

images/FMI-03.png

**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

**TIMIȘOARA23
2023**

Cuprins

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 si facilitățile aplicației

3.1 Cazuri de utilizare pentru utilizatorii standard

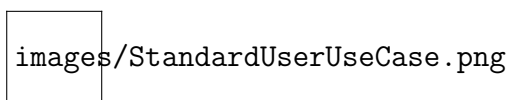


Figura 3.1: Cazuri de utilizare pentru utilizatorii standard

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

3.2 Cazuri de utilizare pentru administratori

images/AdminUserUseCase.png

Figura 3.2: Cazuri de utilizare pentru Administratori

Administratorii sunt utilizatorii cu privilegii ridicate, care au posibilitatea și responsabilitatea să gestioneze accesul celorlalți utilizatorii 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 fie 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.

3.3 Design-ul bazei de date

3.3.1 Privire de ansamblu

images/DB_Diagram.png

Figura 3.3: Diagrama bazei de date

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

3.3.2 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(Licența sau Master)

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

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 lasat comentariul.

¹ în 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 malicios care a obținut acces la baza de date

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 sa trimita 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 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 datorită tehnologiilor precum React Native și NodeJS;

React

React 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, acestea 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 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, deleghează 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 caruia acesta este recunoscut si 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 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 sta 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 aceeași ruta, important este însă faptul că express ne permite să adăugăm și o listă de funcții intermediare care vor fi apelate secvențial, oricare dintre acestea putând opri prematur executia(de regula în cazuri de eroare) sau de a apela următoarea funcție din lanț, optional acestea pot și adăuga/modifica date din cererea primită înainte de a apela următoarea 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 este un sistem de gestionare a bazelor de date de tip NoSQL bazat pe documente (echivalentul rândurilor într-o bază de date de tip SQL) și colecții (echivalentul tabelor 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 față 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 structură 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 librerie de mapare a datelor in obiecte (Object-Data Modeling), ODM-urile ofera o interfata de tip OOP in limbajul de programare ales pentru a interactiona cu baza de date NoSQL aleasa, conditiia fiind evident ca ODM-ul ales sa suporte atat limbajul de programare ales cat si baza de date. Mongoose ofera o interfata in javascript pentru a interactiona cu baze de date de tip MongoDB, pe langa aceasta convenienta de a putea comunica cu baza de date scriind cod direct in javascript si suita de functii care abstractizeaza detalii de implementare ale unor nevoi comune(de exemplu findByIdAndDelete), acesta mai are un beneficiu foarte important deoarece ajuta in diminuarea principalului dezavantaj al MongoDB-ului, mai exact lipsa de consistenta. Mongoose interactioneaza doar cu colectiile din baza de date pentru care are un model definit, la baza acestui model sta o schema definita in JSON pe care obiectele din aceea colectie ar trebui sa o respecte. Orice incercare de a crea / modifica obiecte in asa fel incat nu vor respecta schema definita va genera o eroare, acest lucru nu rezolva in totalitate problema din pacate insa deoarece valideaza doar operatiile de scriere pe care acesta le primeste, baza de date putand fii inca modificata direct si astfel aceastqa ar putea ajunge intr-o stare inconsistenta. Practic mongoose garanteaza consistenta bazei de date doar daca scrierea in aceasta se fac in 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 intre clienti si serverele web, acesta urmeaza modelul "cerere-raspuns", in care clientul si server-ul stabliesc o conexiune, clientul apoi trimite o cerere serverului iar serverul trimite inapoi un raspuns, dupa care conexiunea este inchisa, protocolul HTTP sta la baza web-ului acesta fiind protocolul folosit pentru marea majoritate a traficului web, acesta are o limitare insa, cum conectiunea poate fii initiata doar de catre client, iar aceasta este inchisa automat odata cu primirea raspunsului din partea serverului cererile de tip HTTP nu sunt potrivite pentru cazurile de utilizare unde se doreste ca serverul sa trimita mesaje catre client, cum ar fii in cazul de fata functionalitatea pentru mesagerie, in care se doreste ca serverul sa trimita clientului mesajele primite in timp real. Exista cateva "solutii" pentru a suporta aceasta functionalitae, cum ar fii HTTP polling unde clientul trimite cereri catre server pentru a obtine mesajele noi la un interval fix de timp, problema cu aceasta abordare fiind echilibrul dintre responsivitate si performanta, in aceasta abordare daca timpul de asteptare dintre cererii este prea lung atunci experienta utilizatorului se degradeaza considerabil, insa daca cereriile sunt prea frecvente atunci performanta sistemului va avea de suferit, crescand drastic numarul de cererii trimise in timp ce majoritatea acestora vor fii redundante.

