

**ROMÂNIA**  
**MINISTERUL APĂRĂRII NAȚIONALE**  
**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE  
CIBERNETICĂ**

**Specializare: Calculatoare și Sisteme Informatice pentru Apărare și  
Securitate Națională**



**APLICAȚIE DE TIP COLABORATIV PROTEJATĂ ÎN CLOUD**

**CONDUCĂTOR ȘTIINȚIFIC:  
Col. prof. univ. dr. ing. Mihai TOGAN**

**ABSOLVENT:  
Stud. plt. Ionuț-Alexandru PAVEL**

Conține \_\_\_\_\_ file  
Inventariat sub nr. \_\_\_\_\_  
Poziția din indicator: \_\_\_\_\_  
Termen de păstrare: \_\_\_\_\_

**BUCUREȘTI  
2022**

NECLASIFICAT

NECLASIFICAT

NECLASIFICAT  
2 din 72

NECLASIFICAT

NECLASIFICAT

## ABSTRACT

Collaboration tools involve multiple types of software or services that allow people to work together. Collaboration tools can route work through a process, distribute pieces and tasks to involved parties, and help to coordinate activities.

This paper presents the implementation of the web-based collaborative real-time application using ReactJs and NodeJs frameworks. The application provides a secure, controlled environment for real-time document review, coauthoring and redaction. Also, it allows the creation of private or group conversation between users and allows text message transfer. Documents can be saved in encrypted format using server storage and personal data is stored using MySql relational database. The application also ensures communication between users by creating private or public chat channels. Communication between users is secure and allows the exchange of text messages or the transmission of media files. Another communication method provided by the application is video calling, allowing users to interact directly with each other. The software solution also provides the functionality for users with an admin role to monitor the activity of other users and restrict their access to the application.

The final goal of the project is the development and operationalization of the presented software solution based on the studied concepts. An important aspect to take into account is that the application at the moment is not a final product to be put into use, but it is a demo version that puts the theoretical concepts into practice.

NECLASIFICAT

NECLASIFICAT  
6 din 72

## REZUMAT

Această lucrare prezintă o aplicație software de tip colaborativ ce are ca funcționalități schimbul de mesaje text sau de fișiere multimedia, editarea de text în timp real, stocarea de fișiere la distanță și relizarea de apeluri video și audio între participanți.

Capitolul 1 prezintă importanța dezvoltării unui astfel de sistem colaborativ propriu în locul alegerii unei soluții deja existente.

Capitolul 2 descrie soluții software similare și prezintă tehnologiile specifice folosite de acestea la momentul actual. Printre aceste tehnologii se află algoritmi folosiți în cadrul editării de text colaborative și metode de comunicație utilizate în cadrul acestor aplicații.

În Capitolul 3 sunt prezentate bibliotecile și API-urile folosite pentru dezvoltarea sistemului. Acest capitol fundamentează bazele implementării software alături de Capitolul 4, unde sunt prezintate cerințele aplicației, arhitectura și modul acestora de funcționare folosind diagrame de componente.

Capitolul 5 prezintă implementarea software a principalelor module din aplicație, explicând relațiile dintre componente și modul prin care interacționează la nivelul sistemului.

Capitolele 6 și 7 prezintă testarea sistemului prin utilizarea de diagrame UML și prin prezentarea unui plan de testare configurat pe baza cerințelor funcționale.

NECLASIFICAT

# Cuprins

LISTĂ DE ABREVIERI .....	11
TABELĂ FIGURI .....	12
1. Introducere.....	13
1.1 Importanța temei.....	13
1.2 Scopul lucrării .....	13
1.3 Utilitatea sistemului.....	13
2 Stadiul actual .....	15
2.1 Software colaborativ.....	15
2.2 Sisteme similare .....	15
2.2.1 Microsoft Teams.....	15
2.2.2 Slack .....	16
2.2.3 Discord .....	16
2.2.4 Telegram.....	17
2.2.5 Google Docs .....	18
2.3 Principii de funcționare .....	19
2.3.1 Editarea colaborativă .....	19
2.3.2 OT.....	20
2.3.3 CRDT .....	22
2.3.4 WebRTC.....	23
2.3.5 Signal Protocol .....	24
3 API-uri folosite.....	25
3.1 Aplicația Server.....	25
3.2 Aplicația Client.....	26
3.3 SocketIo.....	28
3.4 Quill.....	29
4 Structura proiectului .....	31
4.1 Cerințele sistemului .....	31
4.1.1 Cerințe Non-Funcționale .....	31
4.1.2 Cerințe Funcționale .....	31
4.2 Arhitectura sistemului .....	32
4.2.1 Aplicația Web Client.....	33
4.2.2 Aplicației Web Server .....	35
4.2.3 Structura bazei de date.....	36
5 Implementarea software .....	39
5.1 Aplicația Web Client.....	39
5.1.1 Definirea Componentelor .....	39
5.1.2 React-Redux .....	45
5.1.3 Definirea Request-urilor.....	46

## NECLASIFICAT

5.2	Aplicația web server.....	47
5.2.1	Tratarea request-urilor.....	47
5.2.2	Baza de date MySQL.....	48
5.3	Metode de securizare.....	49
5.3.1	Protecție împotriva atacurilor CSRF .....	49
5.3.2	OpenVPN .....	51
5.3.3	Autentificare mutuală .....	52
6	Testarea sistemului .....	55
6.1	Utilizarea aplicației.....	55
6.2	Testarea sistemului .....	62
6.3	Diagrame UML .....	65
6.3.1	Diagrama cazurilor de utilizare .....	65
6.3.2	Diagrama de activități.....	66
7	Concluzii .....	67
7.1	Probleme întâmpinate.....	67
7.2	Rezultate obținute.....	67
7.3	Dezvoltări ulterioare.....	68
8	Bibliografie.....	69
9	Anexe .....	70
9.1	Anexa A.....	70
9.2	Anexa B.....	70

## LISTĂ DE ABREVIERI

1.	WebRTC	Web Real-Time-Comunication
2.	VoIP	Voice over Internet Protocol
3.	AES	Advanced Encryption Standard
4.	MitM	Man-in-the-Middle
5.	ALTS	Application Layer Transport Security
6.	TLS	Transport Layer Security
7.	OT	Operational Transformation
8.	CRDT	Conflict-free Replicated Data Type
9.	API	Application Programming Interface
10.	HTTP	Hypertext Transfer Protocol
11.	ICE	Interactive Connectivity Establishment
12.	STUN	Session Traversal Utilities for NAT
13.	TURN	Traversal Using Relay NAT
14.	UDP	User Datagram Protocol
15.	NAT	Network Address Translation
16.	DTLS	Datagram Transport Layer Security
17.	TLS	Transport Layer Security
18.	E2EE	End-to-End Encryption
19.	JSON	JavaScript Object Notation
20.	MVC	Model View Controller
21.	DOM	Document Object Model
22.	URL	Uniform Resource Locator
23.	CORS	Cross-Origin Resource Sharing
24.	CRUD	Create, Read, Update, Delete
25.	CSS	Cascading Style Sheets
26.	LESS	Leaner Style Sheets
27.	JWT	Json Web Token
28.	NPM	Node Package Manager
29.	VPN	Virtual Private Network
30.	SQL	Structured Query Language

## TABELĂ FIGURI

Figură 2.1 Microsoft Teams (Versiunea Desktop) – HomePage [3] .....	16
Figură 2.2 Discord (Versiunea Desktop) - Homepage[4].....	17
Figură 2.3 Google Docs - Editare colaborativă.....	18
Figură 2.4 Creare de conflict prin accesare concurențială .....	20
Figură 2.5 Mecanism Transformare Operațională[10] .....	21
Figură 2.6 WebRTC - Transmisia datelor[12].....	23
Figură 3.1 Modelul asincron de tratare al evenimentelor - NodeJS.....	25
Figură 3.2 Inițierea Protocolului WebSocket.....	28
Figură 3.3 Parametrii handShake pentru conexiunea de tip Socket.IO [18].....	29
Figură 4.1 Reprezentarea arhitecturii sistemului software.....	33
Figură 4.2 Diagrama de componente a sistemului.....	34
Figură 4.3 Fluxul de date în cadrul unui canal de comunicație Socket.IO .....	35
Figură 4.4 Structura bazei de date.....	36
Figură 5.1 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea 1 .....	40
Figură 5.2 Trimiterea request-ului de conectare la sesiunea de editare colaborativă .....	41
Figură 5.3 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea 2 .....	43
Figură 5.4 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea 3 .....	44
Figură 5.5 Principiu de funcționare al sistemului React Redux.....	45
Figură 5.6 Componentele de tip Reducer de la nivelul apluației client .....	46
Figură 5.7 Exemplu request de tip post pentru adăugare unui utilizator nou în grup .....	47
Figură 5.8 Folosirea funcției middleware pentru verificarea autenticității request-ului .....	47
Figură 5.9 Interogare SQL pentru returnarea listei de participanți din cadrul unui grup.....	49
Figură 5.10 Diagramă atac de tip CSRF .....	50
Figură 5.11 Securizarea sesiunii folosind JWT token.....	50
Figură 5.12 Rularea serviciului la nivelul utilizatorului pentru accesarea aplicației .....	51
Figură 5.13 Sesiune de administrare a serverului folosind conexiune SSH .....	52
Figură 5.14 Mod de interacțione folosind comunicația securizată.....	52
Figură 5.15 Fișierul de configurare al serverul de Nginx .....	52
Figură 5.16 Accesarea aplicației fără un deținerea certificatului de utilizator.....	53
Figură 5.17 Solicitarea certificatului de utilizator la nivelul browser-ului .....	53
Figură 6.1 Pagina de Login a aplicației .....	55
Figură 6.2 Prezentate Fereastra de Homepage.....	56
Figură 6.3 Accesarea meniului de navigație .....	56
Figură 6.4 Prezentare funcționalitate de schimbare a profilului .....	57
Figură 6.5 Prezentare funcționalitate de editare a datelor de utilizator.....	57
Figură 6.6 Accesarea serviciului de mesagerie .....	58
Figură 6.7 Exemplu de trimitere a unui fișier în cadrul conversației.....	58
Figură 6.8 Prezentarea funcționalităților din cadrul ferestrei de mesagerie.....	59
Figură 6.9 Prezentarea sistemului de stocare pentru fișiere.....	59
Figură 6.10 Sesiune de editare colaborativă a textului .....	60
Figură 6.11 Salvarea documentului generat în cadrul sesiunii de editare.....	60
Figură 6.12 Prezentarea sesiunii de comunicație video-audio .....	61
Figură 6.13 Fereastra de control destinată administratorilor .....	61
Figură 6.14 Diagrama cazurilor de utilizare .....	65
Figură 6.15 Diagrama de activități.....	66

# 1. Introducere

## 1.1 Importanța temei

Progresul tehnologic și necesitatea diminuării timpul de muncă împreună cu nevoia oamenilor de a menține comunicația la distanță au dus la dezvoltarea de aplicații software colaborative care să satisfacă aceste cerințe într-o manieră ușor de gestionat pentru utilizator. Astfel aplicațiile colaborative sunt dezvoltate cu scopul de a ajuta persoanele implicate în proiecte comune să își atingă obiectivele minimizând resursele necesare de spațiu și timp. Toate aceste acțiuni trebuie să fie desfășurate într-un mediu care oferă siguranță atât pentru utilizator, cât și pentru datele care sunt prelucrate.

## 1.2 Scopul lucrării

Prezenta lucrare are ca obiectiv detalierea și formarea cunoștiințelor necesare pentru dezvoltarea și operaționalizarea unei soluții software „on-premise” (implementată folosind resurse hardware proprii) care să asigure activitatea colaborativă la nivelul unei organizații guvernamentale. Lucrarea va cuprinde informații despre realizarea arhitecturii aplicației software, detalii despre tehnologiile folosite, informații despre mecanismele de securizare folosite. De asemenea, în proiect vor fi prezentate cazurile de utilizare și se va întocmi un raport de testare.

## 1.3 Utilitatea sistemului

Organizațiile guvernamentale lucrează cu date clasificate, care nu pot fi prelucrate folosind soluții software externe din rațiuni de securitate. În același timp există riscul ca o aplicație pusă la dispoziție de un dezvoltator extern să colecteze date cu privire la utilizator, precum locația acestuia, adresa IP sau activitatea acestuia la nivelul dispozitivului pe care este folosită. Astfel, apare necesitatea unei soluții dezvoltate și administrate de organizația în cauză, pentru a reduce riscul colectării și interceptării datelor.

În același timp folosirea unei aplicații realizate în mediul intern oferă control absolut asupra fluxului de date (datele sunt stocate folosind resurse locale), dar și asupra utilizatorilor - în cazul în care este detectată o activitate suspectă poate fi restricționat accesul asupra aplicației. Regulile pe care le respectă organizațiile guvernamentale nu permit folosirea de resurse Cloud externe pentru stocarea informațiilor, deoarece furnizorii de servicii Cloud nu pot dovedi integritatea datelor pe care le administrează, dar și din rațiuni politico-geografice. Soluția propusă va îndeplini criteriile de securitate specifice organizațiilor guvernamentale, punând accent pe securizarea canalului de comunicație, stocarea datelor, dar și monitorizării activității utilizatorilor.

NECLASIFICAT

## 2 Stadiul actual

### 2.1 Software colaborativ

Aplicațiile software colaborative sunt concepute cu scopul de a optimiza munca în echipă prin accesarea unui mediu de lucru comun unde angajații pot împărtăși cunoștințe, date și documente pentru a rezolva probleme specifice. Printre funcționalitățile cel mai des puse la dispoziție de aplicațiile collaborative se numără: comunicația dintre utilizatori folosind canale de mesagerie instantă, managementul documentelor, împărțirea de sarcini, portabilitatea pe diferite tipuri de device-uri și măsurile de securitate [1].

Lucrarea de față își propune crearea unei aplicații collaborative cu interfață web de tip „on premise” care să îndeplinească funcționalități de comunicație și de editare text în timp real, folosind doar servicii implementate intern. Spre deosebire de alte aplicații puse la dispoziție pe piață, soluția software propusă va oferi acces nelimitat despre modul de funcționare, întrucât este implementată folosind resurse proprii, iar clientul va avea acces și la codul sursă al acesteia. Marea majoritate a aplicațiilor cu scop colaborativ oferite de dezvoltatori externi nu oferă servicii „on-premise”, datele clienților fiind administrate de dezvoltatorii în cauză.

### 2.2 Sisteme similare

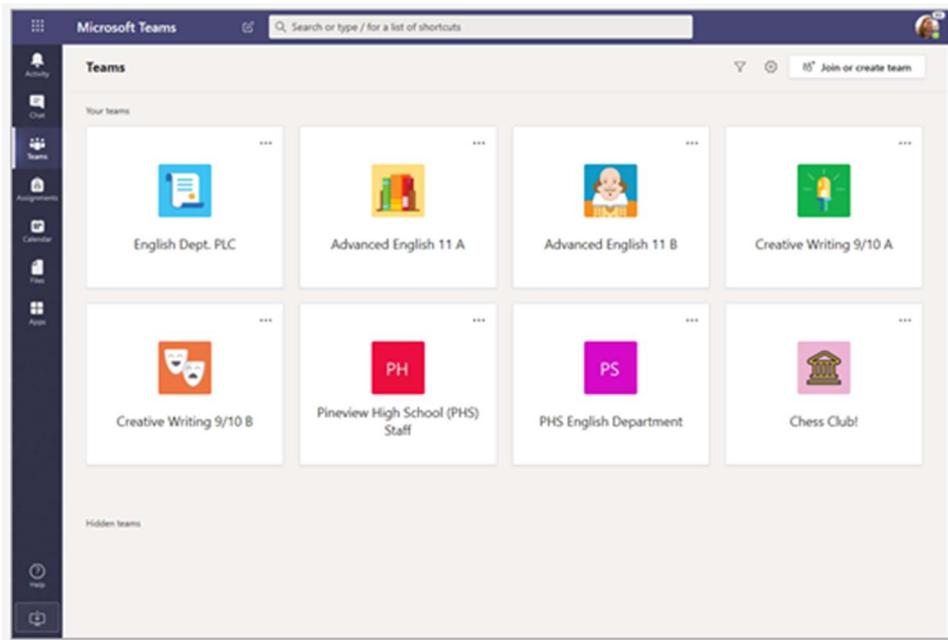
În momentul actual există o diversitate destul de mare de aplicații care oferă suport pentru munca colaborativă, cele mai multe dintre ele fiind implementate folosind Cloud extern. Acest aspect devine un criteriu important de selecție atunci când organizația își dorește o securitate ridicată și lucrează cu informații clasificate care nu trebuie să părăsească mediul intern al organizației. Printre aplicațiile collaborative des întâlnite amintim: Microsoft Teams, Slack, Discord, Asana și Google Docs.

#### 2.2.1 Microsoft Teams

Microsoft Teams este o aplicație colaborativă lansată la finalul anului 2016 care înglobează toate serviciile de comunicație (mesagerie instantă, apeluri video și audio, gestionare de documente), oferind suport ca aplicație web, aplicație mobilă și aplicație desktop. Acest utilitar se integrează cu suita de servicii Microsoft 365 și alte servicii furnizate de Microsoft (spațiul de stocare Cloud)[2].

Principalele avantaje ale aplicației Microsoft Teams sunt portabilitatea între dispozitive, capabilitatea de a gestiona grupuri cu un număr ridicat de utilizatori și diversitatea serviciilor puse la dispoziție. Acest serviciu este destinat în principal companiilor sau organizațiilor cu un număr ridicat de utilizatori, pornind de la abonamente gratuite, până la abonamente contra cost în funcție de necesitățile și activitatea organizației. Popularitatea de care se bucură serviciul

Teams este dată de suportul și sustenabilitatea oferite de firma Microsoft, acesta ocupând mare parte a spațiului educațional și antreprenorial.



