



**UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ: INFORMATICĂ
APLICATĂ**

LUCRARE DE LICENȚĂ

COORDONATOR:
Conf. Dr. Fortiș Florin

ABSOLVENT:
Vidican Vlad-George

**TIMIȘOARA
2023**

**UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ: INFORMATICĂ
APLICATĂ**

ThesisHub: Platforma web pentru lucrări de licență/Dizertație

COORDONATOR:
Conf. Dr. Fortiș Florin

ABSOLVENT:
Vidican Vlad-George

**TIMIȘOARA
2023**

Rezumat

Arguably one of the most important parts of studenthood is writing a thesis, it being the first big undertaking for most students it is not an uncommon experience for them to feel overwhelmed by the challenge that this presents, whilst not necessarily a "career deciding" moment, it is still a choice not to be taken lightly, as one must commit to spending a considerable amount of effort in developing and learning in one specific area, most then would naturally prefer that their thesis would align as well as possible with their future goals and interests, for some that might sadly prove an even greater challenge as it is not unheard of for last year students to not have a clear path in mind yet, thus having to take a "leap of faith".

That given the case, it can understandably be a stressful experience for students but eventually a fulfilling one as it will hopefully result in personal growth and a sense of achievement. However I do think that removing some of the unnecessary stress, frustration and ambiguity out of this process is a goal worth pursuing, and this is what my thesis aims to achieve.

Thesishub specifically tries to provide a clear process on how to proceed in finding a suitable thesis, having both the option to easily browse and apply for the ones in which they find interest, or if they have a clear goal in mind on what they would like to achieve, they can simply post their own proposal describing their idea and they can just wait for a professor who also shares an interest in their idea to apply, rather than to go "knocking on all closed doors"

Cuprins

1	Introducere	2
	Introducere	2
1.1	Descrierea problemei	2
2	Solutii Similare	3
	Solutii similare	3
2.1	Servicii de consultanta	3
2.1.1	Gradcoach	3
2.1.2	ThesisHelpers	3
2.1.3	Comparație	3
2.2	Platforme de matchmaking	4
2.2.1	Fiverr	4
2.2.2	Freelancer	4
2.2.3	MentorNet	4
2.2.4	Comparatie	4
3	Arhitectura și facilitățile aplicației	6
3.1	Privire de ansamblu	6
3.2	Cazuri de utilizare pentru utilizatorii standard	7
3.3	Cazuri de utilizare pentru administratori	9
3.4	Design-ul bazei de date	11
3.4.1	Privire de ansamblu	11
3.4.2	Analiza colecțiilor din baza de date	11
4	Detalii de implementare	14
4.1	Tehnologiile alese	14
4.1.1	Frontend	14
4.1.2	Backend	15
4.1.3	General	17
4.2	Manual de Implementare	18
4.2.1	Backend	18
4.2.2	frontend	23
4.3	manual de utilizare	27
5	Concluzii și direcții viitoare	37
5.1	direcții viitoare	37
5.2	Concluzii	37

Capitolul 1

Introducere

1.1 Descrierea problemei

Tema aleasă de către mine constă în dezvoltarea unei platforme online destinată studenților în anii terminali la unul dintre ciclurile de învățământ superior și profesorilor universitari în scopul coordonării sau găsirii unui coordonator pentru elaborarea lucrării de licență sau dizertație. Principalul mod în care platforma în cauză își propune să ajute în această privință este prin a facilita comunicarea dintre studenți și profesori, utilizatorii având posibilitatea de a-și posta propriile propuneri de teme sau de a aplica pentru unele din cele disponibile și active în acel moment, reducând astfel potențialul timp mort apărut pentru student în momentul aplicării pentru propuneri de teme a căror disponibilitate nu le este încă cunoscută dar și reducând comunicarea redundantă de ambele părți, propunerile având o secțiune de comentarii unde se pot acoperi eventualele întrebări frecvente.

Capitolul 2

Solutii Similare

2.1 Servicii de consultanta

2.1.1 Gradcoach

*Gradcoach*¹ este o platformă online care oferă suport contra cost studenților în realizarea lucrării de licență/dizertație, suportul variază în funcție de planul ales putând presupune:

- Consiliere one-on-one cu un mentor specialist în domeniul temei alese. În timpul întâlnirii se pot discuta: progresul, eventuale puncte de blocare și sugestii de îmbunătățire.
- Crearea și distribuirea de chestionare. colectarea și compilarea acestor răspunsuri. Transcripție în cazul în care datele colectate vin în format audio sau audio-video și codarea datelor.
- Ajustări pentru variantă finală a lucrării, cum ar fi repararea greșelilor gramaticale, asigurarea consecvenței și respectării standardelor universității în ceea ce privește redactarea lucrării (font, spațiere, cuprins, citate și referințe, numerotarea paginii).

2.1.2 ThesisHelpers

*Thesishelpers*² este o platformă online similară cu gradcoach, în sensul că și aceasta oferă servicii studenților pentru realizarea lucrării de licență/dizertație, aceste servicii fiind:

- Ajustări pentru varianta finală a lucrării.
- Editare ușoară (restructurarea textului, rescrierea anumitor propoziții pentru o alegere mai bună a cuvintelor păstrând însă ideea din spate intactă).
- Realizarea întregi lucrări, clientul fiind responsabil doar de setarea cerințelor pentru această lucrare.

2.1.3 Comparatie

Soluția mea este diferită față de cele două platforme menționate mai sus, deoarece aceasta va fi o platformă internă, non-profit care își propune doar să faciliteze comunicarea dintre

¹<https://gradcoach.com/services/>

²<https://www.thesishelpers.com/about>

studenți și profesori în scopul realizării lucrării necesare pentru absolvirea ciclului de studii ales în timp ce platformele menționate mai sus oferă servicii contra cost a unor consultanți externi, unele dintre aceste servicii presupunând chiar plagiarism și implicit încălcarea integrității academice

2.2 Platforme de matchmaking

O platformă de matchmaking are ca scop fundamental potrivirea de persoane. Matchmaking-ul este un termen-umbrelă destul de vast, care acoperă o multitudine de servicii/platforme, toate aceste servicii având însă ca scop potrivirea unei persoane care are o problemă/nevoie cu o persoană care poate oferi o soluție. În cazul de față, platforma dezvoltată de mine poate fi considerată o platformă de matchmaking, deoarece potrivește studenți care au nevoie de consiliere în scopul realizării unei lucrări de licență/dizertație cu un profesor coordonator care poate oferi această consiliere. Câteva astfel de exemple sunt:

2.2.1 Fiverr

*Fiverr*¹ este o platforma web în care utilizatorii au posibilitatea de a posta servicii pe care aceștia le oferă (ex: proof-reading, voice-over, colectare de date, etc) și prețurile pe care le cer pentru aceste servicii.

2.2.2 Freelancer

*Freelancer*² este o platforma pe care utilizatorii postează proiecte în care detaliază serviciile de care au nevoie și un preț maxim pe care sunt dispuși să îl plătească pentru acel serviciu, iar ceilalți utilizatori pot licita pentru acel proiect (concurând cu ceilalți utilizatori care doresc să preia acel proiect prin prețuri cât mai competitive).

2.2.3 MentorNet

*MentorNet*³ este o organizație non-profit care își propune să pună studenți în contact cu specialiști în domeniul acestora de studiu pentru dezvoltarea unei relații de mentorat. După procesul inițial de asociere a unui student cu un mentor, aceștia vor purta discuții săptămânale pentru a discuta despre proiecte, planuri de carieră, și alte topicuri relevante pentru viitorul studentului.

2.2.4 Comparatie

Platforma dezvoltată de mine se deosebește de primele două exemple, deoarece nu este concepută ca o piață de desfacere pentru servicii. S-ar putea ca unii utilizatori să ofere servicii de consultanță asemănătoare cu *Gradcoach* sau *Thesishelpers* pe aceste platforme, dar în esență acestea se aseamănă doar prin faptul că sunt platforme de matchmaking.

Există o asemănare mai puternică însă între *MentorNet* și platforma dezvoltată de mine, deoarece ambele au ca scop fundamental să faciliteze stabilirea unor relații de mentorat

¹<https://www.fiverr.com>

²<https://www.freelancer.com/jobs>

³<https://greatmindsinstem.org/mentornet/>

în folosul studenților, platforma mea diferențiindu-se de către aceasta prin specificul său. **MentorNet** își asumă o abordare generalistă asupra mentoratului și oferă astfel doar sistemul de matchmaking, care permite inițierea relației de mentorat și un mijloc de comunicare. În schimb, platforma aceasta este concepută în jurul realizării lucrării de licență/dizertație, oferind astfel facilități care să asiste în acest proces. Aceasta este de asemenea concepută drept o platformă internă spre deosebire de **MentorNet**, accesul fiind astfel oferit doar profesorilor și studenților din aceeași universitate, lucru garantat prin faptul că administratorii platformei sunt responsabili de integrarea noilor utilizatori, aceștia fiind nevoiți să atașeze dovezi de identitate odată cu cererea de creare a unui cont, cerere care va fi ulterior validată de un administrator.

Capitolul 3

Arhitectura și facilitățile aplicației

3.1 Privire de ansamblu

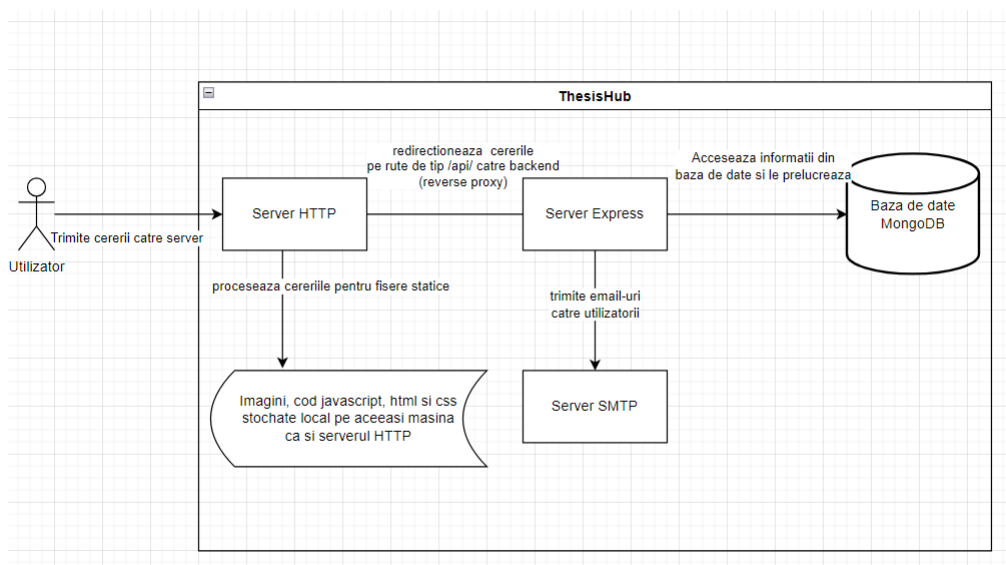


Figura 3.1: Privire de ansamblu asupra sistemului

Aplicația poate fi împartită în următoarele componente:

- Un server http, în momentul de față este cel oferit de către mediul de dezvoltare react, însă acesta poate fi înlocuit cu server http dedicat optimizat pentru ”productie”, acesta va reprezenta punctul de intrare în sistem, toate cererile fiind interceptate de către el, cele pentru resurse statice vor fi rezolvate de către acesta iar cererile pe rutele de tipul /api/ vor fi redirectionate către serverul express.
- Un server de tip express(backend), acesta poate fi considerat piesa centrală a aplicației, fiind cel responsabil pentru procesarea majorității acțiunilor utilizatorilor cât și de validarea acestora și de obținerea datelor necesare pentru ca clientul să poată genera paginile dorite.
- Gestionarea stării este o necesitate pentru orice proiect non-trivial, iar o bază de date face acest lucru fezabil oferind atât persistența datelor cât și o interfață de a lucra cu acestea.

