



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



PROIECT DE DIPLOMĂ
Janina Constantina Cocci

COORDONATOR ȘTIINȚIFIC
Ș. L. Dr. Ing. Cătălin Cerbulescu

Iulie 2025
CRAIOVA



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



CampusConnect. Aplicație web pentru chat
Janina Constantina Cocei

COORDONATOR ȘTIINȚIFIC
Ș. L. Dr. Ing. Cătălin Cerbulescu

Iulie 2025
CRAIOVA

„Sub mâna celui învățat, uneltele devin punți spre zări de cunoaștere.”

Autor necunoscut

DECLARAȚIE DE ORIGINALITATE

Subsemnatul JANINA CONSTANTINA COCEI, student la specializarea CALCULATOARE ÎN LIMBA ROMÂNĂ din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul CAMPUSCONNECT. APLICAȚIE WEB PENTRU CHAT,
- coordonată de Ș. L. DR. ING. CĂTĂLIN CERBULESCU
- prezentată în sesiunea IULIE 2025.

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele și referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

Semnătura candidatului,



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică
Departamentul de Calculatoare și Tehnologia Informației

Aprobat la data de
Șef de departament,
Prof. dr. ing.
Nicolae Enescu

PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului/-ei:	Cocei Janina Constantina
Enunțul temei:	CampusConnect. Aplicație web pentru chat
Datele de pornire:	<p>Pentru dezvoltarea aplicației CampusConnect, am avut ca punct de plecare:</p> <ul style="list-style-type: none">• analiza unor aplicații de comunicare deja existente (precum WhatsApp, Slack sau Teams), pentru a identifica funcționalitățile esențiale într-un mediu colaborativ;• interesul crescut pentru digitalizarea interacțiunii în spațiul academic;• cunoștințele tehnice dobândite în timpul facultății.
Conținutul proiectului:	Proiectul este structurat în următoarele capitole: introducere, tehnologii și framework-uri utilizate, instrumente software folosite, proiectarea și arhitectura aplicației, implementarea aplicației, instalarea aplicației și concluzii.
Material grafic obligatoriu:	Diagrame, capturi de ecran.
Consultații:	Periodice
Conducătorul științific (titlul, nume și prenume, semnătura):	Ș. L. Dr. Ing. Cerbulescu Cătălin
Data eliberării temei:	15.10.2024
Termenul estimat de predare a proiectului:	18.06.2025
Data predării proiectului de către student și semnătura acestuia:	



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică

Departamentul de Calculatoare și Tehnologia Informației

REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele și prenumele candidatului/-ei:

Specializarea:

Calculatoare

Titlul proiectului:

CampusConnect. Aplicație web pentru chat

Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):

În facultate ☐

În producție ☐

În cercetare ☐

Altă locație:

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
Tipul proiectului		Cercetare <input type="checkbox"/>	Proiectare <input type="checkbox"/>	Realizare practică <input type="checkbox"/>	Altul
Aparatul matematic utilizat		Simplu <input type="checkbox"/>	Mediu <input type="checkbox"/>	Complex <input type="checkbox"/>	Absent <input type="checkbox"/>
Utilitate		Contract de cercetare <input type="checkbox"/>	Cercetare internă <input type="checkbox"/>	Utilare <input type="checkbox"/>	Altul
Redactarea lucrării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
Partea grafică, desene		Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
Realizarea practică	Contribuția autorului	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Mare <input type="checkbox"/>	Foarte mare <input type="checkbox"/>
	Complexitatea temei	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input type="checkbox"/>
	Analiza cerințelor	Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
	Arhitectura	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input type="checkbox"/>
	Întocmirea specificațiilor funcționale	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
	Implementarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
	Testarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
	Funcționarea	Da	Parțială	Nu	

		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Rezultate experimentale	Experiment propriu <input type="checkbox"/>	Preluare din bibliografie <input type="checkbox"/>		
Bibliografie	Cărți	Reviste	Articole	Referințe web
Comentarii și observații				

În concluzie, se propune:

ADMITEREA PROIECTULUI <input type="checkbox"/>	RESPINGEREA PROIECTULUI <input type="checkbox"/>
---	---

Data,

Semnătura conducătorului științific,

REZUMATUL PROIECTULUI

Proiectul a avut ca scop dezvoltarea unei aplicații web menite să faciliteze comunicarea la nivel academic între studenți și între studenți și tutori.

Aplicația a fost dezvoltată folosind tehnologii precum React pentru frontend și Spring Boot pentru backend, asigurând astfel o arhitectură robustă și scalabilă. Integrarea WebSocket a facilitat comunicarea în timp real.

În timpul dezvoltării, s-au întâlnit multe provocări, în special în ceea ce privește gestionarea conexiunilor WebSocket și sincronizarea datelor, dar acestea au fost rezolvate printr-o organizare meticuloasă a codului, teste și documentare constantă. Utilizarea GitHub a simplificat, de asemenea, administrarea versiunilor.

Prin contribuția personală am implementat funcționalități cheie ca sistemul de mesagerie în timp real, administrarea rolurilor, și autentificarea utilizatorilor. Lucrarea ilustrează capacitatea de aplicare practică a cunoștințelor teoretice acumulate în timpul studiilor și poate constitui fundament pentru dezvoltări ulterioare.

Termenii cheie: aplicație web, comunicare academică, digitalizare, React, Spring Boot, WebSocket, mesagerie în timp real.

MULȚUMIRI

Doresc să îmi exprim întreaga mea recunoștință față de conducătorul științific, Ș. L. Dr. Ing. Cerbulescu Cătălin, pentru îndrumarea competentă, răbdarea și sprijinul oferit pe parcursul elaborării acestei lucrări de licență. Profesionalismul și sfaturile domniei sale au avut un rol esențial în structurarea și finalizarea acestei etape importante din parcursul meu academic.

Mulțumiri deosebite adresez și cadrelor didactice din cadrul Facultății de Automatică, Calculatoare și Electronică, pentru formarea și cunoștințele transmise de-a lungul anilor de studiu, contribuind la dezvoltarea mea intelectuală și profesională.

De asemenea, doresc să mulțumesc familiei mele, pentru sprijinul necondiționat, încurajare și încrederea pe care mi-au oferit-o constant. Fără susținerea lor morală și emoțională, finalizarea acestei lucrări nu ar fi fost posibilă.

Tuturor celor care, direct sau indirect, au contribuit la realizarea acestei lucrări, le adresez sincere mulțumiri.

CUPRINSUL

1	INTRODUCERE.....	1
1.1	SCOPUL.....	1
1.2	MOTIVAȚIA	1
2	TEHNOLOGII ȘI FRAMEWORK-URI UTILIZATE	2
2.1	REACTJS.....	2
2.2	MATERIAL UI (MUI).....	2
2.3	REDUX.....	3
2.4	REACT ROUTER DOM.....	4
2.5	AXIOS	4
2.6	FRAMER MOTION	5
2.7	SOCKJS & STOMP.JS.....	5
2.7.1	<i>SockJS</i>	5
2.7.2	<i>STOMP.js</i>	6
2.7.3	<i>Scenarii de integrare în aplicații web</i>	6
2.8	RECHARTS	6
2.9	YUP & REACT HOOK FORM	7
2.10	MICROLINK	7
2.11	SPRING BOOT	7
2.12	SPRING WEB.....	8
2.13	SPRING DATA JPA.....	8
2.14	SPRING SECURITY + JWT.....	9
2.15	SPRING WEBSOCKET.....	9
2.16	POSTGRESQL	10
2.17	LOMBOK.....	10
2.18	JAKARTA VALIDATION.....	11
3	INSTRUMENTE SOFTWARE FOLOSITE.....	12
3.1	WEBSTORM.....	12
3.2	REDUX DEVTOOLS.....	12
3.3	INTELLIJ IDEA.....	13
3.4	POSTMAN	13
3.5	PGADMIN	14
3.6	GITHUB	14
4	PROIECTAREA ȘI ARHITECTURA APLICAȚIEI.....	15

4.1	PREZENTAREA GENERALA	15
4.2	DESCRIEREA FUNCȚIONALITĂȚILOR SISTEMULUI.....	15
4.3	MODELAREA INTERACȚIUNII UTILIZATORILOR PRIN DIAGrame	16
4.3.1	<i>Introducere în modelarea UML</i>	16
4.3.2	<i>Caz de utilizare: Procesul de autentificare</i>	17
4.3.3	<i>Caz de utilizare: Rolul studentului</i>	18
4.3.4	<i>Caz de utilizare: Rolul tutorelui</i>	19
4.3.5	<i>Caz de utilizare: Funcții specifice administratorului</i>	20
4.4	STRUCTURA BAZEI DE DATE.....	21
4.4.1	<i>Modelul conceptual și relațional</i>	21
4.4.2	<i>Tehnologii utilizate pentru stocare</i>	21
4.4.3	<i>Maparea obiect-relațională (ORM)</i>	21
4.4.4	<i>Considerații legate de performanță și consistență</i>	22
4.4.5	<i>Utilizarea pattern-urilor de proiectare</i>	23
5	IMPLEMENTAREA APLICAȚIEI	25
5.1	SECURITATEA APLICAȚIEI	25
5.2	SISTEMUL DE AUTENTIFICARE	26
5.2.1	<i>Autentificarea unui utilizator</i>	27
5.2.2	<i>Înregistrarea unui nou utilizator</i>	29
5.3	SISTEMUL DE GESTIONARE A PROFILULUI UTILIZATORULUI.....	31
5.4	SISTEMUL DE GESTIONARE A CONEXIUNII WEBSOCKET.....	34
5.4.1	<i>Descriere generală (Frontend)</i>	35
5.4.2	<i>Componentele principale</i>	35
5.4.3	<i>Integrarea în aplicație</i>	37
5.4.4	<i>Descriere generală (Backend)</i>	37
5.4.5	<i>Componentele principale</i>	37
5.5	LISTA CONVERSAȚIILOR.....	38
5.6	SISTEMUL DE CREARE ȘI ADMINISTRARE A UNUI GRUP	41
5.7	SISTEMUL DE TRIMITERE A UNUI MESAJ	42
5.7.1	<i>Structura generală a interfeței de conversație</i>	43
5.7.2	<i>Componenta ChatFooter</i>	43
5.8	SISTEMUL DE AFIȘARE A UNUI MESAJ	45
5.9	SISTEMUL DE AFIȘARE INFORMAȚII (CONTACT VS. CONTACTGROUP).....	47
5.9.1	<i>Componenta Contact</i>	47
5.9.2	<i>Componenta ContactGroup</i>	48
5.9.3	<i>Integrarea componentelor în aplicație</i>	49

5.10	SISTEMUL DE MANAGEMENT.....	49
6	INSTALAREA APICAȚIEI	52
6.1	CLONAREA CODULUI SURSĂ	52
6.2	CONFIGURAREA BAZEI DE DATE POSTGRESQL	53
6.3	INSTALAREA ȘI RULAREA BACKEND-ULUI	53
6.4	INSTALAREA ȘI RULAREA FRONTEND-ULUI.....	53
7	CONCLUZII	54
8	BIBLIOGRAFIE.....	55

1 INTRODUCERE

1.1 Scopul

Scopul principal al acestei lucrări a fost dezvoltarea unei aplicații menite să sprijine îmbunătățirea comunicării în mediul academic, în special între studenți și între studenți și tutori. CampusConnect a fost concepută ca o platformă internă, dedicată comunității universitare pentru a promova schimbul de mesaje, fotografii, link-uri și documente într-un mod eficient, organizat și accesibil.

Dincolo de componenta practică, proiectul a reprezentat și o oportunitate valoroasă de a îmbunătăți și aplica cunoștințele dobândite pe parcursul studiilor la specializarea Calculatoare, în limba română, din cadrul Facultății de Automatică, Calculatoare și Electronică din Craiova. Procesul de dezvoltare a presupus implicare activă în toate etapele: de la analiză și proiectare, până la implementare și testare.

Redactarea acestei lucrări a avut ca scop documentarea detaliată a întregului parcurs tehnic, de la alegerile arhitecturale și tehnologice până la provocările întâmpinate pe parcurs. Această lucrare reflectă aplicarea concretă a cunoștințelor teoretice, subliniind dorința de a le extinde prin integrarea soluțiilor moderne și adaptându-le pe nevoile reale ale mediului academic printr-un proces de digitalizare continuă.

1.2 Motivația

Alegerea acestei teme a fost determinată de dorința de a contribui activ la procesul de digitalizare a comunicării în mediul academic. Într-un context în care interacțiunile dintre studenți și tutori devin mai complexe și dinamice, am identificat nevoia unei soluții moderne, care să faciliteze colaborarea într-un mod eficient și accesibil.

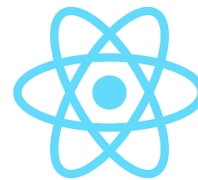
CampusConnect s-a conturat din convingerea că digitalizarea nu ar trebui să se limiteze doar la platformele educaționale formale, ci poate ar trebui să se extindă către aspectele cotidiene ale vieții universitare. Printr-un sistem dedicat de mesagerie și partajare de resurse, aplicația își propune să simplifice și să organizeze mai bine comunicarea între membrii comunității academice.

Totodată, această temă a reprezentat pentru mine o oportunitate valoroasă de dezvoltare personală și profesională. Realizarea unui proiect complet, cu aplicabilitate concretă, mi-a oferit șansa de a aprofunda cunoștințele dobândite în timpul facultății și de a le pune în practică într-un mod relevant, tehnic și util.

2 TEHNOLOGII ȘI FRAMEWORK-URI UTILIZATE

2.1 ReactJS

React este o bibliotecă JavaScript care este utilizată pentru construirea interfețelor de utilizator, atât web cât și native. (Inc. F. , 2023) Aplicațiile dezvoltate cu React sunt construite din componente independente, fiecare responsabilă de o anumită parte a interfeței, de la un simplu buton până la o pagină completă. (Inc. F. , 2023) Această abordare simplifică logica aplicației și facilitează testarea, mentenanța și extinderea codului.



Unul dintre principiile fundamentale ale React este Virtual DOM (Document Object Model). În loc să interacționeze direct cu DOM-ul real, React utilizează o copie virtuală, ceea ce reduce semnificativ numărul de manipulări directe ale DOM-ului. Acest lucru îmbunătățește performanța aplicațiilor, deoarece React efectuează un proces de reconciliere prin care identifică diferențele între starea anterioară și cea curentă a interfeței, actualizând doar elementele afectate.

Componentele în React sunt reprezentate fie prin componente funcționale (funcții JavaScript care returnează JSX), fie prin clase (în versiunile mai vechi). În prezent, componentele funcționale sunt preferate datorită suportului pentru React Hooks, introdus în versiunea 16.8. (Abramov D. , 2019) Acesta permite gestionarea stării și a efectelor colaterale direct în componentele funcționale, fără a fi necesare clase.

Ecosistemul React este susținut de numeroase biblioteci complementare precum Redux (pentru gestionarea stării globale), React Router (pentru rutare), MUI (Material UI – pentru interfețe vizuale) etc. (Banks A. &, 2020)

2.2 Material UI (MUI)

MUI, cunoscut anterior sub denumirea de Material-UI, este o bibliotecă de componente UI pentru React, construită pe baza principiilor de design promovate de Material Design, un sistem de design dezvoltat de Google. (Inc. G. , 2023) Scopul acestei biblioteci este de a facilita dezvoltarea interfețelor moderne, coerente și accesibile, accelerând procesul de dezvoltare frontend prin oferirea de componente predefinite, personalizabile și bine documentate. (Team M. , 2024)



MUI oferă o gamă largă de componente, de exemplu: butoane, formulare, tabele, bare de navigare, dialoguri, alerte, etc. toate implementate ca și componente React. Acestea respectă standardele de accesibilitate (ARIA) și sunt optimizate pentru performanță și responsivitate. (Larsen, 2021)

Arhitectura modulară a MUI permite importul selectiv al componentelor necesare, contribuind la optimizarea performanței aplicației prin reducerea dimensiunii bundle-ului final. De asemenea, biblioteca este compatibilă cu instrumente moderne de styling precum Emotion (biblioteca implicită de CSS in JS), dar oferă și opțiuni pentru integrarea cu alte soluții de stilizare, cum ar fi Styled Components sau Tailwind CSS. (Team M. , 2024)

MUI este utilizat pe scară largă în aplicații comerciale și academice datorită documentației extinse, comunității active și integrării excelente cu ecosistemul React. (Larsen, 2021)

2.3 Redux

Redux este o bibliotecă JavaScript open-source utilizată în mod obișnuit pentru gestionarea stării aplicațiilor frontend, în special în cadrul ecosistemului React, dar nu exclusiv. A fost dezvoltată inițial de Dan Abramov și Andrew Clark în 2015 și se bazează conceptual pe modelul fluxului unidirecțional de date inspirat de arhitectura Flux dezvoltată de Facebook. (Abramov D. ș., 2024)



Principiul de bază al Redux este acela de a menține o singură sursă de adevăr („single source of truth”) prin intermediul unui „store” global în care este salvată întreaga stare a aplicației. Interacțiunea cu această stare se realizează prin acțiuni („actions”) care sunt procesate de funcții pure denumite „reduceri” („reducers”). Acest model determină o predictibilitate crescută a comportamentului aplicației și facilitează testarea și depanarea codului. (Ng, 2017)

Un flux simplificat în Redux implică următoarele etape:

- Dispatcharea unei acțiuni: un obiect JavaScript care descrie o intenție de modificare a stării.
- Procesarea acțiunii de către reducer: o funcție care primește starea curentă și acțiunea și returnează o nouă stare.
- Actualizarea store-ului: noua stare este salvată în store și componentele abonate sunt notificate.

Redux încurajează o arhitectură predictibilă și scalabilă, de aceea este preferat în aplicații complexe unde starea aplicației este distribuită pe multiple componente.

Un avantaj important al utilizării Redux constă în facilitarea funcționalităților avansate precum:

- time-travel debugging (prin redux-devtools),
- serializarea și rehidratarea stării,
- logarea și urmărirea modificărilor de stare. (Alex Banks, 2016)

Deși a fost criticat uneori pentru boilerplate-ul necesar, apariția unor extensii precum redux-thunk, redux-saga sau redux-toolkit a simplificat considerabil scrierea și gestionarea logicii aplicațiilor moderne. (Ng, 2017)

2.4 React Router DOM

React Router DOM este o bibliotecă esențială pentru gestionarea rutării în aplicațiile dezvoltate cu React, oferind suport pentru navigarea între pagini fără reîncărcarea completă a aplicației. Aceasta implementează funcționalități de tip *Single Page Application (SPA)*, permițând comutarea dinamică între diferite componente sau *views*, pe baza adresei URL, fără pierderea stării aplicației. (Inc. R. S., 2024)



Această bibliotecă este construită pe baza arhitecturii React Router, dar adaptată pentru medii bazate pe DOM, cum ar fi aplicațiile web. Începând cu versiunea 6, React Router a introdus o serie de îmbunătățiri semnificative, inclusiv un API¹ declarativ mai intuitiv, suport îmbunătățit pentru *nested routes*² și utilizarea *data loaders*³ pentru preluarea datelor înainte de randarea componentelor. (Fulton, 2022)

Componentele de bază oferite de React Router DOM includ:

- **BrowserRouter:** componentă care folosește API-ul HTML5 pentru a păstra sincronizarea între interfață și URL.
- **Routes și Route:** definesc rutele aplicației și componentele asociate.
- **Link:** permite navigarea internă fără reîncărcare.
- **useNavigate, useParams, useLocation:** hook-uri pentru manipularea programatică a rutării și accesarea parametrilor din URL. (Banks R. &., 2023)

React Router DOM contribuie astfel la crearea unei experiențe de utilizare fluentă și interactivă, fiind un instrument indispensabil în dezvoltarea aplicațiilor moderne React care necesită navigare multiplă, autentificare pe bază de rute și gestionare de layout-uri dinamice.