O alta solutie ar fii HTTP long polling, unde se trimite o singura cerere, iar serverul blocheaza acel fir de executie(fiind creat un fir de executie diferit pentru fiecare cerere) pana cand acesta are date de trimis clientului, dupa care raspunsul este trimis si conexiunea este inchisa, iar apoi clientul initiaza o noua astfel de cerere, aceasta abordare este mai eficienta decat prima insa este tot suboptima, deoarece o buna parte din resursele serverului vor fii 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 ofera posibilitatea unei comunicari bidirectionale si cu latenta redusa intre client si server, facand astfel aplicatiile web care necesita comunicare in timp real fezabile, in acest protocol conexiunea dintre client si server este proiectata sa persiste iar atat clientul cat si serverul pot sa trimita evenimente celuiilalt oricand iar aceste mesaje sunt mult mai rapide deoarece acestea nu trebuie sa includa ”bagajul” necesar pentru stabilirea unei conexiuni noi spre deosebire de http

4.2 Manual de Implementare

4.2.1 Backend

Arhitectura sistemului

fișierele care contin codul sursa sunt impartite logic in urmatoarele categorii:

- Models: acest folder contine modelele mongoose aferente colectiilor din baza de date
- Controller: in acest folder se regasesc functiile definite pentru a trata cererile primite
- Routes: aici sunt definite rutele pe care le accepta serverul si sunt legate de functiile care ar trebui sa ruleze la apelarea rutelor respective

dupa cum se poate observa in figura 4.1 exista o mapare de aproape 1:1 intre modele si controllere, arhitectura implementata a fost in mare parte inspirata de catre MVC(Model View Controller), functiile care creaza/returneaza/modifica sau sterg obiecte ce apartin de un anumit model sunt grupate sub un singur fisier numit controller legat logic de modelul aferent, aceasta implementare nu respecta in totalitate arhitectura MVC totusi, deoarece backend-ul in acest caz este un rest api care comunica in mod exclusiv prin JSON fisierele statice(HTML, CSS si javascript) nefiind sub responsabilitatea acestuia, partea de View lipsind astfel complet din aceasta arhitectura. livrarea fisierelor statice ce tin de frontend sunt delegate in momentul de fata serverului http local care vine la pachet cu mediul de dezvoltare react, acesta este conceput insa doar ca o solutie pentru testarea si dezvoltarea locala, pentru a livra in productie aceste fisiere ce tin de frontend mai intai codul react trebuie compilat intr-un build optimizat (cu ajutorul aceluiasi mediu de dezvoltare) iar apoi livrat de catre un server http dedicat. O solutie posibila ar fii configurarea serverului de express pentru a livra si aceste fisiere statice, insa aceasta solutie nu este favorabila deoarece reduce scalabilitatea intregului sistem si ar reprezenta deasemenea un ”single point of failure”, o solutie mai buna ar fii folosirea unui server http dedicat pentru servirea de continut static precum NGINX si configurarea acestuia ca si un reverse proxy pentru serverul de express, permitand astfel adaugarea mai multor instante ale serverului de back end la nevoie, serverul reverse proxy avand atat scopul de a livra continutul static dar functionand si ca un load balancer

- Config: aici se regasesc functii pentru conectarea la baza de date si conectarea serverului de express cu serverul de SMTP pentru functionalitatiile ce implica trimiterea de email-uri
- Middleware: acest folder contine functiile intermediare rulate inainte de cele din controller, in mare parte functionalitate pentru validarea request-ului dar sunt si unele functii menite pentru a adauga informatii in request-ul primit de la client