Figură 2.1 Microsoft Teams (Versiunea Desktop) – HomePage [3]

## 2.2.2 Slack

Slack este un serviciu software de mesagerie instantanee lansat în anul 2013 și conceput special pentru mediul de lucru din cadrul organizațiilor. Potrivit deținătorului companiei, numele aplicației este un acronim pentru „Searchable Log of All Conversation and Knowledge”, astfel dezvăluind principala caracteristică a produsului, aceea ca toate conversațiile și fișierele partajate să poată fi căutate mai ușor față de alte dispozitive colaborative[3]. Un alt avantaj major al acestei platforme este că elimină nevoia conversațiilor bazate pe email la nivelul organizației.

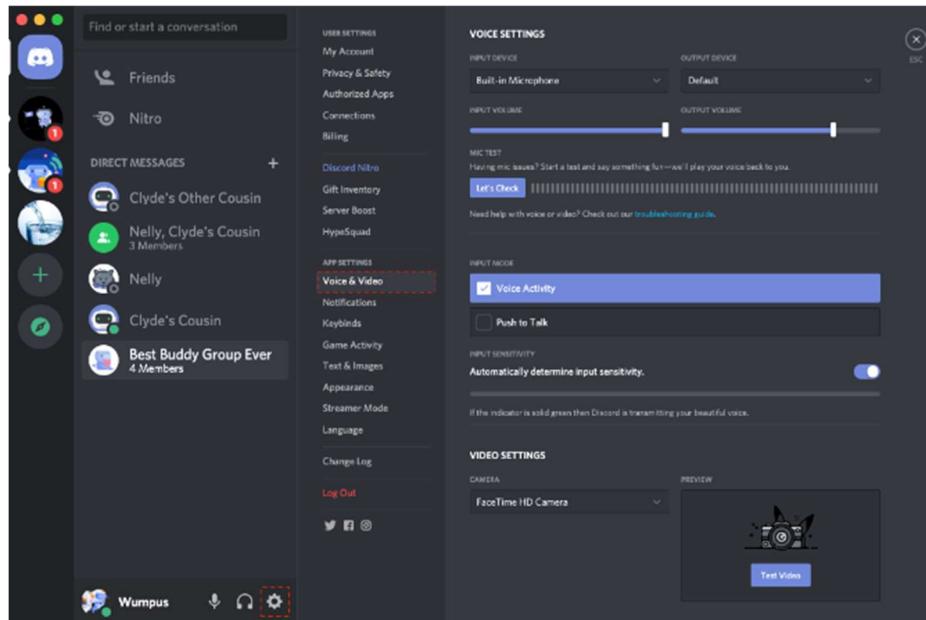
Slack oferă versiuni ale aplicației compatibile desktop și mobile, dar și o versiune de browser. Din punct de vedere al scalabilității, aplicația se remarcă prin faptul că nu are o limită superioară al numărului de conversații deschise, față de competitorul său Teams care are o limită superioară de maxim 200 de canale la nivelul unei echipe.

## 2.2.3 Discord

Discord este o aplicație cross-platform gratuită de mesagerie text, voce și video folosită cel mai mult în scop recreativ. Popularitatea aplicației este dată de faptul că este gratuită, iar limita maximă a utilizatorilor dintr-un grup poate ajunge până la 250.000, cu mențiunea că acest număr poate fi mărit. Limita de participanți pentru un apel video este de 25 de utilizatori, iar pentru o conversație audio este de până la 122 de participanți. Pentru asigurarea comunicației, Discord, folosește tehnologia WebRTC oferită la nivelul browser-ului. Tehnologia WebRTC este

disponibilă în toate browserele moderne. Pentru versiunile aplicației care rulează pe alte dispozitive (desktop, mobile) este utilizat un modul media dezvoltat în C++ programat pe baza bibliotecii native WebRTC. Datorită acestui aspect unele caracteristici ale serviciului funcționează mult mai bine pe aplicațiile instalate decât în aplicația de tip browser.

Unul din dezavantajele aplicației aplicației sunt problemele de securitate și incidentele petrecute în trecut precum cele de tipul phishing și ransomware. De asemenea, aplicația nu eforă un serviciu de criptare end-to-end precum alte aplicații de chat.



Figură 2.2 Discord (Versiunea Desktop) - Homepage[4]

## 2.2.4 Telegram

Spre deosebire de celelalte aplicații prezentate până acum, Telegram este o aplicație colaborativă de tip open source ce prestează servicii de mesagerie text și VoIP. Aplicația se remarcă prin numărul foarte mare de participanți ce pot face parte dintr-un grup la un moment dat, în momentul de față acesta fiind de până la 100.000 de utilizatori. Ca și particularități de interes tehnic mesajele transmise cu ajutorul aplicației sunt transportate client-server criptat folosind criptare simetrică AES-256, după care sunt salvate la nivelul serverului pentru a putea fi descărcate oricând pe orice alt dispozitiv la care se conectează utilizatorul.

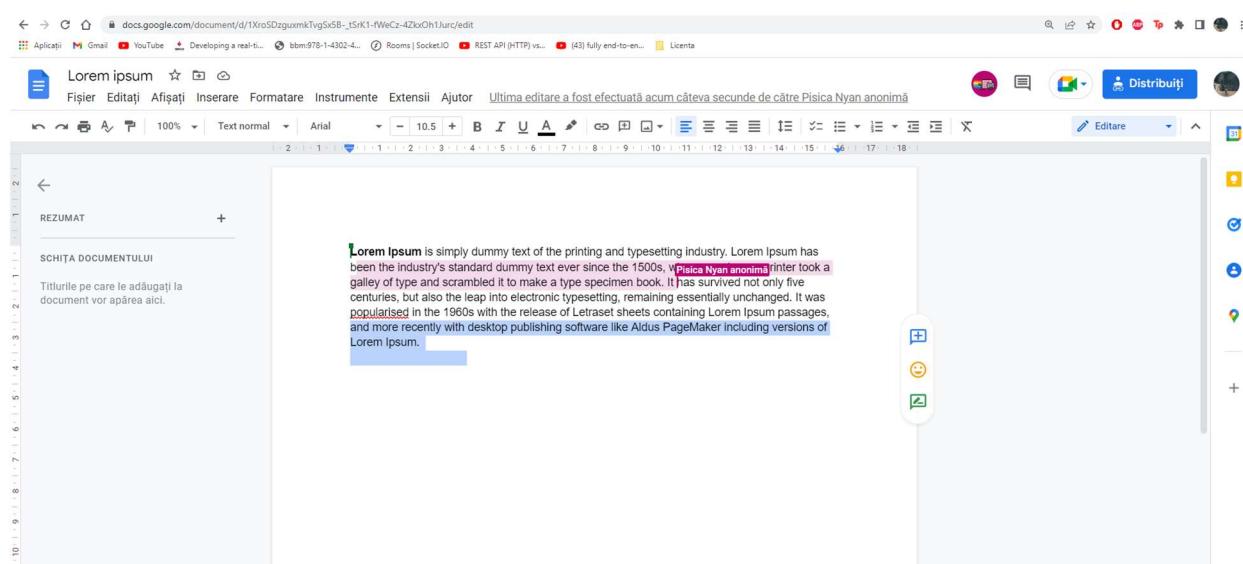
Pentru purtarea de conversații sigure, aplicația pune la dispoziție opțiunea de conversații „secrete” de tip text și apeluri vocale criptate end-to-end doar între doi utilizatori online, dar nu și pentru grupuri sau canale cu mai mult de doi utilizatori. Dacă utilizatorul optează pentru acest mod de conversație „secretă” atunci mesajele nerecepționate nu pot fi retrimise destinatarului, iar acestea nu sunt nici salvate pentru back-up. Pe lângă aceste specificații aplicația atrage controverse prin folosirea unei scheme de criptare proprie derivată din schimbul de chei Diffie-Hellman, care a fost criticată de specialiști. Această schemă de criptare este

descrișă pe site-ul celor de la Telegram, purtând numele de protocolul MTProto[5]. Acest protocol este în prezent la versiune 2.0, versiuna MTProto 1.0 fiind considerată depreciată de la sfârșitul anului 2017. Într-o lucrare publicată de Marino Miculan și Nicola Vitacolonna este făcută o analiză teoretică asupra protocolului MTProto 2.0 și rezultatul obținut concluzionează că algoritmul este vulnerabil la anumite tipuri de atacuri MitM. [6]

## 2.2.5 Google Docs

Google Docs este face parte din suita de aplicații și servicii puse la dispoziție de către Google. Acest serviciu este accesibil prin intermediul browser-ului de internet ca aplicație web, dar este disponibil și ca aplicație mobilă pentru Android, iOS sau desktop doar pe sistemul de operare Chrome OS dezvoltat de Google. Aplicația, Google Docs, este destinată pentru realizarea și editarea de documente online de către mai mulți utilizatori simultan. Modificările aduse de fiecare utilizator sunt salvate într-un istoric propriu folosit pentru revizuire. Poziția cursorului specifică fiecărui utilizator este evidențiată cu ajutorul unei culori, iar un sistem de permisiuni reglementează ceea ce pot face utilizatorii. În plus aplicația, dispune de o secțiune specială pentru distribuirea de sarcini către utilizatori, iar modificările aduse fișierelor sunt transmise serverului în mod automat.

Serviciile Google folosesc pentru autentificare și transport un protocol criptografic numit ALTS, dezvoltat special de Google pentru securizarea serviciilor sale, în detrimentul protocolului standard TLS. Ca principale avantaje ale folosirii protocolul ALTS se remarcă utilizarea protocolului de tip buffer (format de date cross-platform gratuit și open-source destinat serializării datelor) pentru a serializa certificatele și mesajele de protocol, în timp ce TLS utilizează certificate X.509 codificate cu ASN.1.[7]



Figură 2.3 Google Docs - Editare colaborativă

## 2.3 Principii de funcționare

Aplicațiile software colaborative sunt sisteme informatiche care sprijină persoanele angajate în sarcini de interes comun prin oferirea de aplicații cu interfață partajată. Aceste instrumente trebuie să respecte anumite criterii de funcționare pentru a furniza utilizatorilor forme de interacțiune la un nivel optim, facilitând controlul, coordonarea și colaborarea la nivelul echipei.

Potrivit articolului „Collaborative Systems: Characteristics and Feature” apărut în anul 2012[8], sistemele colaborative pot fi clasificate pe baza interacțiunii și comunicăției cu utilizatorului. Prin urmare, instrumentele colaborative pot fi clasificate pe baza locului unde se desfășoară interacțiunea (ambii utilizatori sunt prezenți în același loc sau de la distanță) și pe baza timpului (dacă activitatea efectuată de utilizatori este sincronă sau asincronă):

- Sisteme sincrone. Sunt instrumente de colaborare care necesită un timp de răspuns cât mai scăzut precum serviciile de mesagerie sau teleconferință;
- Sisteme asincrone. Aceste instrumente nu necesită un timp de răspuns aproape instant. Exemple pentru această categorie de instrumente asincrone sunt: transmiterea de email-uri, calendarele de activități sau aplicațiile de tip forum.

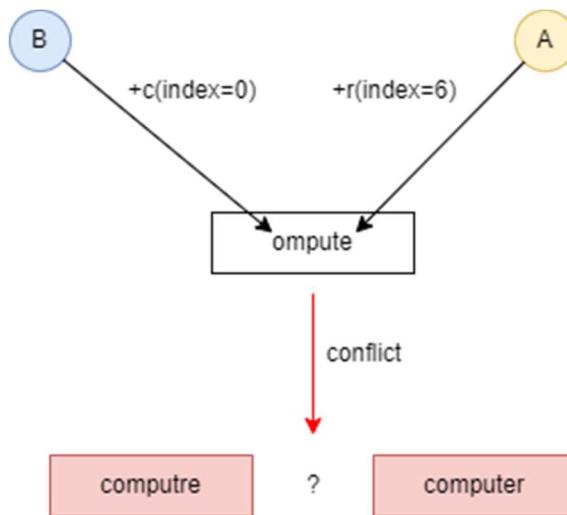
În continuare, voi descrie o parte din conceptele care stau la baza sistemelor colaborative din cadrul editării de text, partajării de flux video și asigurarea confidențialității.

### 2.3.1 Editarea colaborativă

Editorul de text colaborativ permite mai multor utilizatori să vizualizeze și să editeze simultan un document partajat. Dezvoltarea unui editor colaborativ necesită luarea în considerare a următoarelor cerințe de sistem:

- Timpul de răspuns trebuie să fie adecvat operațiunilor de editare simultane;
- Asigurarea coereneței textului în timpul actualizărilor simultane;
- Posibilitatea fiecărui utilizator de a se întoarce la o stare precedentă;
- Evidențierea și indicarea prezenței celorlalți utilizatori la nivelul documentului.

Principalul aspect ce trebuie luat în considerare la dezvoltarea unei aplicații colaborative bazate pe editare de text este că atunci când se dorește editarea simultan de către mai mulți utilizatori, apar probleme de control concurențial. Pentru detalierea acestui caz voi reluarea exemplul descris în capitolul „Group Editors” din lucrarea „Computer Supported Co-operative Work” publicată de către profesorul Michel Beaudouin-Lafon.



Figură 2.4 Creare de conflict prin accesare concurențială

Un document conține un sir de caractere „ompute”. Să presupunem că utilizatorul A încearcă să insereze caracterul „r” după caracterul „e”. Majoritatea editoarelor text folosesc funcții de inserare care utilizează un parametru pentru indexarea poziției din document, mai exact o funcție care va adăuga litera „r” la poziția 6. Dar să presupunem că între momentul în care operația este generată de datele de intrare ale lui A și momentul în care este executată, un alt utilizator, B, încearcă să adauge litera „c” înaintea literei „o”. Dacă este executată operația lui B înaintea operației lui A, sirul afișat ar fi „computre”, în loc de rezultatul dorit, „computer”[9].

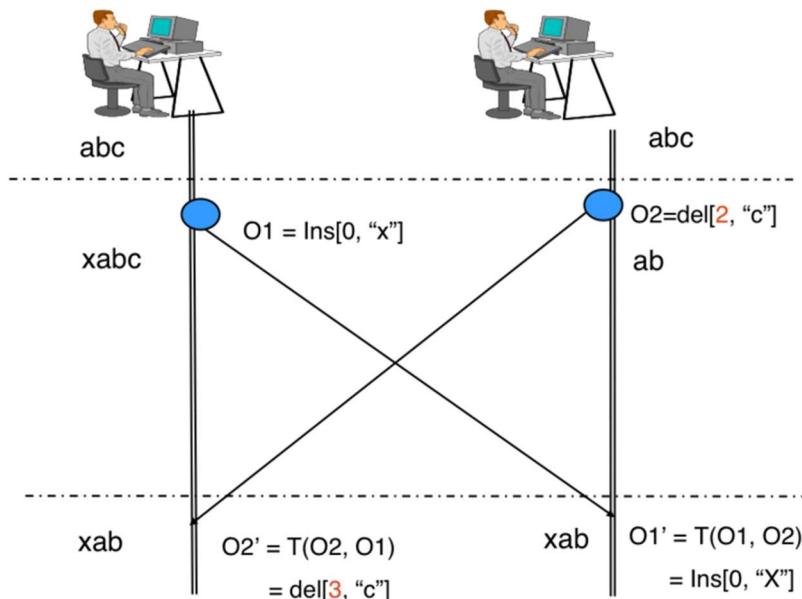
Autorul lucrării propune diferite scheme de control al accesului pentru rezolvarea conflictelor printre care: controlul concurențial pesimist, controlul concurențial pozitiv sau posibilitatea ca modificările participanților să comute între ele. În cazul pesimist restricționează accesul utilizatorilor la editare până ce documentul este reactualizat pentru fiecare utilizator, acest algoritm fiind nefezabil din cauza generării timpilor mari de așteptare ca rezultat al latenței rețelei. Pentru accesul concurențial pozitiv, o operație este executată imediat pe copia locală și apoi este transmisă spre restul utilizatorilor pentru a fi executată. Toate operațiile de actualizare sunt mai întâi marcate în timp, astfel încât două operații oarecare să fie consecvent reordonate în toate copiile, chiar dacă sunt primite în aceeași ordine sau nu. Pentru reordonare, fiecare editor trebuie să mențină o listă de istoric a operațiilor ce au fost făcute asupra documentului. În continuare urmează descrierea a doi algoritmi dezvoltăți pentru rezolvarea controlului concurențial la nivelul editorului.

### 2.3.2 OT

OT (Operational Transformation) este o tehnică de menținere a consistenței pentru sistemele de editare colaborativă folosită de aplicații distribuite pentru sprijinirea interacțiunii om-calculator și a colaborării prin rețele de comunicații. Teoria cauzalității a stat la baza tuturor sistemelor OT anterioare, dar este

inadecvată pentru a satisface cerințele esențiale ale OT în funcționalitate și corectitudine. Capacitățile sale au fost extinse, iar aplicabilitățile sale au fost extinse pentru a include rezolvarea conflictelor, notificarea și comprimarea operațiilor, conștientizarea grupului, partajarea aplicațiilor și instrumentelor de proiectare media asistată de calculator în colaborare[10]. Această tehnologie este folosită de către editorul Google Docs pentru rezolvarea conflictelor.

Sistemele de colaborare care utilizează OT folosesc de obicei stocarea replicată a documentelor, astfel fiecare aplicație client are propria copie a documentului. Modificările de text sunt făcute de utilizatori pe copiile lor locale într-o manieră cursivă, fără blocaje, iar modificările sunt apoi propagate la restul clientilor. Acest mecanism asigură o capacitate de reacție ridicată a clientului în medii cu latență ridicată, precum internetul. Atunci când un client primește modificările propagate de la un alt client, acesta transformă modificările în funcție de copia sa locală, înainte de a le executa. Transformarea asigură menținerea criteriilor de coerență dependente de aplicație de către toate site-urile. Acest mod de funcționare are ca rezultat un sistem potrivit pentru implementarea funcțiilor de colaborare, cum ar fi editarea colaborativă a documentelor.



