stable”.

Astfel aceste modificări aduse algoritmului clasic nu mărește timpul de execuție polinomial, acesta fiind tot $O(n^2)$, în cel mai ineficient caz, unde n reprezintă numărul de femei sau de bărbați, acesta fiind egal.

1.3 Algoritmul de „stable matching” utilizat de aplicație

Pentru aplicația web realizată de mine, am ales să integrez algoritmul „stable matching” cu liste incomplete de preferințe, deoarece caracteristicile acestuia se pliază cu modul de funcționare al „wishlist”-ului unui utilizator.

Lista de cărți, pe care le preferă un utilizator și dorește să le închirieze atunci când ele vor fi disponibile, nu este obligatorie ca să fie distinctă cu a altui utilizator. Se poate întâmpla ca de cele mai multe ori ea să fie nulă sau să conțină un număr redus de elemente. Astfel modul de funcționare al unui „wishlist” este echivalentul unei liste de preferințe descris de algoritmul de „stable matching” cu liste incomplete.

Correspondentul celor două mulțimi pe care se realizează cuplarea din derivația algoritmul clasic (reprezentată prin mulțimea de bărbați ce dorește să își găsească o pereche din mulțimea de femei) este reprezentat prin mulțimea de utilizatori ce au un „wishlist” ce nu este gol și mulțimea de cărți formată prin reuniunea conținutului fiecărui „wishlist”.

În situația platformei realizate definesc o stare stabilă ce respectă caracteristicile de mai jos:

- 2 mulțimi: U a utilizatorilor cu listă de preferințe ce nu este vidă, și C a cărților formată prin reuniunea cărților din fiecare „wishlist”
- există perechea formată din (u, c) , cu $u \in U, c \in C$
- u' nu are pereche, dar preferă cartea c
- c nu preferă pe u' mai mult decât pe u

Spre deosebire de relația de preferință a unui utilizator ce este clar fixată de acesta, criteriul pe care îl urmează o carte pentru a alege perechea sa este definit pe baza perioadei de închiriere la care se adaugă sau se scade un coeficient de încredere al acestuia.

Pentru a preciza condiția de decizie a cărți în vederea distingerii între doi utilizatori, voi defini următoarele notații :

- m_s = momentul de începere al închirierii
- m_t = momentul de terminare al închirierii
- m_p = momentul predării cărții
- $p_i = m_t - m_s$, perioada de închiriere a cărții
- $p_r = m_p - m_s$, perioada de timp până la returnarea cărții
- $c = \sum_{k=1}^n 100 - \left(100 * \frac{p_{r_k}}{p_{i_k}} \right)$, unde n = nr de cărți returnate,

c = coeficient de returnare, este reprezentat ca o medie între diferențele de timp dintre momentul când expiră închirierea și momentul de timp când se returnează cartea.

Acest coeficient poate fi și negativ, semnul reprezentând că utilizatorul returnează de obicei cartea cu întârziere, iar valoarea acestuia evidențiază perioada în procente cu care întârzie.

Analog, în cazul în care procentul este pozitiv este echivalent cu o returnare mai devreme.

Ținând cont de notațiile definite mai sus decizia este reprezentată prin formula:

- $D = p - \left(\frac{c}{100} * p \right)$

Astfel, o carte din motive de eficiență dintre doi utilizatori ce o doresc, logic ar fi ca aceasta să se grupeze cu acel „user” care are un „D” (dat prin formula de mai sus) cât mai mic.

„D”- ul din formulă reprezintă o predicție a perioadei de închiriere a utilizatorului, bazându-se pe istoricul de returnare al acestuia.

2 Integrarea algoritmului de „stable matching” în aplicația web

2.1 Tehnologii folosite

a) Celery

Este o coadă distributivă de „task”-uri asincrone. Este folosit de obicei pentru execuția „task”-urilor consumatoare de resurse sau timp în aplicații web deoarece acesta permite realizarea lor în „background”. Astfel Celery nu suprimă timpul de răspuns al unui „request”, oferind o experiență plăcută utilizatorului menținându-se fluenta aplicației.

b) Celery Beat

Acesta este un „scheduler” ce generează periodic un „task”. Intervalul de timp la care se execută este setat din înainte, iar „task”-ul creat este executat de Celery prin intermediul unui „worker” disponibil în acel timp.

c) RabbitMQ

Un „message broker” îndeplinește rolul de „middleman” pentru un serviciu, de cele mai multe ori acel serviciu este o aplicație web. Acesta este folosită cu scopul de a reduce „load”-ul aplicației stocând „task”-urile și totodată se ocupă de „routing”-ul acestora către un „worker” din Celery.

d) Python 3.6

Limbajul de programare folosit de mine este Python deoarece este un limbaj ușor de folosit fiind încadrat ca limbaj de „high-level”. Acesta este simplist și de aceea este foarte „readable”, oferind claritate codului scris.

e) PostgreSQL

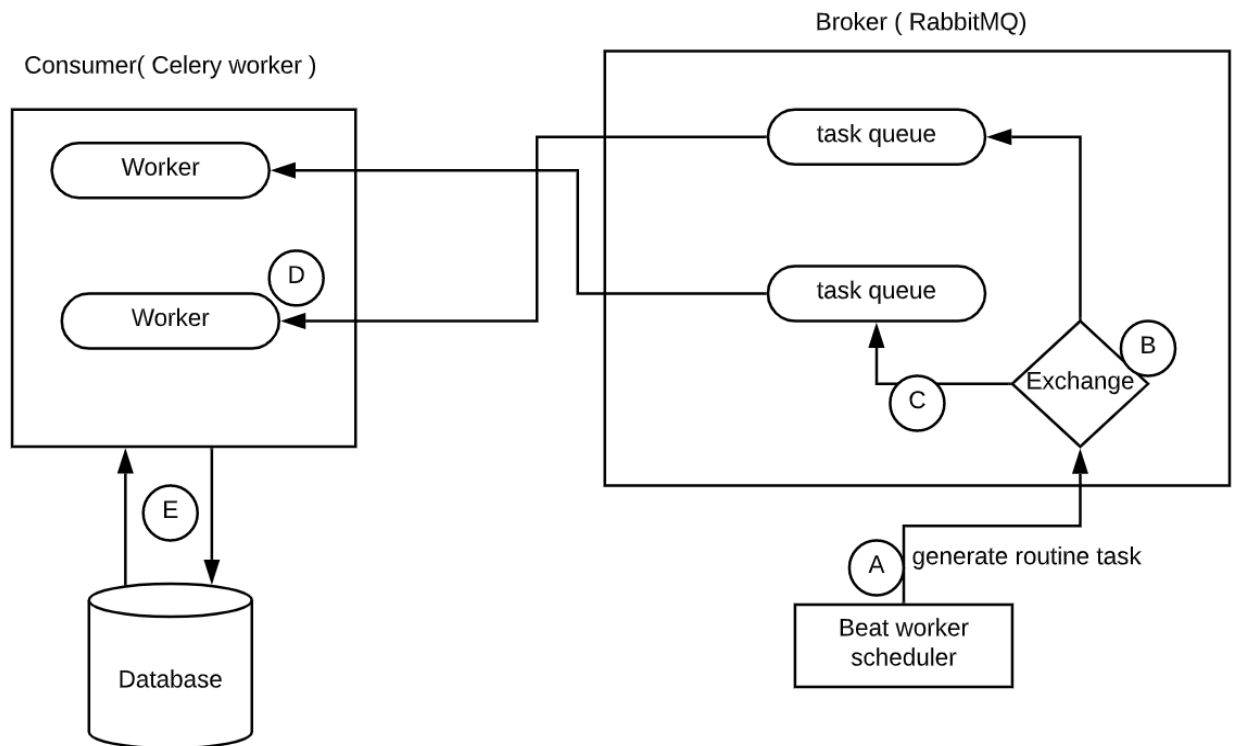
Este o bază de date relațională, este gratuită fiind „open-source”. De asemenea este stabilă și suportă lucrul cu un volum mare de informații.

f) SQLAlchemy

Acesta reprezintă un „ORM”, compatibil cu limbajul de programare Python și cu baza de date PostgreSQL. Am folosit acest „tool” ce reprezintă un

„layer” intermediar între comunicarea bazei de date și aplicația mea web.
SQLAlchemy este cel mai răspândit „ORM” din cele prezente în aplicațiile scrise în Python.