- Un server smtp, acesta permite trimiterea de email-uri de catre sistem utilizatorilor aplicatiei, email-urile reprezentand in acest caz un mod de a comunica cu utilizatorii fara a fii necesara o conexiune intre cei doi.

3.2 Cazuri de utilizare pentru utilizatorii standard

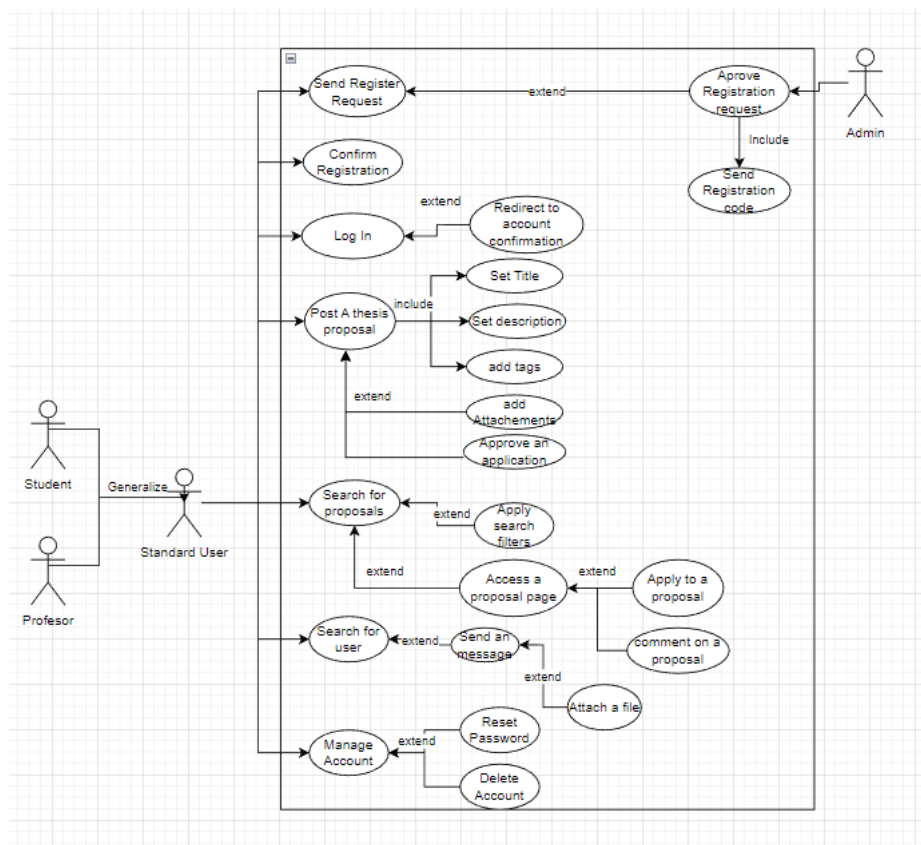


Figura 3.2: Cazuri de utilizare pentru utilizatorii standard

În diagrama de mai sus am ales să generalizez utilizatorul de tip Student și utilizatorul de tip Profesor ca un actor de tip „Utilizator standard” dat fiind faptul ca facilitățile disponibile celor două tipuri de conturi sunt aceleași, diferențierea dintre cele două tipuri de conturi fiind totuși necesară din punct de vedere semantic(nu ar avea sens ca aplicația să permită unui student să coordoneze lucrări, sau unui profesor să aplice pentru tema propusă de alt profesor). Am încercat să cuprind în această diagramă toate cazurile de utilizare disponibile unui utilizator standard cu scopul de a avea o imagine generală asupra facilităților oferite de către platforma, voi analiza detaliat însă publicarea pe platforma a unei noi propunere de lucrări:

Acest caz de utilizare descrie procesul prin care noile propuneri de lucrări sunt adăugate pe platforma

- Actorii:

1. Utilizator standard

- Precondiții:

1. Utilizatorul este de tip Profesor sau Student
2. Contul acestuia trebuie să fi trecut atât de aprobarea înregistrării de către un administrator al platformei cât și confirmarea acesteia de către utilizator
3. Utilizatorul este autentificat pe platformă

- Scenariu:

1. Accesarea paginii pentru crearea unei propuneri de licență
2. Introducerea unui Titlu
3. Introducerea unei Descrierii
4. Selectarea etichetelor potrivite temei propuse din lista de etichete disponibile
 - (a) utilizatorul este nevoit să selecteze cel puțin o etichetă pentru a continua
5. Opțional: Adăugarea eventualelor Atașamente utile pentru descrierea temei propuse celorlalți utilizatorii interesați
6. Dacă oricare dintre câmpurile menționate mai sus este invalid (gol sau prea scurt) atunci utilizatorul este avertizat
7. Noua propunere de lucrare este creata pe platformă

- Postcondiții:

1. Noua propunere este adăugată în baza de date a platformei și devine vizibilă celorlalți utilizatori ai platformei, ei putând interacționa cu această

3.3 Cazuri de utilizare pentru administratori

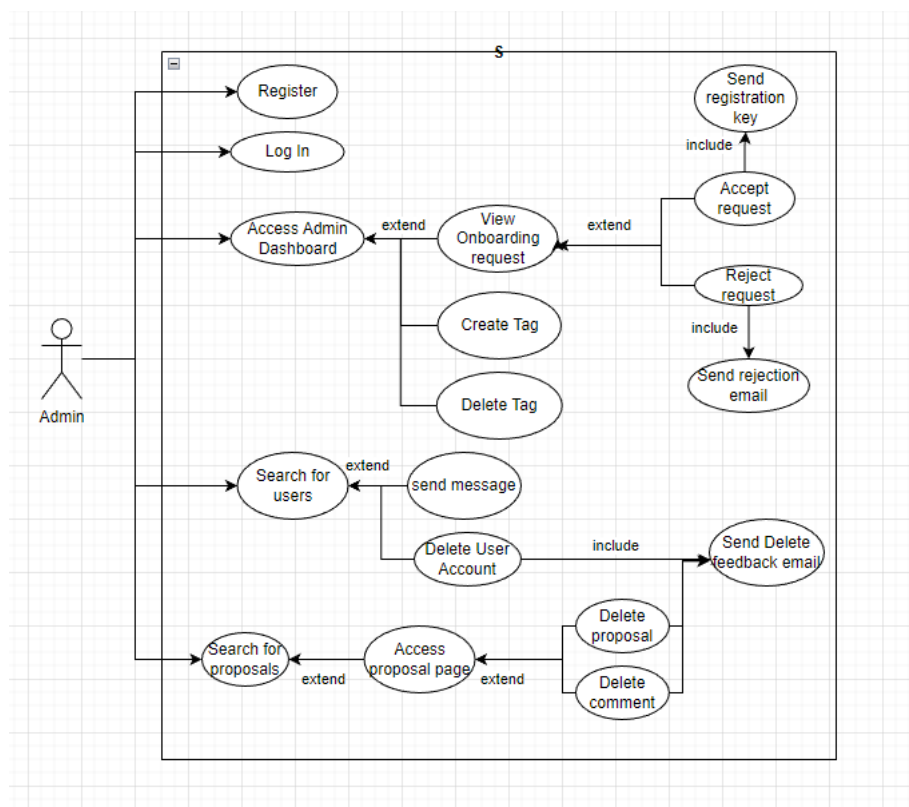


Figura 3.3: Cazuri de utilizare pentru Administratori

Administratorii sunt utilizatorii cu privilegii ridicate, care au posibilitatea și responsabilitatea să gestioneze accesul celorlalți utilizatori la platformă și să modereze conținutul acesteia pentru a păstra calitatea conținutului platformei la standardele universității și a unui mediu academic (eliminarea potențialelor postări/comentarii inadecvate scopului platformei), acest tip de conturi este gândit ca unul strict administrativ neavând astfel capacitatea de a:

- Publica propuneri de lucrări.
- A aplica pentru propunerile existente sau de a lasă comentarii asupra acestora.

funcționalitățile comune cu conturile de utilizator standard fiind doar:

- Autentificarea.
- Căutarea și accesarea propunerilor de lucrări de pe platformă.
- Trimiterea de mesaje celorlalți utilizatori.

Am ales de asemenea că pentru diagramă de mai sus să abstractizez funcționalitatea ce ține de autentificare și trimiterea de mesaje, această fiind identică cu cea a conturilor standard reprezentate deja mai detaliat în diagrama precedentă, mai jos se poate vedea o analiză mai în detaliu a cazului de utilizare pentru aprobarea sau respingerea unei cereri de înregistrare.

- Actorii:

1. Administrator.

- Precondiții:

1. Utilizatorul are un cont de administrator pe platformă.
2. Contul acestuia trebuie să fi trecut atât de aprobarea înregistrării de către un administrator al platformei cât și confirmarea acestuia de către utilizator.
3. Utilizatorul este autentificat pe platformă.
4. Există cel puțin o cerere de înregistrare pe platformă.

- Scenariul standard:

1. Utilizatorul accesează bordul de gestionare al administratorilor.
2. Utilizatorul alege una dintre cererile de înregistrare.
3. Utilizatorul verifică datele afișate în cererea de înregistrare.
4. Utilizatorul verifică dovezile de identitate atașate de persoana care a inițiat cererea.
5. Utilizatorul validează cererea.
6. un email automat va fi trimis de către sistem utilizatorului care a inițiat cererea conținând un cod necesar pentru confirmarea înregistrării.

- Postcondiții:

1. Cererea o dată procesată va dispărea din bordul de gestionare al administratorilor.
2. Un cod de confirmare a înregistrării este generat de către sistem și primit de către persoana care a inițiat cererea pe adresa de email folosită de către aceasta la crearea contului.
3. Utilizatorul care a inițiat cererea are acum potențialul de își confirma contul, având apoi acces la restul funcționalităților de pe platformă.

- Scenariu alternativ:

1. Utilizatorul accesează bordul de gestionare al administratorilor.
2. Utilizatorul alege una dintre cererile de înregistrare.
3. Utilizatorul verifică datele afișate în cererea de înregistrare.
4. Utilizatorul verifică dovezile de identitate atașate de persoana care a inițiat cererea.
5. Utilizatorul respinge cererea.
6. Utilizatorul este nevoit să justifice decizia de a respinge cererea.
7. Un email de informare împreună cu justificarea respingerii oferită de către administratorul care a procesat cererea respectivă este trimis.

- Postcondiții:

1. Cererea o dată procesată va dispărea din bordul de gestionare al administratorilor.
2. Contul nu a fost creat și un email de informare a fost trimis de către sistem.