Figură 2.5 Mecanism Transformare Operațională[10]

Mecanismul de funcționare al transformării operaționale este reprezentat în figura de mai sus folosind un scenariu simplu: fiecare utilizator este conectat la sesiunea de editare ce conține textul „abc”. Primul utilizator introduce caracterul „x” la poziția 0 a șirului, definind astfel operația  $O1 = \text{insert}[0, 'x']$ . Cel de al doilea utilizator dorește să șteargă caracterul „c” de la poziția 2, definind operația  $O2 = \text{insert}[2, 'c']$ .

Să pleacă de la premiza că cele două operații sunt executate în ordinea  $O1$  și  $O2$ , pentru primul utilizator. După executarea lui  $O1$ , textul formatat devine "xabc". Pentru a putea fi executată  $O2$  după ce a fost executată  $O1$ , mai întâi trebuie ca  $O2$  să fie transformată față de  $O1$  pentru a deveni:  $O2' = \text{Delete}[3, 'c']$ ,

al cărui parametru pozitonal este incrementat cu o unitate datorită inserării caracterului „x” de către operația O1. În final, după O2' pe „xabc” se șterge caracterul corect „c” și documentul devine „xab”. Cu toate acestea, dacă O2 este executată fără transformare, aceasta șterge în mod incorrect caracterul „b” în loc de caracterul „c”. Ideea de bază a transformării operaționale este transformarea parametrilor operațiilor de editare în funcție de efectele operațiilor concurente executate anterior, astfel încât operația transformată să poată obține efectul corect și să fie menținută coerentă documentului.

### 2.3.3 CRDT

În calcul distribuit, CRDT (Conflict-free replicated data type) este o structură de date care poate fi replicată pentru mai mulți utilizatori, replicile putând fi actualizate independent și simultan fără coordonare între ele, iar inconsecvențele care pot apărea pot fi mereu rezolvate din punct de vedere matematic. Conceptul CRDT a fost definit în mod oficial în 2011, iar dezvoltarea a fost motivată inițial de editarea colaborativă de text și de informatică mobilă. Sistemele CRDT au fost utilizate, de asemenea, în sistemele de mesagerie online și în jocurile de noroc online[11]. Există două abordări pentru sisteme CRDT, ambele putând asigura o coerentă puternică eventuală: sisteme CRDT bazate pe operații și sisteme CRDT bazate pe stări. Cele două alternative sunt echivalente din punct de vedere teoretic, deoarece una poate emula pe celalaltă. Cu toate acestea, există diferențe practice.

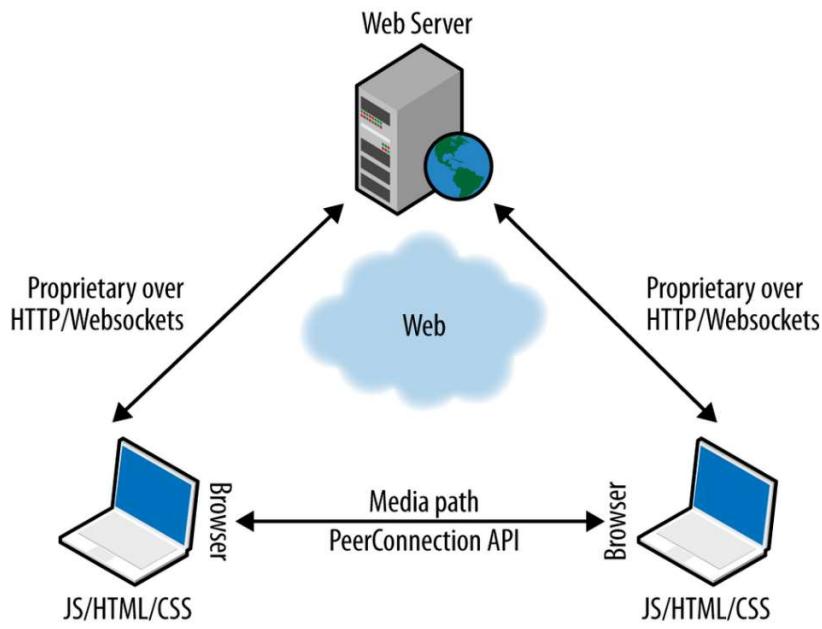
Sistemele CRDT bazate pe stare sunt adesea mai simplu de proiectat și de implementat; singura lor cerință de la substratul de comunicație fiind mesajele despre starea celorlalte sisteme replicate. Dezavantajul lor este că întreaga stare a fiecărei CRDT trebuie transmisă în cele din urmă fiecărei alte replici, ceea ce poate fi costisitor. În schimb, sistemele CRDT bazate pe operații transmit doar operațiile de actualizare, care sunt de obicei mici. Cu toate acestea, sistemele CRDT bazate pe operații necesită garanții din partea protocolului de transmisie că operațiile nu sunt abandonate sau duplicate atunci când sunt transmise către celelalte replici și că sunt livrate în ordine cauzală[11].

Sistemele CRDT bazate pe operații sunt, de asemenea, numite tipuri de date replicate comutative. Replicile CRDT de tip comutativ propagă starea prin transmiterea doar a operației de actualizare. Cu toate acestea, ele nu sunt neapărat neschimbate. Prin urmare, infrastructura de comunicații trebuie să se asigure că toate operațiile care aparțin unei replici sunt transmise celorlalte replici, fără duplicate, dar în orice ordine. Sistemele CRDT bazate pe stare se numesc tipuri de date replicate convergente. Acestea, spre deosebire de celelalte, trimit starea lor locală completă către alte replici, unde stările sunt fuzionate utilizând o funcție care trebuie să fie comutativă, asociativă și imutabilă.

### 2.3.4 WebRTC

Web Real-Time Communication (WebRTC) este o colecție de standarde, protocole și API-uri JavaScript care face posibilă partajarea de date audio și video între browsere, în mod peer-to-peer. Semantica clasică a arhitecturii web se bazează pe o paradigmă client-server, în care browserele trimit o cerere de conținut HTTP către serverul web și primesc ca răspuns informațiile solicitate.

WebRTC extinde semantica clasică de tip client-server prin introducerea unei paradigmă de comunicație peer-to-peer între browsere. În cazul aplicațiilor colaborative acest standard este foarte popular datorită modului facil de implementare al canalului de comunicație. Înainte de stabilirea conexiunii dintre cele două browsere implicate în conexiune, este necesar trimitera de mesaje intermediare pentru schimbul de adrese ale clientilor. Aceste mesaje se mai numesc și mesaje de semnalizare. Acestea sunt transportate folosind protocolul HTTP sau WebSocket prin intermediul serverelor web care le pot modifica, traduce sau gestiona în funcție de necesități. Este de remarcat faptul că semnalizarea dintre browser și server nu este standardizată în WebRTC, deoarece este considerată ca făcând parte din aplicație. În ceea ce privește calea de transmitere a datelor, conexiunea peer-to-peer permite ca mediile să circule direct între browsere, fără intervenția serverului, reprezentare în figura 2.6.



Figură 2.6 WebRTC - Transmisia datelor[12]

Odată stabilită o conexiune între omologii, fluxurile media (asociate local cu obiecte MediaStream definite ad-hoc) pot fi trimise direct către browserul de la distanță. Conexiunea peer-to-peer utilizează protocolul ICE împreună cu serverele STUN și TURN pentru a permite fluxurilor media bazate pe UDP să traverseze sistemele NAT și sistemele firewall. Protocolul ICE permite browserelor să descopere suficiente informații despre topologia rețelei în care sunt implementate pentru a găsi cea mai bună cale de comunicație exploataabilă. Utilizarea ICE oferă,

de asemenea, o măsură de securitate, deoarece împiedică paginile web și aplicațiile care nu sunt de încredere să trimită date către gazde care nu se așteaptă să le primească.

Protocolul STUN (definit în RFC5389) permite unei aplicații găzdui să descopere prezența unui traductor de adrese de rețea și, în acest caz, să obțină combinațiile de adrese IP și porturi publice alocate pentru conexiune. Pentru a face acest lucru, protocolul necesită asistență din partea unui server STUN configurat, de treță parte, care trebuie să se afle în rețeaua publică.

Protocolul TURN (definit în RFC5766) permite unei gazde aflate dincolo de NAT să obțină o adresă IP publică și un port de la un server relu care se află pe internetul public. Datorită adresei de transport retransmise, gazda poate primi apoi medii de la orice omolog care poate trimite pachete către internetul public.

Fluxul media de date efectuat la nivelul standardului WebRTC sunt criptate cu ajutorul protocolului DTLS (definit în RFC6347), acesta fiind conceput pentru a preveni interceptarea, modificarea sau falsificarea mesajelor în transportul de datagrame oferit de protocolul UDP și având la bază protocolul TLS cu specificații de securitate asemănătoare.

### 2.3.5 Signal Protocol

Criptarea E2EE este un tip de criptografie asimetrică, care protejează datele, făcându-le disponibile doar destinatarului. La momentul actual, E2EE este considerat cel mai sigur mecanism de protecție a datelor, întrucât participanții sunt direct implicați în schimbul de mesaje, ofuscarea datelor realizându-se doar la nivelul lor fără intervenția serverului sau al altor entități. În acest procedeu, serverul are scopul doar de a realiza schimbul de chei publice dintre participanți fără să aibă posibilitatea de decriptare a mesajelor.

Protocolul Signal este un protocol dezvoltat de Open Whisper Systems în 2013 prin intermediul căruia se poate realiza o comunicație securizată E2EE. Etapele protocolului Signal sunt următoarele:

- Schimbul inițial de chei, sau protocolul X3DH (extended triple Diffie-Hellman), care combină chei Diffie-Hellman cu nivel de persistență în timp scurt, mediu și îndelungat pentru a genera cheia de bază partajată;
- O etapă asimetrică de tip „ratchet” (termen care în traducere înseamnă clichet, în literatura de specialitate fiind folosit pentru a exprima proprietatea sistemului de a funcționa într-un singur sens, fără posibilitate de inversare a operațiilor), în care utilizatorii trimit alternativ noi chei de tip Diffie-Hellman efemere pe baza cheilor private de bază generate anterior pentru a genera o serie de chei de sesiune;
- O etapă simetrică cu clichet de tip „ratchet”, în care utilizatorii nu iau nicio entropie suplimentară, ci folosesc în schimb funcții de derivare pentru a genera chei de criptare simetrice.

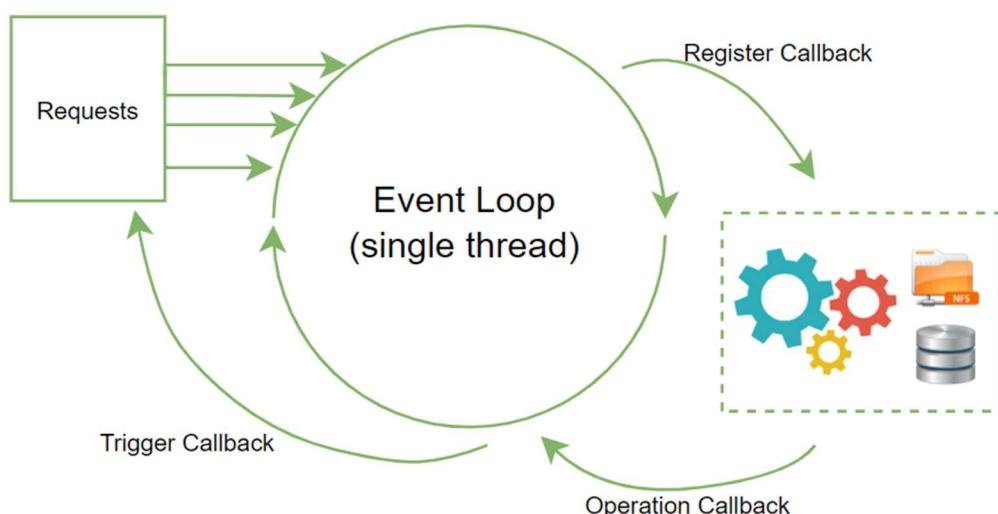
Mecanismele de tip „ratchet” generează o nouă cheie privată pe baza celei precedente, oferind sistemului confidențialitate în cazul în care una din cheile private de bază este compromisă. Astfel, un posibil atacator nu va reuși să decripteze decât o serie minimă de mesaje[13].

### 3 API-uri folosite

Pentru realizarea proiectului s-a folosit limbajul de programare orientat obiect JavaScript însorit de bibliotecile aferente acestuia. Pentru aplicația server a fost utilizat NodeJS, iar pentru realizarea aplicației client ReactJS. Pentru persistența datelor am ales folosirea unei baze de date MySQL de tip relațional, din necesitatea de a reține relațiile de interacțiune dintre utilizatori sub forma unei reprezentări grafice bine definite. Limbajul de programare JavaScript facilitează folosirea formatului de date JSON, un format inteligibil pentru oameni și folosit pentru transmiterea datelor structurate prin rețea.

#### 3.1 Aplicația Server

NodeJS este un mediu de execuție JavaScript asincron bazat pe evenimente, open-source și cross-platform. Spre deosebire de cele mai multe alte medii moderne, un proces Node nu se bazează pe multithreading, pentru a susține execuția concurențială a proceselor, ci se bazează pe un model asincron de desfășurare a evenimentelor Input/Output (vezi figura 3.1). Limbajul JavaScript se potrivește excelent pentru această abordare pentru că suportă returnările de evenimente. Natura funcțională a JavaScript îl face extrem de facil pentru crearea obiectelor funcționale anonime ce pot fi înregistrate ca gestionări de evenimente. Printre altele, NodeJS, în comparație cu alte medii de execuție, prezintă și capacitatea de a gestiona un număr foarte mare de conexiuni simultane cu un debit ridicat de date, ceea ce echivalează cu o scalabilitate ridicată[14].



Figură 3.1 Modelul asincron de tratare al evenimentelor - NodeJS

Express.js este un framework creat pentru mediul de execuție NodeJS și utilizat pentru reducerea timpului de dezvoltare al aplicațiilor și organizarea arhitecturii aplicației după modelul MVC. În cadrul proiectului, am folosit acest framework, deoarece oferă un mecanism avansat de rutare ce ajută la păstrarea stării paginii web cu ajutorul URL-ului lor și de asemenea pune la dispoziție mai multe caracteristici utilizate în mod obișnuit de NodeJS sub formă de „middleware” care pot fi apelate oriunde la nivelul programului[15]. Principalele componente „middleware” folosite pentru realizarea aplicației server sunt următoarele:

- *body-parser* – analizează corpul request-ului înainte de prelucrarea acestuia;
- *cookie-parser* – parsează antetul Cookie din request sub formă de perechi cheie-valoare;
- *cors* – folosit pentru activarea politicilor CORS cu diferite opțiuni;
- *dotenv* – încarcă variabilele de mediu, globale, din fișierul „.env” în „process.env”;
- *date-and-time* – colecție minimalistă de funcții pentru prelucrarea datei și orei;
- *multer* – permite manipularea „multipart/form-data” cu scopul încărcării și salvării de fișiere;
- *mysql* – folosit pentru interogarea bazei de date MySQL;
- *socket.io* – permite comunicația bidirectională în timp real bazată pe evenimente folosind protocolul WebSocket;
- *rand-token* – generează token-uri aleatoare pe baza input-ului furnizat de utilizator;
- *moment* – folosit pentru analizarea, validarea și formatarea formatului de timp;
- *uuid* – furnizează id-uri aleatoare folosite la identificarea obiectelor;
- *crypto-js* – furnizează o colecție de algoritmi criptografici folosiți pentru aigurarea confidențialității la nivelul aplicației;
- *fs* – permite creare de foldere și fișiere atât în mod sincron, cât și asincron;
- *node* – oferă utilitate pentru lucrul cu căile de acces la fișiere și directoare.

## 3.2 Aplicația Client

Pentru implementarea aplicației client, am ales biblioteca ReactJS *v18.2.0*, datorită popularității sale la momentul actual. Această bibliotecă apartanentă JavaScript este utilizată pentru dezvoltarea de interfețe interactive cu utilizatorul și se remarcă prin dezvoltarea componentelor modulare care pot fi reutilizate,

încorporând modelul de proiectare MVC. Alte componente API notabile folosite pentru realizarea aplicației client sunt:

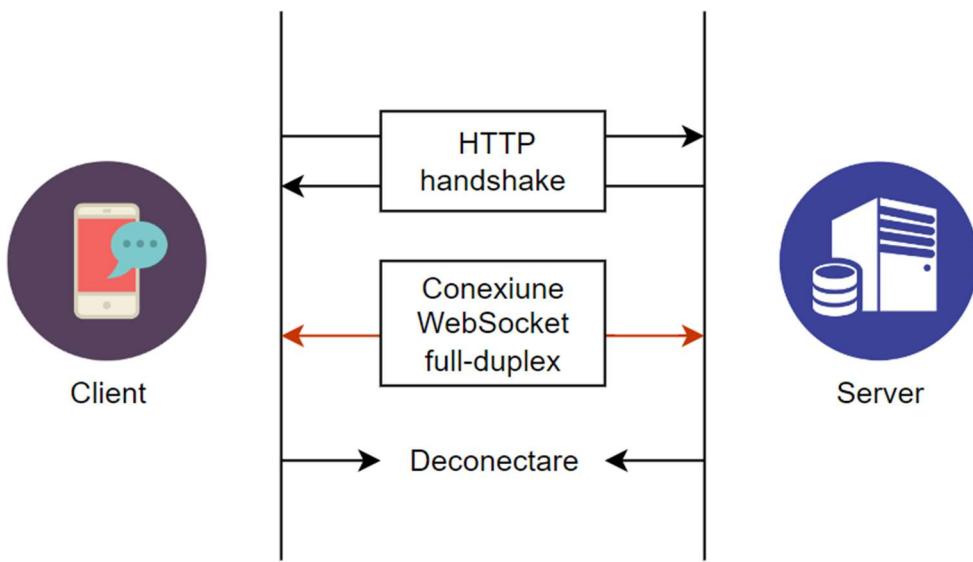
- *React Hooks* – este o funcționalitate introdusă în versiunea React 16.8. Aceasta permite utilizarea stării componentelor specifice React fără folosirea unui obiect de tip clasă (de menționat că acestea nu funcționează în interiorul unei clase). Denumirea de „Hooks” denotă caracteristica acestora de a se „agăța” de starea și de caracteristicile ciclului de viață din componentele funcționale[16];
- *React Redux* – permite componentelor React să acceseze variabile salvate într-un mediu de stocare numit Redux Store. Atunci când variabilele sunt modificate, toate componentele ce depind de folosirea lor sunt redate;
- *bootstrap* – framework de tip front-end ce cuprinde elemente de design HTML și CSS predefinite pentru dezvoltarea interfețelor web;
- *axios* – modul specific mediului NodeJS, ce facilitează trimiterea de request-uri HTTP asincrone către server cu scopul efectuării de operații CRUD;
- *crypto-browserify* – modul ce asigură funcțiile modulului *crypto-js*, dar pentru utilizarea în browser;
- *font-awesome* – este un set de pictograme și fonturi bazat pe CSS și LESS;
- *simple-peer* – bibliotecă utilizată pentru crearea de conexiuni de tip peer-to-peer între aplicațiile de tip browser. De menționat că nu include funcționalitate de semnalizare print intermediul serverului. Semnalizarea se referă la schimbul de mesaje de start pentru stabilirea conexiunii;
- *socket.io-client* – permite conectarea clientului de la nivelul browser-ului la conexiunea bazată pe protocolul WebSocket;
- *quill* – este un modul specific JavaScript ce implementează un editor de text cross-platform; acesta suportă un set extins de operații dedicate formatării de text;
- *quill-cursors* – este un modul *quill* dedicat pentru afișarea cursorilor altor utilizatori pentru asigurarea unei experiențe interactive de editare colaborativă;
- *MediaDevices.getUserMedia()* – solicită utilizatorului permisiunea de a utiliza intrările media (cameră, microfon sau serviciu de partajare ecran) care produc un flux media cu outputul generat de acestea.