- Utils: aici se regaseste de regula cod necesar/util care nu potriveste insa nici unei alte locatii din punct de vedere semantic. In cazul de fata o clasa care extinde clasa de baza "Error", care primeste ca si atribut pe langa un mesaj de eroare si un code de status util in generarea si tratarea erorilor intampinate de un server web

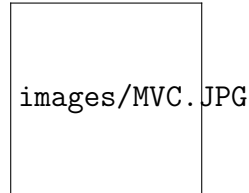


Figura 4.1: Structura codului backend

fluxul de control

Pentru a putea intelege mai bine modul de functionarea a serverului, consider ca ar fii utila urmarirea cap-coada a ciclului de viata a unui request, in cazul de fata o cerere care consta in aprobarea unei aplicatii ar fii ceea mai potrivivita.

Odata cu pornirea serverului, acesta creaza o instanta noua de express, si ii specifica acesteia ca pentru oricare cerere a carei destinatii incepe cu "api/proposal" sa foloseasca rutele definite in proposalRoutes.

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

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

In acest set de rute o avem pe urmatoarea definita: "":"/proposalID/approve" unde "proposalID" este un parametru, acesta nu are o valoare predefinita, ci este mai degraba un substituent, practic orice request de tipul: /api/proposal/id/approve unde id poate sa ia orice valoare va intra pe aceasta ruta, valoarea acestuia fiind facuta disponibila functiilor care trateaza cererea, acestea putand 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;
```

odata intrat un astfel de request, prima functie care se apeleaza este cea de autorizare, pentru a vorbi despre aceasta insa, consider ca o scurta explicatie despre Json Web tokens (JWT) este necesara deoarece aceasta tehnologie sta la baza sistemului de autentificare si autorizare.

In urma unei autentificari reusite serverul genereaza un hash pe baza incarcaturii token-ului (payload-ul, in cazul de fata id, tip cont si ciclu de studii) timpul si o cheie secreta data serverului ca si variabila de mediu (environment variable), token-ul nu poate fi descifrat fara a avea access la cheia secreta folosita pentru a genera hash-ul iar token-ul nu poate fi alterat de catre un eventual actor malitios deoarece oricare alterare va rezulta intr-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 modificare pot fi aduse unui token deja existent pentru a ii modifica payload-ul), odata acest token primit clientul il salveaza in memoria locala si il ataseaza la fiecare request, serverul cunoscand secretul cu care a fost "semnat" token-ul il poate descifra pentru a avea access la acest payload si astfel identifica cine este utilizatorul care a initiat request-ul in cauza. Revenind la functia noastra, aceasta verifica daca request-ul primit contine un header de autorizare de tip "Bearer token", si daca da incearca sa il descifreze folosind codul secret, in cazul in care header-ul este prezent si poate fi descifrat (este valid) atunci adauga in request payload-ul token-ului si apeleaza urmatoarea functie din lant, in caz contrar trimite clientului un raspuns de tip eroare si opreste 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"})
  }
}
```

urmatorul pas este sa verificam daca id-ul primit ca si parametru este unul valid, mai intai verificam daca s-a primit o valoare pentru acest ID sau acesta este null iar in caz afirmativ, obtinem din baza de date obiectul aferent acelui ID si il atasam cererii in acelasi mod ca si in pasul anterior iar intr-un final apelam urmatoarea functie din lant. In cazul in care nu

a fost oferit un ID sau nu exista acest ID in baza de date, returnam 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);
  }
}
```

urmatorul pas este sa verificam daca utilizatorul care a initiat cererea este si detinatorul propunerii de teme, in caz afirmativ, se adauga un nou atribut in request, numit auth si este setat ca adevarat. La final se apeleaza urmatoarea functie din lant indiferent de rezultatul conditiei