2.2 Comunicarea tehnologiilor si rolul îndeplinit de acestea



Figură 2 – Interacțiunea dintre „Beat worker scheduler”, „Message Broker RabbitMQ” și „Celery Consumer”

În Figură 2 se pot observa următoarele :

- „Scheduler”-ul generează o dată la un interval de 2 minute un „task”, acesta este redenumit de mine ca „routine”. El este realizează mai multe acțiuni printre care și execuția algoritmului de „stable matching”. Voi reveni ulterior la această „routine”, unde o voi prezenta în detaliu la etapa F.
- În „exchange” „task”-ul generat de „beat worker scheduler” este preluat de către RabbitMQ . „Exchange”-ul acționează ca un „router” asignând „task”-ului un „task queue” disponibil.

- C. După decizia „exchange”, „task”-ul este asignat uneia dintre cozile „message broker”-ului, decizia se face în funcție de disponibilitatea acestuia.
- D. „Task”-ul ce urmează a fi executat realizează următoarele etape, acestea sunt evidențiate și în Figură 3 :
- i. Invalidează cărți neacceptate — Cărțile ce au fost asignate de către algoritmul de „stable matching”, în urma execuției acestuia și care nu au fost acceptate în intervalul precizat de 3 ore.
 - ii. Actualizarea timp cărți — Se face „update” la timpul rămas pentru cărțile închiriate.
 - iii. Actualizare „input” pentru „stable matching” — Se preia din baza de date informațiile necesare executării algoritmului (utilizatori, liste de preferințe, număr de cărți).
 - iv. Execuția algoritmului propriu-zis — Se încearcă găsirea unei perechi între utilizator și una dintre cărțile dorite din lista de preferințe incomplete.
 - v. Scriere rezultate în baza de date — Înainte de trecerea rezultatelor în baza de date se face o verificare a corectitudinii rezultatului, deoarece există un decalaj de timp între momentul actualizării „input”-ului pentru algoritmul de „stable matching” și scrierea efectivă a rezultatelor. Totodată verificarea este necesară deoarece „task”-ul se realizează asincron cu „request”-urile aplicației web, putând apărea modificări în „wishlist”-uri între timp.
- E. Interacțiunea cu baza de date în timpul execuției „task”-ului — Toate etapele din structura unui „task” interacționează cu baza de date pentru a realiza modificări asupra datelor din aceasta, exceptând etapa iii. și etapa iv. În etapa iii. se fac interogări asupra bazei de date pentru a salva informațiile necesare algoritmului, iar în etapa v. se trec rezultatele „stable matching”-ului.

```

@celery.task()
def stable_matching_routine():
    clean_unaccepted_books()

    update_remaining_book_time()

    users_with_wishlist, trust_coeffs, book_count = update_state()
    stable_matching = Stable_matching(users_with_wishlist, trust_coeffs, book_count)
    stable_matching.run()
    matched = stable_matching.get_match()
    del stable_matching

    write_result_of_matching(matched)
    logger.info("Routine done")

```

Figură 3 – Prezentarea etapelor unui „task”

În Figură 3 se pot observa etapele pe care le parcurge un „task” despre care am vorbit în detaliu mai sus, dar pe lângă acest lucru este prezent și decoratorul pentru „celery”. Acesta transformă o funcție obișnuită într-o funcție de tip „celery” ce se execută asincron în „background”. La sfârșitul funcției este prezent un „logger” ce afișează în consolă un mesaj, specificând terminarea executării „task”-ului.

2.3 Dificultăți întâlnite

Consider că principala dificultate în realizarea acestei aplicații web este reprezentată de integrarea algoritmului de „stable matching” într-un mediu dinamic, în care există posibilitatea ca „input”-ul să se schimbe în mod constant, chiar și în timpul unei iterații ale acestuia.

Mediul dinamic este determinat de natura asincronă a aplicației web, dar și funcționalitățile pe care aceasta i le conferă utilizatorilor. Funcționalitățile ce intervin în schimbarea „input”-ului algoritmului sunt reprezentate de următoarele:

- Utilizatorul poate modifica „wishlist”-ul – În orice moment, un utilizator poate modifica conținutul listei de preferințe ce duce la modificarea datelor pe baza cărora se realizează „stable matching”. Acesta poate adăuga o carte în lista de preferințe sau poate șterge una. El mai poate realiza și o modificare a proprietăților unei cărți reprezentate prin „rank” sau perioadă. Chiar și o

simplă operație de actualizare a importanței sau a perioadei de închiriere reprezintă o modificare a stării bazei de date.

- Bibliotecarul poate modifica o carte – Un utilizator cu rol de bibliotecar are acces la realizarea de operațiuni asupra unei cărți. El poate adăuga o carte nouă sau poate șterge una deja existentă. Există posibilitatea de realizării operației de actualizare a unei cărți deja existente, reprezentată prin modificarea titlului cărții sau categoria din care face parte aceasta, modificarea numelui autorului etc. Una dintre funcționalitățile pe care le poate realiza un bibliotecar asupra unei cărți duce la o modificare a „input”-ului algoritmului de „stable matching” invalidând posibilitatea de execuție al acestuia.

Soluția găsită de mine la această problemă constă în utilizarea de stări, astfel încât o stare are un input constant și pe baza căruia se poate executa algoritmul. Introducerea noțiunii de stare contracarează inconvenienta produsă de mediul dinamic.

Utilizarea stărilor permite executarea algoritmului, dar soluția la rândul ei aduce o nouă dificultate deoarece rezultatul produs de „stable matching” este generat pe baza unei stări, iar aceasta poate să difere de starea actuală definită prin „input”-ul din baza de date.

Astfel, utilizarea de stări nu garantează producerea unui „output” corect și de aceea este necesar ca atunci când se realizează scrierea rezultatului algoritmului, în baza de date, să se realizeze o verificare a acestuia.

Verificarea „output” presupune următoarele:

- Cartea există în baza de date – Acesta verificare are rolul de a preveni posibilele consecințe ale funcționalităților unui bibliotecar.
- Cartea există în „wishlist” – Se verifică dacă cartea asignată în urma execuției algoritmului există în lista de preferințe, totodată se realizează și o verificare asupra caracteristicilor acesteia astfel încât să corespundă „rank”-ul și perioada. Aceste verificări sunt realizate cu scopul de a contracara inconvenientele produse de funcționalitățile la care are acces utilizatorul.