ReactJS s-a remarcat ca fiind un sistem eficient și performant datorită conceputului de DOM virtual pe care îl folosește. Cu alte cuvinte, menține un model de DOM virtualizat în interiorul memoriei, iar atunci când o modificare trebuie să fie adusă paginii web, în loc să se actualizeze instantaneu sistemul DOM de navigare, sunt efectuate mai întâi modificări la nivelul sistemului de

DOM virtual din cadrul ReactJS. După realizarea modificărilor la nivelul de DOM virtual, se aplică un algoritm diferențial care compară modelul virtual și modelul de navigare, iar apoi actualizează doar nodurile relevante și dorite din arborelui DOM din browser. Fluxul datelor la nivelul ReactJS este permis de sus în jos, unidirecțional și susținut. În cazul în care o anumită modificare urmează să fie făcută asupra datelor din componente superioare, componente care utilizează acele date se vor reda automat pentru a actualizare[17].

### 3.3 SocketIO

WebSocket este un protocol de comunicații care oferă canale de comunicație full-duplex utilizând o singură conexiune TCP. WebSocket este distinct de HTTP. Ambele protocole se află la nivelul 7 în modelul OSI și depind de TCP la nivelul 4. Deși sunt diferite, RFC 6455 afirmă că „este conceput pentru a funcționa pe porturile HTTP 443 și 80, precum și pentru a suporta proxy-uri și intermediari HTTP”, făcându-l astfel compatibil cu HTTP. Pentru a obține compatibilitatea, handshake-ul WebSocket utilizează antetul HTTP Upgrade pentru a trece de la protocolul HTTP la protocolul WebSocket.



Figură 3.2 Inițierea Protocolului WebSocket

Socket.IO este o bibliotecă care permite comunicația, bidirectională, fără întârzieri și bazată pe evenimente între Client și Server. Necesitatea folosirii acestei biblioteci în cadrul proiectului este dată de nevoia transmisiei de mesaje cu latență cât mai scăzută. Aceasta este construit pe baza protocolului WebSocket și oferă garanții suplimentare de transmitere a datelor, cum ar fi revenirea la o interogare de tip HTTP sau reconectarea automată[18]. Socket-ul folosit folosit pentru realizarea conexiunii emulează un socket de rețea prin diferite mecanisme de transport și are diferite etape în ciclul său de funcționare care sunt următoarele:

- *connecting*;
- *connected*;

- *disconnecting*;
- *disconnected*;

Socket-ul se stabilește doar după ce clientul trimită o cerere de conectare către server și se inițiază un handshake. După ce handshake-ul este finalizat, se deschide o conexiune folosind protocolul de transport negociat, iar starea socket-ului este setată ca *connected*. Pentru a verifica dacă socketul este conectat serverul poate solicita trimiterea de mesaje *heartbeat* de la client la server în intervale regulate. În absența unui astfel de mesaj sau în cazul deconectării layer-ului transport, socket-ul va fi deconectat. În acest caz, clientul va iniția o reconectare, iar în cazul în care conexiunea este restabilită în intervalul de timp așteptat, conexiunea se va restabili, iar mesajele stocate în memorie vor fi trimise. În cazul în care conexiunea nu este restabilită, clientul va iniția o nouă cerere de conexiune, prin inițierea unui nou handshake. Socket-ul este închis atunci când metoda *close()* este apelată fie de client, fie de server. Odată ce conexiunea de transport este stabilă, schimbul de mesaje dintre client și server se realizează doar prin intermediul socket-ului.

```
{
  "sid": "FSDjX-WRwSA4zTZMALqx",
  "upgrades": ["websocket"],
  "pingInterval": 25000,
  "pingTimeout": 5000
}
```

Figură 3.3 Parametrii handShake pentru conexiunea de tip Socket.IO [18]

Parametrii stabiliți la handshake (figura 3.3) reprezintă:

- sid este id-ul sesiunii, care trebuie inclus în parametrul de interogare sid în toate cererile HTTP ulterioare;
- upgrades conține lista protocolelor de transport stabilite cu serverul;
- valorile pingInterval și pingTimeout sunt utilizate în mecanismul heartbeat.

### 3.4 Quill

Quill este un editor de text modern și cross-platform construit pentru compatibilitate și extensibilitate. Acest modul API oferă suport și pentru aplicațiile web, reprezentând un punct de plecare foarte important pentru construirea funcționalității de editare colaborative. Conținutul editorului este reprezentat în format JSON, fiind ușor de extras și de prelucrat. Modificările aduse de către utilizator sunt extrase sub formă de obiecte cu denumirea de *delta*[19].

Obiectele de tip *delta* sunt reprezentate într-un format simplu, dar expresiv, care poate fi utilizat pentru a descrie conținutul și modificările din Quill. Formatul este un subansamblu strict de tip JSON, lizibil pentru oameni și ușor de analizat

de către program. Obiectele *delta* pot descrie orice document generat de Quill, include toate informațiile de text și formatare, fără ambiguitatea și complexitatea formatului HTML. Funcționalitățile editorului folosite în proiect sunt:

- *getContents()* - preia conținutul editorului, cu date de formatare, reprezentate de un obiect *delta*;
- *getLength()* – returnează lungimea conținutului editorului;
- *getModule()* – returnează un modul care a fost adăugat editorului, în contextul actual este folosit pentru adăugarea modulului de afișare a cursorului;
- *updateContents()* – actualizează conținutul editorului pe baza obiectului *delta*;

De asemenea, modulul Quill pune la dispoziție un mecanism de detectarea a evenimentelor generate atunci când acesta își schimbă starea. Aceste evenimente pot fi tratate prin intermediul funcției „*quill.on(event, function(delta, oldDelta, source)*” . Evenimentele folosite în cadrul aplicației sunt:

- *text-change* – se emite atunci când conținutul Quill s-a schimbat. Sunt furnizate detalii privind modificarea, reprezentarea conținutului editorului înainte de modificare, precum și sursa modificării. Sursa va fi „*user*” dacă provine de la utilizator;
- *selection-change* – se emite atunci când un utilizator sau API determină modificarea selecției, cu un interval care reprezintă limitele selecției. Un interval *null* indică pierderea selecției (cauzată, de obicei, de pierderea focalizării editorului).

## 4 Structura proiectului

### 4.1 Cerințele sistemului

Aplicația propusă are scopul de a asigura munca de tip colaborativ dintre membrii unei organizații care lucrează cu documente clasificate. Din necesitatea de a-și asigura confidențialitatea, organizația nu poate utiliza servicii software existente care funcționează pe sisteme Cloud externe și este nevoie de o soluție software „on-premises”. Pentru a răspunde acestui caz de utilizare am proiectat și dezvoltat o aplicație de tip colaborativ, web based, care va răspunde setului de cerințe definite mai jos. O aplicație de tip web based, răspunde cerințelor de mobilitate impuse de activitatea membrilor organizației.

#### 4.1.1 Cerințe Non-Funcționale

Soluția software propusă va fi formată din două componente: aplicația client, de tip interfață web și aplicația server. Cerințele non-funcționale ce trebuie îndeplinite de sistem sunt prezentate mai jos.

**Ergonomie** – aplicația client trebuie să pună la dispoziție o interfață intuitivă prin care utilizatorul poate accesa ușor funcționalitățile puse la dispoziție.

**Confidențialitate** – datele utilizatorului sunt protejate prin anonimizarea accesului (datele personale ale utilizatorului sunt modificate).

**Securitate** – canalul de comunicație va fi securizat prin criptarea mesajelor fișierelor și a fluxului video. Accesul la aplicație se va face prin filtrarea adreselor IP legitime, iar în cazul încercărilor de acces neautorizat se vor produce alerte. Se va asigura un mecanism de protecție împotriva atacurilor de tip Cross-Site Request Forgery prin folosirea token-urilor de acces JWT.

**Performanță** – aplicația va suporta utilizarea simultană de către mai mulți utilizatori cu timpi de răspuns scăzuți pentru a nu afecta performanța serviciului și cu experiența utilizatorului.

**Portabilitate** – funcționalitățile asigurate de aplicație vor putea fi accesate din browser folosind orice dispozitiv de tip desktop conectat la internet.

**Flexibilitate** – toate datele vor fi salvate în baza de date și vor putea fi accesate la fiecare nouă sesiune a utilizatorului.

#### 4.1.2 Cerințe Funcționale

Aplicația propusă spre dezvoltare va trebui să respecte următoarele cerințe pentru atingerea scopului și asigurarea unui instrument facil de lucru:

**SRS<sup>1</sup>** – Soluția propusă este capabilă să fie accesată prin intermediul browser-elor Google Chrome, Mozilla Firefox și Microsoft Edge folosind sisteme hardware de tip desktop.

---

<sup>1</sup> SR – abreviere pentru Software Requirements Specification

**SRS 2** – Accesul în aplicație se va face folosind autentificare mutuală pe bază de certificate, respectiv credențiale (email și parolă). Fiecare utilizator beneficiază de un cont de utilizator creat de administrator și nu este disponibilă opțiunea de register din motive de securitate.

**SRS 3** – Căutarea de utilizatori pentru începerea conversațiilor de tip privat sau de conversații deja existente, rezultatele obținute în urma căutării fiind ordonate alfabetic în funcție de subșirul de căutare.

**SRS 4** – Asigurarea comunicației bidirectionale la nivel de mesaje text între doi utilizatori folosind canale de comunicație private.

**SRS 5** – Crearea unui mediu comun de stocare al fișierelor sub formă de folder pentru participanții unei conversații de tip privat. Acest *folder* va fi reprezentat utilizatorilor purtând numele interlocutorului.

**SRS 6** – Crearea de grupuri cu mai mult de doi participanți. La crearea unui grup nou, va fi creat și un mediu de stocare care va fi partajat cu fiecare nou utilizator adăugat.

**SRS 7** – Opțiune de adăugare de noi participanți la grupurile publice. Aceștia vor avea acces la spațiul de stocare specific grupului.

**SRS 8** – Trimiterea de fișiere la nivelul convesației sub formă de mesaj și posibilitatea de a vizualiza detalii despre acestea la nivelul convesației. Fișierele respective vor fi afișate și în spațiul comun de stocare, însotite de detalile aferente (denumire, dată creare, autor).

**SRS 9** – Fișierele partajate vor putea fi descărcate, atât de la nivelul convesației, cât și din spațiul de stocare comun de la nivelul sistemului de fișiere.

**SRS 10** – Utilizatorii pot iniția apeluri video-audio prin accesarea unei pagini dedicate din canalul de mesagerie. Pentru conectarea utilizatorilor la un apel video-audio este suficient accesarea adresei URL din browser care poate fi partajată și cu alți utilizatori care nu aparțin aceluiași grup sau convesații.

**SRS 11** – Funcționalitate de creare a fișierelor text și de editare simultană de către doi sau mai mulți utilizatori.

**SRS 12** – Conectarea utilizatorilor la fereastra de editare colaborativă se va face prin accesarea adresei URL a ferestrei prin intermediul browser-ului.

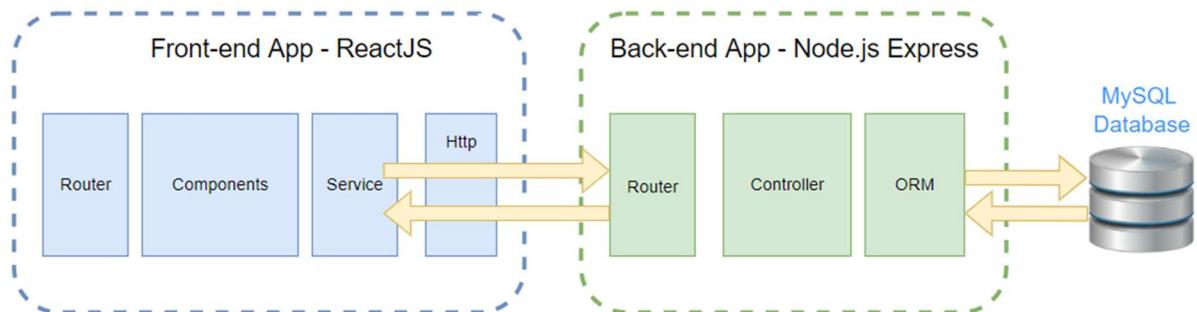
**SRS 13** – Interacțiunea simultană a utilizatorilor asupra documentelor de tip text va fi evidențiată prin metode specifice.

**SRS 14** – Fișierele de tip text, editate la nivelul sesiunii collaborative pot fi salvate și accesate oricând din spațiul de stocare de către membrii care au permisiuni asupra folderului în care este salvat.

## 4.2 Arhitectura sistemului

Soluția dezvoltată este formată din aplicația front-end, reprezentată în partea stângă a figurii 4.1 și aplicația server reprezentată în partea din dreapta. Utilizatorul interacționează cu aplicația interfață pentru a beneficia de funcționalitățile puse la dispoziție de aceasta, care la rândul său face request-uri

către server pentru procesare. Utilizatorul nu are acces în niciun moment la infrastructura serverului și nu poate interacționa cu aceasta.



Figură 4.1 Reprezentarea arhitecturii sistemului software

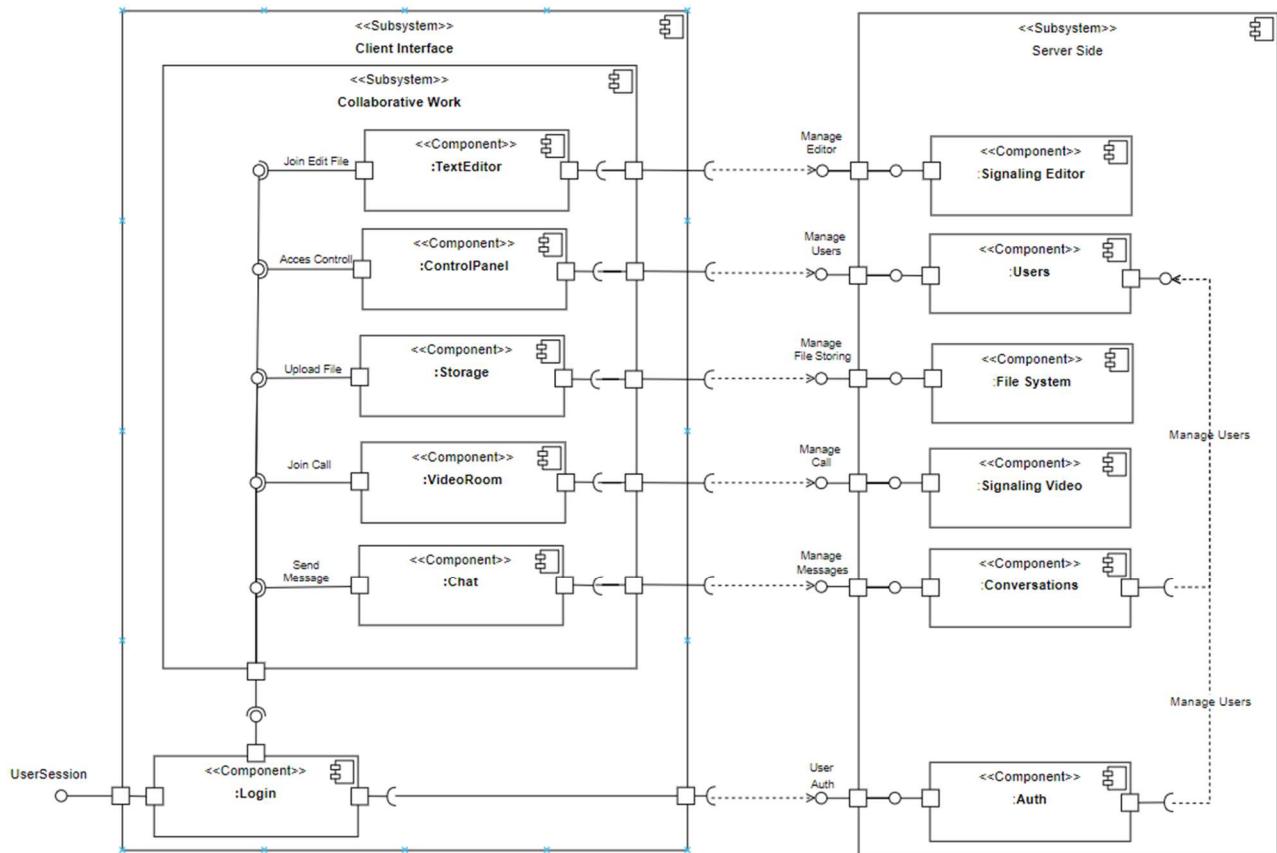
Pentru intermedierea accesului dintre utilizator și interfață voi folosi un server web Nginx care controlează accesul utilizatorilor prin solicitarea certificatului de utilizator la momentul accesării paginii. Astfel, accesul va fi permis doar pentru accesul la interfața web, iar serverul Nginx având rolul de API gateway. Reprezentarea pentru modelul infrastructurii va fi reprezentată în secțiunea de implementare a aplicației. În continuare voi descrie principalele părți componente ale sistemului folosind reprezentarea grafică de tip UML.