3.4 Design-ul bazei de date

3.4.1 Privire de ansamblu

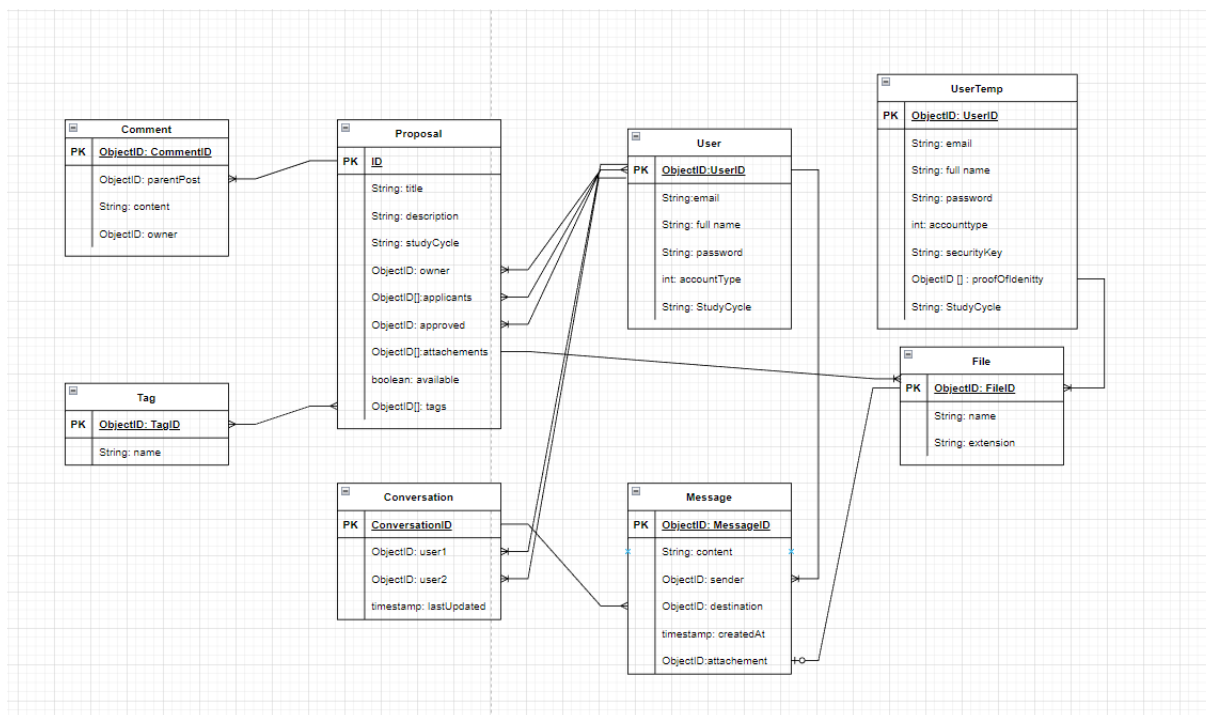


Figura 3.4: Diagrama bazei de date

Mai jos se poate regăsi o scurtă descriere a fiecărei colecții regăsite în diagrama de mai sus

3.4.2 Analiza colecțiilor din baza de date

User

În această colecție vor fi stocate toate documentele care conțin informații despre utilizatorii aplicației, informații precum:

- Nume.
- Email.
- Parola.
- Tip de utilizator (Profesor, Student sau Administrator).
- Ciclu de studiu (Licență sau Master)

parolele sunt de asemenea trecute printr-un algoritm de „hashing”¹ înainte de a fi stocate din motive de securitate.

¹În acest context un proces unidirecțional și determinist de a mapa un șir de caractere la unul nou, de lungime fixă și cu un aspect complet aleatoriu pentru un eventual actor malițios care a obținut acces la baza de date

UserTemp

În această colecție sunt stocate cererile de înregistrare pe platformă, pe lângă câmpurile din colecția „User” avem un câmp în plus care va stoca o cheie de confirmare și referințe către dovezile de identitate aferente cererii din colecția de fișiere, în momentul când utilizatorul creează o cerere de înregistrare pe platformă un document nou va fi creat în această colecție având câmpul cheii de confirmare nul, o dată validată cererea de către administrator o cheie de confirmare va fi generată pentru acest utilizator și transmisă acestuia prin email, odată cu confirmarea înregistrării de către utilizator folosind cheia primită un document nou va fi creat în colecția „User” iar cel aferent cererii de înregistrare va fi șters din colecția „UserTemp”.

Comment

În această colecție sunt stocate comentariile de pe platformă, pe lângă conținutul comentariului se mai stochează și o referință către utilizatorul care a postat comentariul și către propunerea de tema asupra căreia a fost lăsat comentariul.

Tag

În aceasta colecție se găsesc etichetele pe care un utilizator le poate atașa unei propuneri la creare, ștergerea și adăugarea de documente noi în aceasta colecție este rezervată exclusiv administratorilor

Conversation

Această colecție servește funcționalității de mesagerie a platformei, aici sunt stocate conversațiile (doar membrii, nu și conținutul) de pe platformă, împreună cu o data care reprezintă data și timpul ultimului mesaj nou în aceea conversație(necesar pentru afișarea conversațiilor unui utilizator în ordine cronologică.

Message

Această colecție stochează mesajele trimise de către utilizatorii, atributul „sender” conține id-ul utilizatorului care a trimis mesajul iar cel de „destination”, conține id-ul conversației în care a fost trimisă, utilizatorul care dorește să trimită un mesaj nou trebuie evident să fie un membru al acelei conversații.

File

Această colecție stochează numele fișierelor aferente platformei și extensia acestora, fișierele în sine fiind stocate în sistemul de fișiere al serverului, această colecție stocând doar referințe către acestea.

Proposal

această colecție stochează propunerile de teme de pe platformă, aceasta deține câmpuri primitive precum:

- Descriere.
- Titlu.

- Ciclu de studiu.
- Disponibilitate.

dar și referințe către alte colecții precum:

- „owner” reprezintă id-ul utilizatorului care a publicat propunerea de tema.
- „applicants” reprezintă un vector de id-uri ale unor utilizatori care au aplicat pentru propunerea în cauză.
- „attachaments” un vector de id-uri către fișierele aferente propunerii.
- „tags” conține referințe către etichetele atașate propunerii.
- „approved” conține o referință către utilizatorul a cărui aplicație a fost aprobată de către „owner” această referință este nula la creare și rămâne așa până când o aplicație este aprobată.

Capitolul 4

Detalii de implementare

4.1 Tehnologiile alese

4.1.1 Frontend

JavaScript

JavaScript[5] este un limbaj de programare interpretat conceput inițial pentru a rula în browser cu scopul de a manipula conținutul paginilor web și de a facilita astfel aplicații web cu conținut dinamic.

Acesta este un limbaj de programare de nivel înalt, oferind facilități precum : tipizare dinamica, gestionare automată a memoriei și suport atât pentru programare funcțională, tratând funcțiile ca obiecte de ordin prim, cât și pentru programarea orientată pe obiecte. În plus JavaScript dispune de o bibliotecă standard vastă care să acopere majoritatea nevoilor dezvoltatorilor cu privire la funcționalități ”low level”, atât expresivitatea limbajului cât și timpul relativ scurt de învățare a acestuia au contribuit la creșterea popularității sale.

În momentul de față JavaScript se regăsește ca unul dintre cele mai folosite limbaje de programare, acesta fiind între timp extins în afara limitelor unui browser datorita tehnologiilor precum React Native și NodeJS;

React

React[1] este un framework pentru dezvoltarea interfețelor web care are la bază conceptul unui DOM virtual care să servească ca o interfață/intermediar a DOM-ului browser-ului, Acest DOM virtual este reprezentat nativ în JavaScript și stocat în memoria sistemului client, făcând astfel manipularea acestui DOM virtual de către JavaScript mult mai puțin costisitoare.

Un alt concept fundamental pentru react sunt componentele, acestea stând la baza oricărui proiect react. Componentele au în definiția lor atât o reprezentare în DOM-ul virtual cât și stare, starea în acest context poate fi interpretată ca și elementele non-stactice dintr-o pagină web, care depind de utilizator și acțiunile sale. odată ce starea unei componente a fost modificată este inițiat un proces de reconciliere între DOM-ul virtual și DOM-ul browser-ului în urma căruia doar părțile care diferă dintre cele două DOM-uri sunt rerandate în pagina web, celelalte părți rămânând intacte. Datorită acestui fapt aplicațiile react tind să ofere o experiență de utilizare fluidă în schimbul unui consum de memorie crescut.

Componentele de tip react pot conține în definiția lor atât componente primitive cât și alte componente compuse iar componentele pot împărtăși în mod selectiv părți din starea

lor cu componentele din care sunt compuse, în acest fel, componentele care depind de stare comună între ele nu sunt nevoite să gestioneze această stare în paralel, ci, în mod uzual starea comună împreună cu responsabilitatea gestionării ei este elevată la cel mai apropiat părinte comun al acestora iar starea este transmisă mai apoi în cascadă spre componentele care depind de ea.

Atât procesul simplu de gestionare a stării cât și posibilitatea de îmbricare a componentelor duc spre două principii fundamentale în filosofia react, mai exact compozabilitate și reutilizabilitate, aceste 2 principii dovedindu-se extrem de utile în gestionarea complexității. Rata de adopție a react-ului a fost una foarte abruptă datorită funcționalităților inovative pentru timpul lor pe care le-a adus acesta fiind în momentul de față de departe cel mai popular frontend framework.

CSS

CSS (Cascading Style Sheets) este un limbaj pentru descrierea stilului paginilor web. Deși HTML-ul permite stilizarea elementelor individuale folosind atributul de stil (Inline Styling) această abordare devine inefficientă în cazul unui proiect de complexitate ridicată. În mod uzual, HTML-ul este utilizat pentru a defini doar scheletul paginii web, în timp ce aspectul paginii este delegat CSS-ului.

CSS oferă funcționalități precum moștenirea stilurilor, posibilitatea de a grupa elementele, aplicarea de stiluri pe grupuri de elemente și un sistem intuitiv de rezolvare a conflictelor.

Apariția CSS a reprezentat un pas uriaș în evoluția dezvoltării web, revoluționând practic acest domeniu. Deși există alternative precum Sass, Less și Stylus, acestea sunt, în esență, doar extensii ale CSS-ului, iar codul scris în aceste limbaje este compilat în CSS înainte de a fi procesat de către browser. CSS nu are, astfel, vreun competitor real în această perspectivă

4.1.2 Backend

NodeJS

NodeJS[3] este un mediu de rulare dezvoltat pentru a permite execuția programelor scrise în Javascript direct de către sistemul de operare, acesta a fost dezvoltat cu asincronicitate ca o prioritate, în special pentru operațiile de Intrare/Ieșire de date (I/O) ele fiind principala motivație pentru dezvoltarea acestei tehnologii.

Creatorul nodeJS era nemulțumit de abordarea clasică din aceea perioadă de a gestiona operațiile de Intrare/Ieșire pe serverele web, mai exact abordarea în care pentru fiecare cerere primită un nou fir de execuție era creat, această abordare este inefficientă deoarece acele fire de execuție sunt blocate până la finalizarea operației I/O, consumând totuși resurse între timp(memoria rezervată firului de execuție și timpul de procesare folosit pe schimbarea de context).

NodeJS a fost proiectat în schimb să ruleze pe un singur fir de execuție având la baza o buclă de evenimente, interpretorul de Javascript trece prin fiecare linie de cod și când acesta întâmpină cod asincron, precum citirea dintr-o bază de date, delegă operațiile în cauza sistemului de operare.

Bucula de evenimente rulează continuu și verifică dacă operațiile delegate sistemului de operare și-au terminat execuția, în caz afirmativ aceasta apelează o funcție definită să ruleze odată cu finalul execuției codului asincron aferent (Callback function), Arhitectura NodeJS-

ului este principalul motiv datorita căruia acesta este recunoscut și apreciat pentru scalabilitatea sa.

Deși o tehnologie relativ recentă, aceasta a avut o creștere în popularitate remarcabilă, fiind astăzi unul dintre cele mai populare tehnologii pentru backend, atât popularitatea deja existentă a Javascript-ului cât și ușurința de a îl învăța, împreună cu conveniența de a folosi un singur limbaj de programare atât pe backend cât și pe frontend sunt principalele cauze ale popularității sale, coborând de astfel considerabil bariera de intrare pentru dezvoltarea web, fiind un favorit pentru începătorii din acest domeniu.

Express

Express[2] este un framework minimalist și neopinionat al nodeJS-ului dedicat pentru dezvoltarea serverelor web, acesta poate fi considerat minimalist deoarece oferă o suită relativ mică de facilități, focusându-se pe anumite funcționalități generale, de exemplu: rutarea cererilor http, ușurând procesul de a lucra cu rute dinamice și oferind posibilitatea de a grupa rutele, un sistem de gestiune a fluxului cerere-răspuns, și suport pentru șablonarea paginilor în cazul aplicațiilor unde randarea se face pe server.