O altă problemă întâlnită a fost una de natură tehnică, reprezentată prin găsirea unei tehnologii compatibile cu necesitățile cerințelor determinate de soluția găsită. „Tool”-ul

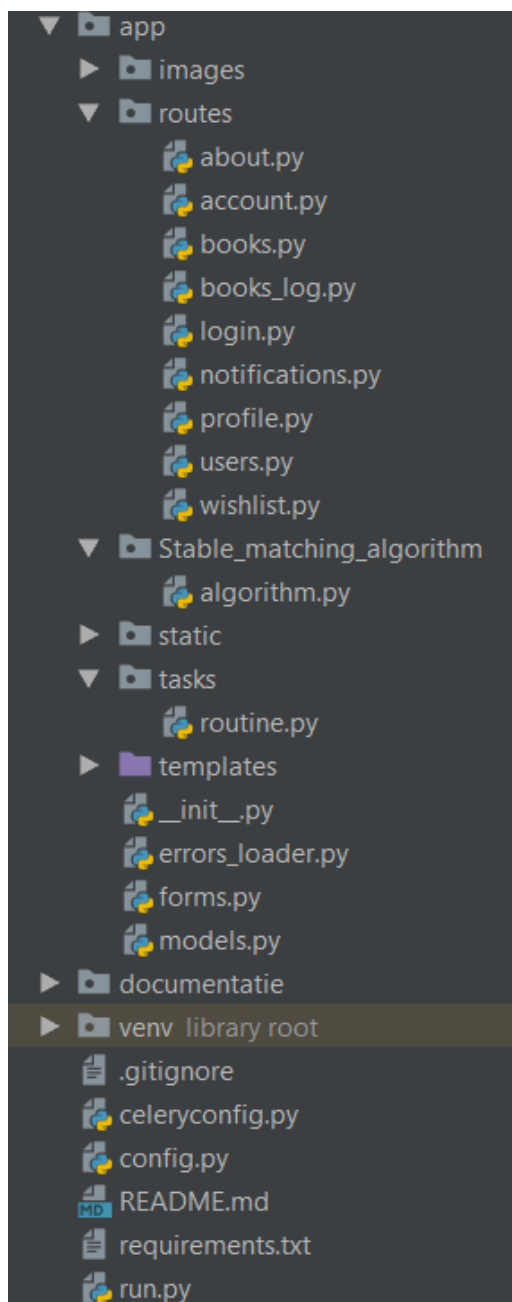
căutat trebuie să permită executarea algoritmului, dar totodată să nu intervină în execuția aplicației web sau să reducă performanțele acesteia. Totodată execuția „task”-ului trebuie să se realizeze în „background”, astfel încât el să se realizeze periodic, iar începerea executării acestuia să nu fie declanșat de un „request” al aplicației web.

De asemenea au fost probleme și la pornirea execuției „worker”-ului. Pornirea clasică a acestuia nu mai era compatibilă cu noua lui versiune și de aceea am rezervat această problemă prin utilizarea următorului link (4).

Astfel am ales tehnologiile prezentate în subcapitolul 2.2, unde am prezentat în detaliu interacțiunile dintre acestea. Un „scheduler” generează periodic un „task”, acesta printr-o operațiune de „exchange” este preluat de un „message broker”. Acesta din urmă salvează „task”-ul pentru o scurtă perioadă de timp într-un „task queue”. „Message broker”-ul se ocupă de „routing”-ul „task”-ului spre un „worker” disponibil aflat în „Consumer”.

3 Structura aplicației web

3.1 Prezentare



Figură 4 – Structura aplicației

În Figură 4 se observă structura aplicației mele web. „Root”-ul proiectului cuprinde modulul de „app”, fișiere de configurare „celeryconfig.py” (unde sunt specificate setări pentru „Celery” și message brokerul „RabbitMQ”), fișierul de configurare „config.py”(se realizează configurarea aplicației, conectarea la baza de date).

De asemenea în „root” este prezent fișierul de „requirements.txt” ce conține dependențele aplicației, fișierul de „README.md” în care sunt evidențiate pașii de configurare ai proiectului și fișierul de „run.py” prin intermediul căruia se realizează pornirea execuției aplicației web.

Modulul de „app” este la rândul lui format din următoarele module :

- „images” - În acest modul sunt copiate pozele utilizatorilor atunci când se creează un nou profil de utilizator și totodată „path”-ul este adăugat în baza de date ca ulterior să poate fi accesată.
- „routes” – Modul în care este prezent orice „controller” al aplicației ce se ocupă de „routing” în urma unui „request”.
- „Stable_matching_algorithm” – După cum sugerează și denumirea acestuia, modulul conține algoritmul propriu-zis de „stable matching”.
- „static” – Este modulul „public” al aplicației web ce este divizat în alte module („css” – fișiere de „css” pentru fiecare „template” în parte, „img”- conține imagini utilizate de aplicație diferite de profilul utilizatorilor, „js” – conține fișiere de „script”-uri pentru fiecare „template”)
- „tasks” – Conține fișierul „routine.py” în care se află „task”-ul ce este executat de către „Celery”
- „templates” – Modulul cuprinde „template”-urile aplicației
- „__init__.py” – Este „constructor”-ul modului „app” în care se creează instanțe ale aplicației, a bazei de date, a „ORM”-ului
- „errors_loader.py” – Funcții ce sunt returnate în caz de erori ale aplicației web.
- „forms.py” – Aplicația folosește „form”-uri securizate de tipul „cross site validation” și sunt stocate în acest modul.
- „models.py” – Conține orice „model” al aplicației web.

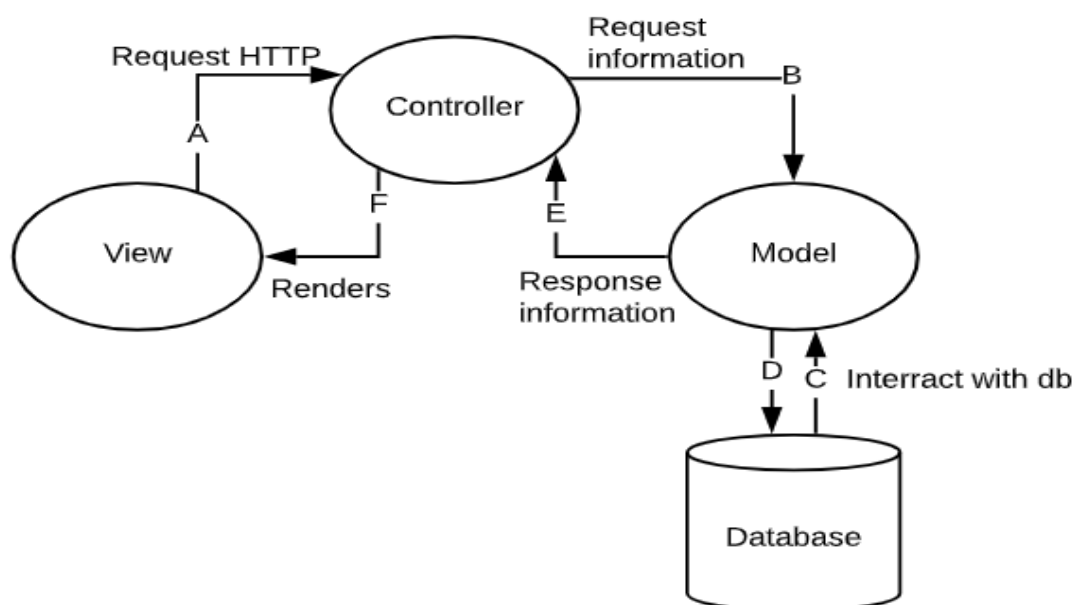
3.2 Folosirea „design pattern”-ul „MVC”

În Figură 4 se observă că folosesc „design pattern”-ul „MVC” în arhitectura aplicației, acesta folosește modulele:

- Models - În aplicația mea are ca reprezentant modulul de „models.py”
- View – Acesta are ca echivalent modulul de „templates”(am folosit notația de „templates” în detrimentul celei de „views” deoarece este des întâlnită în cazul aplicațiilor web scrise în Python).
- Controller – Acesta este reprezentat de modulul de „routes” ce conține fiecare „controller” utilizat de aplicație.