2.5 Axios

Axios este o bibliotecă JavaScript open-source utilizată pentru efectuarea cererilor HTTP din aplicații web sau mobile. Este construită pe baza Promise API, ceea ce o face ideală pentru lucrul asincron, în special în cadrul aplicațiilor dezvoltate cu React, Vue, Angular sau alte framework-uri JavaScript moderne. (Contributors, 2024)



Axios oferă o interfață simplificată pentru trimiterea cererilor de tip GET, POST, PUT, DELETE și altele, către servere RESTful, facilitând comunicarea între client și server într-un mod scalabil.

¹ API (Application Programming Interface) reprezintă un set de reguli și protocoale care permit aplicațiilor software să comunice între ele. În contextul aplicațiilor web, un API expune funcționalități ale serverului ce pot fi accesate de către client prin cereri HTTP. (GeeksforGeeks, 2024)

² Nested routes sunt rute imbricate într-o aplicație web, în care o pagină principală conține subpagini, fiecare reprezentată de o rută copil.

³ Data loaders sunt funcții asociate rutelor React Router care încarcă datele necesare unei componente înainte ca aceasta să fie randată, oferind o experiență de utilizare mai fluentă și coerentă.

Datorită suportului nativ pentru interceptoare, Axios permite tratarea globală a erorilor, adăugarea de token-uri de autentificare în antetul cererilor, sau logarea răspunsurilor.

Funcționalități esențiale oferite de Axios includ:

- Suport pentru transformarea automată a datelor JSON.
- Configurarea implicită a antetelor HTTP.
- Compatibilitate cu API-urile browserelor moderne și cu Node.js. (Zhang, 2022)

Axios este utilizat frecvent în aplicațiile care comunică cu un backend REST, precum cele dezvoltate cu Spring Boot, deoarece facilitează trimiterea de date serializate, gestionarea sesiunilor sau a token-urilor JWT și manipularea răspunsurilor de la server într-un mod eficient.

2.6 Framer Motion

Framer Motion este o bibliotecă de animații declarative pentru React, dezvoltată de echipa Framer. Aceasta permite dezvoltatorilor să creeze tranziții și animații fluide într-un mod intuitiv și performant, folosind sintaxa JSX familiară. Este considerată una dintre cele mai populare și moderne soluții pentru animarea interfețelor web în aplicații React, fiind potrivită atât pentru animații simple (fade, slide, scale), cât și pentru interacțiuni complexe și micro-animații. (Atori, 2021)



Caracteristici principale:

- API declarativ și intuitiv: Permite definirea animațiilor direct în JSX, fără a apela imperativele CSS sau JavaScript.
- Performanță ridicată: Utilizează requestAnimationFrame și layout animations optimizate pentru redare fluidă.
- Integrare ușoară cu React: Se integrează natural în fluxul aplicației, fără a fi nevoie de logică suplimentară complexă. (Framer, 2024)

2.7 SockJS & STOMP.js

Pentru aplicațiile moderne care necesită actualizări în timp real, cum ar fi aplicațiile de chat, colaborare sau notificări live, utilizarea WebSocket-urilor este esențială. Totuși, din cauza limitărilor de compatibilitate cu anumite browsere sau rețele, au fost dezvoltate soluții de fallback care să mențină o conexiune bidirecțională între client și server. Printre cele mai populare soluții se numără SockJS, în combinație cu protocolul STOMP, accesat prin biblioteca STOMP.js.

2.7.1 SockJS

SockJS este o bibliotecă JavaScript care oferă o interfață WebSocket, dar cu fallback-uri automate către alte protocoale atunci când WebSocket-ul nu este disponibil. Aceasta asigură

compatibilitate extinsă și o conexiune fiabilă între client și server, indiferent de condițiile de rețea sau browser. (SockJs, 2024)

Caracteristici principale:

- Oferă fallback la protocoale precum XHR-streaming, XHR-polling, EventSource etc.
- Funcționează fără întreruperi în medii cu restricții de rețea (ex: firewall-uri, proxy-uri).
- Utilizată frecvent în aplicații în care fiabilitatea conexiunii în timp real este critică.

2.7.2 STOMP.js

STOMP (Simple Text Oriented Messaging Protocol) este un protocol simplu, bazat pe text, pentru comunicarea între clienți și servere prin mesaje. STOMP.js este o bibliotecă JavaScript care implementează acest protocol, permițând frontend-ului să interacționeze cu un server STOMP (de exemplu, Spring WebSocket) într-un mod standardizat. (WebSocket Support, n.d.) (STOMP, n.d.)

Funcționalități cheie:

- Abonarea la canale (topics) și trimiterea de mesaje către destinații (queues/topics).
- Suport pentru reconectare și heartbeat-uri.
- Permite o separare clară între logica de transport (SockJS/WebSocket) și modelul de mesagerie.

2.7.3 Scenarii de integrare în aplicații web

Combinarea **SockJS** + **STOMP.js** este utilizată frecvent în aplicații bazate pe Spring Boot, care oferă suport nativ pentru WebSocket și STOMP prin modulul *spring-websocket*. Această soluție permite dezvoltarea de aplicații cu funcționalități avansate de comunicare în timp real, cum ar fi:

- transmiterea de mesaje în timp real;
- notificări live;
- actualizări sincronizate între utilizatori.

2.8 Recharts

Recharts este o bibliotecă de vizualizare a datelor construită special pentru React, bazată pe D3.js. Aceasta permite dezvoltatorilor să creeze cu ușurință grafice declarative, reutilizabile și responsive. Sintaxa sa simplificată, component-based, se aliniază cu paradigma React, oferind în același timp o flexibilitate ridicată pentru personalizarea graficelor. (Recharts, n.d.)

Caracteristici principale:

- Grafică SVG pentru o randare clară și scalabilă;
- Suport pentru o varietate de tipuri de grafice: linii, bare, pie chart, area chart, radar etc.;
- Ușor de personalizat cu ajutorul props-urilor și al stilizării dinamice

2.9 Yup & React Hook Form

React Hook Form este o bibliotecă JavaScript open-source destinată gestionării formularelor în aplicațiile React, recunoscută pentru simplitatea sa, performanța ridicată și integrarea nativă cu mecanismul de *hooks* introdus în React 16.8. Aceasta permite înregistrarea ușoară a câmpurilor de formular, validarea acestora și controlul erorilor, totul cu un minim de re-randăr. (Form, 2024)

Biblioteca se remarcă printr-un API simplificat, folosind funcții precum `useForm()`, `register()`, `handleSubmit()` sau `formState`, oferind o alternativă performantă față de metodele tradiționale de gestionare a stării formularelor în React.

Pentru validare, React Hook Form poate fi integrat cu diverse biblioteci externe, printre care cea mai populară este Yup, o bibliotecă de validare schema-based pentru JavaScript. (Jquense, 2023) Yup permite definirea unor reguli de validare declarative, sub formă de *schema objects*, ceea ce contribuie la o logică clară și reutilizabilă de validare a datelor.

2.10 Microlink

Microlink este un serviciu și o bibliotecă JavaScript open-source care permite generarea de previzualizări pentru link-uri printr-o singură cerere HTTP. Aceasta returnează informații precum titlul paginii, imaginea principală, descrierea, favicon-ul și alte elemente relevante, folosind standarde precum Open Graph sau Twitter Cards. Microlink este potrivită pentru aplicații moderne ce necesită îmbogățirea conținutului text cu elemente vizuale dinamice, fiind ușor de integrat atât pe backend, cât și direct în frontend. (Microlink, 2024)



2.11 Spring Boot

Spring Boot este un framework open-source dezvoltat de Pivotal Software, parte a ecosistemului Spring, care facilitează crearea rapidă a aplicațiilor Java enterprise. Scopul principal al Spring Boot este de a simplifica procesul de dezvoltare și configurare a aplicațiilor backend, oferind o abordare opinată („opinionated”) care elimină necesitatea configurațiilor XML extinse sau setărilor complicate. (Walls, 2022)



Spring Boot oferă suport pentru o serie de caracteristici esențiale în dezvoltarea modernă de aplicații web, printre care:

- **Embedded servers** (Tomcat, Jetty, Undertow), care permit rularea aplicației fără a instala un server separat.
- **Autoconfiguration**, prin care framework-ul configurează automat componentele necesare pe baza dependențelor din proiect.
- **Starter dependencies**, care grupează automat bibliotecile necesare pentru diverse funcționalități (ex: `spring-boot-starter-web`, `spring-boot-starter-data-jpa`).

Aplicațiile Spring Boot sunt adesea utilizate pentru dezvoltarea de API-uri RESTful, datorită integrării excelente cu Spring MVC și suportului pentru serializare JSON, validare, autentificare și interacțiune cu baze de date prin JPA/Hibernate. (Long, 2021)

Datorită simplității sale și integrării cu instrumente moderne precum Spring Security, Spring Data, Spring Cloud și altele, Spring Boot a devenit una dintre cele mai utilizate tehnologii backend în industrie și în proiecte academice, oferind o bază solidă pentru aplicații scalabile și ușor de întreținut. (Long, 2021)

2.12 Spring Web

Spring Web este un modul esențial din cadrul ecosistemului Spring Framework, destinat construirii de aplicații web robuste, scalabile și ușor de întreținut. Acesta oferă suport complet pentru dezvoltarea aplicațiilor web tradiționale și a serviciilor RESTful, prin integrarea cu module precum Spring MVC (Model-View-Controller).

Componenta principală a Spring Web este Spring MVC, un framework care urmează arhitectura de tip model-vizualizare-controler. Acesta permite separarea clară a logicii de afaceri de interfața utilizator și oferă un control fin asupra fluxului de date și a comportamentului aplicației. (Walls, 2022)

2.13 Spring Data JPA

Spring Data JPA este un submodul din ecosistemul Spring Data, conceput pentru a simplifica interacțiunea dintre aplicațiile Java și bazele de date relaționale. Acesta oferă o abstracție de nivel înalt peste Java Persistence API (JPA), permițând dezvoltatorilor să creeze rapid și eficient straturi de persistență printr-un model declarativ bazat pe interfețe.

Unul dintre avantajele majore ale Spring Data JPA este posibilitatea de a defini interfețe de tip Repository (precum JpaRepository, CrudRepository sau PagingAndSortingRepository) fără a fi necesară implementarea explicită a metodelor CRUD (Create, Read, Update, Delete), acestea sunt generate automat la runtime de către framework. (Meier, 2022) În plus, Spring Data JPA permite:

- Crearea de interogări derivate pe baza convenției de denumire a metodelor (findByName, findByEmailAndStatus etc.).
- Utilizarea interogărilor personalizate cu JPQL sau SQL nativ prin adnotarea @Query.
- Gestionarea relațiilor dintre entități (OneToMany, ManyToOne, etc.) și a tranzacțiilor.

Prin automatizarea operațiunilor repetitive și expunerea unui API intuitiv, Spring Data JPA contribuie la reducerea cantității de cod boilerplate, sporind astfel productivitatea și lizibilitatea aplicațiilor enterprise. (Schmidt, 2021)

2.14 Spring Security + JWT

Spring Security este un framework puternic și extensibil din ecosistemul Spring, destinat autentificării și autorizării în aplicațiile Java. Acesta oferă un set complet de mecanisme pentru securizarea aplicațiilor web și RESTful, fiind compatibil cu standarde moderne precum OAuth2, JWT și LDAP. (Walls, 2022)

În contextul aplicațiilor REST, o abordare des utilizată este securizarea prin JSON Web Token (JWT). JWT este un standard deschis (RFC 7519) care permite transmiterea sigură a informațiilor între client și server sub forma unui obiect JSON. (Jones, 2015) Într-un flux tipic de autentificare JWT cu Spring Security:

- Utilizatorul trimite datele de autentificare către un endpoint (/login sau /authenticate).
- Serverul validează credențialele și generează un token JWT, semnat criptografic.
- Clientul stochează acest token (de regulă în localStorage sau sessionStorage) și îl atașează în antetul Authorization: Bearer al fiecărei cereri viitoare.
- Serverul validează token-ul și extrage informațiile despre utilizator (fără a menține o sesiune pe server), implementând astfel o arhitectură stateless.

Această abordare este preferată pentru aplicații distribuite, microservicii sau aplicații mobile, datorită scalabilității și separării dintre client și server. (Rahman, 2021) În Spring Security, integrarea JWT se face de obicei printr-un filtru personalizat (OncePerRequestFilter) care validează token-ul și setează contextul de securitate (SecurityContextHolder).

Combinatia Spring Security + JWT permite o securizare robustă și flexibilă, fără necesitatea gestionării sesiunilor pe server, facilitând dezvoltarea aplicațiilor moderne, conforme cu cerințele actuale de performanță și siguranță.

2.15 Spring WebSocket

Spring WebSocket este un modul din ecosistemul Spring Framework care oferă suport pentru comunicarea bidirecțională, asincronă, între client și server, utilizând protocolul WebSocket. Acesta permite dezvoltarea de aplicații în timp real, cum ar fi chat-uri, dashboard-uri sau jocuri multiplayer, unde actualizarea imediată a datelor este esențială. (Walls, 2022)

Protocolul WebSocket oferă o conexiune persistentă între client și server, permițând trimiterea de mesaje în ambele direcții, fără a necesita o cerere HTTP separată pentru fiecare schimb de informații. Comparativ cu modelul tradițional request-response, WebSocket reduce latența și utilizarea resurselor serverului în aplicațiile interactive.

Spring WebSocket este construit peste specificația Java WebSocket API (JSR 356) și oferă, de asemenea, integrare cu STOMP (Simple Text Oriented Messaging Protocol), un protocol sub WebSocket care facilitează modelul de mesagerie publish-subscribe. Pentru gestionarea mesajelor, Spring utilizează o infrastructură bazată pe broker intern (SimpleBroker) sau extern (precum

RabbitMQ), iar pentru configurarea rutelor și endpoint-urilor se utilizează adnotări precum `@MessageMapping` și `@SendTo`.

Un exemplu tipic de flux în Spring WebSocket:

- Clientul inițiază o conexiune WebSocket către un endpoint definit (/ws).
- După stabilirea conexiunii, se abonează la un topic (ex: /topic/messages).
- Serverul primește mesajele trimise de client (ex: pe /app/chat) și le difuzează altor clienți abonați la topicul respectiv.

Această arhitectură facilitează dezvoltarea unor aplicații scalabile și interactive, oferind un model de comunicare performant și eficient pentru aplicațiile moderne în timp real. (Mak, 2024)

2.16 PostgreSQL

PostgreSQL este un sistem de gestiune a bazelor de date relaționale (RDBMS) open-source, avansat, care pune accent pe conformitatea cu standardele SQL, extensibilitate. A fost inițial dezvoltat la Universitatea din California, Berkeley, și continuă să fie întreținut de o comunitate activă de dezvoltatori sub licență PostgreSQL. (Group, PostgreSQL 16 Documentation, 2024)



Printre caracteristicile definitorii ale PostgreSQL se numără:

- Extensibilitatea ridicată, oferind posibilitatea de a defini tipuri de date proprii, funcții, operatori și chiar limbaje de programare;
- Sistemul avansat de gestionare a permisiunilor, care permite control granular al accesului la resurse;
- Suport pentru JSON și XML, făcându-l potrivit atât pentru aplicații relaționale, cât și semi-structurate;

În contextul aplicațiilor moderne Java Spring Boot, PostgreSQL este adesea utilizat în combinație cu JPA/Hibernate, datorită compatibilității excelente și suportului pentru dialect PostgreSQL.

2.17 Lombok

Lombok este o bibliotecă open-source pentru Java care automatizează generarea codului tipic (boilerplate) prin intermediul unor adnotări, simplificând astfel semnificativ dezvoltarea aplicațiilor Java. Aceasta este frecvent utilizată în cadrul aplicațiilor Spring Boot pentru a reduce cantitatea de cod necesară pentru definirea claselor model, DTO-urilor sau serviciilor. (Mokito, 2021)

Prin utilizarea adnotărilor precum `@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`, `@NoArgsConstructor`, `@AllArgsConstructor` sau `@Builder`, Lombok generează automat metodele corespunzătoare în timpul compilării, fără a le scrie explicit în codul sursă. Acest lucru conduce la o creștere a lizibilității codului și o întreținere mai eficientă a proiectului.

Un avantaj major îl reprezintă integrarea transparentă cu IDE-uri moderne precum IntelliJ IDEA sau Eclipse, datorită pluginurilor dedicate care permit recunoașterea și validarea codului generat de Lombok. (Project, 2024)

2.18 Jakarta Validation

Jakarta Validation, cunoscut anterior sub numele de Bean Validation (JSR 380/303), este o specificație standardizată pentru validarea datelor în aplicațiile Java. Este parte a ecosistemului Jakarta EE și este utilizată frecvent în cadrul aplicațiilor Spring Boot, în special împreună cu Hibernate Validator, implementarea de referință a specificației. (EE, 2023)

Această tehnologie oferă un set de adnotări standard pentru a defini regulile de validare aplicabile asupra câmpurilor unei clase. Printre cele mai utilizate se numără:

- **@NotNull, @NotBlank, @NotEmpty** : pentru validarea prezenței valorilor;
- **@Size**: pentru dimensiuni minime și maxime ale colecțiilor, șirurilor de caractere sau array-uri;
- **@Min, @Max, @Positive, @Negative**: pentru validarea valorilor numerice;
- **@Email, @Pattern**: pentru validări pe expresii regulate sau email-uri.

Validarea se realizează în mod automat, de obicei la nivel de controller, prin adnotarea parametrului cu **@Valid** sau **@Validated**. În cazul unei erori de validare, Spring generează un răspuns HTTP corespunzător, ce poate fi gestionat prin mecanisme de tratare globală a excepțiilor (ex: **@ControllerAdvice**).

Un beneficiu major al Jakarta Validation este capacitatea de a extinde sistemul cu validatori personalizați prin implementarea interfeței **ConstraintValidator**, oferind astfel un cadru flexibil și robust pentru asigurarea integrității datelor. (Seacord, 2021)

3 INSTRUMENTE SOFTWARE FOLOSITE

3.1 WebStorm

WebStorm este un mediu de dezvoltare integrat (IDE) dezvoltat de compania JetBrains, specializat în dezvoltarea aplicațiilor front-end și full-stack, în special utilizând JavaScript și framework-urile moderne asociate (React, Angular, Vue etc.). Este considerat unul dintre cele mai avansate IDE-uri pentru dezvoltarea aplicațiilor web datorită suportului extins pentru tehnologii moderne, a integrării cu sisteme de control al versiunilor și a funcțiilor inteligente de completare a codului, refactorizare și depanare. (JetBrains, 2024)



WebStorm oferă o serie de funcționalități care îmbunătățesc semnificativ productivitatea dezvoltatorilor:

- Debugging integrat: instrumente de depanare pentru JavaScript, TypeScript și Node.js;
- Suport pentru testare: integrare cu framework-uri de testare precum Jest, Mocha sau Cypress;
- Tooling modern: suport pentru ESLint, Prettier, Git, Webpack, npm și alte instrumente esențiale;
- Live Editing: actualizare în timp real a aplicației în browser în timpul dezvoltării.