```
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 verifica daca utilizatorul este autorizat sau nu pentru a initia cererea in cauza, acest lucru se face verificand daca vre-o functie anterioara lui "isAuthorized" a setat deja atributul "req.auth" ca adevarat, in caz afirmativ acesta apeleaza doar functia urmatoare din lant, iar in caz contrar returneaza o eroare de tipul "permisiuni insuficiente" aceasta abordare simplifica semnificativ implementarea unui sistem de permisiuni complex, dat fiind modularitatea sa acesta presupune doar definirea unor middleware-uri care verifica unele conditii simple, si apoi la definirea rutei acestea se pot inlantui in functie de nevoie, un exemplu pentru asta ar putea fii: o ruta pentru stergerea unui comentariu si o ruta pentru editarea unui comen-

tariu, ar avea sens ca atat persoana care a postat comentariul, cat si persoana care detine postarea in care a fost postat comentariul, cat si un administrator sa poata sterge comentariul in cauza insa doar persoana care a postat comentariul ar trebui sa aibe permisiunea de a ii modifica continutul, in acest context ambele cazuri se pot rezolva implementand urmatoarele functii: `isCommentOwner`, `isProposalOwner`, `isAdmin` in acelasi mod ca si in pasul precedent iar apoi pentru ruta de stergere apelam urmatoarea lista de functii "..., `isCommentOwner`, `isProposalOwner`, `isAdmin`, `isAuthorized`, ..." iar pentru cea de editare apelam 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});
  }
}

```

dupa autorizare, urmatorul pas este validarea cererii, functia verifica daca propunerea respectiva are deja o aplicatie aprobata si daca id-ul utilizatorului aplicant lipseste din baza de date, daca cel putin una dintre aceste doua sunt adevarate atunci functia va returna o eroare si va oprii executia in caz contrar, se apeleaza ultima functie din acest lant

```

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 postari este modificat din null in id-ul utilizatorului a carui aplicatie a fost aprobata si apoi functia trimite un raspuns cu un mesaj de success 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 impartit in urmatoarele categorii:

- Pages: aici se regasesc componentele React aferente paginilor platformei, acestea sunt componentele la cel mai inalt nivel de abstractizare, fiind compuse la randul lor din alte componente React de nivel inalt.
- css: In acest folder se regasesc toate fisierele ce tin de stilizarea componentelor, codul sursa de css a fost impartit in fisiere in functie de pagina de care apartine
- api: Acest folder contine functiile folosite pentru trimiterea de cererii http catre backend, functiile respective sunt importate in anumite componente React si apelate in urma unor evenimente(de exemplu la incarcarea initiala a componentei sau in urma unei apasari de buton de catre utilizator). Comunicarea cu server-ul pentru functionalitatea de mesagerie nu se regaseste aici insa, pentru aceasta componenta comunicarea cu serverul este realizata folosind protocolul WebSockets ci nu http, evenimentele dupa care client-ul "asculta" pot fii definite doar dupa stabilirea unei conectiuni cu clientul, din acest motiv logica ce tine de mesagerie a fost incapsulata in pagina dedicata pentru mesagerie.
- Images: Aici se regasesc imaginile ce tin de aplicatia web in sine, acest folder este diferit de cel ce contine atasamentele aferente mesajelor/propunerilor de pe platforma. In cazul de fata acest folder contine logo-ul aplicatiei
- Utils: In acest folder se regasete functionalitatea de validare de pe front, aceste functii sunt apelate inainte de trimiterea unor cererii catre server. Este de preferat identificarea cererilor gresite preventiv pe cat de mult posibil, deoarece daca suntem intr-o

situatie in putem stii sigur ca cererea va rezulta intr-o eroare inca de pe front-end(un camp obligatoriu lipseste de exemplu), atunci clientul poate sa il anunte pe utilizator direct, evitand astfel trimiterea unei cererii si asteptarea unui raspuns pentru ea. reducand in acest fel atat timpul de asteptare al utilizatorului cat si incarcatura pe care o suporta server-ul