Comunicarea între module este prezentată în Figură 5, și în detaliu după cum urmează:

- A. Prin intermediul unui „View” utilizatorul interacționează cu aplicația web și trimite un „request HTTP” către un „controller” ce îl procesează.
- B. „Controller”-ul la rândul lui comunică cu „Model” pentru a face rost de informație.
- C. „Model” interoghează baza de date pentru informațiile necesare.
- D. „Model” primește informațiile cerute.
- E. „Controller”-ul primește ca „response” informația cerută anterior „Model”-ului.
- F. „Controller”-ul realizează face „render” la informație în „View” pentru a fi văzută de utilizator.



Figură 5 – „MVC” arhitectură

Am ales să folosesc în arhitectura aplicației mele „design pattern”-ul „MVC” deoarece acesta conferă o izolare între logica aplicației și interfața vizibilă utilizatorului. Astfel se creează noi nivele de abstractizare ce sunt independente, iar modificări ulterioare la unele dintre acestea nu influențează nivelele ce nu au suferit modificări.

Totodată „MVC” face ca aplicația web să fie foarte scalabilă astfel încât se pot adăuga noi funcționalități cu ușurință, adăugând fiecărui modul din „pattern” („model”, „view” sau „controller”) noua componentă caracteristic acestuia.

4 „Back-end”-ul aplicației web

4.1 Tehnologii folosite

a) Python 3.6

Limbajul de programare folosit de mine este Python deoarece este un limbaj ușor de folosit fiind încadrat ca limbaj de „high-level”. Acesta este simplist și de aceea este foarte „readable”, oferind claritate codului scris.

b) Flask

Acesta este un „micro-framework” web compatibil cu limbajul de programare Python. Este un „micro-framework” deoarece acesta nu necesită anumite „tool”-uri specifice, abstractizări ale bazei de date, validări de „form”. Astfel am ales Flask deoarece este foarte „customisable” prin adăugarea diferitor extensii și module sau „third-party” librării.

c) PostgreSQL

Este o bază de date relațională, este gratuită fiind „open-source”. De asemenea este stabilă și suportă lucrul cu un volum mare de informații.

d) SQLAlchemy

Acesta reprezintă un „ORM”, compatibil cu limbajul de programare Python și cu baza de date PostgreSQL. Am folosit acest „tool” ce reprezintă un „layer” intermediar între comunicarea bazei de date și aplicația mea web. SQLAlchemy este cel mai răspândit „ORM” din cele prezente în aplicațiile scrise în Python.

4.2 „Design”-ul bazei de date

În arhitectura bazei de date am ținut cont de criteriile de normalizare a unei baze de date. Astfel informația stocată respectă următoarele proprietăți:

- Informația din celule este atomică - Nu există celulă în baza de date care să conțină două elemente distincte ce puteau fi introduce în doua celule separate. Se respectă proprietatea de 1NF.

- Orice atribut din „header”-ul tabelului este dependent de cheia primară a acestuia – Proprietatea evidențiază faptul că fiecare atribut trebuie să fie dependent de în mod direct de cheia primară corespunzătoare a acestuia și nu prin intermediul unui alt atribut. Se respectă proprietatea de 2NF.
- Orice element dintr-o coloană ce nu este cheie(primară sau străină) să fie independentă de orice altă coloană – Ajută la evitarea cazului în care modifici un element dintr-o coloană să nu trebuiască să se realizeze schimbări și în altă coloană. Astfel se evită stocarea de informație derivată în baza de date, respectând proprietatea 3NF.
- Orice informației dintr-un să depindă doar de cheia primară a acestuia – Proprietatea este asemănătoare cu cea precedentă, doar că există o diferență evidențiată prin situația în care într-un tabel sunt două sau mai multe chei candidate compuse, ce au anumite elemente comune. Această proprietate este BCNF.

Am ținut cont și de integritatea informației în „design”-ul bazei de date, astfel încât o cheie primară este unică și nu există alta identică în același tabel. Mai mult aceasta nu poate să fie nulă, de asemenea și cheile străine trebuie să aibă corespondent o cheie primară în alt tabel și să nu fie nici ea nulă.

În Figură 6 se poate observa o parte din arhitectura bazei de date, am ales să o împart în două imagini diferite deoarece aceasta conține tabelul de „user” și celelalte tabele ce se află în relație directă.

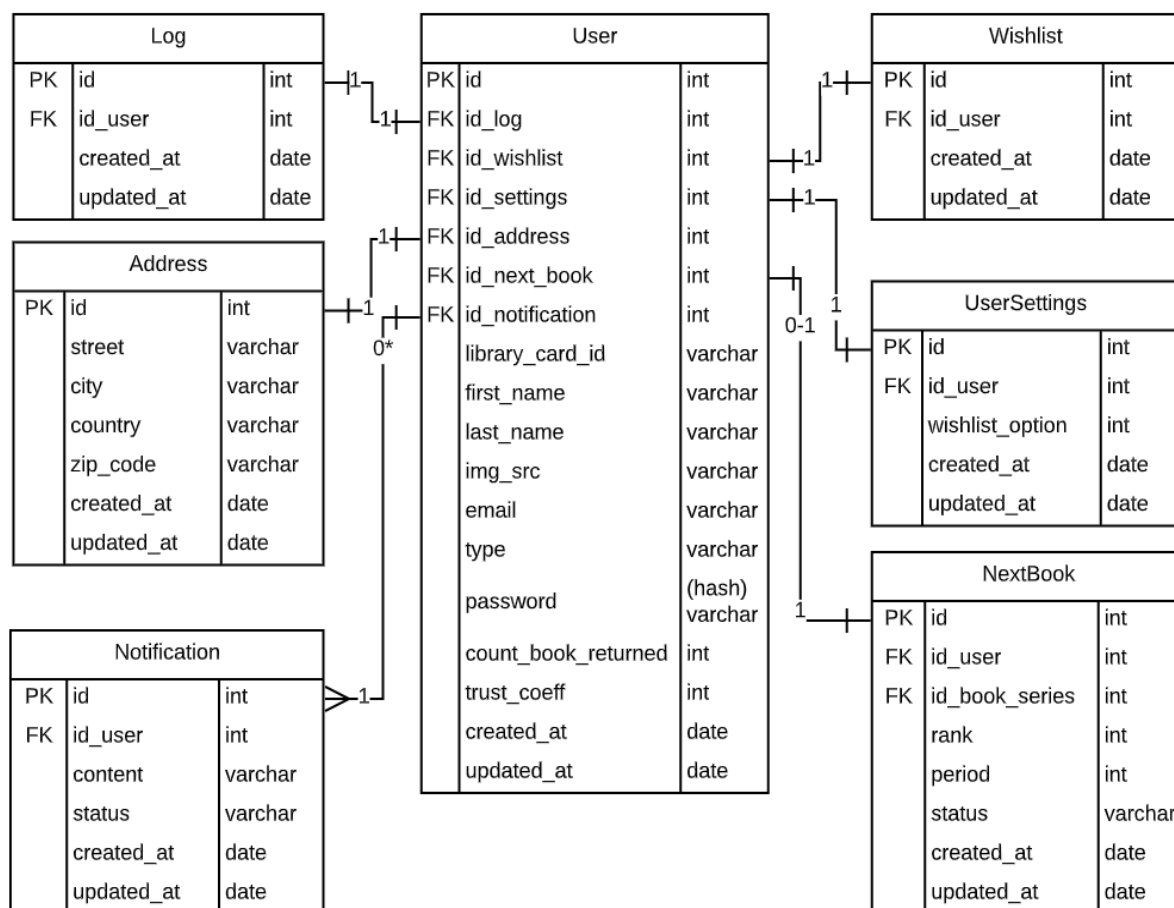
Tabele ce se află în relație de „one to one” cu tabelul de „User” sunt:

- Log – Tabel intermediar între cel de „User” și cel de „EntryLog”. Fiind unul intermediar nu conține informațiile cu privire la istoricul cărților închiriate de un utilizator, acestea fiind stocate de fapt în „EntryLog”. Previne existența unei relații de „many to many”.