Un avantaj important al WebStorm este integrarea sa profundă cu ecosistemul React, oferind suport pentru JSX, React Hooks și chiar pentru TypeScript, ceea ce îl face ideal pentru aplicații moderne dezvoltate cu aceste tehnologii.

3.2 Redux DevTools

Redux DevTools este un instrument esențial pentru dezvoltatorii care utilizează biblioteca Redux în aplicații frontend, oferind facilități avansate pentru monitorizarea, inspectarea și depanarea stării aplicației. Acesta permite vizualizarea în timp real a stării globale (store), a acțiunilor (actions) și a reducerilor (reducers), fiind disponibil sub forma unei extensii pentru browserele moderne precum Chrome și Firefox. (DevTools, 2024)

Funcționalități principale:

- Time-travel debugging: permite navigarea între stările anterioare și urmărirea modificărilor în fluxul aplicației;
- Vizualizarea acțiunilor: fiecare acțiune dispecerizată este înregistrată, împreună cu starea aplicației de dinainte și de după;
- Export/import de sesiuni: utile pentru partajarea cazurilor de testare sau depanarea colaborativă;
- Filtrarea și limitarea acțiunilor: pentru o analiză mai eficientă a fluxului aplicației în aplicații mari.

Redux DevTools contribuie semnificativ la creșterea transparenței și predictibilității aplicației, facilitând procesul de testare și mentenanță, în special în aplicații complexe care utilizează o stare distribuită și numeroase interacțiuni asincrone.

3.3 IntelliJ IDEA

IntelliJ IDEA este un mediu integrat de dezvoltare (Integrated Development Environment - IDE) dezvoltat de JetBrains, recunoscut pentru suportul său avansat în dezvoltarea aplicațiilor Java și a altor limbaje de programare. Este apreciat pentru facilitățile sale care cresc productivitatea dezvoltatorilor, printre care se numără suportul extins pentru refactorizare, analiză statică a codului, completare inteligentă, navigare rapidă și integrarea facilă cu sisteme de control al versiunilor precum Git.



IntelliJ IDEA oferă suport nativ pentru tehnologiile utilizate în dezvoltarea aplicațiilor backend Java, inclusiv Spring Boot, Hibernate, și alte framework-uri populare, facilitând astfel crearea, testarea și mentenanța aplicațiilor complexe. De asemenea, include instrumente puternice pentru depanare și gestionare a bazelor de date, ceea ce îl face o alegere preferată atât în mediile academice, cât și industriale.

Prin interfața intuitivă și numeroasele pluginuri disponibile, IntelliJ IDEA sprijină un flux de lucru eficient și colaborativ, contribuind la reducerea timpului necesar dezvoltării și îmbunătățirea calității codului. (JetBrains, 2025)

3.4 Postman

Postman este o platformă populară utilizată pentru dezvoltarea, testarea și documentarea API-urilor (Application Programming Interfaces). Aceasta oferă o interfață prietenoasă și intuitivă pentru realizarea cererilor HTTP (GET, POST, PUT, DELETE etc.) și vizualizarea răspunsurilor primite de la servere, facilitând astfel procesul de testare și depanare a serviciilor web. (Inc. P. , 2025)



Postman suportă crearea de colecții de cereri, ceea ce permite organizarea și reutilizarea ușoară a testelor API, dar și automatizarea acestora prin intermediul scripturilor pre și post-request scrise în JavaScript. De asemenea, include instrumente pentru validarea răspunsurilor, gestionarea variabilelor de mediu și generarea de documentație API.

Prin integrarea cu sisteme de control al versiunilor și instrumente CI/CD, Postman devine un aliat important în fluxurile de dezvoltare moderne, sprijinind echipele în livrarea rapidă și calitativă a serviciilor web.

3.5 PgAdmin

PgAdmin este o aplicație open-source de administrare și gestionare a bazelor de date PostgreSQL. Este cel mai popular instrument grafic folosit pentru administrarea serverelor și bazelor de date PostgreSQL, oferind o interfață intuitivă și ușor de utilizat pentru utilizatori de toate nivelurile.



Cu PgAdmin, dezvoltatorii și administratorii de baze de date pot crea, modifica și interoga baze de date, gestiona utilizatori și permisiuni, vizualiza structura și conținutul tabelor, precum și executa scripturi SQL direct din interfața grafică. Instrumentul include facilități avansate precum monitorizarea performanței, generarea de rapoarte și suport pentru backup și restaurare.

PgAdmin este compatibil cu multiple platforme (Windows, macOS, Linux) și facilitează lucrul colaborativ prin opțiuni de configurare a conexiunilor și securității, fiind un instrument indispensabil în administrarea bazelor de date PostgreSQL. (Team P. D., 2025)

3.6 GitHub

GitHub este o platformă populară de găzduire a proiectelor software care implementează sistemul de control al versiunilor Git. GitHub facilitează colaborarea, versiunea codului și urmărirea modificărilor în timp, fiind astfel esențială pentru asigurarea unui flux de lucru organizat și eficient. (Chacon, 2014)



Git este un sistem distribuit de control al versiunilor care permite urmărirea precisă a fiecărei modificări în codul sursă și oferă posibilitatea de a lucra simultan pe mai multe ramuri (branches), asigurând izolarea dezvoltării și testarea funcționalităților înainte de integrarea lor în versiunea principală. (Git, 2019)

Prin intermediul GitHub am beneficiat de:

- Centralizarea codului: Toate componentele aplicației, atât frontend (React), cât și backend (Spring Boot), au fost stocate și gestionate într-un depozit (repository) comun, accesibil și securizat.
- Monitorizarea istoricului: Commit-urile descriptive au documentat clar etapele dezvoltării, facilitând întoarcerea la versiuni anterioare și analiza evoluției codului.
- Integrarea cu instrumente CI/CD: Automatizarea procesului de testare și livrare prin GitHub Actions a contribuit la menținerea calității și stabilității aplicației.

Astfel, utilizarea GitHub a fost fundamentală pentru un management eficient al proiectului, asigurând atât trasabilitatea modificărilor, cât și un proces colaborativ fluid și transparent.

4 PROIECTAREA ȘI ARHITECTURA APLICAȚIEI

4.1 Prezentarea generala

Dezvoltarea aplicației CampusConnect a fost ghidată de o serie de cerințe generale, menite să asigure utilitatea și performanța acesteia într-un mediu academic. Aplicația trebuie să răspundă nevoilor principale ale utilizatorilor (studenți și tutori) în ceea ce privește comunicarea, partajarea de resurse și colaborarea.

În cadrul procesului de analiză, am identificat următoarele cerințe generale:

- Aplicația trebuie să ofere un sistem de autentificare securizat;
- Comunicarea între utilizatori trebuie să fie rapidă, cu suport pentru mesagerie instant;
- Sistemul trebuie să permită transmiterea de fișiere, inclusiv imagini și documente;
- Aplicația trebuie să fie accesibilă din browser, fără a necesita instalare locală;
- Interfața trebuie să fie intuitivă, prietenoasă și adaptabilă la diferite dimensiuni de ecran;
- Datele utilizatorilor trebuie să fie stocate în siguranță, iar interacțiunile să fie gestionate în mod organizat;
- Sistemul trebuie să permită extinderea ulterioară, fiind scalabil și modular;

Aceste cerințe au fost conturate în urma analizei nevoilor concrete din mediul universitar, completate de consultarea unor studii de specialitate și bune practici din domeniul dezvoltării software. De asemenea, a existat dorința de a propune o soluție scalabilă, cu aplicabilitate reală în contextul educațional actual.

4.2 Descrierea funcționalităților sistemului

Specificațiile funcționale definesc comportamentul așteptat al aplicației din perspectiva utilizatorului final. Acestea descriu în mod concret ce acțiuni poate realiza un utilizator și cum răspunde sistemul la acestea. CampusConnect oferă un set de funcționalități esențiale pentru susținerea unui mediu de comunicare eficient în cadrul comunității universitare.

Funcționalitățile principale implementate sunt:

- **Autentificare și înregistrare utilizatori:** Aceștia își pot crea un cont sau se pot autentifica utilizând doar mail-urile instituționale. Sistemul validează datele și oferă acces în funcție de rolul utilizatorului (student, tutore sau administrator).
- **Gestionarea profilului de utilizator:** Fiecare utilizator are un profil care poate fi personalizat (fotografie de profil, descriere etc.).
- **Mesagerie individuală și de grup:** Utilizatorii pot trimite și primi mesaje în timp real, fie în conversații private, fie în cadrul grupurilor.

- **Mesagerie cu asistent AI:** Utilizatorii pot interacționa într-un chat dedicat cu un asistent virtual bazat pe inteligență artificială, care oferă răspunsuri automate în timp real, pe baza unui model antrenabil.
- **Crearea și administrarea grupurilor:** Tutorii pot crea grupuri tematice (ex. pe bază de materie sau proiecte), pot adăuga membri și pot gestiona setările grupului.
- **Partajarea de fișiere:** Aplicația permite trimiterea și descărcarea de imagini și documente (ex. PDF, Word), atât în conversații private, cât și în grupuri.
- **Căutare și filtrare conversații:** Utilizatorii au posibilitatea de a căuta conversații existente după nume sau de a iniția altele noi. De asemenea, conversațiile importante pot fi evidențiate prin funcționalitatea de „fixare” (pinned), fiind afișate prioritar în partea superioară a listei de conversații.
- **Marcarea mesajelor importante:** Este posibilă evidențierea unor mesaje prin opțiuni precum „starred” (favorit), pentru o accesare mai rapidă ulterioară.
- **Vizualizarea și organizarea conținutului media:** Toate mesajele ce conțin imagini, fișiere, documente sau linkuri sunt grupate într-o secțiune dedicată pentru fiecare conversație, facilitând astfel accesul rapid la aceste resurse.
- **Previzualizarea conținutului linkurilor:** Linkurile trimise în conversații pot fi afișate cu o previzualizare a conținutului.
- **Funcționalități avansate pentru administratori:** Administratorii pot vizualiza statistici privind activitatea utilizatorilor (ex. numărul de mesaje trimise într-un grup) și sunt singurii care pot modifica rolurile membrilor dintr-un grup.
- **Profil cu avatar personalizabil:** Utilizatorii beneficiază de un profil care include un avatar generat automat pe baza inițialei numelui. Ulterior, aceștia au posibilitatea de a-și personaliza imaginea de profil, încărcând o fotografie proprie pentru o identificare mai facilă în cadrul aplicației.

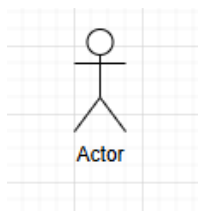
4.3 Modelarea interacțiunii utilizatorilor prin diagrame

4.3.1 Introducere în modelarea UML

Diagramele cazurilor de utilizare reprezintă un instrument esențial în analiza și proiectarea sistemelor software, fiind parte integrantă a limbajului de modelare UML (Unified Modeling Language). Acestea oferă o reprezentare grafică a interacțiunilor dintre actorii externi (utilizatori sau alte sisteme) și sistemul propriu-zis, evidențiind funcționalitățile pe care acesta le oferă. Prin intermediul acestui tip de diagramă, se poate obține o înțelegere clară a cerințelor funcționale, permițând o comunicare eficientă între analiști, dezvoltatori și utilizatori finali. Fiecare caz de utilizare descrie un scenariu specific de interacțiune, contribuind la definirea comportamentului așteptat al sistemului în diverse situații.

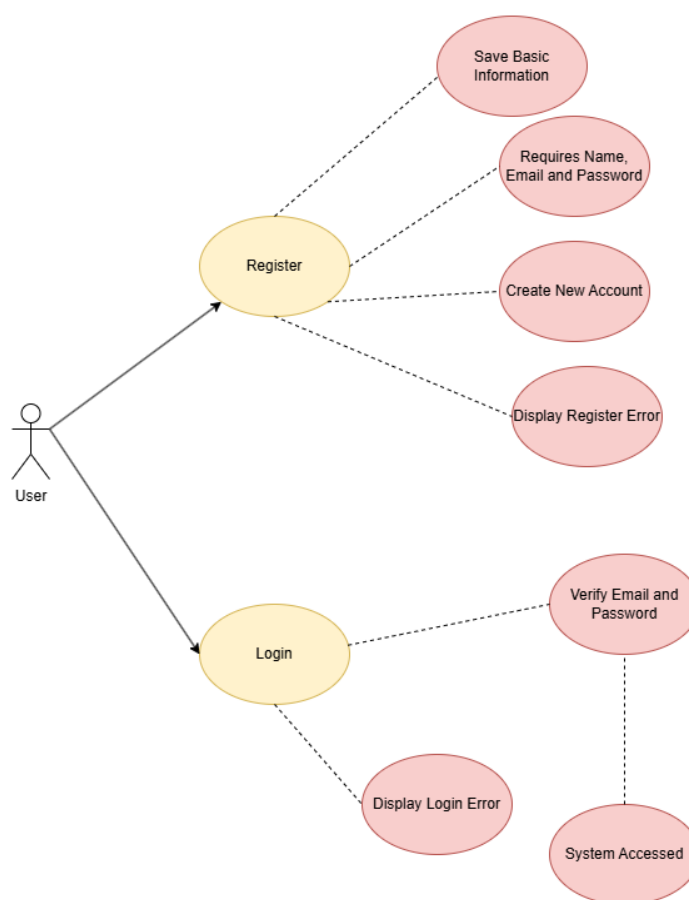
Potrivit lui Booch, Rumbaugh și Jacobson, autorii principali ai UML, „o diagramă de cazuri de utilizare este utilizată pentru a arăta setul de cazuri de utilizare și actorii sistemului, precum și relațiile dintre aceștia”. (Booch, 2005)

Actorul reprezintă o entitate externă sistemului, care interacționează cu acesta. Poate fi un utilizator uman, un alt sistem software sau un dispozitiv. Actorii pot fi primari (inițiază interacțiunea) sau secundari (participă pasiv).



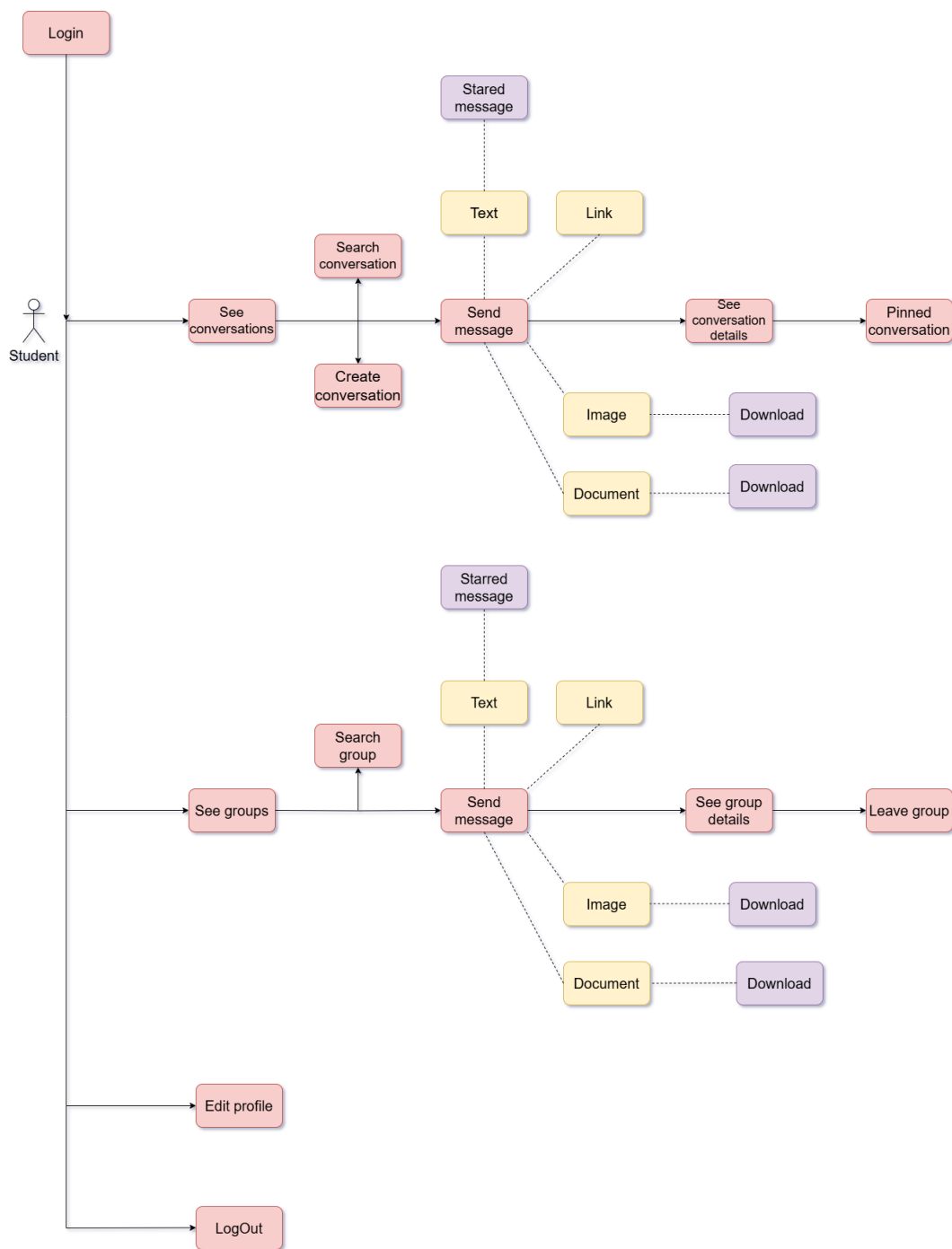
Cazul de utilizare (Use Case) reprezintă o funcționalitate specifică pe care sistemul o oferă unui actor. Este descris printr-un oval și denotă o acțiune valoroasă din perspectiva utilizatorului, precum „Autentificare” sau „Trimitere mesaj”.