#### 4.2.1 Aplicația Web Client

Aplicația va pune la dispoziție o pagină pentru autentificarea în sistem a utilizatorului. Dacă se încearcă accesarea unei alte resurse din aplicație prin accesarea URL-ului respectiv, fără ca utilizatorul să fie deja autentificat, atunci acesta va fi redirecționat spre pagina de autentificare sau Login. După validarea credențialelor, utilizatorul poate accesa funcționalitățile aplicației reprezentate în diagrama componentelor reprezentate în figura 4.2.

Principalele componente ale aplicației sunt modulele:

- *Chat*;
- *VideoRoom*;
- *Storage*;
- *ControlPanel*;
- *TextEditor*.



Figură 4.2 Diagrama de componente a sistemului

**Chat** permite utilizatorului să acceseze conversații existente sau să creeze conversații noi la nivelul cărora să trimită mesaje de tip text sau fișiere multimedia. De asemenea, utilizatorul are opțiunea de a crea grupuri noi și de a adăuga noi utilizatori sau de a șterge grupurile deja existente.

**VideoRoom** oferă funcționalitatea de a genera apeluri video-audio la nivelul grupurilor, dar și adăugarea altor participanți din afara grupurilor prin accesarea adresei URL specifică paginii.

**Storage** pune la dispoziție un spațiu de stocare unde sunt salvate fișierele trimise la nivelul conversațiilor, iar pe lângă acestea utilizatorul poate uploada și fișiere proprii, private.

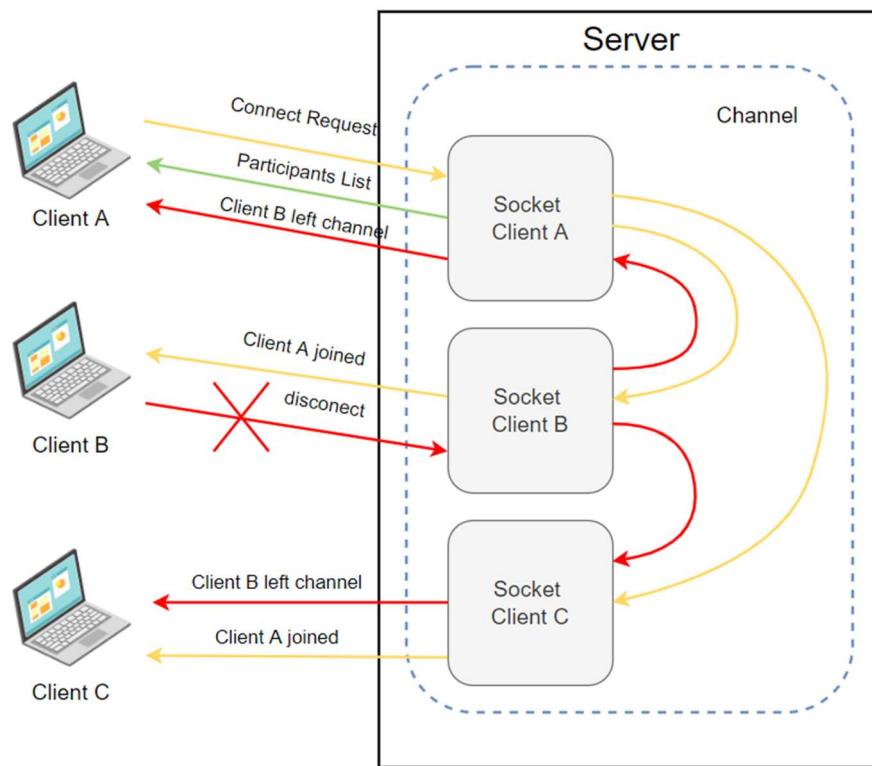
**ControlPanel** poate fi accesat doar de utilizatorii cu rol de administrator. Această pagină reprezintă un panou unde doar utilizatorii cu rol de administrator pot filtra restul utilizatorilor în funcție de grupurile în care sunt înrolați și restricționa accesul.

**TextEditor** este modulul destinat editării documentelor în mod colaborativ. Acesta permite utilizatorilor să interacționeze prin intermediul unui editor text, fiecare având un cursor personalizat pentru interacțiunea cu restul utilizatorilor. La finalul sesiunii de lucru, documentul este salvat în spațiul de stocare al autorului și va putea fi accesat doar de utilizatorii care au permisiuni de acces. De menționat că, pe timpul sesiunii de lucru pot fi adăugați oricără de mulți utilizatori prin distribuirea link-ului de acces al pagini.

#### 4.2.2 Aplicației Web Server

În partea dreaptă a figurii 4.2 sunt reprezentate componentele aplicației server. Back-end-ul aplicației este format din două servicii API de tratare al request-urilor care asigură partea de procesare a aplicației. Primul serviciu este sistemul pentru tratarea request-urilor de tip POST și GET, iar cel de-al doilea serviciu este sistemul de comunicație dintre utilizatori, bazat pe protocolul Websocket. Acesta este folosit pentru trimitera de mesaje, inițierea apelurilor video și editarea colaborativă.

Folosirea unui serviciu de comunicații bazat pe protocolul WebSocket este justificat de necesitatea transmisiei datelor aproape instant către grupurile corespunzătoare. Socket.IO, permite diferențierea fiecărui terminal conectat prin atribuirea unui *ID* unic care poate fi folosit la diferențierea și gruparea utilizatorilor pe canale de comunicație foarte ușor.



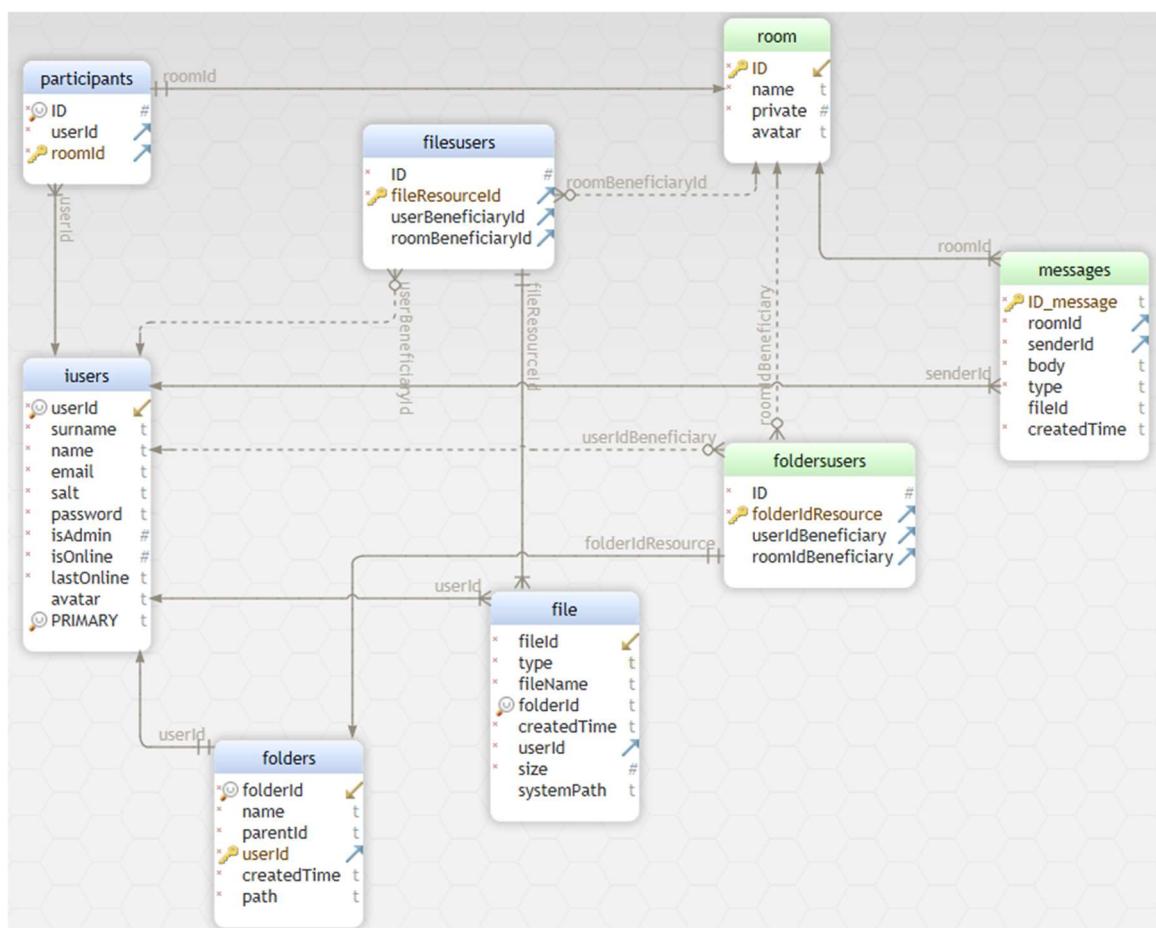
Figură 4.3 Fluxul de date în cadrul unui canal de comunicație Socket.IO

Astfel, pentru fiecare conversație de tip chat, modulul *Conversations* creează câte un nou canal pe baza *ID-ului* specific grupului, iar atunci când un utilizator se autentifică, acesta este conectat la toate canalele asociate conversațiilor sale din baza de date. După validarea criteriilor de verificare, se salvează la nivelul serverului *ID-ul* utilizatorului împreună cu *ID-ul* socket-ului și tipul canalului într-o lista pentru un managementul canalelor. În aceeași manieră, atunci când un utilizator se alătură unui apel video, acesta conectat la canalul respectiv prin intermediul modulului *Signaling Video* pentru a primii lista cu participanții care sunt deja conectați, Fig. 4.3. Pentru participarea la editarea unui document se

procedează în mod asemănător, dar folosind modului de Signaling Editor. În momentul în care socket-ul este deconectat este semnalat evenimentul la nivelul serverului pentru a-l exclude din lista utilizatorilor conectați.

#### 4.2.3 Structura bazei de date

Pentru stocarea datelor am ales utilizarea unei baze de date relaționale de tip MySQL, deoarece pe baza relațiilor ce se stabilesc între tabele, datele pot fi organizate în mod intuitiv, iar aplicația nu necesită stocarea unui volum foarte mare de date. Pentru crearea și administrarea bazei de date am utilizat utilitarul XAMPP, deoarece pune la dispoziție interfață grafică ușor de folosit. Baza de date este formată din opt tabele legate între ele prin intermediul constrângерilor de tip cheie-valoare.



Figură 4.4 Structura bazei de date

Tabelul *iusers* reține datele despre utilizatori. Pentru stocarea datelor despre conversații s-a folosit tabelul *room*. Pentru reprezentarea participării unui utilizator la o conversație este necesară completarea tabelului *participants* cu ID-ul grupului, respectiv al utilizatorului. Astfel, în cazul unei conversații private vor exista două intrări diferite în tabelul *participants*.

Tabelul *folders* reprezintă datele ce emulează structura de tip folder din cadrul spațiului de stocare. Pentru ca un utilizator să poată accesa un anumit folder, este necesară completarea unei noi intrări în tabelul *foldersusers*.

Tabelul *file* reține datele despre fișierele trimise de către utilizatori. Pentru a furniza utilizatorilor acces la un anumit fișier (în cazul în care se dorește partajarea unui anumit fișier) este nevoie să fie completat un nou set de date format din ID-ul fișierului, ID-ul utilizatorului și ID-ul grupului. Dacă fișierul aparține unei singure persoane, ID-ul grupului va fi cu valoarea *NULL*.

Pentru stocarea mesajelor va fi utilizat tabelul *messages*, în care vor fi stocate ID-ul grupului căruia aparțin, ID-ul autorului, timpul când a fost trimis și tipul de mesaj (dacă este un mesaje de tip text sau fișier).

NECLASIFICAT

## 5 Implementarea software

Pentru dezvoltarea aplicației am folosit utilitarul Visual Studio Code datorită editorului pus la dispoziție de acesta, ce ușurează munca prin evidențierea sintaxei, potrivirea parantezelor și identarea automată.

Mediul folosit pentru devoltarea aplicației a fost NodeJS *v16.11.0*, asistat de managerul de pachete NPM *v8.1.4* și alte biblioteci, respectiv framework-uri specificce Javascript enumerate în capitolul „API-uri folosite”. Datorită utilizării limbajului Javascript, pentru dezvoltarea aplicației Client am avut de ales paradigmă de programare funcțională în detrimentul celei orientate pe obiecte.

Programarea funcțională este o paradigmă de programare declarativă ce constă în folosirea de funcții pure, mai exact funcții ce primesc variabile de intrare pe care nu le modifică și generează altele noi ca rezultat. Astfel, rezultatul unei funcții pure depinde doar de parametrii de intrare, fără să producă impact asupra celorlalte componente din cod. Programarea orientată pe obiecte este o paradigmă de programare care organizează datele și structura software-ului pe baza conceptului de clase și obiecte.

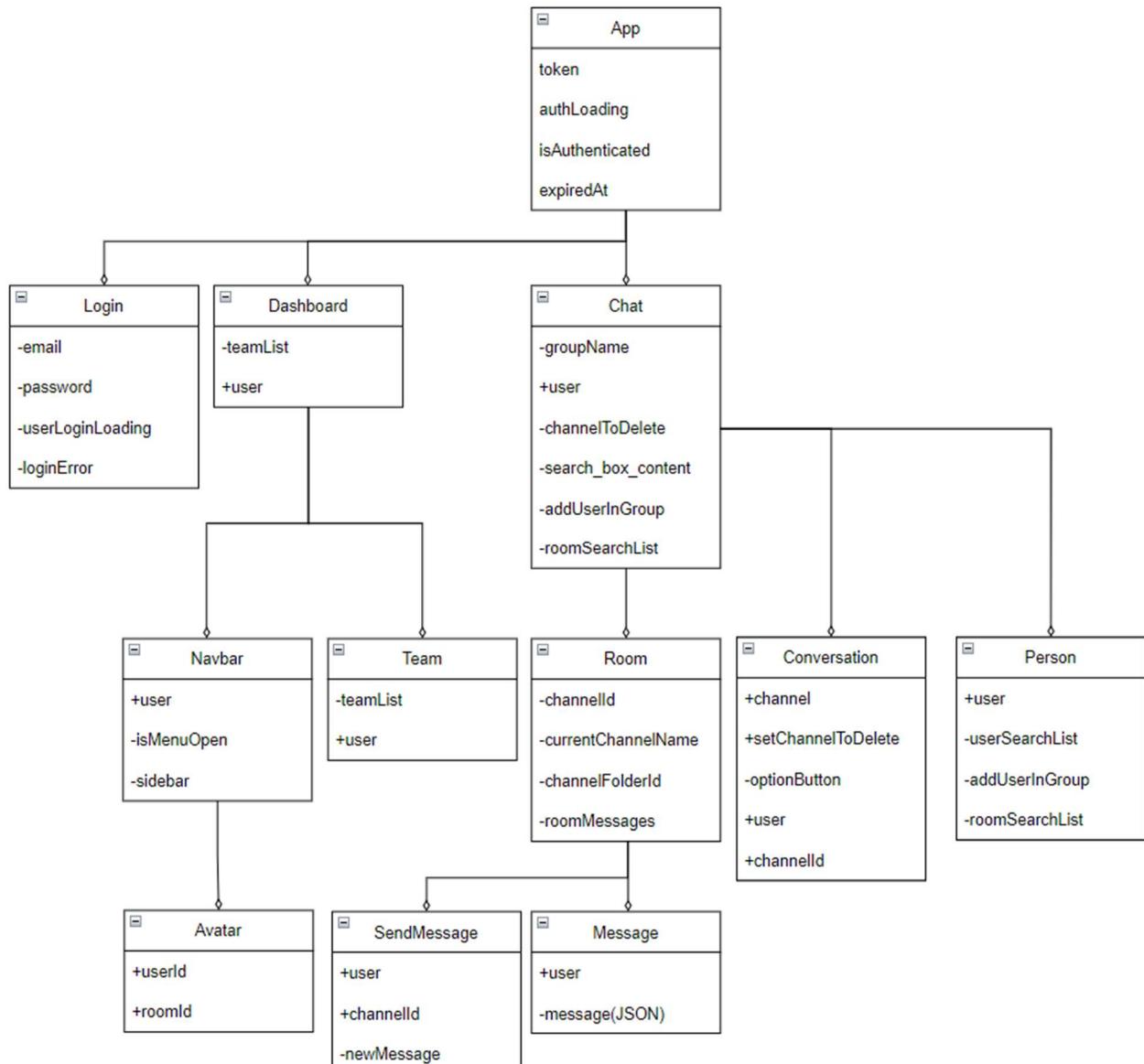
### 5.1 Aplicația Web Client

Componentele permit divizarea interfeței utilizatorului în unități izolate, independente și reutilizabile. O componentă funcțională React se comportă ca o funcție JavaScript prin acceptarea de argumente sub formă de *props* și returnarea rezultatului sub formă de elemente React.

#### 5.1.1 Definirea Componentelor

Pentru implementarea software a aplicației Client am alcătuit diagrama componentelor funcționale React, reprezentată în figurile Fig. 5.1, 5.3 și Fig. 5.4. Funcțiile folosite pentru crearea de request-urilor asupra server-ului se află în directorul *asyncActions*.