- Wishlist – Tabel intermediar între cel de „User” și cel de „EntryWishlist”. Fiind unul intermediar nu conține informațiile cu privire la lista de preferințe, elementele din acestea fiind stocate de fapt în „EntryWishlist”. Previne existența unei relații de „many to many”.
- Address – După cum sugerează numele tabelului, conține informații referitoare la adresa utilizatorului.
- UserSettings – Conține setarea utilizatorului cu privire la administrarea cărților din lista de preferințe ce au fost asignate, dar refuzate. Acestea pot fi: șterge cartea, adaugă cartea la sfârșitul „wishlist”-ului sau adaugă cartea în vechea poziție.
- NextBook – Tabelul conține actuala carte asignată în urma algoritmului de „stable matching”.

Tabele ce se află în relație de „one to many” cu tabelul de „User” sunt:

- Notification – Fiind o relație de „one to many” un utilizator din tabelul „User” poate să aibă mai multe rezultate în tabelul de „Notification”. Acesta după cum sugerează numele conține informații privitoare la notificările utilizatorului. Notificările ajung să fie generate de către „worker”-ul ce execută „task”-ul, ele sunt generate ca o consecință a unei acțiuni (asignare carte în urma „stable matching”-ului sau neacceptarea acesteia, rezervare expirată etc).

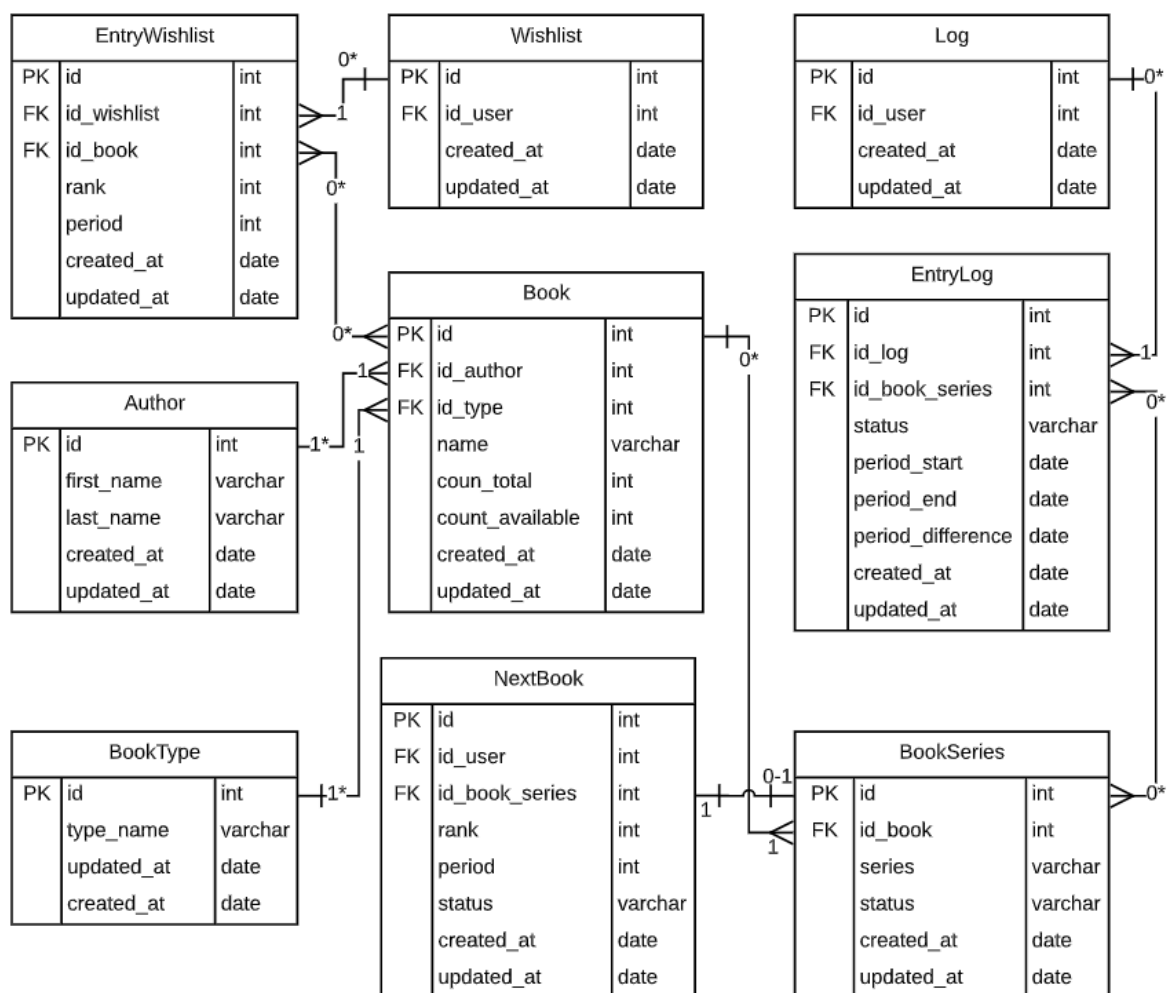


Figură 6 – Partea I din Diagrama bazei de date

În Figură 7 este evidențiată cea de a doua secvență din a bazei de date. În continuare voi descrie „relationship”-urile dintre acestea, la care voi adăuga câteva detalii despre fiecare :

- **Book** – Acest tabel are o relație de „one to many” cu tabelul **BookSeries**, deoarece o carte poate avea mai multe serii. Acesta după cum sugerează numele cuprinde detalii despre cărțile din baza de date, referitoare la nume, numărul total etc.
- **BookType** – Acest tabel conține denumirea tipurile de cărți prezente in baza mea de date. El are o relație de „many to one” cu tabelul de **Book**.
- **BookSeries** – El are o relație de „many to many” cu tabelul de **EntryLog**
- **Author** – Are o relație de „one to many” cu tabelul **Book** și conține informații privitoare la numele și prenumele autorului.
- **Log** – Tabelul se află într-o relație de „one to many” cu tabelul **EntryLog**.

- EntryLog – Acesta are o relație de „many to many” cu tabelul BookSeries. El conține informații referitoare la istoricul de închiriere al unui utilizator.
- Wishlist – Relație de „one to many” cu EntryWishlist.
- EntryWishlist – Relație de „many to many” cu Book. Reprezintă efectiv conținutul unei liste de preferințe a unui utilizator.
- NextBook – Are o relație de „one to one” cu BookSeries. Conține cartea asignată în urma algoritmului de „stable matching”.