Express prioritizează integrabilitatea și extensibilitatea să cu alte librării, delegând comunității datoria de a dezvolta librării pentru a acoperii cazuri de utilizare specifice și dezvoltatorului datoria de a se documenta și alege "uneltele" necesare pentru proiectul său.

Acesta mai este descris și ca neopinionat deoarece își propune să nu impună dezvoltatorilor o anumită paradigmă, arhitectura software sau set de convenții / "bune practici" lăsând dezvoltatorilor atât libertatea cât și responsabilitatea de a alege acestea în funcție de particularitățile proiectului și a echipei în care lucrează.

Abordarea liberală a expressului a rezultat într-un ecosistem vast și în continua creștere în jurul său, fiind astăzi unul dintre cele mai populare librării NodeJS folosite.

O funcționalitate care stă la baza express-ului este support-ul pentru middleware, pentru fiecare ruta definită avem cel puțin o funcție care va fi apelată odată cu intrarea unui cererii pe aceea ruta, important este însă faptul ca express ne permite să adăugăm și o lista de funcții intermediare care vor fi apelate secvențial, oricare dintre acestea putând oprii prematur execuția(de regula în cazuri de eroare) sau de a apela următoarea funcție din lanț, opțional acestea pot și adăuga/modifica date din cererea primită înainte de a apela următoare funcție, acesta reprezentând un mecanism de a persista stare între diferitele funcții implicate în ciclul de viață al unei cererii, acest fapt crescând semnificativ potențiala modularitatea a codului

MongoDB

MongoDB[4] este un sistem de gestionare a bazelor de date de tip NoSQL bazat pe documente (echivalentul rândurilor într-o baza de date de tip SQL) și colecții (echivalentul tabelelor în baze de date SQL), documentele sunt scrise în format BSON (Binary Javascript Object Notation), Acest format suportând obiecte(imbricarea obiectelor fiind suportată de asemenea) și vectorii de valori primitive sau de obiecte în plus fata de tipurile primitive comune cu SQL, Acest fapt oferă o flexibilitate semnificativ mai mare în proiectarea unei baze de date spre deosebire de un DMBS de tip SQL, această flexibilitate vine însă la costul consistenței, o colecție nu impune în mod nativ o schemă asupra documentelor pe care le are în apartenență, astfel în mod teoretic se poate ca două documente din aceeași colecție să aibă o structura diferită, de regulă sunt impuse însă soluții de validare a operațiunilor de creare/modificare pentru a putea asigura o consecvență a datelor.

Mongoose

Mongoose este o librărie de mapare a datelor în obiecte (Object-Data Modeling), ODM-urile oferă o interfață de tip OOP în limbajul de programare ales pentru a interacționa cu baza de date NoSQL aleasă, condiția fiind evident că ODM-ul ales să suporte atât limbajul de programare ales cât și baza de date. Mongoose oferă o interfață în javascript pentru a interacționa cu baze de date de tip MongoDB, pe lângă aceasta conveniența de a putea comunica cu baza de date scriind cod direct în javascript și suita de funcții care abstractizează detalii de implementare ale unor nevoi comune (de exemplu `findByIdAndDelete`), acesta mai are un beneficiu foarte important deoarece ajută în diminuarea principalului dezavantaj al MongoDB-ului, mai exact lipsa de consistență. Mongoose interacționează doar cu colecțiile din baza de date pentru care are un model definit, la baza acestui model stă o schema definită în JSON pe care obiectele din aceea colecție ar trebui să o respecte. Orice încercare de a crea / modifica obiecte în așa fel încât nu vor respecta schema definită va genera o eroare, acest lucru nu rezolvă în totalitate problema din păcate însă deoarece validează doar operațiile de scriere pe care acesta le primește, baza de date putând fi încă modificată direct și astfel aceasta ar putea ajunge într-o stare inconsistentă. Practic mongoose garantează consistența bazei de date doar dacă scrierea în aceasta se face în mod exclusiv prin mongoose.

4.1.3 General

HTTP

HTTP (Hypertext Transfer Protocol) este un protocol de comunicare construit pe baza protocolului TCP/IP cu scopul de a standardiza comunicarea între clienți și serverele web, acesta urmează modelul "cerere-răspuns", în care clientul și server-ul stabilesc o conexiune, clientul apoi trimite o cerere serverului iar serverul trimite înapoi un răspuns, după care conexiunea este închisă, protocolul HTTP stă la baza web-ului acesta fiind protocolul folosit pentru marea majoritate a traficului web, acesta are o limitare însă, cum conexiunea poate fi inițiată doar de către client, iar aceasta este închisă automat odată cu primirea răspunsului din partea serverului cererile de tip HTTP nu sunt potrivite pentru cazurile de utilizare unde se dorește ca serverul să trimită mesaje către client, cum ar fi în cazul de față funcționalitatea pentru mesagerie, în care se dorește ca serverul să trimită clientului mesajele primite în timp real. Există câteva "soluții" pentru a suporta această funcționalitate, cum ar fi HTTP polling unde clientul trimite cereri către server pentru a obține mesajele noi la un interval fix de timp, problema cu această abordare fiind echilibrul dintre responsivitate și performanță, în această abordare dacă timpul de așteptare dintre cereri este prea lung atunci experiența utilizatorului se degradează considerabil, însă dacă cererile sunt prea frecvente atunci performanța sistemului va avea de suferit, crescând drastic numărul de cereri trimise în timp ce majoritatea acestora vor fi redundante.

O altă soluție ar fi HTTP long polling, unde se trimite o singură cerere, iar serverul blochează acel fir de execuție (fiind creat un fir de execuție diferit pentru fiecare cerere) până când acesta are date de trimis clientului, după care răspunsul este trimis și conexiunea este închisă, iar apoi clientul inițiază o nouă astfel de cerere, această abordare este mai eficientă decât prima însă este tot suboptimă, deoarece o bună parte din resursele serverului vor fi blocate la orice moment dat.

WebSockets

WebSockets este un protocol construit cu scopul de a rezolva problemele cauzate de limitările protocolului HTTP, construit tot pe baza TCP/IP acesta oferă posibilitatea unei comunicări bidirecționale și cu latență redusă între client și server, făcând astfel aplicațiile web care necesită comunicare în timp real fezabile, în acest protocol conexiunea dintre client și server este proiectată să persiste iar atât clientul cât și serverul pot să trimită evenimente celuilalt oricând iar aceste mesaje sunt mult mai rapide deoarece acestea nu trebuie să includă ”bagajul” necesar pentru stabilirea unei conexiuni noi spre deosebire de http

4.2 Manual de Implementare

4.2.1 Backend

Arhitectura sistemului

fișierele care conțin codul sursă sunt împărțite logic în următoarele categorii:

- **Models:** acest folder conține modelele mongoose aferente colecțiilor din baza de date
- **Controller:** în acest folder se regăsesc funcțiile definite pentru a trata cererile primite
- **Routes:** aici sunt definite rutele pe care le accepta serverul și sunt legate de funcțiile care ar trebui să ruleze la apelarea rutelor respective

după cum se poate observa în figura 4.1 există o mapare de aproape 1:1 între modele și controllere, arhitectura implementată a fost în mare parte inspirată de către MVC(Model View Controller), funcțiile care creează/returnează/modifică sau șterg obiecte ce aparțin de un anumit model sunt grupate sub un singur fișier numit controller legat logic de model-ul aferent, această implementare nu respectă în totalitate arhitectura MVC totuși , deoarece backend-ul în acest caz este un rest api care comunică în mod exclusiv prin JSON fișierele statice(HTML, CSS și javascript) nefiind sub responsabilitatea acestuia, partea de View lipsind astfel complet din această arhitectură.

- **Config:** aici se regăsesc funcții pentru conectarea la baza de date și conectarea serverului de express cu serverul de SMTP pentru funcționalitățile ce implică trimiterea de email-uri
- **Middleware:** acest folder conține funcțiile intermediare rulate înainte de cele din controller, în mare parte funcționalitate pentru validarea request-ului dar sunt și unele funcții menite pentru a adăuga informații în request-ul primit de la client
- **Utils:** aici se regăsește de regulă cod necesar/util care nu potrivește însă nici unei alte locații din punct de vedere semantic. în cazul de față o clasă care extinde clasa de bază ”Error”, care primește ca și atribut pe lângă un mesaj de eroare și un cod de status util în generarea și tratarea erorilor întâmpinate de un server web

fluxul de control

Pentru a putea înțelege mai bine modul de funcționare a serverului, consider că ar fi utilă urmărirea cap-coadă a ciclului de viață a unui request, în cazul de față o cerere care constă în aprobarea unei aplicații ar fi cea mai potrivită.

Odată cu pornirea serverului, acesta creează o instanță nouă de express, și îi specifică acesteia că pentru oricare cerere a cărei destinații începe cu "api/proposal" să folosească rutele definite în proposalRoutes.

```
import express from "express";
import proposalRoutes from "../routes/proposalRoutes.js";

const app = express();
app.use("/api/proposal", proposalRoutes);
```

În acest set de rute o avem pe următoarea definită: "":"/proposalID/approve" unde "proposalID" este un parametru, acesta nu are o valoare predefinită, ci este mai degrabă un substituent, practic orice request de tipul: /api/proposal/îd_/approve unde id poate să ia orice valoare va intra pe această rută, valoarea acestuia fiind făcută disponibilă funcțiilor care tratează cererea, acestea putând accesa proprietatea req.params.proposalID

```
import express from "express";
import { auth } from "../middleware/auth.js";
import { addProposalToReq ,
        isProposalOwner ,
        validateApproval } from "../middleware/validators/proposalVa
import { approveApplication } from "../Controllers/proposalContro
import { isAuthorized } from "../middleware/validators/userVali

const router = express.Router();
router.put("/:proposalID/approve", auth ,
        addProposalToReq ,
        isProposalOwner ,
        isAuthorized ,
        validateApproval ,
        approveApplication );
export default router;
```

Odată intrat un astfel de request, prima funcție care se apelează este cea de autorizare, pentru a vorbi despre aceasta însă, consider că o scurtă explicație despre Json Web tokens (JWT) este necesară deoarece aceasta tehnologie stă la baza sistemului de autentificare și autorizare.