Principala componentă a aplicației este *App*, aceasta este prima care este redată atunci când se accesează aplicația. Principalul scop al acestei componente este de a redirecționa utilizatorul spre pagina corespunzătoare link-ului URL generat de acțiunile acestuia, folosind *BrowserRouter*, o bibliotecă de rutare standard de tip Javascript. La nivelul acestei pagini se realizează și requestul de verificare al token-ului de acces pentru utilizator. Verificarea se realizează în momentul expirării token-ului sau atunci când pagina suferă un refresh la nivelul browser-ului prin intermediul funcției *verifyTokenAsync()*. În cazul în care token-ul prezent în browser nu este validat de Server, atunci utilizatorul este deconectat de la sesiune și se revine la pagina de Login. Dacă token-ul este validat, atunci acesta este înlocuit cu unul nou și salvat în Redux până la următoarea verificare. De menționat că această verificare se produce la fiecare request al Clientului către server pentru a preveni atacurile de tip Cross-Site Request Forgery.



Figură 5.1 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea I

Paginiile ce pot fi accesate de către utilizator sunt reprezentate de componente:

- *Login*;
- *Dashboard*;
- *Chat*;
- *TextEditor*;
- *VideoRoom*;
- *ControlPanel*;
- *Storage*;
- *Support*.

Componența **Dashboard** afișează toate conversațiile publice ale utilizatorului, oferind posibilitatea de navigare spre acestea. Funcția folosită în cadrul metodei pentru afișarea grupurilor este *getSearchRoomService()*, care întoarce lista grupurilor în format JSON.

Pagina folosită pentru gestionarea mesajelor are ca suport componenta ***Chat***, care este alcătuită din trei componente: *Room*, *Conversation* și *Person*. Funcțiile principale de interacțiune cu aplicația server de la nivelul acestei componente sunt:

- *userSetRoomListAsync()* – întoarce lista sortată a grupurilor ale căror nume conține textul introdus în caseta de căutare;
- *userSearchPersonListAsyn()* – primește ca răspuns sub formă de listă persoanele ale căror nume conține textul introdus în caseta de căutare;
- *CreateNewGroup()* – are ca funcționalitate creare unui grup nou cu numele transmis ca parametru.

Modificările aduse stării din Redux sunt procesate folosind funcțiile:

- *setUserSearchBoxContent()* – salvează valoarea keyword-ului de căutare în reducer-ul *chatRedu*.
- *UpdateAddUserInGroup()* – reține *ID*-ul persoanei ce urmează să fie adăugate în grup. Dacă nu există nici o persoană selectată atunci va fi inițializat cu un sir gol;
- *setPersonSearchList()* – salvează lista cu persoanele căutate de către utilizator.

***Room*** conține lista cu mesaje corespondente fiecărei conversații și mecanismul de trimitere de mesaje noi. Pentru actualizarea mesajelor de la nivelul conversației a fost utilizată funcția *getMessagelistTime()*, cu rolul de a întoarce lista parțială a mesajelor în funcție de un reper temporal și poziția față de acesta. Scopul dezvoltării acestei funcții este actualizarea etapizată a listei de mesaje la evenimentul de scroll executat de utilizator, evitând astfel transmiterea întregului flux de date al mesajelor.

Pagina cu rol de editare colaborativă are la bază componenta ***TextEditor***, ce permite afișarea utilizatorilor care participă la editare și interacțiunea dintre aceștia. Documentul editat este salvat într-o formă persistentă prin utilizarea funcției *saveTextfileAsync()*, ce primește ca argumente datele utilizatorului și conținutul casetei de editorului. Restul căilor de comunicație cu partea de backend sunt asigurate prin intermediul bibliotecii Socket.IO. Pentru conectarea la grupul specific documentului se transmite evenimentul „join room” cu transmisia request-ului ce conține *userId*-ul, *roomId*-ul și tipul grupului.

```
socketRef.current.emit(
  "join room",
  {
    userId: user.userId,
    roomId: fileId,
    type: "edit",
  },
  (error) => {
    if (error) {
      console.log(error);
    }
  }
);
```

Figură 5.2 Trimiterea request-ului de conectare la sesiunea de editare colaborativă

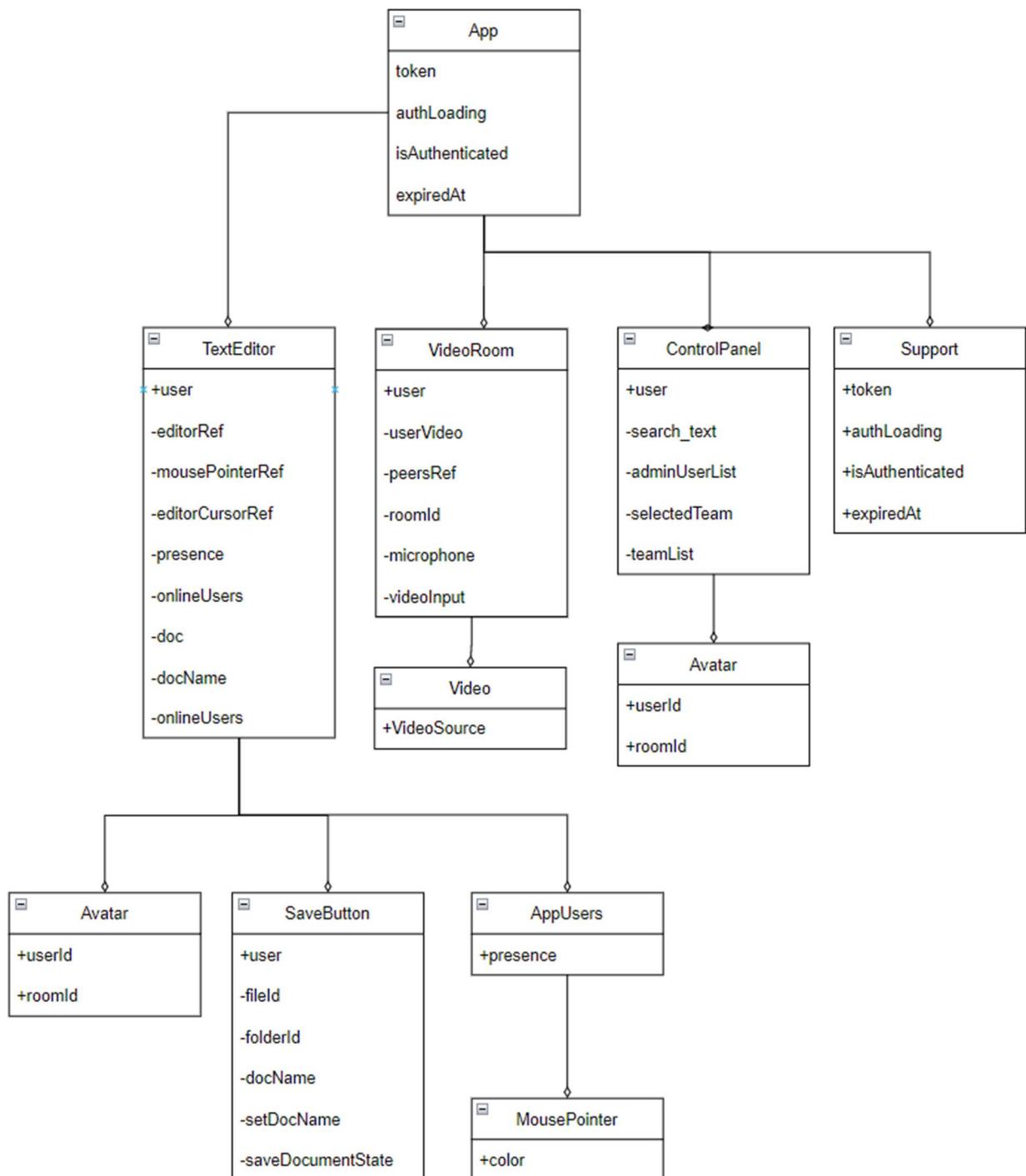
În urma validării făcute de server (verificarea identității utilizatorului), este trimisă lista cu utilizatorii deja conectați la grupul respectiv prin tratarea evenimentului „*all users edit*”. Transmiterea datelor către server și invers are loc în aceeași manieră prin folosirea metodelor `socketRef.current.emit()` și `socketRef.current.on()`, unde `socketRef` reprezintă referința spre socket-ul deschis pentru comunicație. Extragerea și formatarea textului a fost realizată prin folosirea metodelor specifice utilitarului Quill:

- `editorRef.current.on("text-change", (delta, oldDelta, source));`
- `editorRef.current.on("selection-change", (range, oldRange, source));`
- `editorRef.current.updateContents()`, unde `editorRef.current` este referința către obiectul de tip Quill.

Evenimentul „text-change” detectează formatarea textului, și returnează noul format al textului prin variabila delta, iar parametrul `source` reprezintă *sursa de la care a venit modificarea*.

**AppUsers** este folosită pentru afișarea cursoarelor respectivilor utilizatori pe baza coordonatelor absolute preluate de la nivelul containerului în care este definit editorul text, din componenta *TextEditor*.

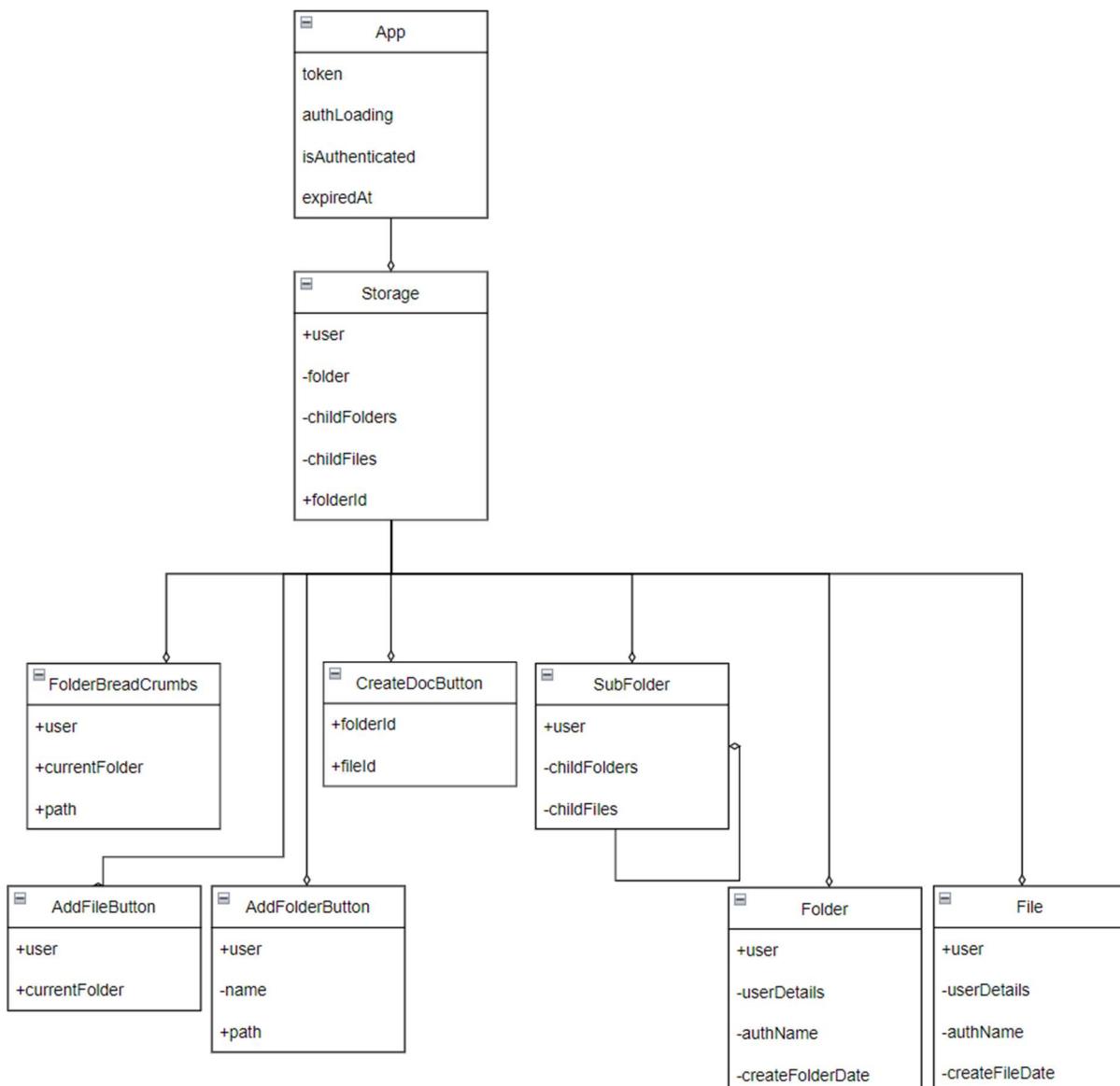
Componenta **Support** are doar rol informativ pentru accesarea repository-ului de Github unde se poate vizualiza codul sursă al proiectului.



Figură 5.3 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea 2

Componenta **VideoRoom** poate fi accesată din cadrul ferestrei de **Chat**, dar și prin simpla accesarea a adresei URL generată anterior de către alt utilizator. La nivelul acestei componente utilizatorul solicită conectarea la canalul grupului de tip video prin intermediul conexiunii asigurate de către biblioteca Socket.IO. După conectare va primii o listă cu utilizatorii înrolați deja în convorbire și va încerca să stabilească conexiuni de tip peer-to-peer cu aceștia pentru transmiterea fluxului video-audio.

Spațiul de administrare al fișierelor este cuprins de componenta ***Storage***. Lista de fișierelor și a folderelor sunt actualizate prin utilizarea funcțiilor: *userSetFolderList()*, respectiv *userSetFileList()*. Principalele funcționalități de la nivelul paginii cu același nume sunt încărcarea de fișiere și crearea de foldere, asigurate de componente ***AddFilebutton***, respectiv ***AddFolderButton***. Fișierele și folderele sunt definite de către componente ***Folder***, respectiv ***Files***. Navigarea la nivelul folderelor se realizează prin intermediul componentei ***FolderBreadCrumbs***, care afișează calea curentă pentru folderul curent și permite accesarea tuturor folderelor precedente prin accesare directă.



Figură 5.4 Diagrama obiectelor de tip React de la nivelul aplicației client – Partea 3

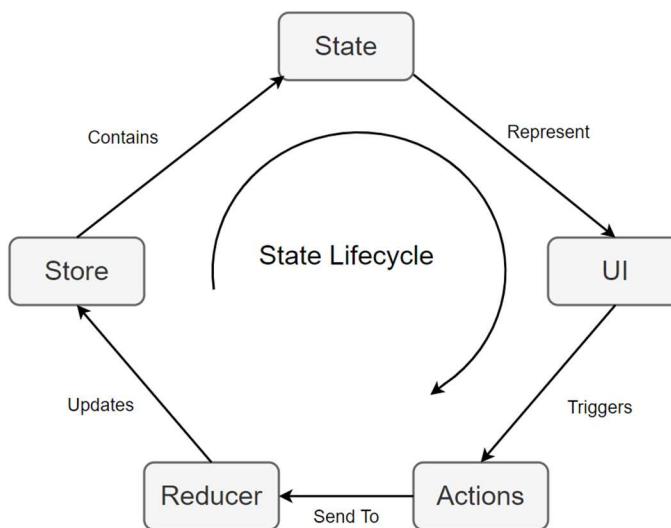
De asemenea, pentru vizualizarea structurii spațiului de stocare este folosită componenta ***SubFolder***. Aceasta primește ca parametru *ID-ul* unui folder și returnează toate subfolderele și fișierele pe care le conține, iar apoi pentru fiecare subfolder este redată din nou componenta ***Subfolder***. Procesul este terminat atunci când subfolderele terminale nu mai conțin alte subfoldere sau fișiere.

Navigarea la nivelul paginilor se face prin intermediul componentei ***Navbar***. Aceasta conține un sidebar cu toate paginile ce pot fi accesate și un al meniu de acțiuni rapide dedicate utilizatorului.

***ControlPanel*** este o componentă cu rol de control asupra conturilor utilizatorilor și poate fi accesată doar de administratori. Aceasta oferă mecanisme de căutare, filtrare, editare și stergere a conturilor de utilizator.

### 5.1.2 React-Redux

Fluxul datelor la nivelul componentelor este unidirecțional, începând de la componente superioare spre componente inferioare. Astfel, nu pot fi transmise date între componentele aflate pe același nivel, ci doar dacă acestea sunt transmise prin intermediul unei componente aflate pe un nivel superior. Pentru rezolvarea acestui inconvenient am folosit librăria React Redux, ce permite stocarea variabilelor într-un mediu de stocare, ***Store***, accesibil de către orice componentă a aplicației. Atunci când se dorește modificarea unei variabile din ***Store***, se folosește modulul ***Reducer***. La fiecare modificare a variabilelor din ***Store***, componentele aplicației sunt redate, aplicând astfel noile modificări. Acest proces este exemplificat în figura Fig. 5.5.



Figură 5.5 Principiu de funcționare al sistemului React Redux

În implementarea aplicației am ales să divizez componenta de ***Reducer*** în trei subcomponente diferite pentru dezvoltarea modulară a proiectului:

- *Auth*;
- *ChatRedu*;
- *FolderRedu*;

***Auth*** are rolul de a menține starea datelor utilizatorului precum: token-ul de sesiune, timpul de valabilitatea al token-ului, datele personale ale utilizatorului și dacă acesta este autentificat sau nu.

***ChatRedu*** menține starea variabilelor corespunzătoare componentelor legate de mesageria text precum: *ID*-ul conversației deschise, numele conversației

deschise, lista mesajelor afişate în conversație, *ID*-ul folderului corespondent conversației, textul după care se face căutarea participanților și lista cu conversațiile deschise ale utilizatorului.

```

import { combineReducers } from "redux";
import auth from "./authReducer";
import chatRedu from "./chatReducer";
import folderRedu from "./folderReducer";

// to combine all reducers together

const appReducer = combineReducers({
  auth,
  chatRedu,
  folderRedu,
});

export default appReducer;

```

*Figură 5.6 Componentele de tip Reducer de la nivelul apluației client*

**FolderRedu** face referire la starea variabilelor legate de sistemul de stocare al fișierelor. Acesta reține starea variabilelor precum: *ID*-ul, numele, și calea folderului spre care a navigat utilizatorul, lista subfolderelor și a subfișierelor corespunzătoare folderului curent în care se află utilizatorul.