Figură 7 – Partea II din Diagrama bazei de date

În implementarea aplicației am utilizat un „Object Relational Mapper”, astfel încât accesul aplicației la informațiile din baza de date se face prin intermediul acestui „layer” de abstractizare.

4.3 Procesarea „request”-urilor și returnarea răspunsului corespunzător

Prin interacțiunea cu aplicația, un utilizator realizează anumite acțiuni, acestea prin intermediul „view”-urile sunt preluate și integrate sub forma unui „request” ce este procesat pe partea de „back-end” de către un „controller”.

„URI”-ul pe care îl procesează acel „router” poate conține diferite acțiuni(„GET”, „UPDATE”, „POST”, „DELETE”) ce sunt realizate asupra informațiilor din baza de date.

O caracteristică importantă a unei aplicații web este reprezentată de securitatea acesteia. Astfel pentru a menține securitatea acesteia, pentru fiecare „request” trebuie verificat dacă utilizatorul are dreptul de a realiza acea acțiune. De obicei dreptul la a realiza o acțiune presupune ca utilizatorul să fie autentificat, rolul sub care este autentificat să aibă drept de a realiza acea acțiune sau informația asupra căreia se realizează modificări să îi aparțină.

```
@app.route("/add_to_wishlist/", methods=['POST'])
@login_required
def add_to_wishlist():
    if current_user.email:
        form = Wishlist_form()
        if form.validate_on_submit():
```

Figură 8 – Segment din „router”-ul ce procesează „URI”-ul adăugării unei cărți în „wishlist”

În Figură 8, se observă o parte din „router”-ul pe care îl utilizează aplicația web pentru a procesa „request” de a adăuga o nouă carte în lista de preferințe.

Astfel funcția „add_to_wishlist()” prin intermediul folosirii decoratorului „@app.route()”(ce face parte din „framework”-ul FLASK) acesta devine un „controller” ce are rolul de procesare al „request”-ului aflat sub forma de „/add_to_wishlist/” executând o acțiune de „POST” prin intermediul căreia este specificat că se adaugă o nouă resursă.

Pentru ca un utilizator să poată realiza această acțiune de adăugare a unei noi cărți în lista de preferințe este necesar ca acesta să fie conectat la aplicație prin intermediul unui cont și parolă validă. În caz contrar utilizatorul nu poate realiza acțiunea prezentată anterior deoarece „router”-ul este securizat prin utilizarea decoratorului „@login_required”, ce face o verificare asupra credențialelor acestuia.

Desigur că verificarea de a fi autentificat nu este singura ce este realizată. Se mai verifică dacă datele introduse în „form”-ul de adăugare al unei cărți sunt valide, adică dacă respectă formatul precizat (numele să nu fie vid, să nu conțină numerele, autorul să nu fie nici el vid sau să nu conțină numere sau alte tipuri de caractere înafara celor permise). În cazul în care nu sunt respectate formatul precizat utilizatorului îi sunt afișate erorile specifice sub câmpul din „form” completat incorect.

În cazul în care validările sunt trecute cu succes, „request”-ul este executat, adică în cazul „router”-ului meu, acesta adaugă o nouă carte în baza de date și returnează un cod de răspuns de succes în cadrul creării (valoarea lui este 201), împreună cu id-ul noii informații adăugate în baza de date.

În caz contrar, adică validările nu se realizează cu succes, sunt returnate diferite coduri „HTTP” de eroare.

De exemplu, în cazul în care utilizatorul realizează acțiunea de a adăuga o nouă carte în lista de preferințe, dar nu este autentificat, „controller”-ul va returna un cod de eroare de tip 401(pentru neautorizat) și în interiorul „view”-ului va fi afișat un mesaj corespunzător.

Astfel este important ca pentru orice „request” să se returneze un cod de eroare în cazul în care nu se trece de etapa validării sau un cod de succes pentru cazul în care acțiunea se realizează.

4.4 Dificultăți întâlnite

Majoritatea problemele pe partea de „back-end” au fost de natura tehnică. În primul rând, un impediment a fost configurarea tuturor tehnologiilor folosite, în principal ordinea în care se realizează această configurare. Pentru a rezolva problema de mai sus am folosit informația din documentația de la link-ul (5). De asemenea util în rezolvarea configurării a fost și următorul blog (6).

O altă problemă destul de importantă a fost referitoare la ștergerea informațiilor din două sau mai multe tabele ce au relații bazate pe chei primare și secundare. Astfel prin ștergerea unui „entry” producea o eroare de „key constraint null” și de aceea era necesar utilizarea unui „force” la ștergerea elementelor sau declararea la momentul creării unui model ca acesta să șteargă elementele ce nu au corespondent definit prin cheie diferită de „null”. Problema ștergerii a fost rezolvată utilizând următorul link (7).

5 „Front-end”-ul aplicației web

5.1 Tehnologii folosite

a) HTML5

Este un limbaj de „HyperText Markup” folosit pentru structurarea, afișarea și formatarea conținutului unei pagini web, compatibil cu orice „browser” existent. Este a cincea versiune și cea mai actuală a „html”-ului.

b) Jinja 2

Este un limbaj de „templating” compatibil cu Python (limbaj folosit pe partea de „backend”). Cu ajutorul acestuia pot crea limbaje de „markup”, cum ar fi HTML5, conținutul „html”-ului este returnat utilizatorului printr-un „request” HTTP. Astfel pot utiliza comparații și iterații prin diferite containere de date, lucru nepermis de HTML5.

c) Css3

Cu ajutorul acestui limbaj stilizez conținutul „html”-ului, descriu ce formatare să aibă textul, asignez diferite comportamente de afișare ale acestora etc.

d) Bootstrap 4

Este un „toolkit” folosit împreună cu CSS3 pentru stilizarea conținutului. Acesta este „open-source” și oferă o multitudine de clase ce sunt deja implementate și ajută la stilizare, făcând ca proprietățile folosite prin intermediul claselor să fie foarte „readable”.