- Components: In acest fisier sunt grupate restul componentelor de tip React. Aici se regasesc atat componente simple (a caror definitii detin doar elemente primitive) cat si componente de nivel inalt (a caror definitie contin alte componente de tip React). Pentru a exemplifica diferenta dintre cele doua tipuri putem compara doua astfel de componente, cum ar fii Checkbox si taglist: dupa cum se poate observa mai jos componenta Checkbox este compusa dintr-un div(container) si un element de tip p (paragraf) ambele putand fii 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 insa are in definitia sa alte componente compuse de tip React dupa cum se poate vedea, din acest motiv poate fii considerata ca o componenta de nivel inalt

```
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>
    </div>
  )
}
```

```

        </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 componenta din ierarhie este App, aceasta incorporeaza defapt intreaga aplicatie, react-ul genereaza "aplicatii cu o singura pagina" (Single Page Applications), mai exact la accesarea pagini se trimite catre client codul necesar pentru a putea genera intreaga aplicatie, in mod clasic clientul face o cerere spre server si primeste codul html, css si javascript pentru fiecare navigare intre pagini, fiecare pagina fiind separata, React insa doar manipuleaza dom-ul browser-ului pentru a randa subcomponente diferite a acestui element de radacina in functie de URL pe care se afla utilizatorul si simuland astfel comportamentul unei aplicatii 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;

```

in cazul de fata pentru fiecare ruta definita este asociata o componenta de tip pagina. Una dintre ele este pagina propunerilor, aceasta este o ruta dinamica, id-ul propunerii fiind dat ca parametru. Aceasta pagina are in componenta sa:

- Bara de navigare: aceasta componenta contine butoane pentru redirectarea spre celelalte pagini si este regasita dealtfel in toate celelalte pagini
- ProposalItem: aceasta componenta contine datele propunerii, mai exact: titlu, descriere, detinator, ciclul de studii, etichete(tags) si atasamentele propunerii
- CommentSection: aceasta componenta contine atat lista de comentarii de pe platforma cat si componenta pentru adaugarea 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" primeste de la parinte id-ul cererii, si la initierea acesteia trimite o cerere spre server pentru a obtine lista de comentarii aferente propunerii, itereaza apoi prin lista primita si genereaza un CommentItem(comentariu) pentru fiecare element din lista.

```

{
  !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 parinte continutul comentariului ca si stare si pe baza acestuia se genereaza continutul comentariului, sintaxa JSX permite definirea de blocuri de cod javascript(intre acolade) aceste blocuri permit sa generam continut dinamic, componentele putand astfel avea variabile in definitiile lor. Aceasta sintaxa ne permite si sa randam in mod conditional un element, dupa cum se poate vedea mai jos, iconita aferenta autorului comentariului este definita conditional folosind operatorul tertiar in functie 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 inregistrare, pentru asta este necesar ca utilizatorul sa acceseze formularul de inregistrare si sa completeze urmatoarele informatii:

- Nume si prenume

- adresa de email
- parola
- tipul de cont (putand alege intre Student Profesor si Administrator)
- in cazul in care tipul de cont este student, atunci mai trebuie ales si ciclul de studii
- atasamente continand dovezi de identitate

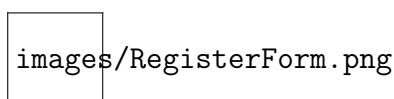


Figura 4.2: formular de inregistrare

Apoi utilizatorul va primi un cod de securitate pe adresa de email folosita la crearea contului, pe care va trebui sa il introduca in interfata deschisa dupa trimiterea cererii, aceasta interfata ii permite de asemenea utilizatorului sa solicite un cod nou de securitate in cazul in care mail-ul initial nu a fost livrat.

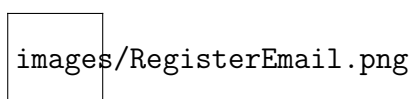


Figura 4.3: exemplu email primit la inregistrare

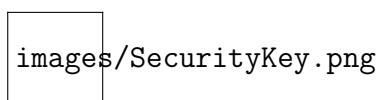


Figura 4.4: interfata pentru introducerea codului de securitate

dupa ce utilizatorul introduce codul secret cererea lui de inregistrare este confirmata, ceea ce inseamna ca aceasta va fii acum vizibila in bordul de gestionare a administratorilor (admin dashboard) si urmeaza sa fie revizuita de catre unul din administratorii platformei, pentru acest pas utilizatorul nu are nimic de facut, odata ce cererea ii va fii aprobata acesta se va putea autentifica pe platforma, acestia vor primi de asemenea un email de informare in legatura cu acest eveniment.

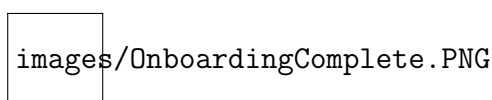


Figura 4.5: exemplu email de informare

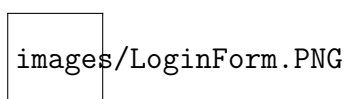


Figura 4.6: Interfata autentificare

Dupa o autentificare reusita utilizatorul este redirectat catre pagina principala, de aici utilizatorul poate sa navigheze printre propunerile active de pe platforma. Din motive de performanta am decis sa implementez paginare pentru cautarea de propuneri, serverul este configurat sa returneze 4 elemente per pagina, in cazul in care nu toate propunerile sunt vizibile lista de propuneri permite scrolling, in partea de jos a pagini se afla interfata pentru navigarea printre paginile de propuneri.

Utilizatorul are optiunea de a merge la pagina precedenta, pagina urmatoarea sau la oricare dintre cele vizibile, apasand pe unul dintre propuneri utilizator va fii redirectionat catre pagina acelei propuneri

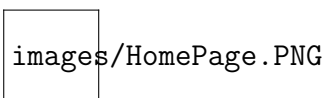


Figura 4.7: sistemul de navigare intre propuneri din pagina principala

Platforma suporta si cautarea filtrata dupa propuneri, apasand pe show filters utilizatorul poate sa filtreze propunerile in functie de urmatoorii factori:

- ciclu de studii pentru care este destinata propunerea.
- ce tip de cont a creat propunerea(student sau professor).
- anumite cuvinte cheie din titlul propunerii.
- anumite cuvinte cheie din descrierea propunerii.
- etichetele in care se incadreaza propunerile. aici utilizatorul poate sa aleaga si care sa fie relatia intre etichetele alese (sa contina cel putin una sau pe toate) daca sunt mai multe.

Dupa ce utilizatorul alege toate filtrele de care este interesat poate apasa pe "Apply filters" pentru a reimprospata pagina doar cu propunerile ce respecta noile filtre, sau poate sa apese la orice moment pe "reset filters" pentru a debifa toate filtrele adaugate

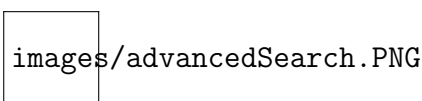


Figura 4.8: sistemul de cautare avansata

Apasand pe una dintre propuneri, utilizatorul va fii dus catre pagina ei, aici are acces la mai multe informatii despre aceasta propunere, cum ar fii:

- Lista de Atasamente.
- Lista de comentarii.
- Aplicatiile curente pe aceasta tema.

si de aici acesta poate sa:

- Aduage un comentariu nou.

- sa aplice pentru aceasta propunere.