### 5.1.3 Definirea Request-urilor

Pentru realizarea request-urilor de comunicație cu partea de back-end s-a folosit biblioteca *Axios*. Cele două sisteme rulează pe aceeași mașină fizică, astfel comunicația dintre ele se poate realiza folosind request-uri de tip HTTP. Funcțiile folosite pentru efectuarea request-urilor sunt de tip asincron, întrucât este necesar așteptarea procesării cererii și răspunsul serverului. Am definit patru rute principale prin care cele două sisteme realizează comunicație de tip request:

- „/users” – efectuează toate cererile ce țin de utilizator și acțiunile sale (cum ar fi autentificarea, verificarea token-ului sau interacțiunea cu gestionarea mesageriei);
- „/room” - canalizează toate cererile specifice conversațiilor și grupurilor, aducerea din memorie a mesajelor sau a listelor cu participanți;
- „/document” – specifică pe request-urile ce țin de previzualizarea imaginilor, accesarea fișierelor, sau descărcarea de documente;
- „/folder” – conține cererile ce țin de sistemul de fișiere, exemplu fiind returnarea listei cu subfoldere sau fișiere, respectiv crearea de foldere noi.

Răspunsul cererii poate fi trimis într-un format personalizat, iar pentru semnalarea erorilor se pot seta coduri de eroare care să fie interpretate de către

aplicația Client. Aceste coduri de eroare reprezintă și un foarte bun mecanism de tratare al erorilor în cazul inconsistenței răspunsului.

```
// insert new member in a group in database
export const addNewMemberInRoomService = async (roomId, userSearchListId) => {
  try {
    return await axios.post(` ${REACT_APP_API_URL}/room/newmember`, {
      roomId,
      userSearchListId,
    });
  } catch (err) {
    return {
      error: true,
      response: err.response,
    };
  }
};
```

Figură 5.7 Exemplu request de tip post pentru adăugare unui utilizator nou în grup

## 5.2 Aplicația web server

### 5.2.1 Tratarea request-urilor

Aplicația server răspunde request-urilor aplicației client prin intermediul serverului de tip HTTP și a serverului de tip socket.io. Organizarea rutelor pentru request-urile HTTP a fost făcută prin intermediul utilitarului *Router()* din cadrul framework-ului express(), rezultând fișierul *routes* cu fișierele: *users.js*, *room.js*, *folder.js* și *file.js*. În aceste fișiere se răspunde fiecărui request specific cu funcția corespondentă din folderul *controllers*. Înainte de prelucrare, se execută funcția middleware de verificare a JWT.

```
router.post("/details", authMiddleware, GetUserDetails); ← controller
router.post("/newuser", authMiddleware, InsertNewUserAccount); ← middleware
router.post("/edit", authMiddleware, EditUserAccountAsync);
```

Figură 5.8 Folosirea funcției middleware pentru verificarea autenticității request-ului

Serverul pentru comunicația webSocket este deschis pentru conectare prin utilizarea metodei „*on*(“connection”, *(socket)* => {} )”. În cadrul parantezelor sunt definite tipurile de request la care răspunde serverul prin metode de tipul: *socket.on(“join room”, *async (request)*)*. Tipurile de request definite sunt:

- „*join room*” – clientul solicită conectarea pentru un grup de tipul chat, video sau edit. Dacă respectă criterile de validare atunci acesta este adăugat în listă și socketul său asociat este conectat la canalul respectiv de comunicație;
- „*send chat message*” – clientul trimită un mesaj în cadrul mesageriei. Dacă procesul este validat, mesajul este redirecționat spre restul membrilor din grup și apoi este stocat la nivelul bazei de date;
- „*typing chat message*” – clientul semnalizează activitatea din cadrul casetei de editare pentru trimitere a mesajelor;

- „*send doc edit*” – clientul trimite modificările efectuate asupra documentului, care sunt trimise printr-un mesaj de tip broadcast către toți clienții conectați în cadrul canalului la momentul respectiv;
- „*send doc pointer*” – clientul trimite coordonatele cursorului de editare către restul participanților din cadrul editării colaborative;
- „*send doc presence*” – clientul trimite coordonatele pointer-ului cu scopul de a semnala interacțiunea la nivelul ferestrei de interacțiune;
- „*sending signal*” – utilizatorii ce au intrat în cadrul unei con vorbiri de tip video efectuează cereri către restul participanților pentru a stabili conexiuni de tip peer-to-peer;
- „*returning signal*” – reprezintă răspunsul de acceptare al conexiunii peer-to-peer de către participanți;
- „*disconnect*” – acest tip de eveniment este declanșat atunci când socket clientului este deconectat. Tratarea unui astfel de eveniment este foarte importantă, deoarece utilizatorul corespunzător socketului trebuie eliminat din lista canalelor de comunicații.

### 5.2.2 Baza de date MySQL

Interacțiunea cu baza de date se realizează prin intermediul funcțiilor din cadrul folderului *services*. Acestea sunt clasificate după modelul de rutare în categoriile: *Files*, *Folder*, *Room* și *User*. Pentru conectarea la baza de date am folosit metoda *createPool()* din cadrul bibliotecii *mysql*, asociate NodeJS. *CreatePool()* creează un spațiu de stocare al conexiunilor cu baza de date, iar atunci când este solicitată o conexiune dintr-un obiect de tip *pool()*, se va primi ca răspuns o conexiune liberă. Când limita de conexiuni este atinsă, se va aștepta eliberarea unei conexiuni pentru execuția interogării SQL.

Prevenirea atacurilor de tip SQL injection se face prin utilizarea metodei *mysql.format()*, ce are rol de sanitizare a string-urilor de tip query. Un astfel de exemplu este redat în Fig. 5.9, interogare ce are rolul de a determina lista participanților din cadrul unei conversații.

```

function GetPartListData(channelId) {
  let selectQuery = "SELECT * FROM ?? INNER JOIN ?? ON ?? = ?? WHERE ?? = ?";
  let query = mysql.format(selectQuery, [
    "participants",
    "iusers",
    "participants.userId",
    "iusers.userId",
    "roomId",
    channelId,
  ]);
  return new Promise((resolve, reject) => {
    sqlPool.pool.query(query, (err, result) => {
      if (err) {
        return reject(err);
      }
      return resolve(result);
    });
  });
}

```

Figură 5.9 Interogare SQL pentru returnarea listei de participanți din cadrul unui grup

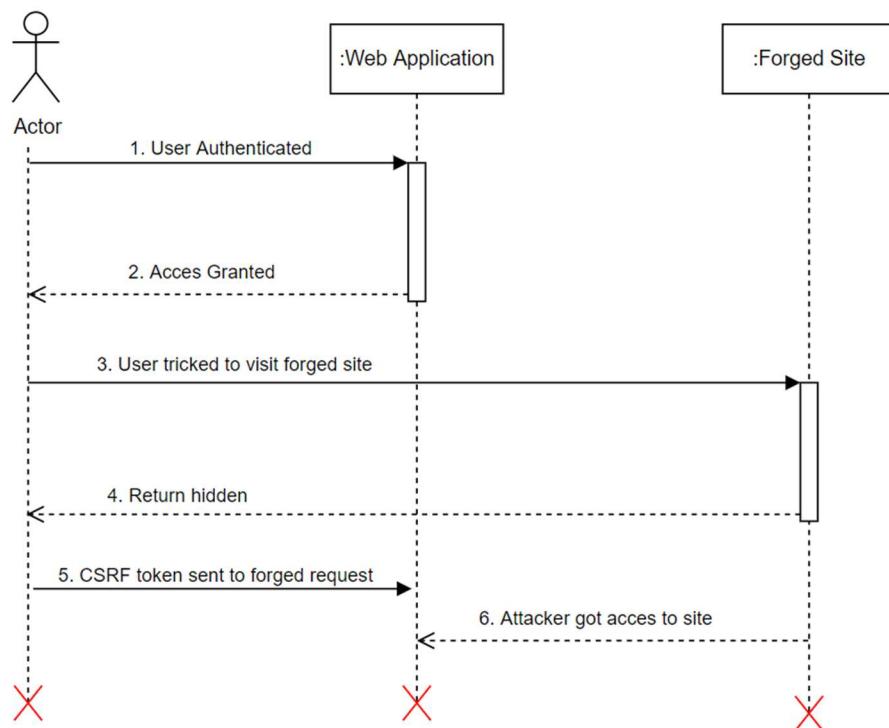
## 5.3 Metode de securizare

### 5.3.1 Protecție împotriva atacurilor CSRF

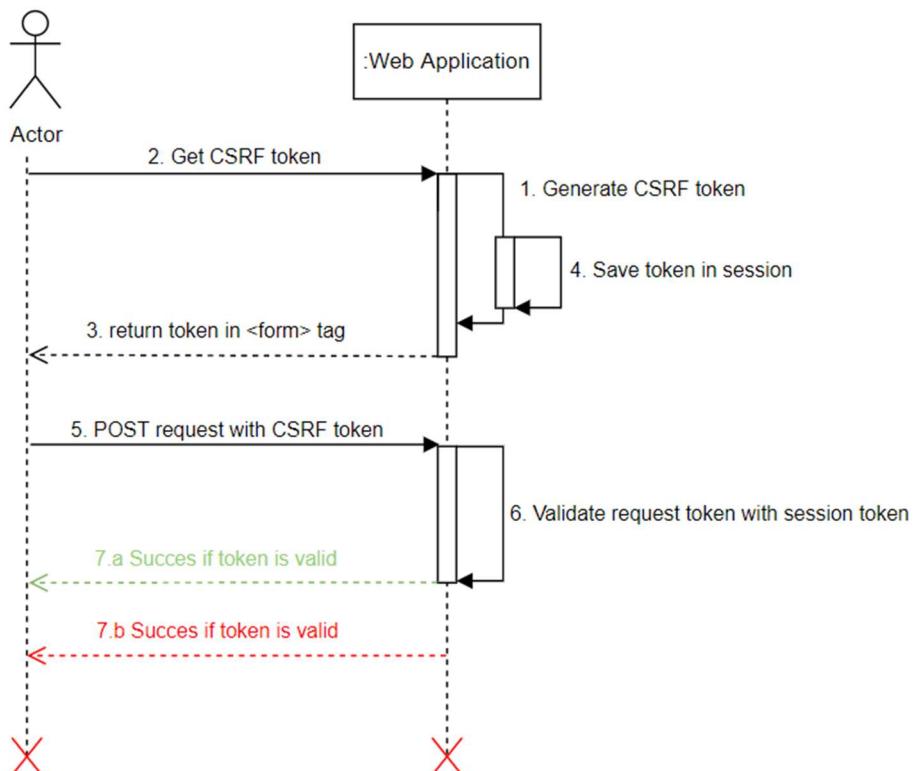
Atacurile de Tip Cross-Site Request Forgery constau în furtul credențialelor de sesiune ale utilizatorilor autentificați și utilizarea lor pentru efectuarea de acțiuni în cadrul site-ului web. Pentru ca un astfel de atac să fie realizat cu succes, site-ul web trebuie să identifice utilizatorii prin intermediul cookie-urilor, iar utilizatorii să acceseze link-ul plasat de către atacator. Pentru prevenirea unor astfel tipuri de atacuri am implementat un sistem bazat pe token-uri de acces Json Web Token.

Sistemul funcționează în felul următor: în momentul autentificării utilizatorului este generat un obiect de tip JWT cu numele *refreshToken*, un *accessToken*. *RefreshToken* este folosit pentru a permite utilizatorului să genereze noi token-uri de tip *accessToken*, cât timp acesta este valid. Odată cu *accessToken* este generat și un timp de expirare al acestuia ce este transmis utilizatorului, iar atunci când expiră clientul va face un request pentru generarea unui nou token de tip *accessToken*. Dacă *refreshToken*-ul a expirat atunci sesiunea utilizatorului va fi opriță și se va solicita autentificarea (*refreshToken*-ul are un timp mai mare).

În momentul când se efectuează un request, token-ul de tip *accessToken* va fi transmis în antetul requestului și va fi verificat prin intermediul unei funcții de tip middleware. Această funcție se execută înainte de procesarea requestului, iar dacă *accessToken*-ul prezent în antetul request-ului nu este validat se răspunde cu un mesaj de eroare.



Figură 5.10 Diagramă atac de tip CSRF

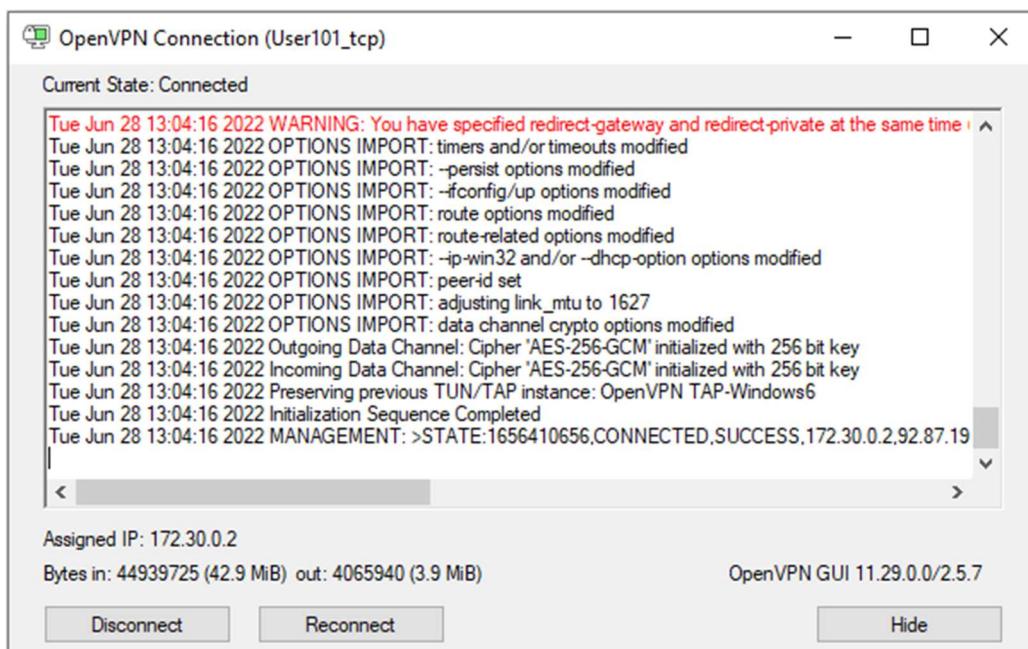


Figură 5.11 Securizarea sesiunii folosind JWT token

### 5.3.2 OpenVPN

OpenVPN este un proiect open-source ce permite crearea de rețele tip VPN. Acest tip de rețea permite comunicația prin utilizarea conexiunilor securizate bazate pe protocole SSL/TSL.

Pentru găzduirea sistemului am configurat un server fizic din infrastructura locală pentru instalarea și rularea sistemului software dezvoltat. Pentru securizarea accesului asupra infrastructurii a fost necesară instalarea serviciului OpenVPN și crearea de profile specifice destinate utilizatorilor aplicației. Pentru configurare a fost necesară generarea datelor pentru autoritatea centrală (cheia privată a CA-ului, respectiv certificatul), iar pentru clienți au fost generate 20 de profile de VPN pe baza configurației serverului de VPN. Pentru conectarea, utilizatorii sunt nevoiți să se conecteze la serviciul de VPN folosind profilul asignat prin utilizarea aplicației OpenVPN. În figura 5.12 se poate observa conectarea utilizatorului *User101\_tcp* la rețeaua de VPN. În cadrul rețelei utilizatorul are atribuită adresa 172.30.0.2.



Figură 5.12 Rularea serviciului la nivelul utilizatorului pentru accesarea aplicației

Adresa IP publică a clientului conectat prin VPN este 92.87.195.226 (adresa publică fără serviciul de VPN este 37.251.223.233), iar aceasta corespunde cu adresa IP a serverului de VPN. În figura 5.13 este redată conexiunea asupra serverului prin intermediul utilitarului Putty.

```

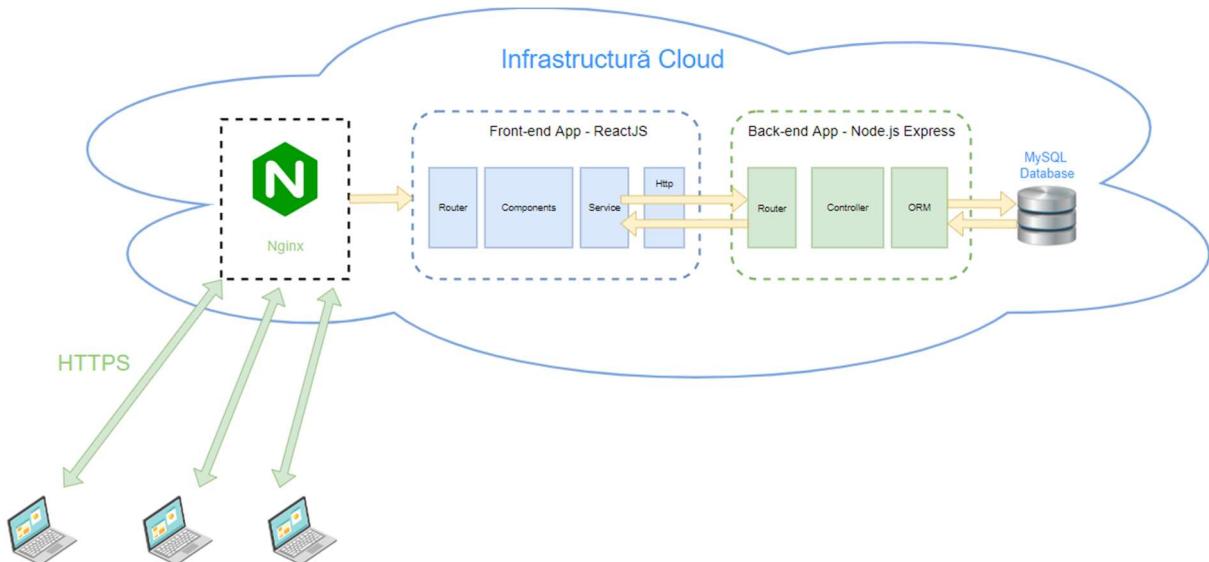
root@practicaATM:~#
root@practicaATM:~# login as: root
[...]
Pre-authentication banner message from server:
| This system is a restricted area. Changes without ATM permissions
| are strictly prohibited. Illegal actions will be prosecuted.
| End of banner message from server
[...]
Authenticating with public key "root@practicaATM"
Passphrase for key "root@practicaATM":
This system is a restricted area. Changes without ATM permissions
are strictly prohibited. Illegal actions will be prosecuted.
Last login: Tue Jun 28 12:29:10 2022 from 172.30.0.2
This system is a restricted area. Changes without ATM permissions
are strictly prohibited. Illegal actions will be prosecuted.
[root@practicaATM ~]# dig +short myip.opendns.com @resolver1.opendns.com
92.87.195.226
[root@practicaATM ~]# 

```

Figură 5.13 Sesiune de administrare a serverului folosind conexiune SSH

### 5.3.3 Autentificare mutuală

Accesul la aplicație este intermediat prin utilizarea unui server Nginx. Acesta are rolul de a controla comunicația dintre utilizator și aplicația client prin realizarea unei conexiuni securizate de tip HTTPS.