În urma unei autentificări reușite serverul generează un hash pe baza încărcăturii token-ului (payload-ul, în cazul de față id, tip cont și ciclul de studii) timpul și o cheie secretă data serverului ca și variabila de mediu (environment variable), token-ul nu poate fi descifrat fără a avea acces la cheia secretă folosită pentru a genera hash-ul iar token-ul nu poate fi alterat de către un eventual actor malițios deoarece oricare alterare va rezulta într-un hash invalid (modificarea unui singur caracter oarecare din cele folosite pentru generarea token-ului va genera un hash complet diferit, astfel nu se poate "ghici" ce modificări pot fi aduse unui token deja existent pentru a îi modifica payload-ul), Odată acest token primit clientul îl salvează în memoria locală și îl atașează la fiecare request, serverul cunoscând secretul cu care a fost "semnat" token-ul îl poate descifra pentru a avea acces la acest payload și astfel identifica cine este utilizatorul care a inițiat request-ul în cauză. Revenind la funcția noastră, aceasta verifică dacă request-ul primit conține un header de autorizare de tip "Bearer token", și dacă da, încearcă să îl descifreze folosind codul secret, în cazul în care header-ul este prezent și poate fi descifrat (este valid) atunci adaugă în request payload-ul token-ului

și apelează următoarea funcție din lanț, în caz contrar trimite clientului un răspuns de tip eroare și oprește procesarea acelui request

```
import jwt from "jsonwebtoken"

function auth (req, res, next){
  if(req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")){
    try {
      let token = req.
        headers.
        authorization
          .split(" ")[1];
      req.user = jwt
        .verify(token, process.env.AUTH.SECRET);
      next();
    } catch (error) {
      console.log(error)
      return res.status(500).json(error);
    }
  }
  else{
    return res.
      status(400)
      .json({msg: "auth_token_not_found_or_invalid"})
  }
}
```

următorul pas este să verificăm dacă id-ul primit ca și parametru este unul valid, mai întâi verificăm dacă s-a primit o valoare pentru acest ID sau acesta este null iar în caz afirmativ, obținem din baza de date obiectul aferent acelui ID și îl atașăm cererii în același mod ca și în pasul anterior iar într-un final apelăm următoarea funcție din lanț. în cazul în care în care nu a fost oferit un ID sau nu există acest ID în baza de date, returnăm din nou eroare

```
async function addProposalToReq(req, res, next){
  try {
    validateString(req.params.proposalID);
    const proposal = await Proposal
      .findById(req.params.proposalID)
      .populate({
        path: "owner",
        model: "User"
      })
    if(!proposal){
      throw new customError('the proposal ID provided
        does not exist', 400);
    }
    req.post = proposal;
    next();
  } catch (error) {
```

```

        if(error instanceof customError){
            return res
                .status(error.statusCode)
                .json({msg: error.message})
        }
        console.log(error);
        return res.status(500).json(error);
    }
}

```

următorul pas este să verificăm dacă utilizatorul care a inițiat cererea este și destinatorul propunerii de teme, în caz afirmativ, se adaugă un nou atribut în request, numit `auth` și este setat ca adevărat. La final se apelează următoarea funcție din lanț indiferent de rezultatul condiției

```

async function isProposalOwner (req , res , next){
    try {
        if(req.user.id == req.post.owner._id){
            req.auth = true;
        }
        next();
    } catch (error) {
        return res.status(500).json({msg: error});
    }
}

```

apoi se verifică dacă utilizatorul este autorizat sau nu pentru a iniția cererea în cauză, acest lucru se face verificând dacă vreo funcție anterioară lui `isAuthorized` a setat deja atributul `req.auth` ca adevărat, în caz afirmativ acesta apelează doar funcția următoare din lanț, iar în caz contrar returnează o eroare de tipul `"permisiuni insuficiente"` această abordare simplifică semnificativ implementarea unui sistem de permisiuni complex, dat fiind modularitatea sa acesta presupune doar definirea unor middleware-uri care verifică unele condiții simple, și apoi la definirea rutei acestea se pot înlănțui în funcție de nevoie, un exemplu pentru asta ar putea fi: o rută pentru ștergerea unui comentariu și o rută pentru editarea unui comentariu, ar avea sens ca atât persoana care a postat comentariul, cât și persoana care deține postarea în care a fost postat comentariul, cât și un administrator să poată șterge comentariul în cauză însă doar persoana care a postat comentariul ar trebui să aibă permisiunea de a îi modifica conținutul, în acest context ambele cazuri se pot rezolva implementând următoarele funcții: `isCommentOwner`, `isProposalOwner`, `isAdmin` în același mod ca și în pasul precedent iar apoi pentru ruta de ștergere apelăm următoarea listă de funcții `"..., isCommentOwner, isProposalOwner, isAdmin, isAuthorized, ..."` iar pentru cea de editare apelăm doar `"...,isCommentOwner, isAuthorized, ..."`

```

async function isAuthorized(req , res , next){
    try {
        if(!req.auth)
            return res
                .status(400)
                .json({msg: '"Insufficient permissions to proceed with this request'" });
        next();
    } catch (error) {

```



```

        console.log(error);
        return res.status(500).json({msg: error});
    }
}

```

după autorizare, următorul pas este validarea cererii, funcția verifică dacă propunerea respectivă are deja o aplicație aprobată și dacă id-ul utilizatorului aplicant lipsește din baza de date, dacă cel puțin una dintre aceste două sunt adevărate atunci funcția va returna o eroare și va opri execuția în caz contrar, se apelează ultima funcție din acest lanț

```

async function validateApproval(req, res, next){
    try {
        if(req.post.approved){
            throw new customError('‘‘you have already approved
            an application for this particular proposal‘‘', 400);
        }
        const user = await User
        .findById(req.body.applicantID);
        if(!user){
            throw new customError('‘‘the provided applicant ID
            does not exist‘‘', 400);
        }
        next();
    } catch (error) {
        if(error instanceof customError){
            return res
                .status(error.statusCode)
                .json({msg: error.message})
        }
        console.log(error);
        return res.status(500).json(error);
    }
}

```

pasul final, atributul "approved" al postării este modificat din null în id-ul utilizatorului a cărei aplicație a fost aprobată și apoi funcția trimite un răspuns cu un mesaj de succes clientului

```

async function approveApplication (req, res){
    try {
        const post = req.post;
        post.approved = req.body.applicantID;
        await post.save()
        return res
            .status(200)
            .json({msg: "application approved!"});
    } catch (error) {
        console.log(error);
        return res.status(500).json(error);
    }
}

```

}

4.2.2 frontend

arhitectura sistemului

Codul sursa este împărțit în următoarele categorii:

- **Pages:** aici se regăsesc componentele React aferente paginilor platformei, acestea sunt componentele la cel mai înalt nivel de abstractizare, fiind compuse la rândul lor din alte componente React de nivel înalt.
- **css:** în acest folder se regăsesc toate fișierele ce țin de stilizarea componentelor, codul sursă de css a fost împărțit în fișiere în funcție de pagina de care aparține
- **api:** Acest folder conține funcțiile folosite pentru trimiterea de cereri http către backend, funcțiile respective sunt importate în anumite componente React și apelate în urma unor evenimente(de exemplu la încărcarea inițială a componentei sau în urma unei apăsări de buton de către utilizator). Comunicarea cu server-ul pentru funcționalitatea de mesagerie nu se regăsește aici însă, pentru această componentă comunicarea cu serverul este realizată folosind protocolul WebSockets și nu http, evenimentele după care client-ul ”asculta” pot fi definite doar după stabilirea unei conexiuni cu clientul, din acest motiv logica ce ține de mesagerie a fost încapsulată în pagina dedicată pentru mesagerie.
- **Images:** Aici se regăsesc imaginile ce țin de aplicația web în sine, acest folder este diferit de cel ce conține atașamentele aferente mesajelor/propunerilor de pe platformă. În cazul de față acest folder conține logo-ul aplicației
- **Utils:** în acest folder se regăsește funcționalitatea de validare de pe front, aceste funcții sunt apelate înainte de trimiterea unor cereri către server. Este de preferat identificarea cererilor greșite preventiv pe cât de mult posibil, deoarece dacă suntem într-o situație în care putem ști sigur că cererea va rezulta într-o eroare încă de pe front-end(un câmp obligatoriu lipsește de exemplu), atunci clientul poate să îl anunțe pe utilizator direct, evitând astfel trimiterea unei cereri și așteptarea unui răspuns pentru ea. reducând în acest fel atât timpul de așteptare al utilizatorului cât și încărcătura pe care o suportă server-ul
- **Components:** în acest fișier sunt grupate restul componentelor de tip React. Aici se regăsesc atât componente simple (a căror definiții dețin doar elemente primitive) cât și componente de nivel înalt (a căror definiție conțin alte componente de tip React). Pentru a exemplifica diferența dintre cele două tipuri putem compara două astfel de componente, cum ar fi Checkbox și taglist: după cum se poate observa mai jos componenta Checkbox este compusă dintr-un div(container) și un element de tip p (paragraf) ambele putând fi considerate elemente primitive.

```
export const Checkbox = ({ text ,
  tagID ,
  checked ,
  onClick }) => {
  function handleChange () {
```

```

        onClick({ tagID , text });
    }

    return (
        <div className = {checked ?
            "checkbox-checked"
            : "checkbox-unchecked"}
            onClick={handleChange}>
            <p>{text}</p>
        </div>
    )
}

```

Componenta TagList însă are în definiția sa alte componente compuse de tip React după cum se poate vedea, din acest motiv poate fi considerată ca o componentă de nivel înalt

```

export const TagList = ({ tags ,
    onChange ,
    resetView ,
    resetTags ,
    categoryID })=>{
    return (
        <div className="taglist">
            <button onClick={resetView}
                className="checkbox-unchecked">
                back to categories
            </button>
            <button onClick={resetTags}
                className="checkbox-unchecked">
                reset tags
            </button>
            <Divider orientation='horizontal' />
            <div className="taglist-tags">
                {
                    tags.map(tag =>{
                        return (tag.category == categoryID)?
                            <Checkbox text={tag.text}
                                tagID = {tag.id}
                                checked = {tag.checked}
                                onClick = {onChange}
                                key = {tag.id} >
                            </Checkbox> : null;
                    })
                }
            </div>
        </div>
    )
}

```

Ierarhia componentelor

prima componentă din ierarhie este App, aceasta încorporează de fapt întreaga aplicație, react-ul generează ”aplicații cu o singură pagină” (Single Page Applications), mai exact la accesarea paginii se trimite către client codul necesar pentru a putea genera întreaga aplicație, în mod clasic clientul face o cerere spre server și primește codul html, css și javascript pentru fiecare navigare între pagini, fiecare pagină fiind separată, React însă doar manipulează dom-ul browser-ului pentru a randa subcomponente diferite a acestui element de rădăcină în funcție de URL pe care se află utilizatorul și simulând astfel comportamentul unei aplicații cu mai multe pagini

```
import { AuthPage } from './Pages/AuthPage';
import { HomePage } from './Pages/HomePage';
import { CreateProposalPage } from './Pages/CreateProposalPage';
import { ProposalPage } from './Pages/ProposalPage';
import { MessagesPage } from './Pages/MessagesPage';
import { AdminDashboard } from './Pages/AdminDashboard';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<HomePage/>} />
        <Route path="/auth" element={<AuthPage/>} />
        <Route path="/proposal/create"
          element = {<CreateProposalPage/>} />
        <Route path="/proposal/:proposalID"
          element = {<ProposalPage/>} />
        <Route path="/messages/"
          element = {<MessagesPage/>} />
        <Route path="/admin/"
          element = {<AdminDashboard/>} />
      </Routes>
    </Router>
  )
}
export default App;
```

În cazul de față pentru fiecare rută definită este asociată o componentă de tip pagină. Una dintre ele este pagina propunerilor, aceasta este o rută dinamică, id-ul propunerii fiind dat ca parametru. Această pagină are în componența sa:

- Bara de navigare: aceasta componenta conține butoane pentru redirectarea spre celelalte pagini și este regăsită de altfel în toate celelalte pagini
- ProposalItem: aceasta componenta conține datele propunerii, mai exact: titlu, descriere, detinator, ciclu de studii, etichete(tags) și atașamentele propunerii
- CommentSection: aceasta componenta conține atât lista de comentarii de pe platforma cât și componenta pentru adăugarea comentariilor noi

```

<div className="ProposalPage">
  <Navbar/>
  <VStack>
    <div className="proposal-page-main-main">
      {!loading && <ProposalItem
        proposalData={proposal}
        updateProposalApplications = {updateProposalApplications}
        updateProposalApproved = {updateProposalApproved}/>
      <CommentSection proposalID={proposalID}/>
    </div>
  </VStack>
</div>

```

componenta "Comment-section" primește de la părinte id-ul cererii, și la înaintarea acesteia trimite o cerere spre server pentru a obține lista de comentarii aferente propunerii, iterează apoi prin lista primită și generează un CommentItem(comentariu) pentru fiecare element din listă.

```

{
  !isLoading &&
  <div className="Comment-section">
    <div className="Comments" ref={commentListRef}>
      {
        commentList.map(comment =>{
          return <CommentItem
            key={comment._id}
            comment={comment}/>
        })
      }
    </div>
    <div className="Add-comment">
      <textarea id="message"
        name="message"
        placeholder= {`press <Shift> + <Enter>
          to send the message ...`}
        ref={textboxRef}
        onKeyDown={(e)=>{handleChange(e)}}>
      </textarea>
    </div>
  </div>
}