- sa accepte una dintre aplicatii daca este detinatorul propunerii.

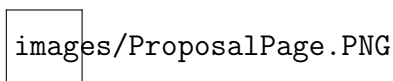


Figura 4.9: pagina de cautare google

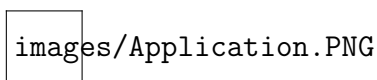


Figura 4.10: exemplu lista de aplicatii a unei propunerii

Utilizatorul poate sa isi creeze propria lui propunere apasand butonul "Create proposal" din bara de navigare mai apoi acesta trebuie sa completeze urmatoarele campuri:

- Titlul propunerii.
- Descrierea propunerii
- Atasamente (optional)
- sa aleaga cel putin o eticheta din cele disponibile

pentru a atasa etichete propunerii utilizatorul trebuie sa aleaga mai intai o categorie din bara din dreapta a paginii si din categoria aleasa poate sa asocieze etichetele disponibile, utilizatorul nu este insa limitat la o singura categorie

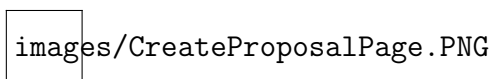


Figura 4.11: pagina pentru crearea propunerilor noi

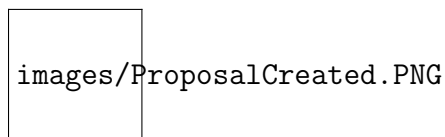


Figura 4.12: Propunere creata cu success

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

Odata ajuns pe pagina de mesagerie utilizatorul poate sa aleaga una dintre conversatiile actuale apasand pe aceasta din bara laterala, sau poate sa apese "Add Users" pentru a initia conversatii noi. Daca utilizatorul doreste sa initieze conversatii noi acesta trebuie doar sa introduca adresa de email a acelei persoane in campul pentru cautarea a utilizatorilor, important de mentionat este ca nu este necesar ca utilizatorii sa cunoasca adresa de email in

totalitate a persoanelor cu care doresc sa initieze o conversatie, sistemul oferind sugestii in functie de literele introduse.

 images/AddUser.PNG

Figura 4.13: Interfata de initiere a unei conversatie

Odata cu alegerea unei conversatii utilizatorii pot trimite mesaje, acest lucru se poate face introducand textul dorit in zona de mesaje si apasand tasta ENTER, sistemul suporta si mesaje ce se intind pe mai multe linii, utilizatorul poate sa inceapa o linie noua apasand tastele SHIFT + ENTER . Sistemul suporta de asemenea si trimiterea de fisiere, acestea sunt legate de mesaje, utilizatorul trebuie astfel doar sa aleaga fisierele dorite si acestea vor fii trimise automat cu urmatorul mesaj.

 images/SendMessages.PNG

Figura 4.14: Interfata Mesagerie

Administratorii au de asemenea posibilitatea de a accesa bordul de gestiune din bara de navigare(dashboard), aici pot vedea toate cererile de inregistrare active de pe platforma, pe langa datele contului administratorii pot si sa revizuieste dovezile de identitate atasate cererilor de inregistrare, in cazul in care administratorul decide sa respinga o cerere atunci acesta este nevoit sa ii ofere utilizatorului si o justificare pentru decizia luata, un email de informare fiind trimis utilizatorului ulterior.


 images/Dashboard.PNG

Figura 4.15: Bordul de gestiune a administratorilor

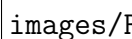
 images/RejectRequest.PNG

Figura 4.16: Exemplu respingerere cerere de inregistrare

Capitolul 5

Concluzii si directii viitoare

5.1 directii viitoare

Desi aplicatia in starea ei actuala ofera functionalitatile propuse aceasta are inca potential de dezvoltare, spre exemplu trecerea spre un sistem care sa suporte mai multe universitati. Desi in momentul de fata accesul nu este restrictionat efectiv unei singure institutii sistemul nu suporta in starea sa actuala separarea logica a entitatilor in functie de universitatea de apartenenta. Aceasta functionalitate este insa necesara pentru a putea fii extinsa, fara aceasta administratorii unei universitati ar avea drepturi elevate in ceea ce priveste gestionarea utilizatorilor asupra tuturor celorlalte universitati integrate in sistem.

Pentru a obtine acest rezultat un prim plan ar fii adaugarea unei colectii de universitati si modificarea schemei colectiilor: user, userTemp si proposal pentru a avea si o referinta catre o universitate, ar mai fii de asemenea nevoie de trecerea la un sistem cu doua tipuri de administratori:

- Administratori locali: acestia vor avea dreptul de a gestiona utilizatorii din aceeasi universitate ca si ei insasi
- Administratori ai aplicatiei: acestia nu apartin unei universitati anume si vor fii responsabili de adaugarea de universitati, categorii si etichete noi in sistem, acestia nu se vor ocupa de gestionarea utilizatorilor in general exceptii fiind adaugarea administratorului initial al unei universitati si adaugarea altor administratori de acest tip

O alta oportunitate de dezvoltare ar fii implementarea unei aplicatii mobile, dat fiind faptul ca serverul de backend este separat de logica generarii de pagini html si livrarii de continut static acesta ar putea fii integrat cu un client de tip mobile fara a necesita nici o schimbare.

Posibilitatea de a folosi react native pentru dezvoltarea aplicatiei mobile este de asemenea un factor important deoarece ar rezulta intr-o portare mai usoara, desi codul nu poate fii mapat unu la unu intre react si react native, aceste tehnologii folosind in spate elemente primitive diferite, acestea au insa un grad de similitudine ridicat