4.3.2 Caz de utilizare: Procesul de autentificare



Figură 1 - Fluxul procesului de autentificare

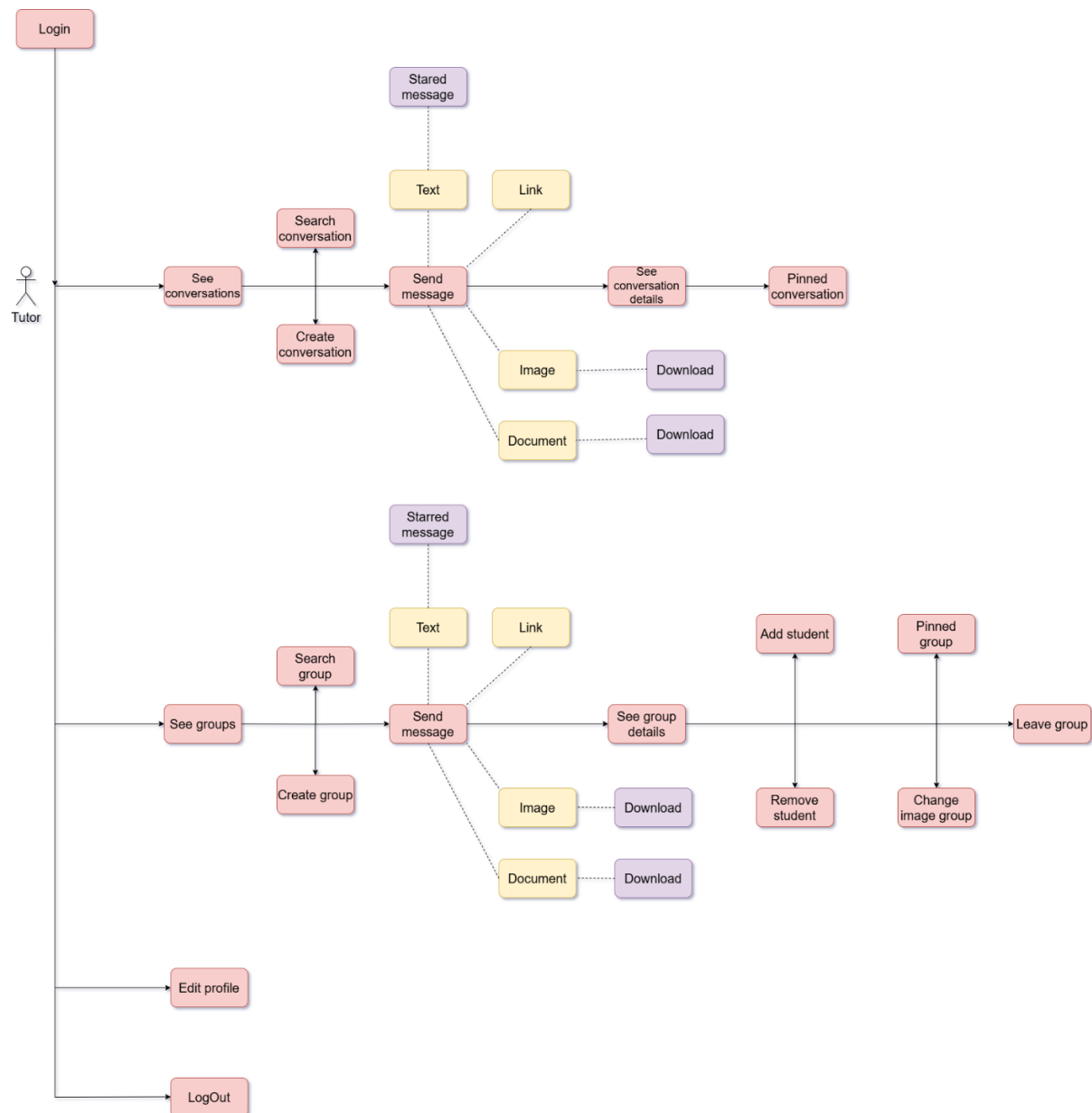
4.3.3 Caz de utilizare: Rolul studentului



Figură 2 - Fluxul de utilizare pentru student

Figura 2 prezintă fluxul de interacțiune disponibil pentru un utilizator cu rol de student după autentificarea în aplicație. Utilizatorul are posibilitatea de a accesa diferite funcționalități, precum inițierea de conversații private cu alți utilizatori, transmiterea de mesaje (text, imagini, documente și link-uri) și participarea activă în grupuri de discuții. În plus, studentul își poate personaliza profilul prin adăugarea unei imagini de profil, a unei descrieri și a unui nickname vizibil în cadrul interacțiunilor din aplicație.

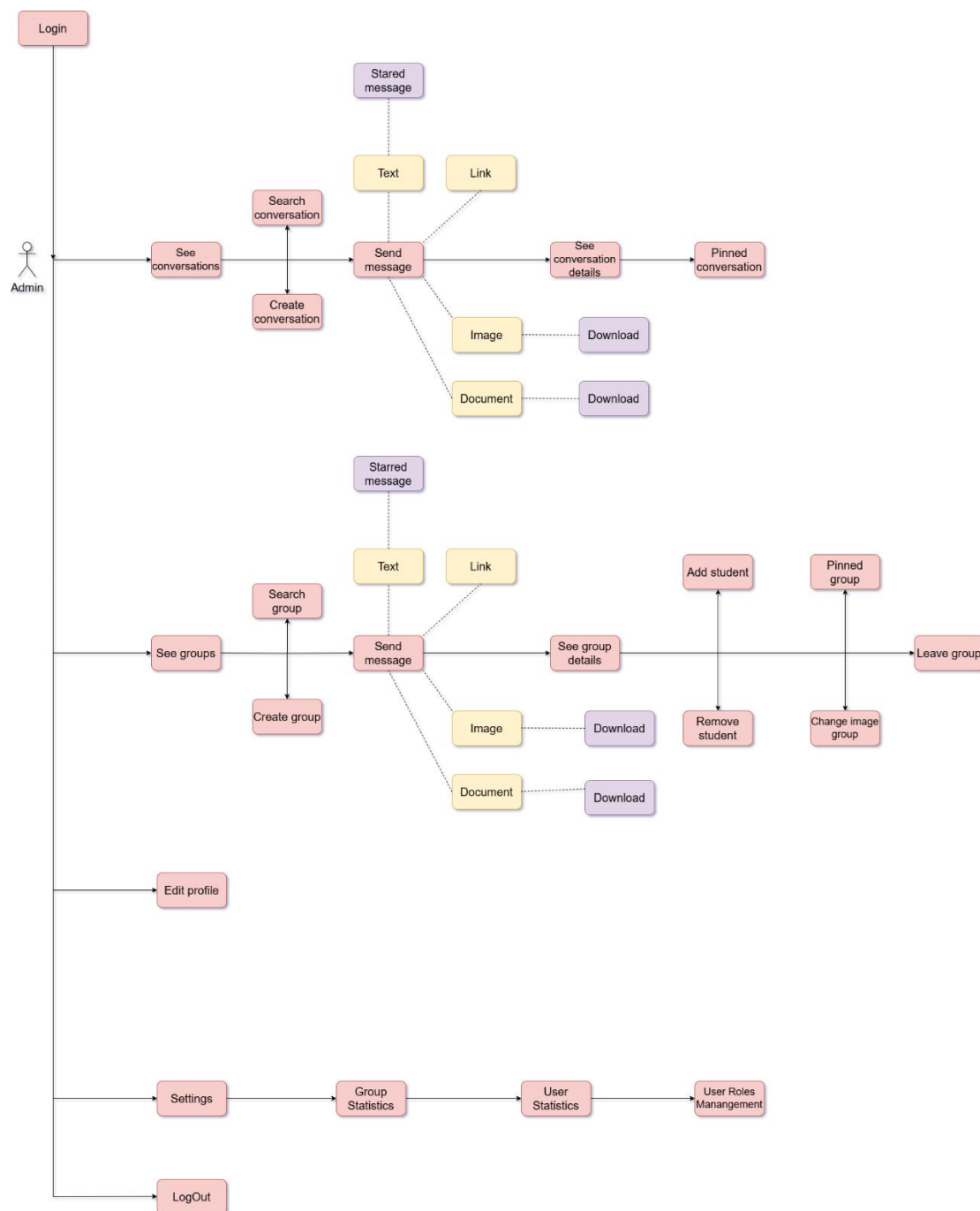
4.3.4 Caz de utilizare: Rolul tutorei



Figură 3 - Fluxul de utilizare pentru tutore

Figura 3 ilustrează diagrama cazurilor de utilizare specifice rolului tutore. Acesta beneficiază de toate funcționalitățile disponibile unui utilizator cu rol de student, precum inițierea conversațiilor private, trimiterea de mesaje (text, imagini, documente, link-uri), participarea în grupuri și personalizarea profilului. În plus, tutorele are drepturi suplimentare, care includ posibilitatea de a crea grupuri de discuții, de a seta imaginea grupului, de a adăuga sau elimina studenți din acestea, având astfel un rol activ în gestionarea colaborării în cadrul platformei.

4.3.5 Caz de utilizare: Funcții specifice administratorului



Figură 4 - Fluxul de utilizare pentru Admin

Figura 4 prezintă diagrama cazurilor de utilizare pentru utilizatorul cu rol de administrator. Acesta dispune de toate funcționalitățile accesibile tutorelui, inclusiv gestionarea conversațiilor private și de grup, trimiterea de mesaje, personalizarea profilului, precum și crearea și administrarea grupurilor. În plus față de tutore, administratorul are acces la funcționalități avansate de management al aplicației, precum vizualizarea statisticilor legate de utilizatori și grupuri, precum și posibilitatea de a modifica rolurile altor utilizatori, atribuindu-le rolul de tutore sau administrator, după caz.

4.4 Structura bazei de date

4.4.1 Modelul conceptual și relațional

Modelul relațional, introdus de Edgar F. Codd (1970), reprezintă fundamentul teoretic al bazelor de date moderne. Acesta organizează datele sub forma unor tabele (relații), fiecare compusă din rânduri (tupluri) și coloane (atribute). Acest model oferă o structură clară și riguroasă, ideală pentru gestionarea relațiilor dintre entități, precum cele existente între utilizatori și mesaje, sau între conversații și grupuri. (Codd, 1970)

Baza de date a fost proiectată cu respectarea principiilor normalizării relaționale, un proces care presupune organizarea datelor astfel încât să se minimizeze redundanța și să se evite anomaliile de inserare, ștergere sau actualizare. (Elmasri, 2017) S-au aplicat formele normale până la cel puțin a treia formă normală (3NF), ceea ce înseamnă că fiecare tabel conține doar atribute ce depind în mod direct de cheia primară și sunt independente tranzitiv.

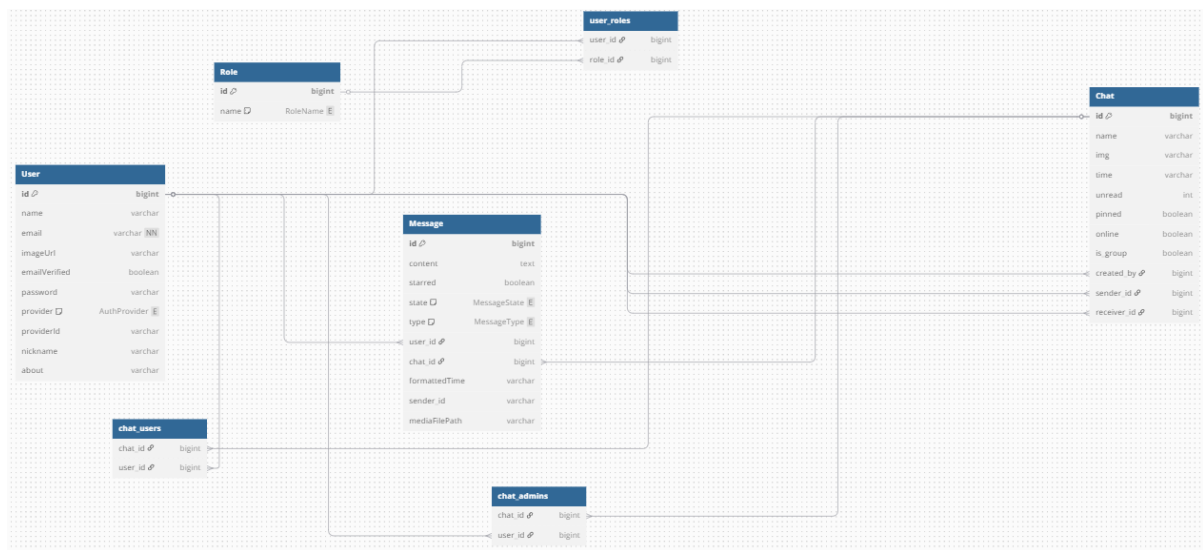
4.4.2 Tehnologii utilizate pentru stocare

Baza de date a fost implementată folosind PostgreSQL, un sistem de gestiune a bazelor de date relaționale open-source, recunoscut pentru suportul avansat pentru tranzacții ACID (Atomicitate, Consistență, Izolare, Durabilitate), extensibilitate, suport JSON, indecși complecși și securitate avansată. (Group, 2024)

Pentru administrarea grafică a bazei de date, s-a utilizat pgAdmin, un instrument oficial dezvoltat pentru PostgreSQL. pgAdmin oferă facilități precum editarea vizuală a tabelelor și relațiilor, rularea interogărilor SQL, monitorizarea performanței și vizualizarea statisticilor, fiind extrem de util în faza de dezvoltare și testare a aplicației. (Team p. , 2024)

4.4.3 Maparea obiect-relațională (ORM)

În cadrul aplicației, persistarea datelor a fost gestionată prin utilizarea JPA (Java Persistence API) în combinație cu Hibernate, unul dintre cele mai utilizate frameworkuri ORM din ecosistemul Java. Prin această abordare, entitățile Java (ex: User, Chat, Message) sunt mapate direct în tabele din baza de date. Relațiile între entități (ex: @ManyToOne, @OneToMany, @ManyToMany) sunt implementate cu ajutorul adnotărilor JPA, ceea ce reduce complexitatea codului SQL și asigură consistența datelor. (Bauer C. &, 2020)



Figură 5 - Baza de date

4.4.4 Considerații legate de performanță și consistență

Pentru a asigura integritatea datelor și performanța sistemului, am adoptat o serie de bune practici:

- Am definit **chei primare** (@Id) și **chei externe** (@JoinColumn) pentru modelarea relațiilor între entități.
- Am utilizat **constrângeri** (@Column(nullable = false), @UniqueConstraint) pentru a preveni introducerea datelor invalide.

```
@Getter
@Setter
@Entity
@Table(name = "users", uniqueConstraints = {
    @UniqueConstraint(columnNames = "email")
})
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;
```

- Am creat indici pe coloane frecvent accesate (ex: email, chat_id) pentru a accelera interogările.
- Selectarea strategiilor adecvate de **fetching** (EAGER sau LAZY), în funcție de frecvența accesării relațiilor.

```
@OneToMany(mappedBy = "chat", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)
@OrderBy("createdDate DESC")
private List<Message> messages = new ArrayList<>();
```

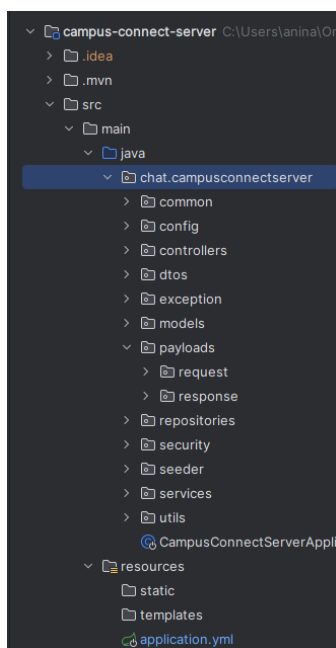
Aceste decizii arhitecturale contribuie la un sistem robust, capabil să gestioneze eficient un volum mare de date și interacțiuni în timp real.

Pe lângă acestea, backend-ul aplicației a fost organizat în mod modular, conform standardelor proiectelor enterprise. Structura pe pachete include:

- **Controllers** care gestionează logica de routing și preluarea cererilor HTTP.
- **Models** care conțin entitățile JPA care definesc schema bazei de date.
- **Repositories** care includ interfețele care extind JpaRepository pentru operații CRUD și interogări personalizate.
- **Security** care gestionează autentificarea și autorizarea utilizatorilor.
- **Services** care sunt intermediare între controllers și repositories, concentrând logica de business.
- **Exception** care gestionează excepțiile.
- **Payloads** care cuprinde atât request cât și response.
- **Utils** care cuprinde clase pentru gestionarea mesajelor de tip media.

Această separare a responsabilităților facilitează testarea, reutilizarea codului și scalarea aplicației.

În figura de mai jos este ilustrată structura modulară:



4.4.5 Utilizarea pattern-urilor de proiectare

Pentru a asigura o arhitectură scalabilă, modulară și ușor de întreținut, aplicația CampusConnect este dezvoltată cu ajutorul unei serii de design pattern-uri consacrate, atât în partea de frontend, cât și în backend:

- a) **Singleton**: Acesta a fost utilizat pentru gestionarea instanței conexiunii WebSocket (ex. WebSocketService), asigurând existența unui singur obiect partajat în întreaga aplicație React.

```
const socket : WebSocketService = new WebSocketService();  
export default socket; Show usages
```

- b) **Observer:** Acesta a fost implementat prin mecanismul WebSocket + STOMP, unde componentele frontend se abonează la diferite canale pentru a primi în timp real evenimente (mesaje).
- c) **Factory:** Aplicat indirect în backend prin Spring, în momentul configurării serviciilor sau a handlerelor WebSocket, unde instanțele sunt create automat și injectate acolo unde este nevoie.
- d) **Strategy:** Utilizat în backend pentru trimiterea mesajelor către utilizatori, fie direct, fie în grup, în funcție de tipul conversației.

```

+ janina280
@Transactional
@MessageMapping("/send-message")
public void sendMessageHandler(@RequestBody MessageRequest req) throws UserException, ChatException {
    User user = userService.findUserProfile(req.getJwtString());

    var message = messageService.sendMessage(req, user);
    var response = messageMapper.toMessageResponse(message);

    for (var m : message.getChat().getUsers()) {
        simpMessagingTemplate.convertAndSendToUser(
            m.getId().toString(),
            destination: "/message/message-send-response",
            response
        );
    }
}

```

- e) **DTO (Data Transfer Object):** Aplicat pentru transferul de date între backend și frontend, evitând expunerea directă a entităților persistente.

```

29 usages + janina280
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserDto {
    private Long id;
    private String name;
    private String email;
    private String imageUrl;
    private Boolean emailVerified = false;
}

```

Aceste pattern-uri contribuie la menținerea clarității codului, la separarea responsabilităților și la extinderea facilă a funcționalităților aplicației.

5 IMPLEMENTAREA APLICAȚIEI

5.1 Securitatea aplicației

Aplicația utilizează framework-ul Spring Security pentru protejarea resurselor expuse de backend. Autentificarea utilizatorilor se realizează pe baza unui token JWT (JSON Web Token), generat în urma unui proces de login și transmis ulterior în fiecare cerere HTTP, prin antetul Authorization.