```

Iar componenta "comment-item" primește de la componenta părinte conținutul comentariului ca și stare și pe baza acestuia se generează conținutul comentariului, sintaxa JSX permite definirea de blocuri de cod javascript(intre acolade) aceste blocuri permit să generăm conținut dinamic, componentele putând astfel avea variabile în definițiile lor. Această sintaxă ne permite și să randăm în mod condițional un element, după cum se poate vedea mai jos, iconița aferentă autorului comentariului este definită condițional folosind operatorul terțiar în funcție de tipul de cont al autorului.

```
import { GiTeacher } from "react-icons/gi";
```

```

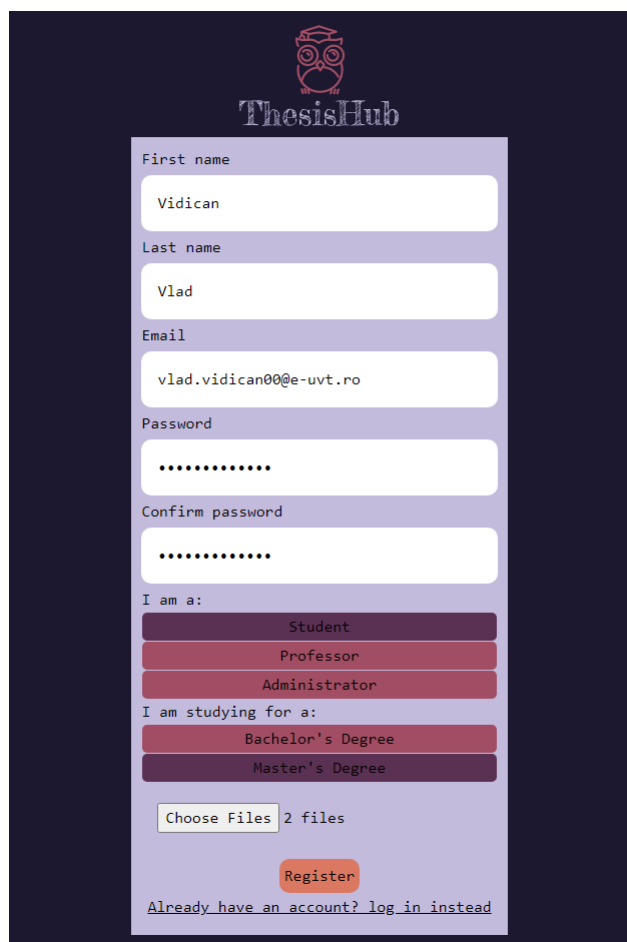
import { FaUserGraduate } from "react-icons/fa";
export const CommentItem = ({ comment }) =>{
  return(
    <div className="comment-item">
      <p>{comment.content}</p>
      <div className="comment-item-footer">
        <div className=
          "comment-item-footer-nameAndType">
          {comment.owner.type === "Student" ?
            <FaUserGraduate/>:
            <GiTeacher/>}
          <p>{comment.owner.name}</p>
        </div>
        <p>{new Date(comment.postedAt)
          .toLocaleDateString("ro-RO",
            {
              day: "numeric",
              month: "numeric",
              year: "numeric"
            })
        }</p>
      </div>
    </div>
  )
}

```

4.3 manual de utilizare

Primul pas necesar pentru a putea folosi platforma este trimiterea unei cereri de înregistrare, pentru asta este necesar ca utilizatorul să acceseze formularul de înregistrare și să completeze următoarele informații:

- Nume și prenume
- adresa de email
- parola
- tipul de cont (putand alege între Student Profesor și Administrator)
- în cazul în care tipul de cont este student, atunci mai trebuie ales și ciclul de studii
- Atașamente continand dovezi de identitate



The image shows a registration form for ThesisHub. At the top is the ThesisHub logo, which features a stylized owl head. Below the logo, the form contains several input fields and buttons. The fields are labeled 'First name', 'Last name', 'Email', 'Password', and 'Confirm password'. The 'First name' field contains 'Vidican', 'Last name' contains 'Vlad', and 'Email' contains 'vlad.vidican00@e-uvt.ro'. The 'Password' and 'Confirm password' fields are filled with dots. Below these fields are two sections of radio buttons. The first section is labeled 'I am a:' and has three options: 'Student', 'Professor', and 'Administrator'. The second section is labeled 'I am studying for a:' and has two options: 'Bachelor's Degree' and 'Master's Degree'. Below these sections is a 'Choose Files' button followed by '2 files'. At the bottom of the form is a red 'Register' button and a link that says 'Already have an account? log in instead'.

Figura 4.1: formular de înregistrare

Apoi utilizatorul va primi un cod de securitate pe adresa de email folosită la crearea contului, pe care va trebui să îl introducă în interfața deschisă după trimiterea cererii, această interfață îi permite de asemenea utilizatorului să solicite un cod nou de securitate în cazul în care mail-ul inițial nu a fost livrat.

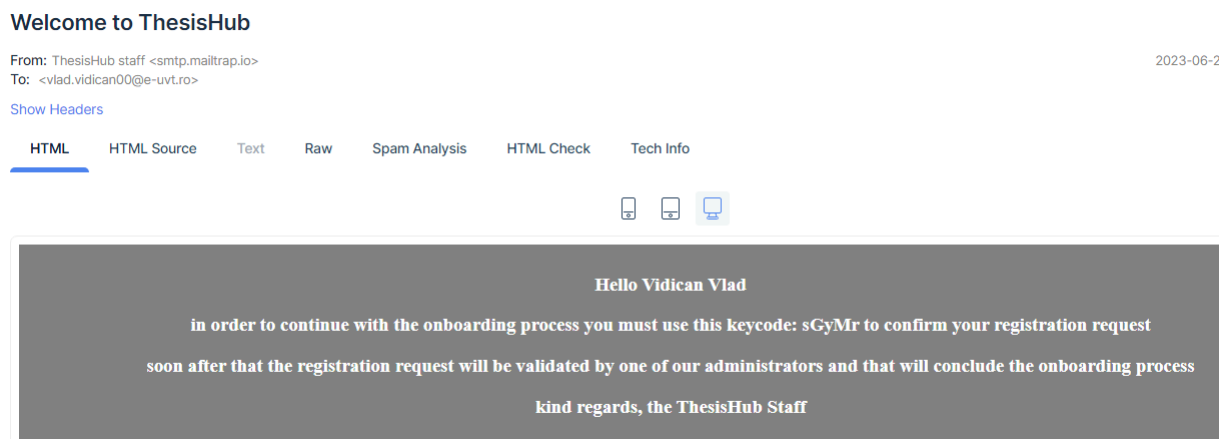


Figura 4.2: exemplu email primit la înregistrare

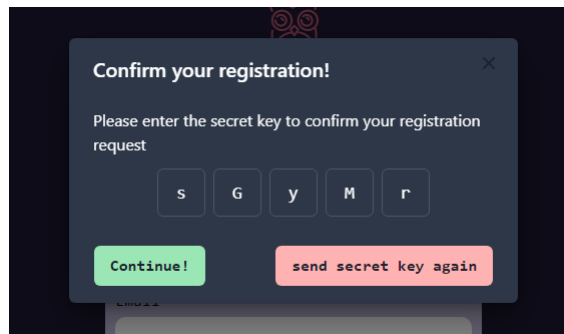


Figura 4.3: interfața pentru introducerea codului de securitate

după ce utilizatorul introduce codul secret cererea lui de înregistrare este confirmată, ceea ce înseamnă că aceasta va fi acum vizibilă în bordul de gestionare a administratorilor (admin dashboard) și urmează să fie revizuită de către unul din administratorii platformei, pentru acest pas utilizatorul nu are nimic de făcut, odată ce cererea îi va fi aprobată acesta se va putea autentifica pe platformă, urmând să primească de asemenea un email de informare în legătură cu acest eveniment.

Onboarding complete :)

From: ThesisHub staff <smtp@mailtrap.io>
To: <vlad.vidican00@e-uvr.ro>

2023-06-

[Show Headers](#)

HTML HTML Source Text Raw Spam Analysis HTML Check Tech Info

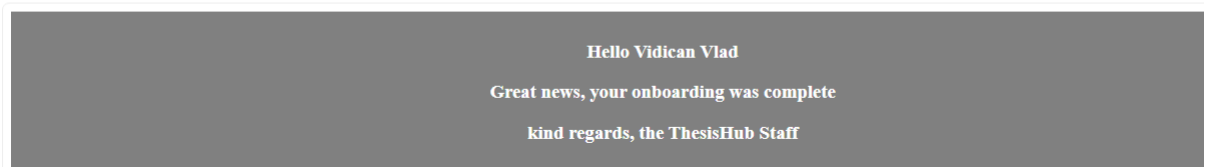


Figura 4.4: exemplu email de informare



Figura 4.5: Interfata autentificare

După o autentificare reușită, utilizatorul este redirectat către pagina principală, de aici utilizatorul poate să navigheze printre propunerile active de pe platformă. Din motive de performanță am decis să implementez paginare pentru căutarea de propuneri, serverul este configurat să returneze 4 elemente per pagină, în cazul în care nu toate propunerile sunt vizibile lista de propuneri permite scrolling, în partea de jos a pagini se află interfața pentru navigarea printre paginile de propuneri.

Utilizatorul are opțiunea de a merge la pagina precedentă, pagina următoare sau la oricare dintre cele vizibile, apăsând pe unul dintre propuneri utilizator va fi redirecționat către pagina acelei propuneri

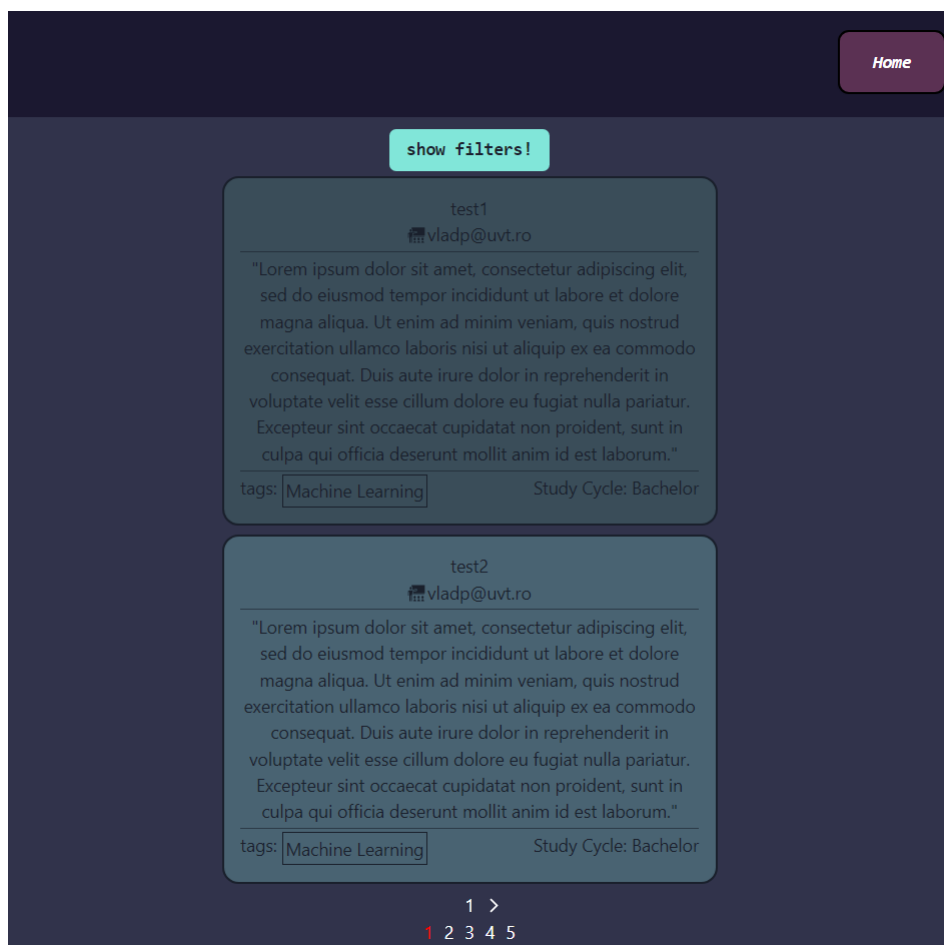


Figura 4.6: sistemul de navigare între propuneri din pagina principala

Platforma suportă și căutarea filtrată după propuneri, apăsând pe show filters utilizatorul poate să filtreze propunerile în funcție de următorii factori:

- ciclu de studii pentru care este destinată propunerea.
- ce tip de cont a creat propunerea(student sau profesor).
- anumite cuvinte cheie din titlul propunerii.
- anumite cuvinte cheie din descrierea propunerii.
- etichetele în care se încadrează propunerile. aici utilizatorul poate să aleagă și care să fie relația între etichetele alese (să conțină cel puțin una sau pe toate) dacă sunt mai multe.

după ce utilizatorul alege toate filtrele de care este interesat poate apăsa pe "Apply filters" pentru a reîmprospăta pagina doar cu propunerile ce respectă noile filtre, sau poate să apese la orice moment pe "reset filters" pentru a debifa toate filtrele adăugate

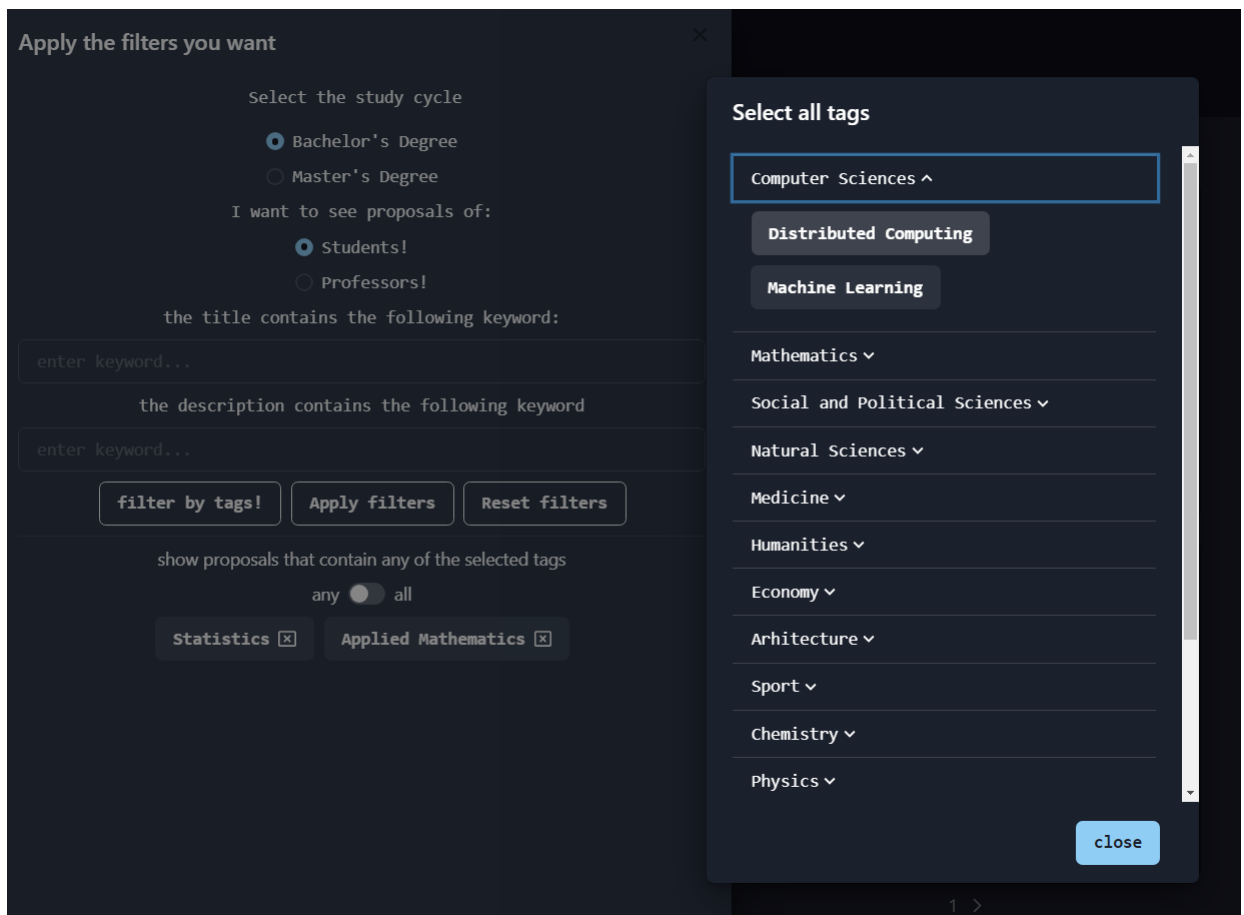


Figura 4.7: sistemul de cautare avanssata

Apăsând pe una dintre propuneri, utilizatorul va fi dus către pagina ei, aici are acces la mai multe informații despre această propunere, cum ar fi:

- Lista de Atasamente.
- Lista de comentarii.
- Aplicațiile curente pe această temă.

și de aici acesta poate să:

- Aduge un comentariu nou.
- să aplice pentru această propunere.
- să accepte una dintre aplicații dacă este deținătorul propunerii.

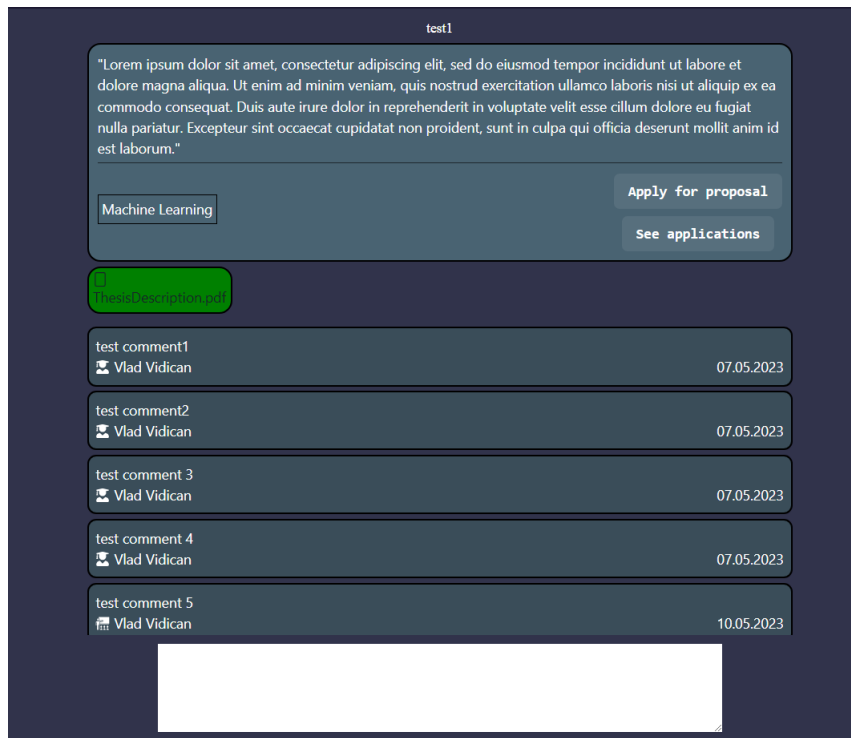


Figura 4.8: pagina de cautare google

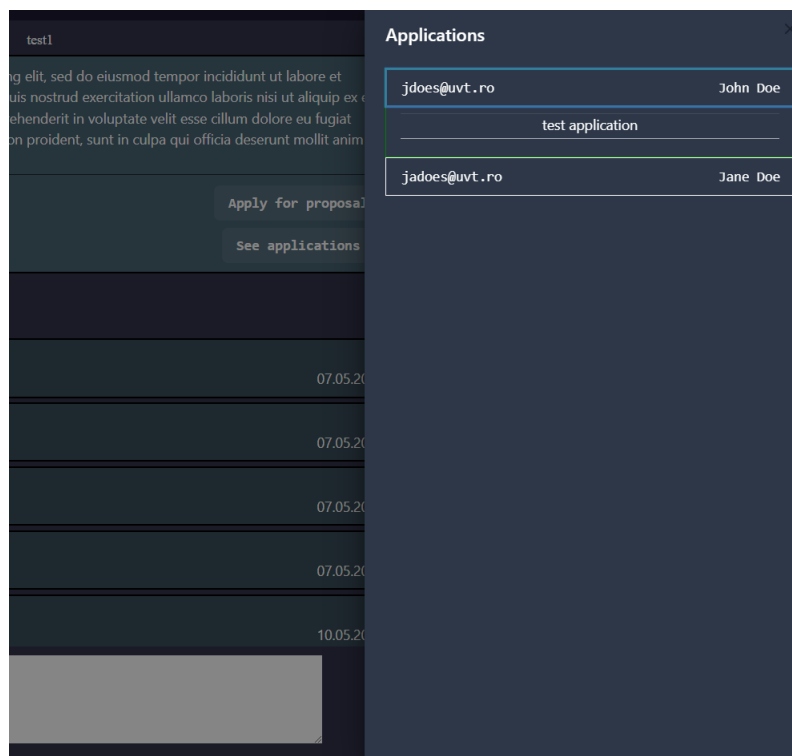


Figura 4.9: exemplu lista de aplicații a unei propuneri

Utilizatorul poate să își creeze propria lui propunere apăsând butonul ”Create proposal” din bara de navigare mai apoi acesta trebuie să completeze următoarele câmpuri:

- Titlul propunerii.

- Descrierea propunerii
- Atașamente (opțional)
- să aleagă cel puțin o eticheta din cele disponibile

pentru a atașa etichete propunerii utilizatorul trebuie să aleagă mai întâi o categorie din bara din dreapta a pagini și din categoria aleasă poate să asocieze etichetele disponibile, utilizatorul nu este însă limitat la o singură categorie

Figura 4.10: pagina pentru crearea propunerilor noi

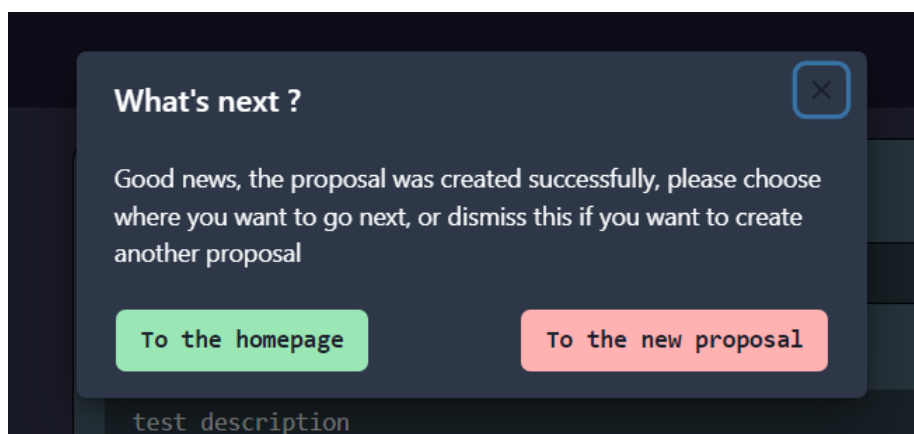


Figura 4.11: Propunere creata cu success

Pentru a putea accesa pagina de mesagerie utilizatorul trebuie să apese butonul "Conversations" din bara de navigare.