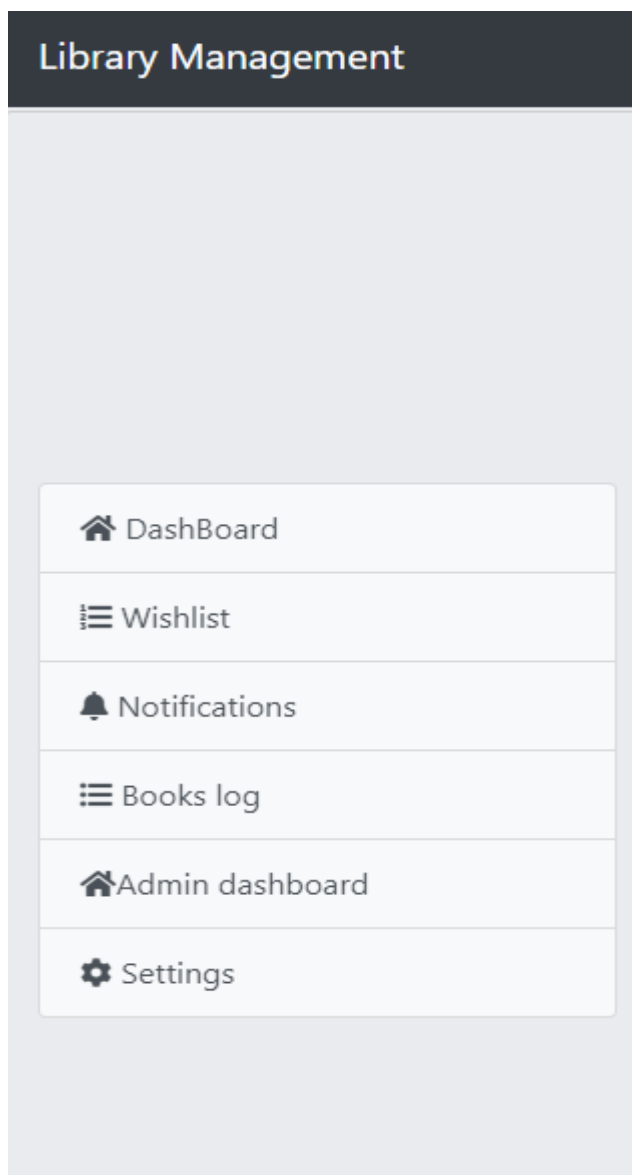
e) JQuery

Este o bibliotecă scrisă în JavaScript. Aceasta face ca „html”-ul din conținutul paginii web să răspundă la „event”-urile create de utilizator, conferă animații acestora. Totodată prin „ajax”-uri se realizează „request”-uri, acestea actualizează informațiile din conținutul paginii fără a face „reload” la pagină, permițând utilizatorului să continue de la locația în pagină de până atunci.

6 Manual de utilizare

6.1 Navigarea prin aplicație

După conectarea cu succes la aplicație orice utilizator este redirecționat pe pagina web „dashboard”, aceasta reprezintă pagina principală a aplicației pentru un utilizator normal.



Figură 9 – „Sidebar” de navigație al aplicației

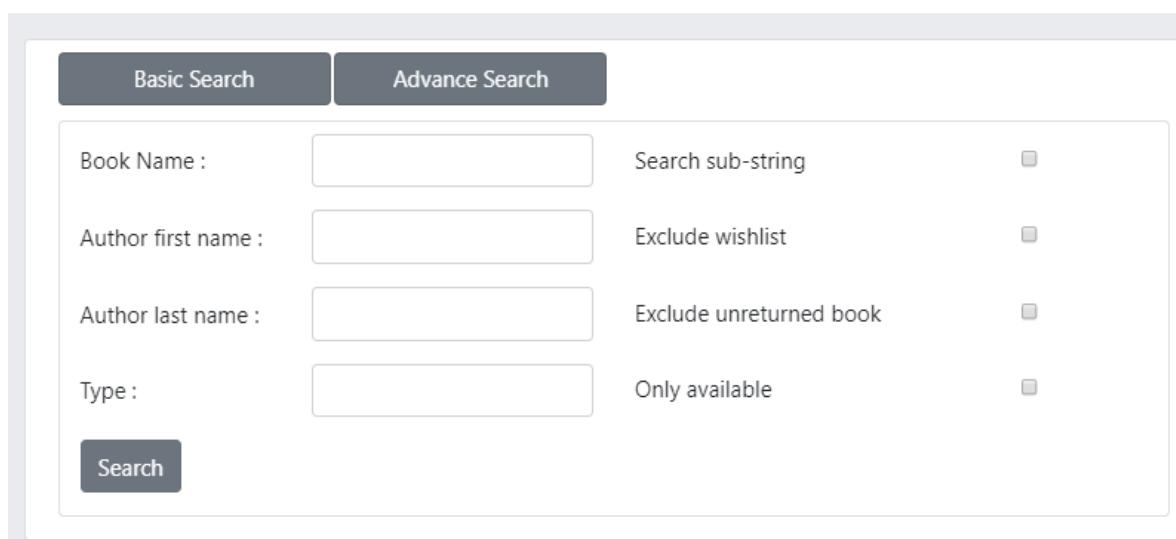
În Figură 9 se poate observa meniul de navigația al aplicației ce cuprinde următoarele :

- „Dashboard” – redirecționează utilizatorul spre pagina principală a aplicației.
- „Wishlist”-ul – după cum sugerează numele este un buton ce permite accesarea liste de preferințe.
- „Notifications” – este pagina de notificări a aplicației.
- „Books log” – buton ce redirecționează utilizatorul spre pagina de istoric al cărților închiriate.
- „Admin dashboard” – utilizatorii ce sunt bibliotecari la conectare sunt redirecționați spre pagina principală la fel ca orice utilizator obișnuit, dar aceștia au în bara de navigație și buton pentru accesarea

funcționalităților de administrator.

- „Settings” - accesează un „pop-up” în care se pot realiza setări pentru „wishlist”

6.2 Căutarea de cărți, rezervarea acestora sau adăugarea în „wishlist”



Basic Search Advance Search

Book Name : Search sub-string ☐

Author first name : Exclude wishlist ☐

Author last name : Exclude unreturned book ☐

Type : Only available ☐


Search

Figură 10 – Prezentarea funcționalităților de căutare a unei cărți

În pagina de „dashboard” utilizatorul poate realiza o căutare normală sau avansată a unei cărți după mai multe criterii prezentate în Figură 10.

Rezultatele căutării sunt prezentate sub formă de tabel în care sunt prezente detalii despre cartea căutată. Dacă statusul acesteia este „Unavailable” prin apăsarea butonului din partea dreapta statusului se deschide un „pop-up” unde cartea respectivă poate fi adăugată la „wishlist” prin completarea câmpului de „rank” și perioadă.

Dacă statusul era „available” cartea se putea rezerva prin apăsarea aceluși buton și completarea câmpurilor referitoare la perioada de închiriere din „pop-up”-ul nou deschis.

No.	Book name	Book type	Author first name	Author last name	Status	
1	New book	Science	New author	New author	Unavailable	

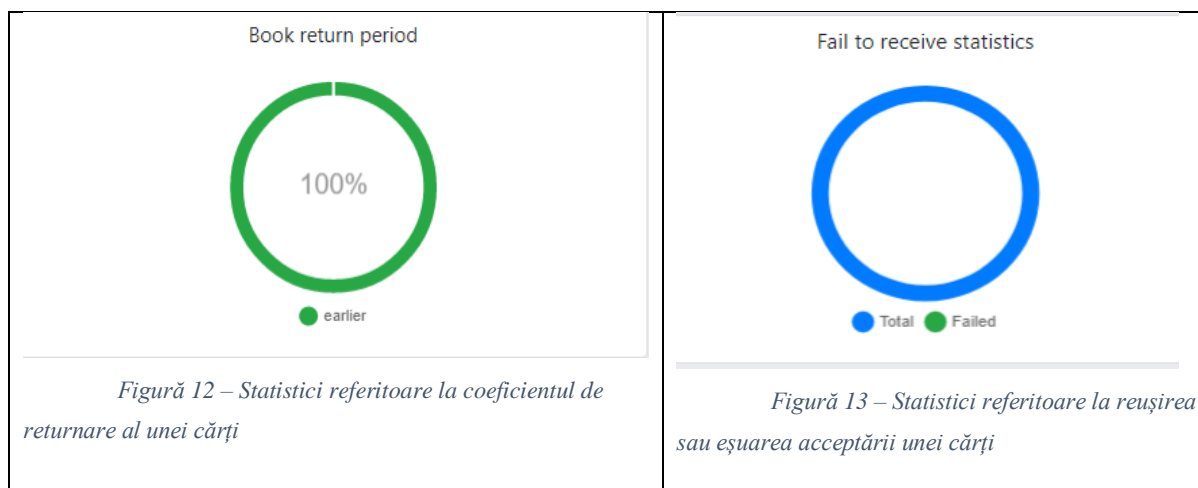
Figură 11 – Prezentarea modului de afișare al rezultatelor

6.3 Interpretarea statisticilor oferite

În pagina de „dashboard” sunt implementate diferite statistici privitoare la istoricul de închiriere al unei cărți. Acestea au rolul de a ajuta utilizatorul pentru a primi cartea dorită din „wishlist”.