Arhitectura componentelor de securitate:

- 1) TokenProvider: Componentă responsabilă pentru generarea, extragerea și validarea token-urilor JWT. Token-ul include în payload identificatorul utilizatorului (sub) și o listă de roluri.
- 2) TokenAuthenticationFilter: Implementare a clasei OncePerRequestFilter care interceptează fiecare cerere HTTP, extrage token-ul JWT, îl validează și setează utilizatorul curent în contextul de securitate (SecurityContextHolder).

```
try {
    String jwt = getJwtFromRequest(request);

    if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
        Long userId = tokenProvider.getUserIdFromToken(jwt);

        UserDetails userDetails = customUserService.loadUserById(userId);
        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
        authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

        SecurityContextHolder.getContext().setAuthentication(authentication);
    }
} catch (Exception ex) {
    logger.error("Could not set user authentication in security context", ex);
}
```

- 3) CustomUserDetailsService: Implementare a interfeței UserDetailsService care permite încărcarea unui utilizator fie pe baza adresei de email, fie pe baza ID-ului. Este utilizată pentru extragerea datelor persistente ale utilizatorului din baza de date.
- 4) UserPrincipal: Implementare custom a interfețelor UserDetails care servește ca reprezentare internă a utilizatorului autentificat. Include informații despre roluri și poate fi convertită în entitate User.
- 5) SecurityConfig: Clasa de configurare principală, care definește un SecurityFilterChain și setează reguli de acces pentru diferite endpoint-uri. De asemenea, adaugă TokenAuthenticationFilter în lanțul de filtre Spring Security.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .cors(Customizer.withDefaults())
        .csrf(csrf -> csrf.disable())
        .formLogin(AbstractHttpConfigurer::disable)
        .httpBasic(AbstractHttpConfigurer::disable)
        .exceptionHandling(exceptionHandling ->
            exceptionHandling.authenticationEntryPoint(new RestAuthenticationEntryPoint())
        )
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/auth/**", "/logout/**", "/app/**", "/ws/**", "/uploads/**").permitAll()
            .anyRequest().authenticated()
        )
    }
}
```

5.2 Sistemul de autentificare

În cadrul procesului de dezvoltare a sistemului de autentificare, am evaluat inițial posibilitatea integrării unui mecanism OAuth2 bazat pe conturi Google, cu restricție de domeniu specific (@campusconnect.com), pentru a controla accesul exclusiv al utilizatorilor aparținând instituției. Pentru a implementa acest lucru, am realizat următorii pași:

- Am obținut un domeniu gratuit prin intermediul GitHub Student Developer Pack, care oferă acces la servicii precum Namecheap pentru înregistrarea unui domeniu timp de un an;
- Am creat aplicație OAuth2 în Google Cloud Console;
- Am activat și configurat API-ului Google Identity Services;
- Am implementat fluxul de autorizare (authorization code flow) pe backend folosind Spring Security (OAuth2UserInfo, CustomOAuth2UserService).

The screenshot shows the configuration page for an OAuth2 client in the Google Cloud Console. At the top, there is a text input field labeled 'Name *' with the value 'Campus Connect'. Below it, a small note states: 'The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.' Below this is a blue information box with a circular icon containing an 'i' and the text: 'The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).' Below the box, the section 'Authorized JavaScript origins' is visible, with a sub-label 'For use with requests from a browser'. It contains a text input field labeled 'URIs 1 *' with the value 'http://localhost:3000' and a '+ Add URI' button. Below this, the section 'Authorized redirect URIs' is visible, with a sub-label 'For use with requests from a web server'. It contains a text input field labeled 'URIs 1 *' with the value 'http://localhost:8080/oauth2/callback/google' and another '+ Add URI' button.

Totodată, pentru a valida exclusiv utilizatorii cu email de domeniu instituțional, am încercat:

- Înregistrarea unui domeniu personalizat și configurarea de email-uri proprii prin servicii de tipul Zoho Mail care aveau până la 5 utilizatori (adică 5 adrese de email gratuite pe domeniul propriu)
- Configurarea manuală a înregistrărilor DNS (MX, SPF, DKIM) pentru validarea domeniului;
- Integrarea acestor adrese în Google Workspace în vederea testării cu OAuth2 și filtrării utilizatorilor.

Cu toate acestea, implementarea completă a acestui mecanism a fost împiedicată de o serie de constrângeri:

- Necesitatea deținerii unui cont Google Workspace valid și verificat, lucru care implică costuri și validare din partea Google;

- Restricții impuse de Google privind aplicațiile care folosesc scopuri OAuth considerate sensibile (ex: profile, email);
- Timpul necesar pentru aprobarea aplicației și verificarea domeniului.

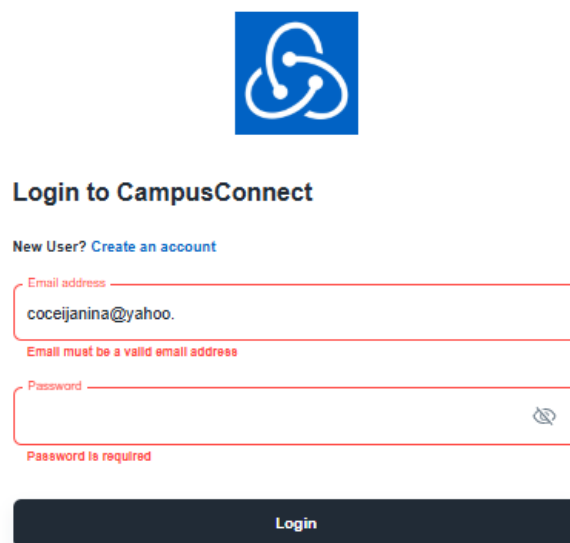
După aproximativ două săptămâni de testare, configurare și depanare, am decis renunțarea temporară la această soluție din considerente de complexitate și compatibilitate.

Ca soluție de compromis, s-a păstrat autentificarea clasică prin email și parolă, cu validarea strictă a domeniului @campusconnect.com în momentul înregistrării, obținând astfel un echilibru între simplitate și controlul accesului.

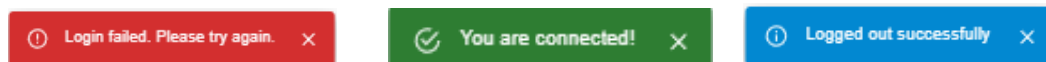
5.2.1 Autentificarea unui utilizator

La accesarea aplicației, utilizatorul este întâmpinat cu interfața de login. Dacă nu deține deja un cont, acesta poate naviga către pagina de înregistrare (Register), unde își completează numele, adresa de email și parola. După finalizarea cu succes a înregistrării, utilizatorul este redirecționat automat către pagina de autentificare.

Validarea formularelor se realizează pe partea de client utilizând biblioteca Yup, care impune reguli stricte pentru email și parolă.



Mesajele de eroare și confirmare sunt afișate sub formă de snackbar-uri, oferind feedback clar și imediat în funcție de acțiunile efectuate.



Formularul de login este integrat cu Redux și utilizează react-hook-form pentru gestionarea stării. La trimiterea formularului, datele sunt transmise asincron prin dispatch(LoginUser(data)) către

backend. În cazul unei autentificări reușite, tokenul JWT primit este salvat atât în localStorage, cât și în Redux store, pentru a păstra sesiunea activă și a permite accesul la rutele protejate.

```
const onSubmit = async (data) : Promise<void> => { Show usages  janina280 *
  setError( value: "");
  try {
    const result = await dispatch(LoginUser(data));

    if (result && result.success === false) {
      setError(result.error);
      reset( values: {email: "", password: ""});
    }
  }
  catch (error) {
    console.error("Login error:", error);
    setError( value: "Something went wrong. Please try again.");
    reset( values: {email: "", password: ""});
  }
};
```

Funcționalități suplimentare de UX includ posibilitatea de a afișa/ascunde parola printr-un toggle vizual, folosind iconițele Eye și EyeSlash din biblioteca Phosphor-react.

Pe backend, procesul de autentificare este deservit de endpoint-ul POST `/auth/login`, care primește un payload de tip `LoginRequest`.

```
@PostMapping("/login")
public ResponseEntity<?> authenticateUser(@RequestBody LoginRequest loginRequest) {
```

Acesta include datele de autentificare ale utilizatorului, respectiv câmpurile email și password, transmise în format JSON.

```
2 usages  janina280 *
@Setter
@Getter
public class LoginRequest {
    private String email;

    private String password;
}
```

Înainte de validarea credențialelor, se efectuează un control al formatului adresei de email, pentru a verifica dacă aceasta aparține domeniului `@campusconnect.com`. În cazul unui format incorect, este returnat un mesaj de eroare.

```
if (!loginRequest.getEmail().endsWith("@campusconnect.com")) {
    return ResponseEntity
        .badRequest()
        .body(new ApiResponse( message: "The email must be of the type @campusconnect.com.", success: false));
}
Authentication authentication = authenticationManager.authenticate(
    new UsernamePasswordAuthenticationToken(
        loginRequest.getEmail(),
        loginRequest.getPassword()
    )
);
```

Dacă validarea trece, se construiește un obiect `UsernamePasswordAuthenticationToken`, care este transmis către `AuthenticationManager`. Dacă autentificarea are succes:

```
SecurityContextHolder.getContext().setAuthentication(authentication);
String token = tokenProvider.createToken(authentication);
var user = userRepository.findByEmail(loginRequest.getEmail());
return ResponseEntity.ok(new AuthResponse(user.getId(), token));
```

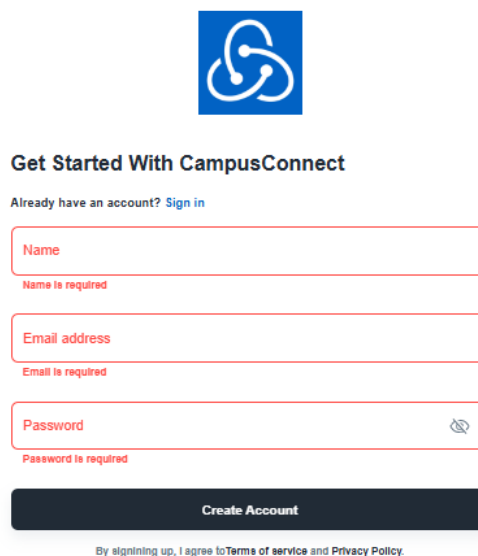
- Contextul de securitate (`SecurityContext`) este populat cu datele utilizatorului.

- Se generează un token JWT prin TokenProvider, care encodează identitatea utilizatorului.
- Răspunsul conține atât tokenul JWT, cât și ID-ul utilizatorului.

Aceste date sunt ulterior utilizate pe frontend pentru a accesa rute protejate, a trimite cereri cu antetul de autorizare și a menține sesiunea autentificată.

5.2.2 Înregistrarea unui nou utilizator

Fluxul de creare a unui cont nou este asemănător cu cel de autentificare. Aceasta folosește aceleași biblioteci moderne precum React Hook Form, Yup (pentru validare) și Redux pentru a colecta și procesa datele introduse de utilizator.



Get Started With CampusConnect

Already have an account? [Sign in](#)

Name
Name is required

Email address
Email is required

Password
Password is required

Create Account

By signing up, I agree to [Terms of service](#) and [Privacy Policy](#).

Datele sunt transmise la trimiterea formularului către RegisterUser, care trimite o cerere POST către backend. În funcție de răspuns:

- Se afișează notificări de tip snackbar (feedback pozitiv sau negativ)
- În caz de succes, utilizatorul este redirecționat către pagina de login

```
export function RegisterUser(formValues, navigate) { Show usages 1 janina280 *
  return async (dispatch, getState) : Promise<void> => {
    dispatch(slice.actions.updateIsLoading({isLoading: true, error: false}));
    await axios
      .post(
        {
          url: "auth/register",
          data: {
            ...formValues,
          },
          config: {
            headers: {
              "Content-Type": "application/json",
            },
          },
        }
      )
      .then(function (response : AxiosResponse<any> ) : void {
        dispatch(
          showSnackBar( (severity,message): {severity: "success", message: response.data.message})
        );
        dispatch(
          slice.actions.updateIsLoading({isLoading: false, error: false})
        );

        navigate("/auth/login")
      })
  }
}
```

- În caz de eroare, formularul este resetat și este afișat un mesaj descriptiv

```
.catch((error) :void => {
  console.log(error);
  dispatch(showSnackBar( (severity,message): {severity: "error", message: error.response.data.message}));
  dispatch(
    slice.actions.updateIsLoading({error: true, isLoading: false})
  );
});
```

Pentru a adăuga dinamism și o experiență vizuală mai fluidă, componenta formularului de înregistrare, dar și cea de autentificare este animată utilizând biblioteca Framer Motion, prin efectul `varBounce().inUp`, aplicat asupra containerului principal.

Pe partea de server, înregistrarea este gestionată de endpoint-ul: `POST /auth/register`.

Acesta primește un obiect JSON de tip `SignUpRequest`, cu următoarele câmpuri:

```
@Getter
@Setter
public class SignUpRequest {
    private String name;
    private String email;
    private String password;
}
```

- name: numele complet al utilizatorului
- email: trebuie să fie de forma `@campusconnect.com`
- password: parola brută, ce va fi criptată cu `passwordEncoder`

Logica din controller efectuează următoarele verificări: validează faptul că emailul are domeniul `@campusconnect.com` și verifică dacă adresa de email există deja în baza de date.

```
if (!signUpRequest.getEmail().endsWith("@campusconnect.com")) {
    return ResponseEntity
        .badRequest()
        .body(new ApiResponse( message: "The email must be of the type @campusconnect.com.", success: false));
}

if (userRepository.existsByEmail(signUpRequest.getEmail())) {
    return ResponseEntity
        .badRequest()
        .body(new ApiResponse( message: "Email address already in use.", success: false));
}
```

Dacă datele sunt valide, așa cum se observa și în imaginea de mai jos, se creează o nouă entitate User. Parola este criptată cu passwordEncoder. Se setează automat rolul ROLE_USER (care corespunde studentului), iar utilizatorul este salvat în baza de date. Se returnează un răspuns 201 Created, cu mesajul „User registered successfully!”.

```
User user = new User();
user.setName(signUpRequest.getName());
user.setEmail(signUpRequest.getEmail());
user.setPassword(signUpRequest.getPassword());
user.setProvider(AuthProvider.LOCAL);

user.setPassword(passwordEncoder.encode(user.getPassword()));
Role roleUser = roleRepository.findByName(Role.RoleName.ROLE_USER)
    .orElseThrow(() -> new RuntimeException("Role ROLE_USER not found!"));
user.setRoles(Collections.singleton(roleUser));

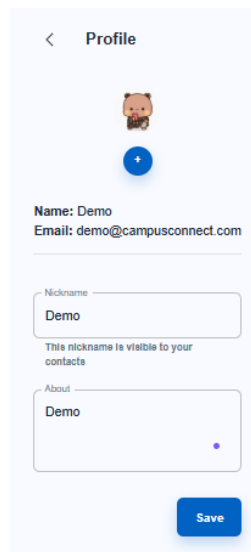
User result = userRepository.save(user);

URI location = ServletUriComponentsBuilder
    .fromCurrentContextPath().path("/user/me")
    .buildAndExpand(result.getId()).toUri();

return ResponseEntity.created(location)
    .body(new ApiResponse( message: "User registered successfully!", success: true));
```

5.3 Sistemul de gestionare a profilului utilizatorului

Pentru a oferi utilizatorilor posibilitatea de a-și personaliza profilul, a fost implementat un formular dedicat unde aceștia pot edita următoarele câmpuri: poreclă, descriere și imagine de profil.



Formularul este construit utilizând React Hook Form împreună cu Yup pentru validarea datelor:

```
const ProfileSchema : ObjectSchema<...> = Yup.object().shape({
  nickname: Yup.string().required( msg: "Nickname is required"),
  about: Yup.string().required( msg: "About is required"),
});
```

Componentele reutilizabile RHFFextField și FormProvider facilitează integrarea cu sistemul de validare și gestionarea stării formularului.

```

<RHTextField
  name="nickname"
  label="Nickname"
  helperText="This nickname is visible to your contacts"
/>
<RHTextField
  multiline
  rows={3}
  maxRows={5}
  name="about"
  label="About"
/>

```

Imaginea de profil este afișată cu ajutorul componentei personalizate CreateAvatar, care generează un avatar colorat în funcție de inițiala numelui dacă nu există o imagine încărcată:

```

<CreateAvatar
  name={user?.name || ""}
  imageUrl={` ${BASE_URL}/${user.imageUrl}?${Date.now()}`}
  size={56}
/>

```

Avatarul este colorat automat în funcție de prima literă a numelui utilizatorului, pe baza unui set predefinit de grupe.

```

const CreateAvatar = ({ name, imageUrl, size : number = 56 }) => {
  const COLORS : { ... } = {
    primary: '#1976d2',
    info: '#0288d1',
    success: '#2e7d32',
    warning: '#ed6c02',
    error: '#d32f2f',
    default: '#9e9e9e',
  };

  const PRIMARY_NAME : string[] = ['A', 'N', 'H', 'L', 'Q', '9', '8'];
  const INFO_NAME : string[] = ['F', '6', 'T', 'I', 'J', '1', '2', '3'];
  const SUCCESS_NAME : string[] = ['K', 'D', 'Y', 'B', '0', '4', '5'];
  const WARNING_NAME : string[] = ['P', 'E', 'R', 'S', 'C', 'U', '6', '7'];
  const ERROR_NAME : string[] = ['V', 'W', 'X', 'M', 'Z'];

  function getFirstCharacter(name) {
    return name && name.charAt(0).toUpperCase();
  }

  function getAvatarColor(name) : string {
    const firstChar = getFirstCharacter(name);

```

Încărcarea imaginii este realizată printr-un `<input type="file" />`, care declanșează acțiunea Redux UploadUserImage. Aceasta trimite fișierul către backend și actualizează imaginea profilului în store-ul global.

```

<Button
  component="label"
  variant="contained"
  sx={{borderRadius: "50%", minWidth: 0, width: 36, height: 36}}
>
  +
  <input
    hidden
    accept="image/*"
    type="file"
    onChange={(e : ChangeEvent<HTMLInputElement> ) : void => {
      if (e.target.files[0]) {
        dispatch(UploadUserImage(e.target.files[0]));
      }
    }}
  />
</Button>

```

La trimiterea formularului, datele sunt ambalate într-un FormData care conține un blob JSON (updatedUserDto), pentru compatibilitate cu uploadul multipart:

```
const dataToSend :FormData = new FormData();
dataToSend.append(
  name: "updatedUserDto",
  new Blob([JSON.stringify(completeForm)], {type: "application/json"})
);
```

Cererea este transmisă către endpoint-ul `/api/user/update` cu metoda PUT și tokenul de autentificare JWT în headerul Authorization.

```
try {
  const response :Response = await fetch( input: `${BASE_URL}/api/user/update`, init: {
    method: "PUT",
    headers: {
      Authorization: `Bearer ${token}`,
    },
    body: dataToSend,
  });

  if (response.ok) {
    await response.json();
    setIsUpdated( value: true);
    dispatch(FetchUserProfile());
    dispatch(showSnackBar( {severity, message}: {severity: "success", message: "Successfully updated"}));
  } else {
    console.error("Error updating profile");
    dispatch(showSnackBar( {severity, message}: {severity: "danger", message: "Error updating"}));
  }
}
catch (error) {
  console.error("Error during submission", error);
}
```

După un răspuns cu succes, este afișat un mesaj de confirmare (Snackbar) și se face re-fetch la datele utilizatorului pentru a actualiza interfața.

Pentru a permite utilizatorilor să își personalizeze profilul, am definit un endpoint PUT care primește: un obiect UserDto serializat ca multipart/form-data, opțional, o imagine (MultipartFile) și utilizatorul autentificat (@CurrentUser).