Odată ajuns pe pagina de mesagerie utilizatorul poate să aleagă una dintre conversațiile actuale Apăsând pe aceasta din bara laterală, sau poate să apese ”Add Users” pentru a iniția conversații noi. Dacă utilizatorul dorește să inițieze conversații noi acesta trebuie doar să introducă adresa de email a acelei persoane în câmpul pentru căutarea utilizatorilor, însă acest lucru nu este necesar ca utilizatorii să cunoască adresa de email în totalitate a persoanelor cu care doresc să inițieze o conversație, sistemul oferind sugestii în funcție de literele introduse.

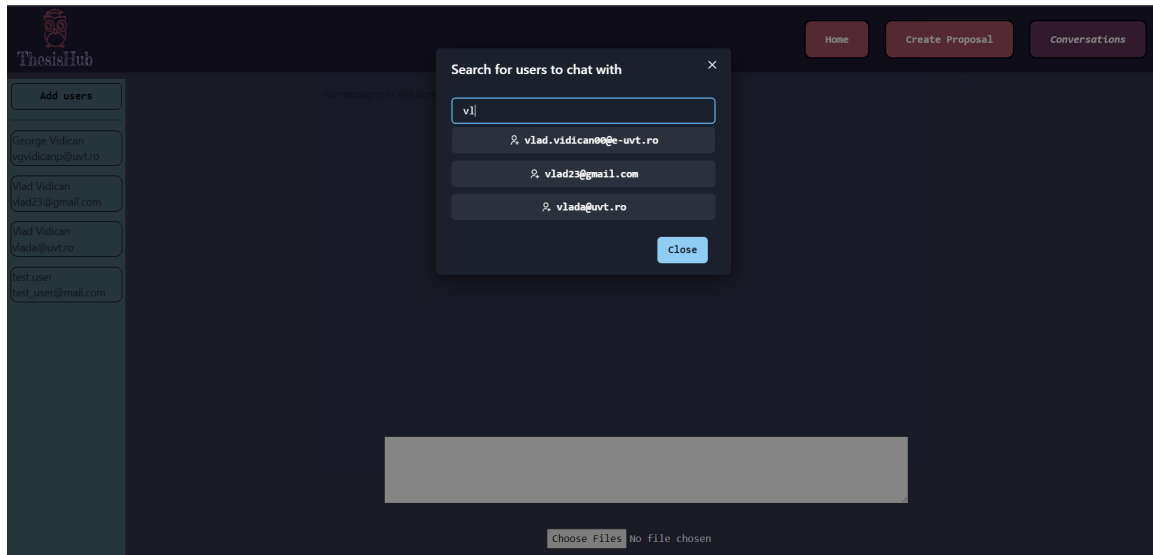


Figura 4.12: Interfata de inițiere a unei conversație

Odată cu alegerea unei conversații utilizatorii pot trimite mesaje, acest lucru se poate face introducând textul dorit în zona de mesaje și apăsând tasta ENTER, sistemul suportă și mesaje ce se întind pe mai multe linii, utilizatorul poate să înceapă o linie nouă apăsând tastele SHIFT + ENTER . Sistemul suportă de asemenea și trimiterea de fișiere, acestea sunt legate de mesaje, utilizatorul trebuie astfel doar să aleagă fișierele dorite și acestea vor fi trimise automat cu următorul mesaj.

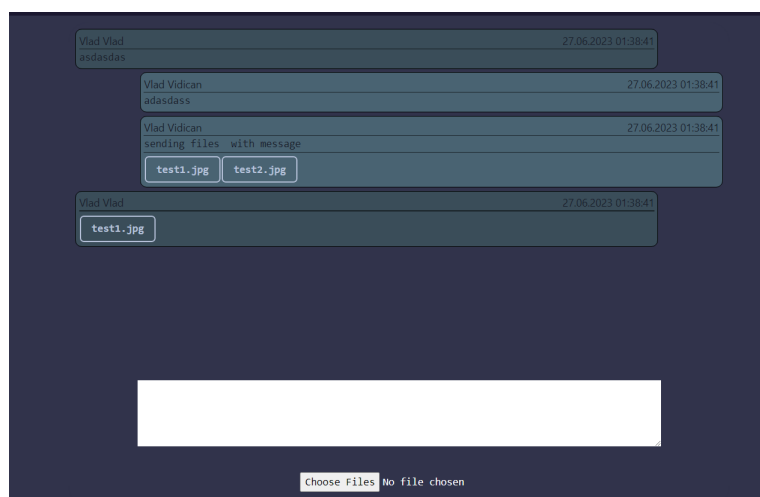


Figura 4.13: Interfata Mesagerie

Administratorii au de asemenea posibilitatea de a accesa bordul de gestiune din bara de navigare(dashboard), aici pot vedea toate cererile de înregistrare active de pe platforma, pe

lângă datele contului administratorii pot și să revizuiască dovezile de identitate atașate cererilor de înregistrare, în cazul în care administratorul decide să respingă o cerere atunci acesta este nevoit să îi ofere utilizatorului și o justificare pentru decizia luată, un email de informare fiind trimis utilizatorului ulterior.

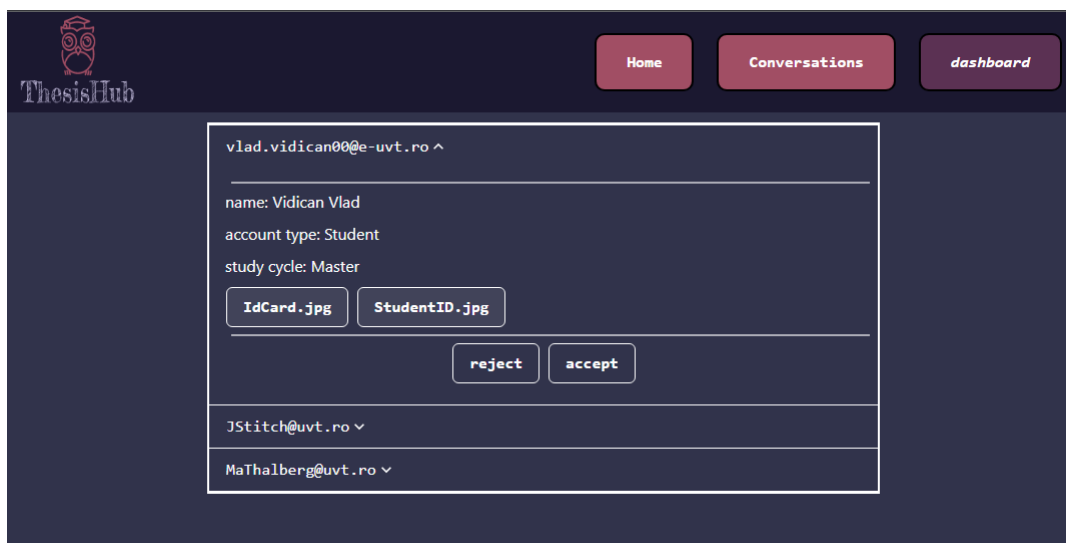


Figura 4.14: Bordul de gestiune a administratorilor

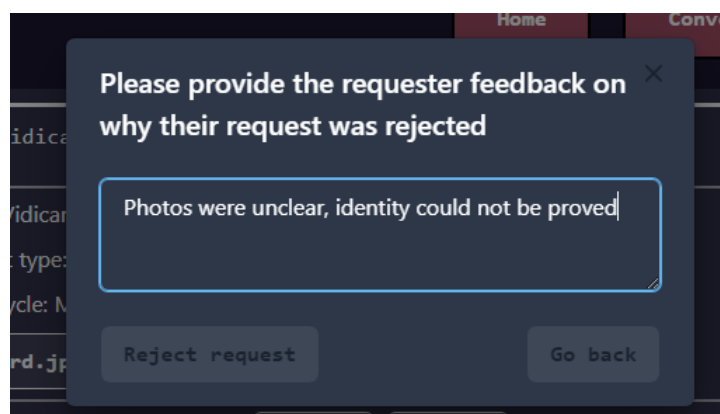


Figura 4.15: Exemplu respingere cerere de înregistrare

Capitolul 5

Concluzii și direcții viitoare

5.1 direcții viitoare

Deși aplicația în starea ei actuală oferă funcționalitățile propuse aceasta are încă potențial de dezvoltare, spre exemplu trecerea spre un sistem care să suporte mai multe universități. Deși în momentul de față accesul nu este restricționat explicit unei singure instituții sistemul nu suportă în starea sa actuală separarea logică a entităților în funcție de universitatea de apartenență. Această funcționalitate este însă necesară pentru a putea fi extinsă, fără aceasta administratorii unei universități ar avea drepturi elevate în ceea ce privește gestionarea utilizatorilor asupra tuturor celorlalte universități integrate în sistem.

Pentru a obține acest rezultat un prim plan ar fi adăugarea unei colecții de universități și modificarea schemei colecțiilor: user, userTemp și proposal pentru a avea și o referință către o universitate, ar mai fi de asemenea nevoie de trecerea la un sistem cu două tipuri de administratori:

- Administratori locali: aceștia vor avea dreptul de a gestiona utilizatorii din aceeași universitate ca și ei înșiși
- Administratori ai aplicației: aceștia nu aparțin unei universități anume și vor fi responsabili de adăugarea de universități, categorii și etichete noi în sistem, aceștia nu se vor ocupa de gestionarea utilizatorilor în general excepții fiind adăugarea administratorului inițial al unei universități și adăugarea altor administratori de acest tip

O altă oportunitate de dezvoltare ar fi implementarea unei aplicații mobile, dat fiind faptul ca serverul de backend este separat de logica generării de pagini html și livrării de conținut static acesta ar putea fi integrat cu un client de tip mobil fără a necesita nici o schimbare.

Posibilitatea de a folosi react native pentru dezvoltarea aplicației mobile este de asemenea un factor important deoarece ar rezulta într-o portare mai ușoară. Deși codul nu poate fi mapat unu la unu între react și react native, aceste tehnologii folosind în spate elemente primitive diferite, acestea au însă un grad de similitudine ridicat

5.2 Concluzii

Această lucrare de licență a avut ca scop dezvoltarea unei posibile soluții pentru una dintre problemele întâmpinate de către studenți, mai exact lipsa unui proces bine stabilit și convenient în ceea ce privește găsirea unui coordonator. Deși universitățile au de regula un proces

bine definit cu precădere la evoluția și asistarea studentului în privința realizării lucrării, acest proces lipsește în ceea ce privește potrivirea studenților cu profesorii coordonatori de regulă fiind asumată mai degrabă o abordare "ad-hoc". Acest Pas ar putea fi însă considerat unul dintre cei mai importanți, deoarece atât compatibilitatea student-profesor cât și interesul real al studentului față de tema lucrării sunt esențiale pentru o lucrare reușită.

Aplicația în starea ei actuală ofera următoarele facilități:

- Posibilitatea de a posta sau aplica la teme de licență/master.
- Căutare avansată care permite un nivel de specificitate ridicat cu scopul de a ușura găsirea temelor de interes.
- Un mediu prielnic pentru comunicarea dintre student și profesor.
- Un proces robust de validare a identității care să permită restricționarea accesului la platforma doar persoanelor vizate.

Datorită celor enumerate mai sus, consider că lucrarea a rezultat cel puțin într-un prototip bun cu potențial de dezvoltare spre o soluție la problema prezentată anterior

Bibliografie

- [1] Banks, A., Parcello, E. [2017]. Learning React. Functional web development with react and redux. USA: O'Reilly Media, Inc.
- [2] Hahn, Evan M. [2016]. Express in Action. Writing, building and testing Node.js applications. New York: Manning Publications Co.
- [3] Young, A. et al. [2017] Node.js in action second edition. New York: Manning Publications.
- [4] Banker, K. et al. [2016] MongoDB in action second edition. New York: Manning Publications. New Tork: Manning Publications.
- [5] Flanagan D. [2011] Javascript: the definitive guide sixth edition. USA: O'Reilly Media, Inc.