De exemplu în Figură 12, utilizatorul poate observa o statistică referitoare la obiceiul său de returnare al unei cărți, astfel în imaginea respectivă acesta returnează cartea de obicei mai devreme cu un procent de „100%” din timpul de închiriere.

În Figură 13 sunt statistici referitoare la numărul de cărți asiguate acestuia din lista de preferințe pe care a reușit să le accepte la timp.

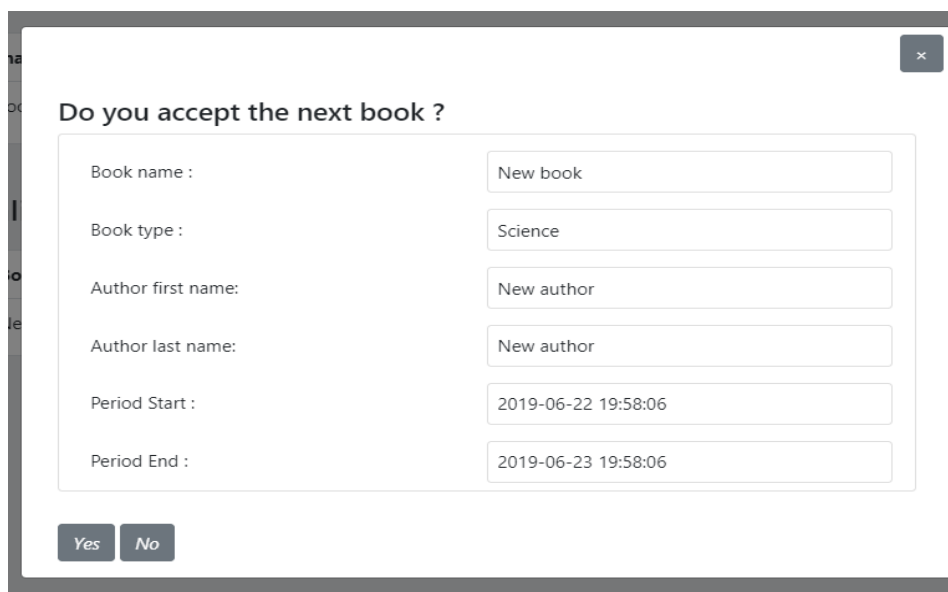


6.4 Acceptarea cărților asiguate din lista de preferințe

În pagina de „wishlist” atunci când unui utilizator îi este asignată o carte, acestuia îi apare la începutul paginii un nou câmp ca în Figură 14.

Accept the next book						
Book name	Book type	Author first name	Author last name	Period Start	Period End	
New book	Science	New author	New author	2019-06-22 19:58:06	2019-06-23 19:58:06	

Figură 14 – Asignarea unei cărți din lista de preferințe



Do you accept the next book ?

Book name :	New book
Book type :	Science
Author first name:	New author
Author last name:	New author
Period Start :	2019-06-22 19:58:06
Period End :	2019-06-23 19:58:06

Yes No

Figură 15 „Pop-up” pentru acceptarea unei cărți din „wishlist”

Prin apăsarea butonului din partea dreaptă, acestuia îi este afișat un „pop-up” ca în Figură 15, prin intermediul căruia se poate accepta cartea asignată, sau se poate refuza. În cazul acceptării aceasta apare în pagina cu istoricul cărților închiriate având statusul de „Unreturned”.

7 Concluzii

Consider că aplicația mea conține funcționalități interesante ce pot optimiza procesul de închiriere al unei cărți prin oferirea unui management automatizat al acestuia.

Prin crearea de rezervări pentru resurse (ce sunt disponibile sau au un număr curent relativ mare, deci nu sunt limitate), sau respingerea acestora de către platformă după un interval de timp, conduce la îmbunătățirea eficienței distribuirii de cărți.

Spre deosebire de cazul acelor cărți pentru care sunt disponibile un număr mare de resurse, în cazul celor limitate administrarea acestora trebuie să fie făcută cu o mai mare atenție. Astfel ca soluție eu am utilizat o listă de preferințe, iar cărțile din interior acesteia sunt asignate pe baza unui algoritm de „stable matching”. Algoritmul atribuie fiecărui utilizator una dintre cărțile dorite în mod eficient ținând cont de istoricul de returnare al acestuia.

Cu toate că am adus noi funcționalități cataloagelor on-line ale bibliotecilor, aplicația mea poate să dispună de unele îmbunătățiri. Acestea pot fi aduse cu ușurință deoarece platforma este scalabilă având o arhitectura bazată pe „design pattern”-ul „MVC”

Funcționalități ce pot fi introduse :

- Adăugarea unui rol de vizitator, prin intermediul căruia un utilizator poate căuta să vadă dacă o carte se află în catalogul bibliotecii, dar nu va putea să realizeze acțiuni precum rezervarea acesteia sau adăugarea la lista de preferințe etc.
- Adăugarea funcționalității de a propune o carte ce nu există deja în catalogul bibliotecii
- Implementarea de notificări pentru noile cărți introduse
- Posibilitatea de închiriere a cărții la sala de lectură în funcție de disponibilitățile sălilor.

Bibliografie

1. *Raport instituțional, Biblioteca Centrală Universitară „Mihai Eminescu”*. Iasi : s.n., 2018.
2. Shapley, D. Gale and L.S. „*College admissions and the stability of marriage*". s.l. : Amer. Math. Monthly, 1962.
3. Kazuo Iwama, Shuichi Miyazaki, Yasufumi Morita, David Manlove. „*Lecture Notes in Computer Science*", capitol „*Stable Marriage with Incomplete Lists and Ties*". s.l. : Springer, Berlin, Heidelberg, 1999. 978-3-540-66224-2.
4. <https://www.distributedpython.com/2018/08/21/celery-4-windows/>.
5. <http://docs.celeryproject.org/en/latest/userguide/configuration.html>.
6. <https://blog.miguelgrinberg.com/post/celery-and-the-flask-application-factory-pattern>.
7. <https://stackoverflow.com/questions/23323947/sqlalchemy-delete-object-directly-from-one-to-many-relationship-without-using-s>.

Index imagini

Figură 1 – Algoritmul de „stable matching” clasic	16
Figură 2 – Interacțiunea dintre „Beat worker scheduler”, „Message Broker RabbitMQ” și „Celery Consumer”	21
Figură 3 – Prezentarea etapelor unui „task”	23
Figură 4 – Structura aplicației	26
Figură 5 – „MVC” arhitectură.....	28
Figură 6 – Partea I din Diagrama bazei de date.....	32
Figură 7 – Partea II din Diagrama bazei de date	33
Figură 8 – Segment din „router”-ul ce procesează „URI”-ul adăugării unei cărți în „wishlist”	34
Figură 9 – „Sidebar” de navigație al aplicației	37
Figură 10 – Prezentarea funcționalităților de căutare a unei cărți	38
Figură 11 – Prezentarea modului de afișare al rezultatelor.....	38
Figură 12 – Statistici referitoare la coeficientul de returnare al unei cărți.....	39
Figură 13 – Statistici referitoare la reușirea sau eșuarea acceptării unei cărți	39
Figură 14 – Asignarea unei cărți din lista de preferințe	39
Figură 15 „Pop-up” pentru acceptarea unei cărți din „wishlist”	40