```
@PutMapping("/{update}")
public ResponseEntity<UserDto> updateProfile(
  @RequestPart(value = "image", required = false) MultipartFile image,
  @RequestPart("updatedUserDto") @Valid UserDto updatedUserDto,
  @CurrentUser UserPrincipal currentUser) {
```

Dacă utilizatorul există, se actualizează porecla și descrierea:

```
User existingUser = userRepository.findById(currentUser.getId())
    .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

existingUser.setNickname(updatedUserDto.getNickname());
existingUser.setAbout(updatedUserDto.getAbout());
```

Dacă a fost trimisă o imagine, aceasta este salvată folosind fileService, iar calea rezultată este atașată obiectului utilizator:

```
if (image != null && !image.isEmpty()) {
  String imagePath = fileService.saveFile(image, Long.valueOf(currentUser.getId().toString()));
  existingUser.setImageUrl(imagePath);
}
```

După modificări, obiectul User este salvat în baza de date și se returnează un UserDto actualizat:

```

        userRepository.save(existingUser);

        UserDto updatedUserResponse = new UserDto(existingUser);
        return ResponseEntity.ok(updatedUserResponse);
    }

```

Metoda saveFile(...) din FileService salvează imaginile de profil într-un folder dedicat fiecărui utilizator:

```

    public String saveFile(
        @NonNull MultipartFile sourceFile,
        @NonNull Long userId
    ) {
        final String fileUploadSubPath = "users" + separator + userId;
        return uploadFile(sourceFile, fileUploadSubPath);
    }

```

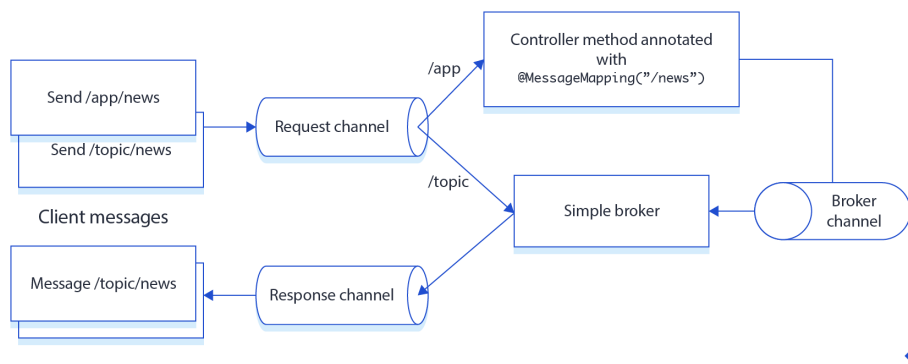
Folder-ul este creat automat pentru fiecare utilizator:

```

    public byte[] getFile(@NonNull Long userId, @NonNull String fileName) {
        final String filePath = fileUploadPath + separator + "users" + separator + userId + separator + fileName;
        Path file = Paths.get(filePath);
        try {
            log.info("Retrieving file from: " + filePath);
            return Files.readAllBytes(file);
        } catch (IOException e) {
            log.error("Failed to read file: " + filePath, e);
            return null;
        }
    }

```

5.4 Sistemul de gestionare a conexiunii WebSocket



Figură 6 - Fluxul de mesaje între client și server folosind STOMP, WebSocket și brokerul simplu în Spring

Diagrama de mai sus ilustrează modul în care mesajele sunt transmise între client și server în cadrul aplicației utilizând WebSocket, STOMP și un broker simplu. Sistemul este structurat în mai multe componente:

- Clientul trimite mesaje fie către /app/... (mesaje care vor fi procesate de server), fie către /topic/... (mesaje difuzate direct de broker).
- Canalul de request preia mesajele de la client.
- Dacă destinația începe cu /app, mesajul este redirectionat către un controller Spring care gestionează mesajul printr-o metodă adnotată cu @MessageMapping.

- Răspunsurile sau mesajele de difuzat sunt transmise către un broker simplu (e.g., SimpleBroker) configurat în backend.
- Brokerul transmite mesajul către toți clienții abonați printr-un canal de răspuns, utilizând destinații de tip /topic.

5.4.1 Descriere generală (Frontend)

Sistemul de gestionare a conexiunii WebSocket din cadrul aplicației frontend CampusConnect asigură comunicarea bidirecțională în timp real între clientul React și backend-ul Spring Boot, utilizând protocolul STOMP și SockJS. Acest sistem facilitează funcționalități precum trimiterea și recepționarea mesajelor, notificări în timp real pentru evenimente (ex. creare conversație, adăugare utilizator în grup etc.) și sincronizarea conversațiilor.

5.4.2 Componentele principale

- a) Clasa WebSocketService acționează ca un singleton care gestionează conexiunea WebSocket și subscripțiile STOMP.

```
import {Client} from "@stomp/stompjs";
import SockJS from "sockjs-client";

const SOCKET_URL :string = "http://localhost:8080/ws";
class WebSocketService { Show usages 1 janina280 +1
  constructor() { Show usages 1 janina280
    this.stompClient = null;
    this.subscriptions = {};
  }
}
```

Metodele principale sunt:

- connect() stabilește conexiunea WebSocket cu serverul

```
connect() :Promise<unknown> { Show usages 1 janina280 +1
  return new Promise( {executor: (resolve, reject) :any | undefined => {
    if (this.stompClient?.connected) {
      console.warn( {message: "WebSocket already connected."});
      return resolve();
    }

    const socket :WebSocket = new SockJS(SOCKET_URL);
    this.stompClient = new Client( {conf: {
      websocketFactory: () :WebSocket | SockJS => socket,
      reconnectDelay: 5000,
      debug: (str :string ) :any | void => console.log(str),
    }});
  });
}
```

- on(topic, callback) se abonează la un topic și execută funcția callback la recepționarea unui mesaj:


```

async on(topic, callback) : Promise<void> { Show usages 1 janina280 +1
  if (!this.stompClient || !this.stompClient.connected) {
    console.warn( {message: "WebSocket is not connected yet. Waiting..."});
    await this.connect();
  }

  const subscription = this.stompClient.subscribe(topic, (message) : void => {
    callback(JSON.parse(message.body));
  });

  this.subscriptions[topic] = subscription;
}

```

- emit(destination, message) trimite un mesaj către un topic specific:

```

async emit(destination, message) : Promise<void> { Show usages 1 janina280 +1
  if (!this.stompClient || !this.stompClient.connected) {
    console.warn( {message: "Cannot send message, WebSocket is not connected. Waiting..."});
    await this.connect();
  }

  this.stompClient.publish({
    destination: destination,
    body: JSON.stringify(message),
  });
}

```

- off(topic) dezabonează de la un topic.
- disconnect() încheie conexiunea și curăță subscripțiile.

```

disconnect() : void { Show usages 1 Madalin Popescu +1
  Object.values(this.subscriptions).forEach((sub) => sub.unsubscribe());
  this.subscriptions = {};
  if (this.stompClient) {
    this.stompClient.deactivate();
    console.log("WebSocket connection closed.");
  }
}

```

- b) Contextul WebSocketContext permite injectarea instanței WebSocket în arborele React printr-un context global.

```

const WebSocketContext : Context<unknown> = createContext();

const WebSocketProvider = ({children}) => { Show usages
  const [isConnected, setIsConnected] = useState( {initialState: false});

  useEffect( {effect: () => {
    socket.connect().then(() : void => {
      setIsConnected( {value: true});
    });

    return () : void => socket.disconnect();
  }, deps: []);

  return (
    <WebSocketContext.Provider value={{isConnected, socket}}>
      {children}
    </WebSocketContext.Provider>
  );
};

const useWebSocket = () => { Show usages
  return useContext(WebSocketContext)
};

export {WebSocketProvider, useWebSocket};

```

5.4.3 Integrarea în aplicație

Componentele funcționale din aplicație folosesc hook-ul `useWebSocket` pentru a accesa conexiunea WebSocket și a asculta evenimente în timp real.

```
const {isConnected, socket} = useWebSocket();
```

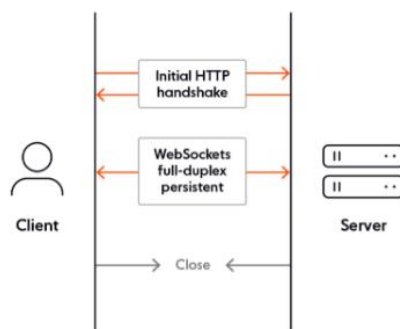
Exemplu: Inițializarea conversațiilor la conectare

Se așteaptă ca WebSocket-ul să fie conectat `isConnected === true`. Trimite o cerere către backend pentru a obține toate conversațiile utilizatorului (`/app/chats`). Primește răspunsul pe topicul `/user/{user_id}/chat/chats-response`. Apoi actualizează store-ul Redux cu conversațiile primite.

```
useEffect( effect: () :void => {  
  
  if (!isConnected) return;  
  socket.emit("/app/chats", "Bearer " + token)  
  
  socket.on(`/user/${user_id}/chat/chats-response`, (data) :void => {  
    dispatch(FetchDirectConversations(data));  
  });  
  
}, deps: [isConnected, dispatch]);
```

5.4.4 Descriere generala (Backend)

Sistemul WebSocket din backend-ul aplicației CampusConnect este construit folosind Spring WebSocket cu suport STOMP (Simple Text Oriented Messaging Protocol). Acesta gestionează recepționarea mesajelor trimise de client, procesarea acestora și transmiterea răspunsurilor țargetate către utilizatori individuali sau către canale publice de mesagerie.



Figură 7 - Procesul de inițializare și comunicare prin WebSocket între client și server

5.4.5 Componentele principale

- Clasa `WebSocketConfig` definește endpoint-ul public și configurarea canalelor de mesagerie STOMP:

```

± janina280
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    no usages ± janina280
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry){
        registry.addEndpoint(...paths: "/ws").setAllowedOriginPatterns("*").withSockJS();
    }
    no usages ± janina280
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry){
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker(...destinationPrefixes: "/group", "/user", "/chat");
        registry.setUserDestinationPrefix("/user");
    }
    no usages ± janina280
    @Override
    public void configureWebSocketTransport(org.springframework.web.socket.config.annotation.WebSocketTransportRegistration registration) {
        registration.setMessageSizeLimit(10 * 1024 * 1024); // 10MB
        registration.setSendBufferSizeLimit(10 * 1024 * 1024); // 10MB
        registration.setSendTimeLimit(60000); // 60 sec
    }
}

```

b) Adnotările:

`@MessageMapping` anotează metodele din controlere care prelucrează mesajele primite prin WebSocket, similar cu `@PostMapping` în contextul REST.

`SimpMessagingTemplate` este componenta Spring utilizată pentru a trimite mesaje înapoi către clienți, fie către utilizatori individuali (prin `convertAndSendToUser(...)`), fie către un topic public.

```

± janina280
@Transactional
@MessageMapping("/chats")
public List<ChatDto> findAllChatByUserHandle(@RequestHeader("Authorization") String jwt) throws UserException {
    var currentUserId = tokenProvider.getUserIdFromToken(jwt.substring(beginIndex: 7));

    var chats = chatService.findAllChatByUserId(currentUserId);

    simpMessagingTemplate.convertAndSendToUser(currentUserId.toString(), destination: "/chat/chats-response", chats);

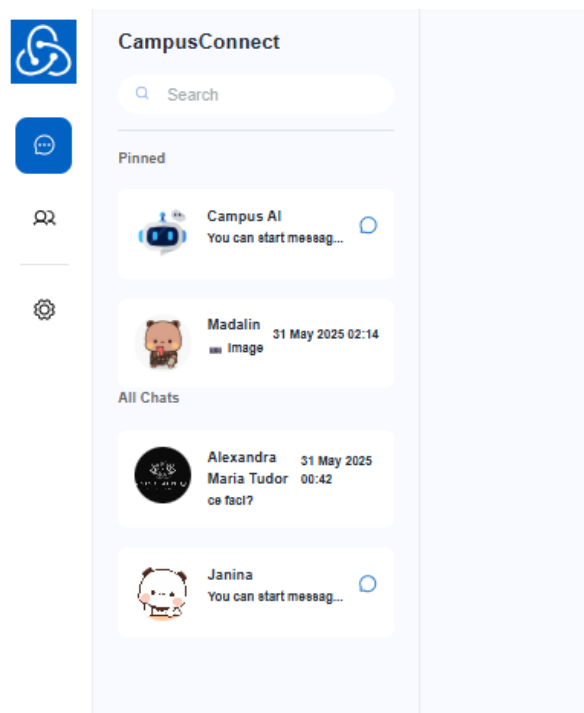
    return chats;
}

```

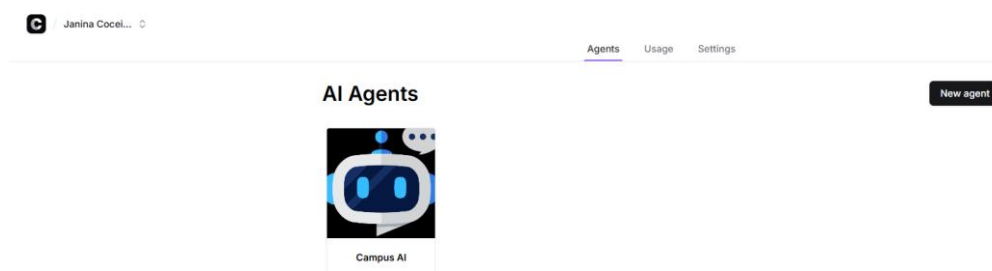
Comunicarea are loc prin endpoint-ul public `/ws`, cu fallback SockJS pentru suport extins, iar mesajele client sunt direcționate către metodele de backend prin prefixul `/app`. Răspunsurile sunt direcționate folosind un broker simplu de mesaje care suportă canale și utilizatori specifici.

5.5 Lista conversațiilor

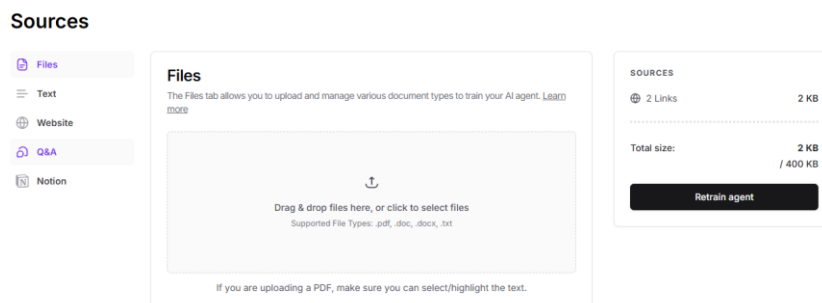
După autentificare, utilizatorul este redirecționat automat către pagina principală a aplicației, unde este afișată lista conversațiilor directe. Această secțiune este organizată în două categorii: conversații „pinned”, cele marcate manual de utilizator pentru a fi afișate prioritar și conversații standard, restul interacțiunilor directe, ordonate cronologic descrescător în funcție de ultima activitate.



În partea superioară a listei este disponibilă conversația implicită cu agentul conversațional AI (Campus AI). Această conversație este tratată ca un canal special, unde utilizatorul poate adresa întrebări direct agentului. Folosind varianta free a platformei Chatbase, am un număr restrâns de întrebări (aproximativ 100/lună).



Pentru a-l antrena pot încărca până la 10 documente care să aibă dimensiunea totală de 400 KB.



Acesta este reprezentat prin intermediul componentei ChatElement care afișează avatarul partenerului de conversație, numele partenerului de conversație, un mesaj și un simbol relevant.

```
<motion.div variants={varHover( scale: 1.05)} whileHover="hover">
  <ChatElement name={"Campus AI"} id={"ai_chat"} img={Robot}></ChatElement>
</motion.div>
```

Pentru a obține conversațiile utilizatorului se trimite o cerere WebSocket (/app/chats).

```
useEffect( effect: () => {

  if (!isConnected) return;
  socket.emit("/app/chats", "Bearer " + token)

  socket.on(`/user/${user_id}/chat/chats-response`, (data) => {
    dispatch(FetchDirectConversations(data));
  });

}, deps: [isConnected, dispatch]);
```

Pe backend, această cerere este interceptată de metoda WebSocket mapată astfel:

```
± janina280
@Transactional
@MessageMapping("/chats")
public List<ChatDto> findAllChatByUserHandle(@RequestHeader("Authorization") String jwt) throws UserException {
    var currentUserId = tokenProvider.getUserIdFromToken(jwt.substring( beginIndex: 7));

    var chats = chatService.findAllChatByUserId(currentUserId);

    simpMessagingTemplate.convertAndSendToUser(currentUserId.toString(), destination: "/chat/chats-response", chats);

    return chats;
}
```

În interiorul metodei, are loc următorul proces:

- Se extrage userId-ul utilizatorului curent din tokenul JWT, folosind componenta tokenProvider.
- Se apelează serviciul chatService.findAllChatByUserId(...) pentru a prelua toate conversațiile directe din baza de date în care este implicat acel utilizator.
- Rezultatul, o listă de obiecte ChatDto, este trimis înapoi către frontend printr-un canal dedicat acelui utilizator: /user/{userId}/chat/chats-response.

Pe frontend, aplicația ascultă acest canal și, în momentul în care primește datele, le salvează în Redux și actualizează interfața cu lista conversațiilor.

Pentru inițierea unei noi conversații directe, utilizatorul poate introduce numele unei persoane în bara de căutare. Pe măsură ce se tastează, aplicația filtrează utilizatorii disponibili și returnează rezultatele în timp real cu ajutorul metodei handleSearch.

```
const handleSearch = (value) => { Show usages
  const trimmedValue = value.trim();
  setQueryChat(trimmedValue);

  if (trimmedValue !== "") {
    dispatch(searchUser( data: {keyword: trimmedValue}));
  }
};
```

Conversațiile sunt afișate cu ajutorul componentei UserElement care afișează avatarul partenerului de conversație și numele acestuia.

Selectarea unui utilizator din rezultate declanșează crearea conversației, operație efectuată prin emiterea unui eveniment WebSocket către backend (/app/chat-create).

```

@janina280
@Transactional
@MessageMapping("/chat-create")
public ChatDto createChatHandle(@RequestBody SingleChatRequest singleChatRequest) throws UserException {
    var currentUserId = tokenProvider.getUserIdFromToken(singleChatRequest.getJwtString().substring(beginIndex: 7));

    var chat = chatService.createChat(currentUserId, singleChatRequest.getUserId());

    for (var c : chat.getUsers()) {
        simpMessagingTemplate.convertAndSendToUser(c.getId().toString(), destination: "/chat/chat-create-response", chat);
    }

    return chat;
}

```

Dacă între cei doi utilizatori există deja o conversație, aceasta va fi reutilizată, evitând duplicarea canalelor de comunicare.

```

const existingChat = conversations.find((chat) => chat.user_id === userId);

if (existingChat) {
    setSelectedChat(existingChat);
    return;
}

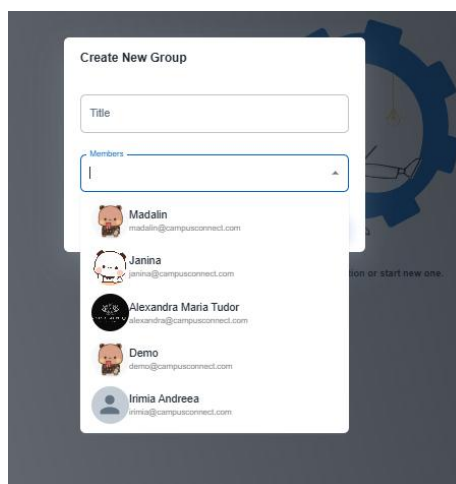
```

Mecanismul de obținere a conversațiilor de grup funcționează pe același principiu ca și în cazul conversațiilor directe.

Frontend-ul trimite o cerere WebSocket către un endpoint specific (*/app/groups*), împreună cu tokenul JWT pentru identificarea utilizatorului. Pe backend, cererea este interceptată de o metodă marcată cu *@MessageMapping*, care extrage *userId* din token, apelează serviciul care returnează toate conversațiile de grup în care este implicat utilizatorul și trimite rezultatul către frontend pe un canal dedicat utilizatorului (*/user/{user_id}/group/groups-response*).

Astfel, utilizatorul primește în timp real lista actualizată a grupurilor din care face parte, exact cum se întâmplă și în cazul conversațiilor directe.

5.6 Sistemul de creare și administrare a unui grup



Când utilizatorul apasă pe butonul de „Create New Group”, se deschide un Dialog modal ce încarcă componenta *CreateGroupForm*, unde utilizatorul introduce un titlu pentru grup și selectează cel

puțin doi membri dintr-o listă autocomplete populată cu toți utilizatorii (prin `FetchAllUsers` din store-ul global).

Funcționalitatea este disponibilă doar pentru utilizatorii cu roluri de `ROLE_TUTOR` sau `ROLE_ADMIN`. Verificarea se face la nivel de interfață, iar opțiunea de creare este afișată doar dacă utilizatorul are unul dintre aceste roluri:

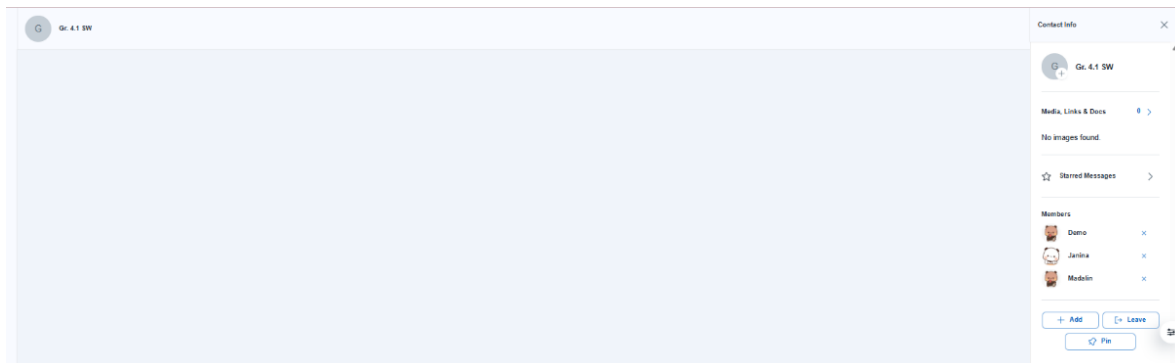
```
const {roles = []} = useSelector(selector: (state) => state.auth);

const isAdmin = roles.includes("ROLE_ADMIN");
const isTutor = roles.includes("ROLE_TUTOR");
const canCreateGroup = isAdmin || isTutor;
```

Dar și în backend în componenta `ChatService` are loc verificarea rolului:

```
1 usage  janina280 *
public ChatDto createGroup(GroupChatRequest req, User reqUser) throws UserException {
    boolean isTutorOrAdmin = reqUser.getRoles().stream()
        .anyMatch(role -> role.getName() == Role.RoleName.ROLE_TUTOR || role.getName() == Role.RoleName.ROLE_ADMIN);
    if (!isTutorOrAdmin) {
        throw new UserException("You do not have permission to create a group. Only tutors or admins can create groups.");
    }
}
```

Doar utilizatorul care a creat grupul poate seta imaginea de profil a acestuia, default este afișat un avatar creat cu ajutorul metodei `createAvatar` întâlnită și la profilul utilizatorului.



De asemenea, doar utilizatorul care a creat grupul poate adăuga un nou utilizator în grup. Acesta are posibilitatea de a scoate un utilizator din grup și de a seta un grup ca fiind “Pin” pentru toți utilizatorii.

5.7 Sistemul de trimitere a unui mesaj

Sistemul de trimitere a unui mesaj în cadrul aplicației `CampusConnect` este implementat utilizând o arhitectură bazată pe `WebSocket`, integrată cu `Redux` pentru gestionarea stării. Funcționalitatea este cuprinsă în mai multe componente și hook-uri, dintre care cele mai relevante sunt `ChatFooter`, `WebSocketContext`, `conversation.js` (slice `Redux`), și componenta principală `ChatComponent`.

5.7.1 Structura general a interfeței de conversație

Componenta principală care este apelată în momentul selectării unui chat este ChatComponent. Aceasta conține trei zone funcționale:

- ChatHeader care afișează informații despre utilizatorul cu care am conversația
- Zona de mesaje (Conversation) care afișează lista de mesaje corespunzătoare conversației curente.
- ChatFooter care conține formularul pentru trimiterea mesajelor.

```
{room_id !== "ai_chat" ? <ChatHeader/> : null}
<Box
  ref={messageListRef}
  width={"100%"}
  sx={{
    position: "relative",
    flexGrow: 1,
    overflow: "auto",

    backgroundColor:
      theme.palette.mode === "light"
        ? "#F0F4FA"
        : theme.palette.background,

    boxShadow: "0px 0px 2px rgba(0, 0, 0, 0.25)",
  }}
>
  <SimpleBarStyle timeout={500} clickOnTrack={false}>
    <Conversation menu={true} isMobile={isMobile}/>
  </SimpleBarStyle>
</Box>

{ /* */ }
{room_id !== "ai_chat" ? <ChatFooter/> : null}
</Stack>
```

5.7.2 Componenta ChatFooter

Componenta ChatFooter este responsabilă pentru introducerea și trimiterea mesajelor dintr-o conversație. Aceasta oferă funcționalități avansate precum:

- trimiterea de mesaje text și linkuri,
- integrarea cu o bibliotecă de emoji (emoji-mart),
- trimiterea fișierelor (imagini și documente) prin selector sau drag & drop,
- integrarea cu WebSocket pentru comunicare în timp real.

Componenta este împărțită în două subcomponente:

- ChatInput: gestionează inputul de text și butoanele auxiliare (emoji, atașamente);
- Footer: orchestrează trimiterea mesajului, adăugarea emoji-urilor și încărcarea fișierelor.

Apăsarea butonului de trimitere (pictograma PaperPlaneTilt) declanșează funcția handleSendMessage. Aceasta construiește un payload care este transmis către backend prin WebSocket:

```
const messageData = {
  jwtString: "Bearer " + token,
  content: value,
  chatId: room_id,
  type: containsUrl(value) ? "link" : "text",
};

socket.emit("/app/send-message", messageData);
set_value(value: "");
```

Funcția containsUrl verifică dacă mesajul conține un link:


```
function containsUrl(text) { Show usages 1 janina280
  const urlRegex = /(https?:\/\/[^\s]+)/g;
  return urlRegex.test(text);
}
```

În cazul în care se dorește trimiterea de mesaje tip fișier acestea pot fi trimise în două moduri:

- Prin selectorul de fișiere din UI (<input type="file" />);
- Prin drag & drop direct peste zona de input.

În ambele cazuri, fișierul este transmis către endpointul *api/message/upload* printr-o cerere POST de tip multipart/form-data:

```
const handleFileUpload = (file) => { Show usages 1 janina280
  const reader = new FileReader();
  reader.onload = () => {
    const formData = new FormData();
    formData.append('name', file);
    axios
      .post(
        'api/message/upload',
        formData,
        {
          headers: {
            Authorization: 'Bearer ${token}',
            'content-type': 'multipart/form-data',
          },
        },
      )
      .then(function (response) {
        const messageData = {
          jwtString: 'Bearer ' + token,
          chatId: room_id,
          type: file.type.startsWith('image/') ? 'image' : 'document',
          media: response.data
        };
        socket.emit('/app/send-message', messageData);
      })
      .catch((error) => {
        console.log(error);
        dispatch(showSnackbar({ severity: 'error', message: error.response.data.message }));
      });
  };
  reader.readAsDataURL(file);
};
```

După încărcare, backend-ul returnează un identificator sau link pentru fișier, care este inclus în payload-ul WebSocket.

Backend-ul aplicației oferă suport complet pentru trimiterea mesajelor în timp real, inclusiv mesaje tip text, linkuri, emoji, fișiere (imagini, documente), folosind WebSocket pentru comunicare bidirecțională instantă între client și server.

Mesajele trimise din frontend sunt direcționate către un endpoint WebSocket configurat în Spring Boot, de forma:

```
1 janina280
@Transactional
@MessageMapping("/send-message")
public void sendMessageHandler(@RequestBody MessageRequest req) throws UserException, ChatException {
  User user = userService.findUserProfile(req.getJwtString());

  var message = messageService.sendMessage(req, user);
  var response = messageMapper.toMessageResponse(message);

  for (var m : message.getChat().getUsers()) {
    simpMessagingTemplate.convertAndSendToUser(
      m.getId().toString(),
      destination: "/message/message-send-response",
      response
    );
  }
}
```

@Transactional: Asigură că salvarea mesajului și transmiterea acestuia se face într-o tranzacție atomică.

@PostMapping("/send-message"): Mapează mesajele WebSocket primite pe ruta /app/send-message (așa cum este apelată din frontend) către această metodă.

@RequestBody MessageRequest req: Mesajul este trimis în format JSON și mapat automat la un obiect de tip MessageRequest.

Utilizatorul expeditor este identificat pe baza tokenului JWT transmis în câmpul jwtString, apoi mesajul este procesat și salvat în baza de date prin serviciul messageService, care preia conținutul mesajului și asociază mesajul cu utilizatorul și conversația(chat). Mesajul este apoi convertit într-un DTO(Data Transfer Object) folosind messageMapper:

var response = messageMapper.toMessageResponse(message);

Mesajul este apoi transmis tuturor participanților la conversație folosind SimpleMessagingTemplate.

Fiecare utilizator primește mesajul printr-un topic WebSocket personalizat (/user/{userId}/message/message-send-response), ceea ce permite clientului să primească doar mesajele relevante pentru el.

Această arhitectură permite trimiterea rapidă și sigură a mesajelor, asigurând scalabilitate și izolare per utilizator, cu un model reactiv bazat pe evenimente.

5.8 Sistemul de afișare a unui mesaj

În funcție de cine trimite mesajul și de tipul mesajului acestea sunt afișate în componenta Conversation.

Mesajele primite sunt interceptate prin contextul WebSocketContext, care ascultă topicul personalizat pentru utilizator:

```
const messageSendEvent = `/user/${user_id}/message/message-send-response`;
socket.on(messageSendEvent, handleMessageSendResponse);
```

Funcția handleMessageSendResponse se declanșează la primirea unui nou mesaj:

```

54     const handleMessageSendResponse = (message) :void => { Show usages
55         const isGroup :boolean = message.group === true;
56         const payload :{...} = {
57             id: message.id,
58             type: "msg",
59             subtype: message.type,
60             message: message.content,
61             outgoing: message.senderId === user_id,
62             senderId: message.senderId,
63             sender: message.sender,
64             state: message.state,
65             createdAt: message.createdAt,
66             media: message.media,
67             formattedTime: message.formattedTime,
68             chatId: message.chatId,
69         };
70
71         if (isGroup) {
72             dispatch(AddDirectMessageGroup(payload));
73         } else if (current_conversationRef.current?.id === message.chatId) {
74             dispatch(AddDirectMessage(payload));
75         }
76     };

```

Mesajele sunt mapate și afișate în ordinea cronologică folosind componente dedicate pentru fiecare tip (text, link, image, document):

```

{current_messages?.map((el, idx) => {
    switch (el.type) {
        case "msg":
            switch (el.subtype) {
                case "image":
                    return (
                        // Media Message
                        <MediaMsg el={el} menu={menu}/>
                    );

                case "document":
                    return (
                        // Doc Message
                        <DocMsg el={el} menu={menu}/>
                    );
            }
        default:
            // Text Message
            return (
                <TextMsg el={el} menu={menu}/>
            );
    }
})}

```

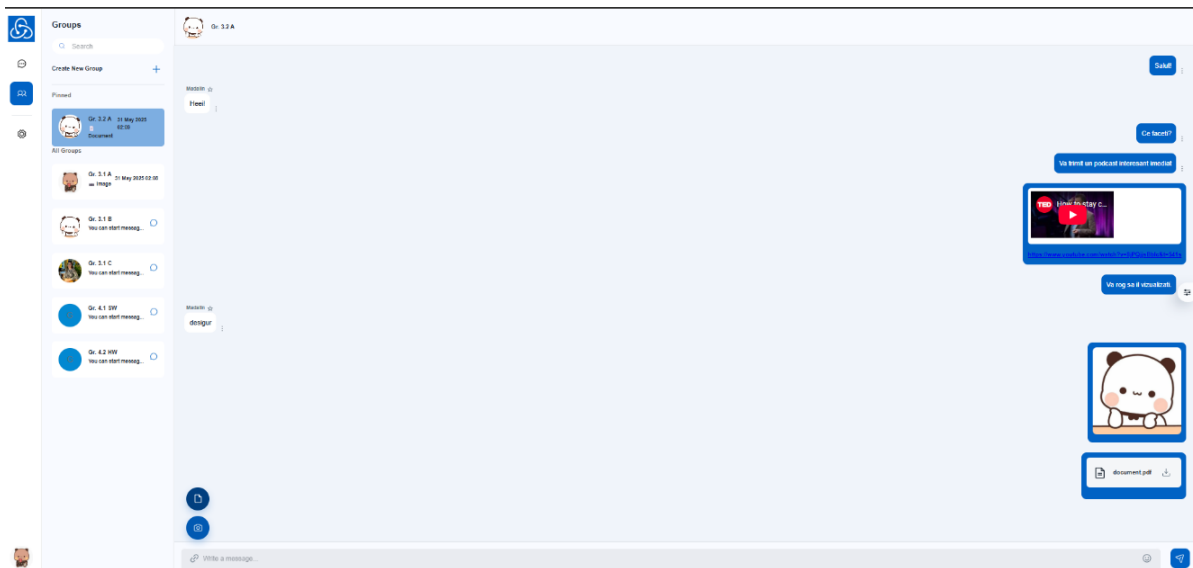
Pentru mesajele de tip text exista posibilitatea de a marca un mesaj ca fiind „star”, acesta fiind afișat ulterior în secțiunea de Contact Info.

```

27     const handleStarMessage = () :void => { Show usages
28         dispatch(starMessage(el.id));
29         dispatch(showSnackBar( {severity: "success",
30             message: "Message starred!",
31         })))
32     };
33     handleClose();
34 };
35
36     const Message_options :{title: string, action: handl...} = [
37         {
38             title: "Star message", action: handleStarMessage
39         },
40     ];

```

Mesajele de tip imagine sunt afișate în conversație, iar link-urile au posibilitatea de a fi vizualizate în interiorul conversației cu ajutorul librăriei Microlink.



Atât imaginile cât și documentele pot fi descărcate apelând metoda `handleDownload`:

```
const handleDownload = () => {
  const link : HTMLAnchorElement = document.createElement( 'a' );
  link.href = image;
  link.download = fileName;
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
};
```

5.9 Sistemul de afișare informații (Contact vs. ContactGroup)

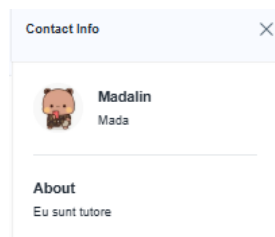
Afișarea informațiilor detaliate despre o conversație, fie aceasta de tip individual sau de grup, este realizată prin două componente React specializate: `Contact` și `ContactGroup`. Aceste componente sunt parte integrantă a panoului lateral (sidebar) și oferă utilizatorilor acces rapid la date relevante despre conversație, conținutul partajat și opțiunile administrative (acolo unde este cazul).

5.9.1 Componenta Contact

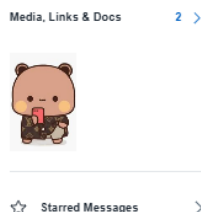
Componenta `Contact` este utilizată pentru a afișa detalii despre o conversație de tip direct (1-la-1), între doi utilizatori. Această componentă extrage datele specifice utilizatorului cu care se poartă conversația și le prezintă într-un format organizat și accesibil. Informațiile sunt redată într-o interfață intuitivă, cu accent pe claritate și accesibilitate.

Funcționalități principale:

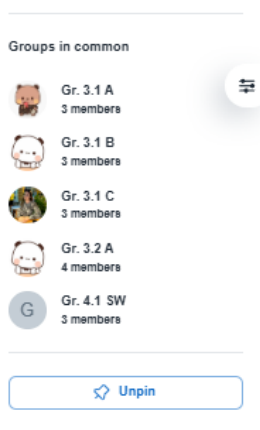
- Afișarea avatarului și numelui complet al utilizatorului cu care se poartă conversația, împreună cu porecla acestuia și descrierea.



- Acces rapid la conținutul media partajat în cadrul conversației (imagini, documente, linkuri).



- Afișarea mesajelor marcate (star), pentru a evidenția conținut important.
- Afișarea grupurilor comune dintre cei doi utilizatori.

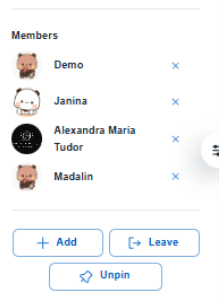


- Posibilitatea de a marca o conversație ca fiind “Pin” pentru a fi afișată întotdeauna în zona dedicată acestor tip de conversații.

Componenta este utilizată exclusiv pentru conversațiile individuale și nu include funcționalități legate de administrarea utilizatorilor, deoarece acest tip de conversație implică doar două entități și nu necesită o structură de management a membrilor.

5.9.2 Componenta ContactGroup

Componenta ContactGroup este responsabilă pentru gestionarea interfeței conversațiilor de grup. Aceasta extinde funcționalitatea componentei Contact, adăugând capabilități suplimentare care sunt relevante în contextul unui grup colaborativ. Rolul principal al acestei componente este de a furniza o interfață completă pentru vizualizarea informațiilor despre grup și pentru efectuarea acțiunilor administrative (acolo unde utilizatorul are permisiunile necesare).



Funcționalități extinse:

- Afișarea listei membrilor grupului, împreună cu informații relevante (nume, imagine).
- Posibilitatea de a adăuga sau elimina membri, funcționalitate disponibilă exclusiv pentru utilizatorii cu rol de tutor sau administrator al grupului.
- Posibilitatea de a părăsi grupul, indiferent de rolul utilizatorului.
- Funcționalități de tip pin/unpin pentru conversație, care permit fixarea acestora în lista principală de conversații.
- Actualizarea imaginii de profil a grupului, acțiune permisă doar utilizatorilor autorizați.

Această componentă este esențială în gestionarea colaborării de grup și permite o administrare flexibilă și eficientă a conversațiilor multiple, în funcție de nivelul de acces al utilizatorului.

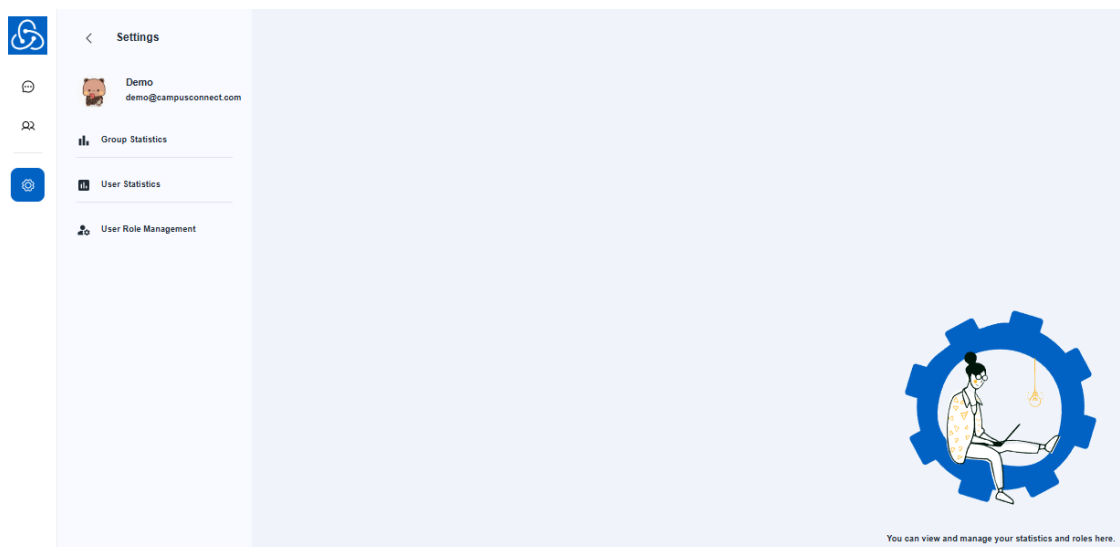
5.9.3 Integrarea componentelor în aplicație

Atât Contact, cât și ContactGroup sunt randate dintr-un container principal (GeneralApp) în funcție de tipul conversației curente:

```
switch (type) {
  case "CONTACT":
    return chat_type === "individual" ? (
      <Contact/>
    ) : (
      <ContactGroup/>
    );
}
```

5.10 Sistemul de management

În cadrul aplicației, componenta **Settings** funcționează ca un panou centralizat de administrare, oferind utilizatorilor cu rol de administrator acces la funcționalități esențiale pentru monitorizarea și gestionarea platformei. Când un administrator accesează această pagină, el este întâmpinat de o interfață curată și organizată în două părți distincte.








Partea din stânga este dedicată navigării și afișării profilului utilizatorului curent. Aici, pe un fundal deschis, apare avatarul personalizat generat din imaginea sau inițialele utilizatorului, însoțit de numele și adresa de email, ceea ce oferă un sentiment de identitate clară și imediată. Sub aceste detalii, se află o listă interactivă de opțiuni disponibile, fiecare reprezentată printr-un icon intuitiv și un titlu descriptiv.

Această listă este construită dinamic, ținând cont de rolul utilizatorului. În cazul unui administrator, opțiunile afișate includ statistici generale ale grupurilor, statistici detaliate pentru utilizatori și gestionarea rolurilor utilizatorilor. Printr-un sistem de stări locale, la apăsarea fiecărei opțiuni se activează un panou de detalii corespunzător, iar conținutul specific este încărcat în partea dreaptă a ecranului.

Partea dreaptă a interfeței este un spațiu dedicat afișării detaliilor și instrumentelor de administrare. În mod implicit, când niciun panou nu este deschis, utilizatorul este întâmpinat cu un mesaj sugestiv și o ilustrație care indică disponibilitatea funcționalităților de monitorizare și gestionare. Când administratorul alege să vizualizeze statistici sau să gestioneze rolurile, acest spațiu se actualizează dinamic pentru a afișa componentele specifice: Statistics, UserStatistics sau UserRoleManagement.

User Role Management

Name	Email	Role
 Madalin	madalin@campusconnect.com	tutor ▾
 Janina	janina@campusconnect.com	user ▾
 Alexandra Maria Tudor	alexandra@campusconnect.com	user ▾
 Demo	demo@campusconnect.com	admin ▾
 Irimia Andreea	irimia@campusconnect.com	user ▾

Close

Pentru o experiență fluidă și plăcută, elementele interactive beneficiază de animații subtile de hover, realizate cu ajutorul bibliotecii Framer Motion, care conferă feedback vizual și încurajează explorarea opțiunilor. De asemenea, componentele și stările sunt bine izolate, iar apelurile pentru date sunt gestionate prin Redux, asigurând o sincronizare corectă cu starea globală a aplicației.

Astfel, sistemul de management din componenta Settings oferă un echilibru între accesibilitate, control și vizibilitate, permițând administratorilor să supravegheze performanța grupurilor, să analizeze activitatea utilizatorilor și să modifice drepturile de acces într-un mod simplu și eficient.

6 INSTALAREA APLICAȚIEI

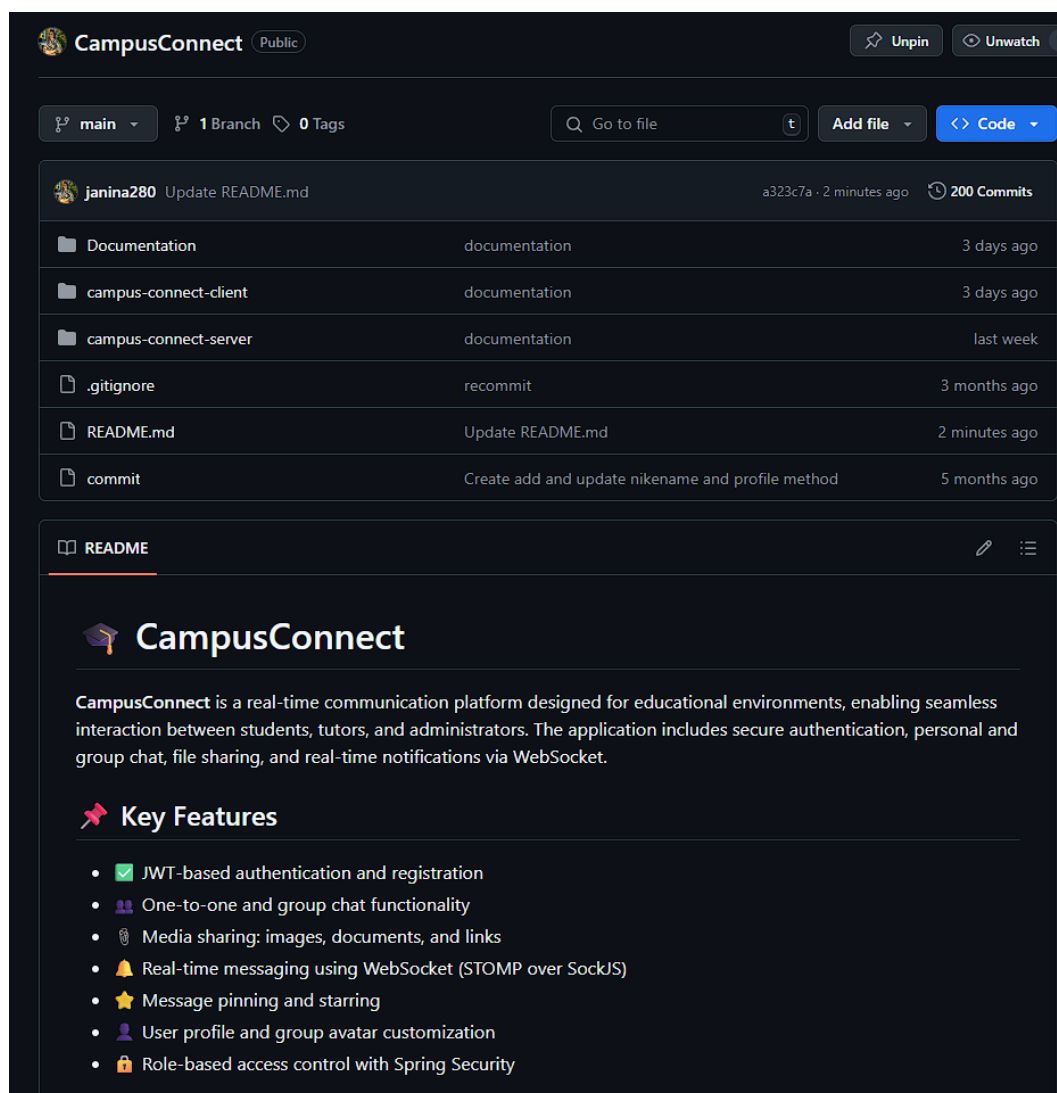
Instalarea aplicației CampusConnect implică configurarea mediului local pentru rularea atât a frontend-ului, cât și a backend-ului, precum și conectarea acestora la o bază de date PostgreSQL.

Aplicația este structurată pe două componente principale:

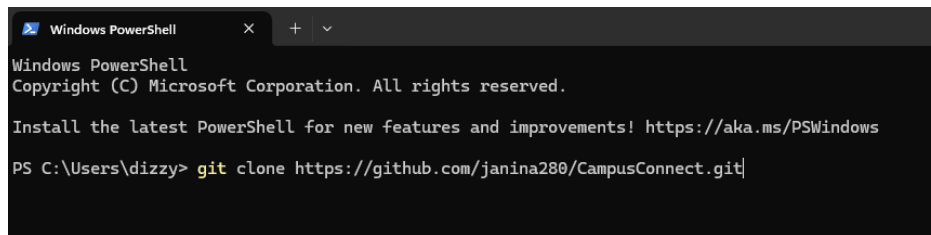
- Frontend: realizat cu React.js, care asigură interfața utilizator.
- Backend: dezvoltat cu Spring Boot, care gestionează logica aplicației și accesul la date, utilizând PostgreSQL ca sistem de gestiune a bazelor de date relaționale.

6.1 Clonarea codului sursă

Pentru a obține codul sursă al aplicației, se accesează depozitul oficial GitHub, disponibil la adresa: <https://github.com/janina280/CampusConnect>.



Se copiază URL-ul de clonare (format HTTPS) disponibil în secțiunea **Code**, iar în terminal sau printr-un client Git se execută comanda:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\dizzy> git clone https://github.com/janina280/CampusConnect.git
```

Aceasta va descărca întregul cod al aplicației în directorul local.

6.2 Configurarea bazei de date PostgreSQL

Aplicația necesită o bază de date PostgreSQL configurată și funcțională. Pentru aceasta, se recomandă:

- Instalarea PostgreSQL pe mașina locală (dacă nu este deja instalat).
- Crearea unei baze de date dedicate aplicației, de exemplu campusconnectdb.
- Configurarea utilizatorului și parolei de acces în PostgreSQL.

Ulterior, fișierul de configurare al backend-ului Spring Boot (application.yml) trebuie actualizat cu datele conexiunii:

- URL-ul bazei de date (ex: jdbc:postgresql://localhost:5432/campusconnectdb)
- Numele utilizatorului și parola corespunzătoare.

6.3 Instalarea și rularea backend-ului

După clonarea codului, se navighează în directorul backend-ului campus-connect-server.

Pentru rularea backend-ului sunt necesare:

- Java JDK instalat.
- Un build tool (Maven), conform configurației proiectului.

6.4 Instalarea și rularea frontend-ului

Frontend-ul este realizat cu React.js și se află în directorul dedicat clientului. Pentru a instala dependențele și a porni interfața se execută: `npm install` și `npm start`.

Aceste comenzi vor instala toate pachetele necesare și vor porni serverul de dezvoltare React, de regulă accesibil pe `http://localhost:3000`.

Odată ce backend-ul Spring Boot și frontend-ul React sunt pornite, se poate accesa în browser pagina aplicației la adresa: `http://localhost:3000`.

7 CONCLUZII

Crearea aplicației CampusConnect a reprezentat o experiență complexă și valoroasă, contribuind semnificativ la consolidarea noțiunilor teoretice și la dezvoltarea competențelor practice în domeniul dezvoltării software. Una dintre cele mai importante provocări întâlnite a fost implementarea sistemului de comunicare în timp real prin WebSocket, care a implicat gestionarea conexiunii persistente, sincronizarea mesajelor între componentele frontend și backend, precum și menținerea unei experiențe fluide pentru utilizator.

Depășirea acestei dificultăți s-a realizat prin consultarea documentației oficiale, studierea resurselor specializate și prin suportul acordat de către coordonator. De asemenea, implementarea iterativă și testarea constantă a funcționalităților au fost esențiale pentru stabilizarea și perfecționarea aplicației.

Totodată, această experiență a conturat o perspectivă mai clară asupra modului în care tehnologiile moderne pot fi orchestrate pentru a genera soluții utile și relevante într-un context real. Finalizarea proiectului confirmă nu doar atingerea obiectivelor propuse, ci și capacitatea de a gestiona complexitatea unui sistem software end-to-end, de la idee la produs funcțional.

SFÂRȘIT.

8 BIBLIOGRAFIE

- Abramov, D. (2019). *Introducing Hooks*. Preluat de pe React Blog: <https://legacy.reactjs.org/docs/hooks-intro.html>
- Abramov, D. ș. (2024). *A JS library for predictable and maintainable global state management*. Preluat de pe Redux: <https://redux.js.org/>
- Alex Banks, E. P. (2016). *Learning React: Functional Web Development with React and Redux*. O'Reilly Media.
- Atori, N. E. (2021). *Introducing Framer Motion*. Preluat de pe <https://www.smashingmagazine.com/2020/10/introduction-to-framer-motion/>
- Banks, A. &. (2020). *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media.
- Banks, R. &. (2023). *Modern Full-Stack Development with React, Redux, and React Router*. O'Reilly Media.
- Bauer, C. &. (2020). *Java Persistence with Hibernate (2nd ed.)*. . Manning Publications.
- Bauer, C. (2016). *Hibernate Validator Reference Guide*. Preluat de pe <https://hibernate.org/validator/>
- Booch, G. R. (2005). *The Unified Modeling Language User Guide (2nd ed.)*. Addison-Wesley.
- Chacon, S. &. (2014). *Pro Git (2nd ed.)*. Apress. Preluat de pe <https://git-scm.com/book/en/v2>
- Codd, E. F. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM.
- Contributors, A. (2024). *Axios – Promise based HTTP client for the browser and node.js*. Preluat de pe <https://axios-http.com>
- DevTools, R. (2024). *Redux DevTools Extension Documentation*. Preluat de pe <https://github.com/reduxjs/redux-devtools>
- Documentation, S. (fără an). *STOMP*. Preluat de pe GitHub: <https://stomp-js.github.io/>
- Documentation, S. (fără an). *WebSocket Support*. Preluat de pe <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#websocket>
- EE, J. (2023). *Jakarta Bean Validation Specification*. Preluat de pe <https://jakarta.ee/specifications/bean-validation>
- Elmasri, R. &. (2017). *Fundamentals of Database Systems (7th ed.)*. Pearson.
- Form, R. H. (2024). *Performant, flexible and extensible forms with easy-to-use validation*. Preluat de pe React Hook Form: <https://react-hook-form.com/>
- Framer. (2024). *Framer Motion – Declarative animations for React*. Preluat de pe <https://www.framer.com/motion/>
- Fulton, A. (2022). *Mastering React Router 6*. Packt Publishing.
- GeeksforGeeks. (2024). *What is an API?* Preluat de pe GeeksforGeeks: <https://www.geeksforgeeks.org/what-is-an-api/>

Git, P. (2019). Preluat de pe Git documentation: <https://git-scm.com/docs>

Group, P. G. (2024). *PostgreSQL 16 Documentation*. Preluat de pe <https://www.postgresql.org/docs/>

Group, P. G. (2024). *PostgreSQL Documentation*. Preluat de pe PostgreSQL: <https://www.postgresql.org/docs/>

Inc., F. (2023). *Quick Start*. Preluat de pe React: <https://react.dev/learn>

Inc., F. (2023). *The library for web and native user interfaces*. Preluat de pe React: <https://react.dev/>

Inc., G. (2023). *Material Design*. Preluat de pe <https://m3.material.io/>

Inc., P. (2025). *Postman – The Collaboration Platform for API Development*. Preluat de pe <https://www.postman.com/>

Inc., R. S. (2024). *React Router – Declarative routing* .

JetBrains. (2024). *WebStorm – The Smartest JavaScript IDE*. Preluat de pe <https://www.jetbrains.com/webstorm/>

JetBrains. (2025). *IntelliJ IDEA – The Java IDE for Professional Developers*. Preluat de pe <https://www.jetbrains.com/idea/>

Jones, M. B. (2015). *JSON Web Token (JWT) – RFC 7519*. Preluat de pe <https://datatracker.ietf.org/doc/html/rfc7519>

Jquense. (2023). *Yup – Dead simple object schema validation*. Preluat de pe Yup: <https://github.com/jquense/yup>

Kurniawan, B. (2022). *Spring Boot for Beginners: A Hands-On Guide to Building Web Applications Using Spring Boot and Spring MVC*.

Larsen, O. &. (2021). *Mastering Material-UI*. Packt Publishing.

Long, J. (2021). *Spring Boot Reference Documentation*. Preluat de pe <https://docs.spring.io/spring-boot/documentation.html>

Mak, J. (2024). *WebSocket Support in Spring Framework*. Preluat de pe <https://docs.spring.io/spring-framework/reference/web/websocket.html>

Meier, R. &. (2022). *Spring Data JPA – Core concepts*. Preluat de pe <https://docs.spring.io/spring-data/jpa/reference/repositories/core-concepts.html>

Microlink. (2024). *Microlink – Turn websites into data*. Preluat de pe <https://microlink.io>

Mokito, S. &. (2021). *Efficient Java Development with Lombok and Spring Boot*. Apress.

Ng, W. (2017). *Pro React 16*. Apress.

Project, L. (2024). *Project Lombok – Java boilerplate code reduction tool*. Preluat de pe <https://projectlombok.org/>

Rahman, B. (2021). *Spring Security in Action*.

Recharts. (fără an). *Recharts - Documentation*. Preluat de pe <https://recharts.org/>

Schmidt, M. (2021). *Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs. 3rd* . Apress.

Seacord, R. (2021). *Effective Java Validation*. Addison-Wesley.

SockJs. (2024). *SockJs*. Preluat de pe GitHub: <https://github.com/sockjs/sockjs-client>

- Team, M. (2024). *Material UI – React components for faster and easier web development*. Preluat de pe <https://mui.com/>
- Team, p. (2024). *pgAdmin 4 Documentation*. Preluat de pe <https://www.pgadmin.org/docs/>
- Team, P. D. (2025). *pgAdmin – The most popular and feature rich Open Source administration and development platform for PostgreSQL*. Preluat de pe <https://www.pgadmin.org/>
- Walls, C. (2022). *Spring Boot in Action. 2nd ed.* . NY: Manning Publications.
- Zhang, X. (2022). *Building Web APIs with Spring Boot and Axios*. Packt Publishing.