Figură 5.14 Mod de interacțiune folosind comunicația securizată

Serverul Nginx redirecționează comunicația de pe portul extern 443 către endpoint-ul aplicației client ( acesta fiind *127.0.0.1:3000*). De asemenea pentru acceptul comunicării clientul trebuie să prezinte un certificat semnat cu credențialele serverului Nginx. Pentru configurația acestei setări trebuie completate următoarele intrări din fișierul „*nginx.conf*”:

```

ssl_certificate "/etc/pki/nginx/server.crt";
ssl_certificate_key "/etc/pki/nginx/private/server.key";
ssl_session_cache shared:SSL:lm;
ssl_client_certificate "/etc/pki/nginx/client_ca.crt";
ssl_verify_client on;

ssl_session_timeout 10m;
ssl_ciphers PROFILE=SYSTEM;
ssl_prefer_server_ciphers on;

```

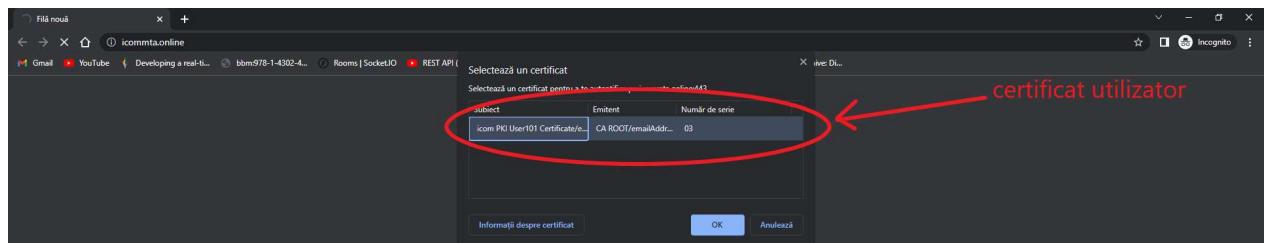
Figură 5.15 Fișierul de configurare al serverului de Nginx

În momentul accesării site-ului, browser-ul solicită utilizatorului să aleagă un certificat valid pentru accesarea paginii.



Figură 5.16 Accesarea aplicației fără un deținerea certificatului de utilizator

Certificatul trebuie importat în browser pentru a putea fi selectat, în caz contrar utilizatorul nu va putea realiza autentificarea mutuală.



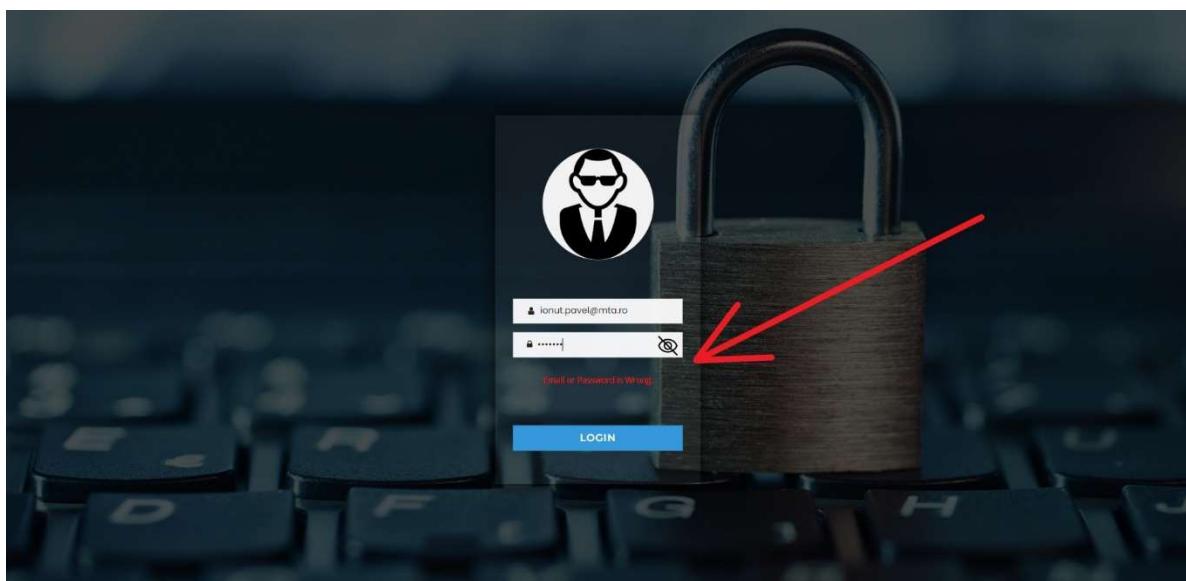
Figură 5.17 Solicitarea certificatului de utilizator la nivelul browser-ului

NECLASIFICAT

## 6 Testarea sistemului

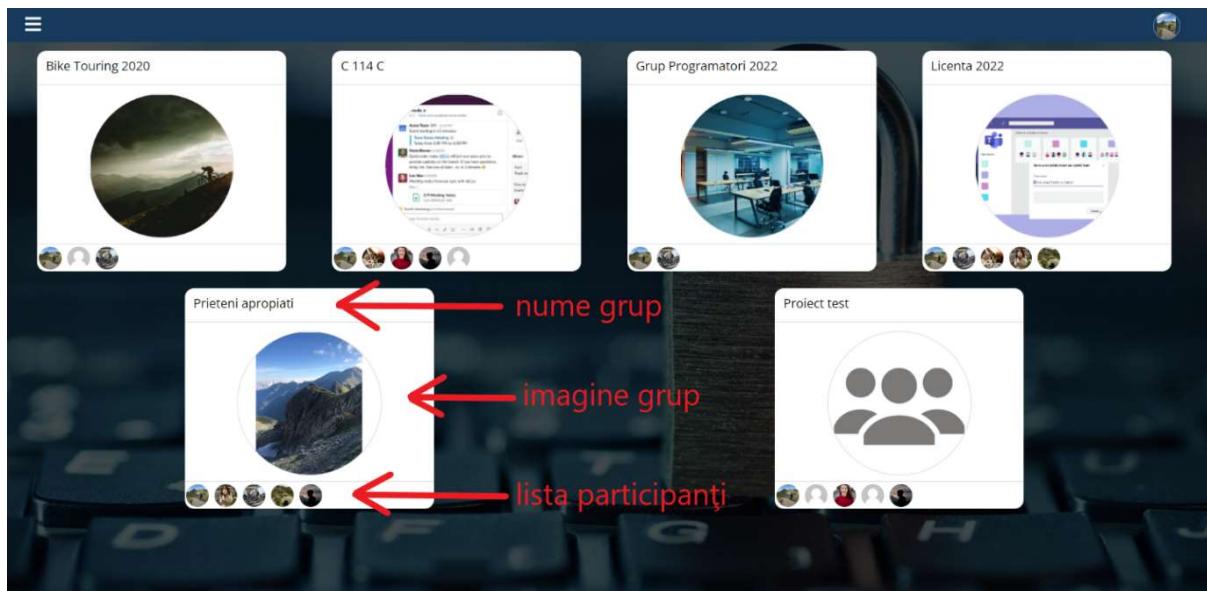
### 6.1 Utilizarea aplicației

Design-ul aplicației este unul intuitiv și ușor de folosit, asemănându-se cu alte aplicații web. Pentru autentificare utilizator este nevoie să-și introducă credențialele contului de utilizator, mai exact adresa de email și parola atribuită contului. În cazul în care utilizatorul introduce greșit credențialele acesta va primi un mesaj de notificare și va fi nevoie să completeze din nou câmpurile. Pe lângă acest pas, este necesar ca utilizatorul să aibă importat în browser certificatul personal și să fie conectat la serviciul de VPN prin folosirea profilului specific. În caz contrar acesta nu va putea accesa platforma.



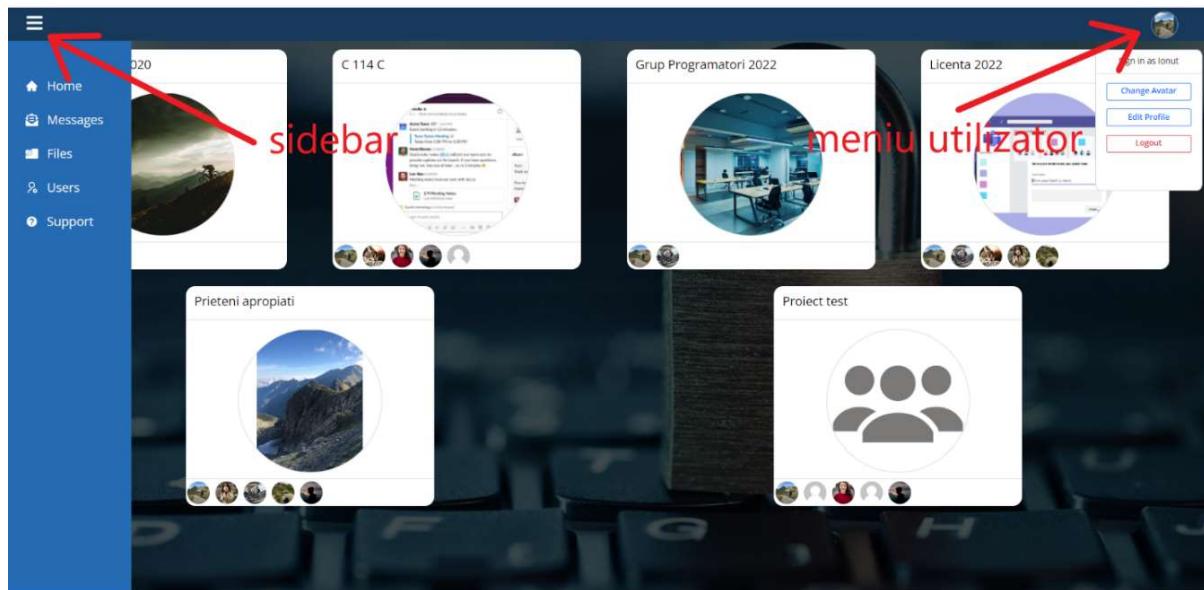
Figură 6.1 Pagina de Login a aplicației

După realizarea autentificării utilizatorul este redirecționat spre *homepage*, unde sunt listate toate grupurile publice (nu și conversațiile private) din care face parte, împreună cu lista de utilizatori corespunzătoare fiecărui. Utilizatorii sunt reprezentați cu ajutorul imaginilor de profil. Utilizatorii sau grupurile care nu au setată o imagine de profil, au atribuită o iconiță setată standard. Pentru accesarea conversațiilor este suficient ca utilizatorul să selecteze unul din grupurile afișate, iar astfel va fi redirecționat spre fereastra de mesagerie a grupului.



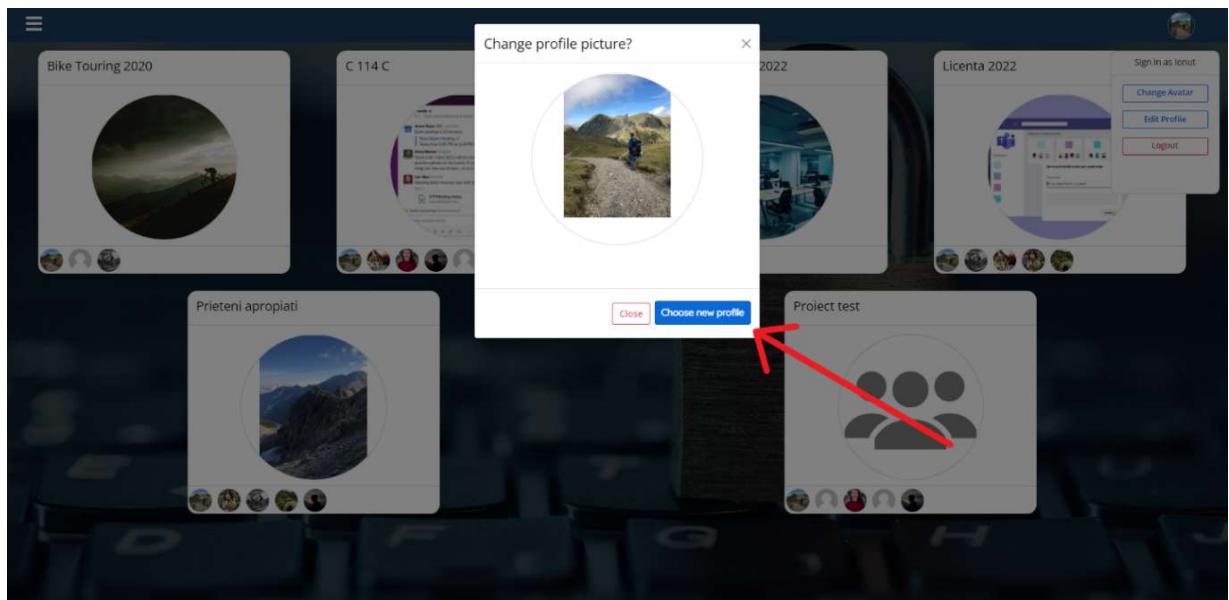
Figură 6.2 Prezentare Fereastra de Homepage

Pentru navigarea spre restul paginilor se folosește sidebar-ul din partea stângă. Acesta poate fi accesat prin apăsarea butonului de meniu prezent în navbar. Prin intermediul acestui meniu se poate naviga între ferestrele: *home*, *messages*, *files*, *support* și *users* (doar pentru utilizatorii cu rol de administrator).



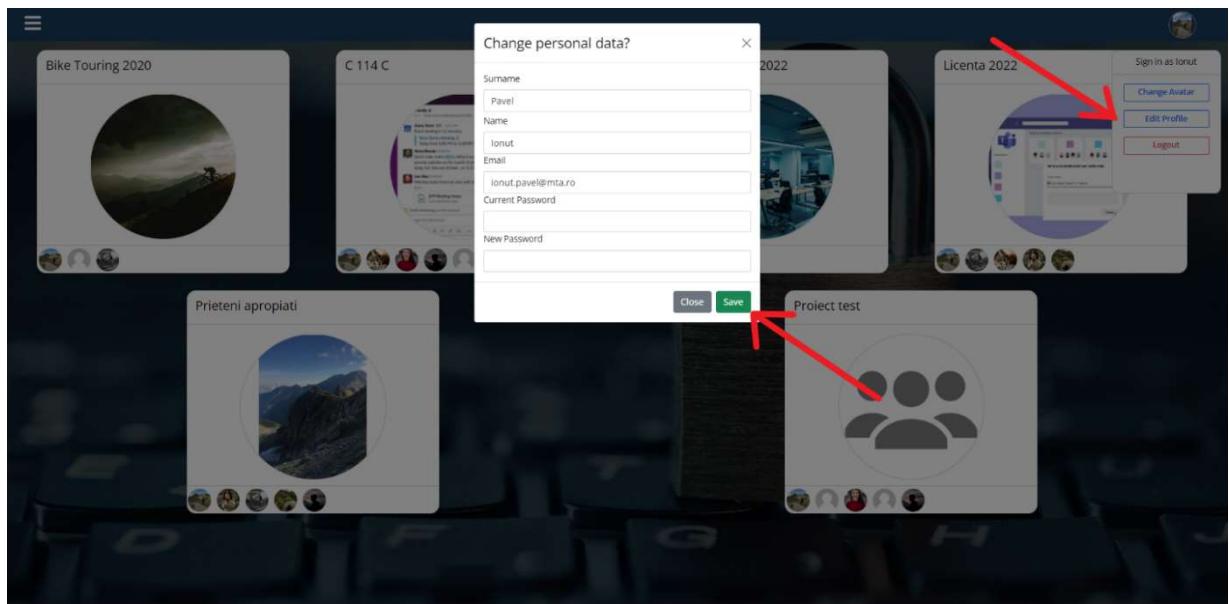
Figură 6.3 Accesarea meniului de navigație

Meniul dedicat utilizatorului poate fi deschis prin click asupra profilului din navbar și prezintă opțiuni de schimbare a imaginii de profil, de editare a datelor personale și de ieșire din aplicație. Aceste meniuri pot fi închise folosind butoanele specifice sau prin click în exteriorul lor. Schimbarea pozei de profil se face prin alegerea unei noi imagini și confirmarea salvării modificărilor.



Figură 6.4 Prezentare funcționalitate de schimbare a profilului

Pentru editarea datelor contului, utilizatorul este nevoit să introducă parola actuală, altfel modificările nu vor fi salvate. Pentru închiderea ferestrei, se poate folosi butonul dedicat sau se poate efectua click în afara componentei de editare.



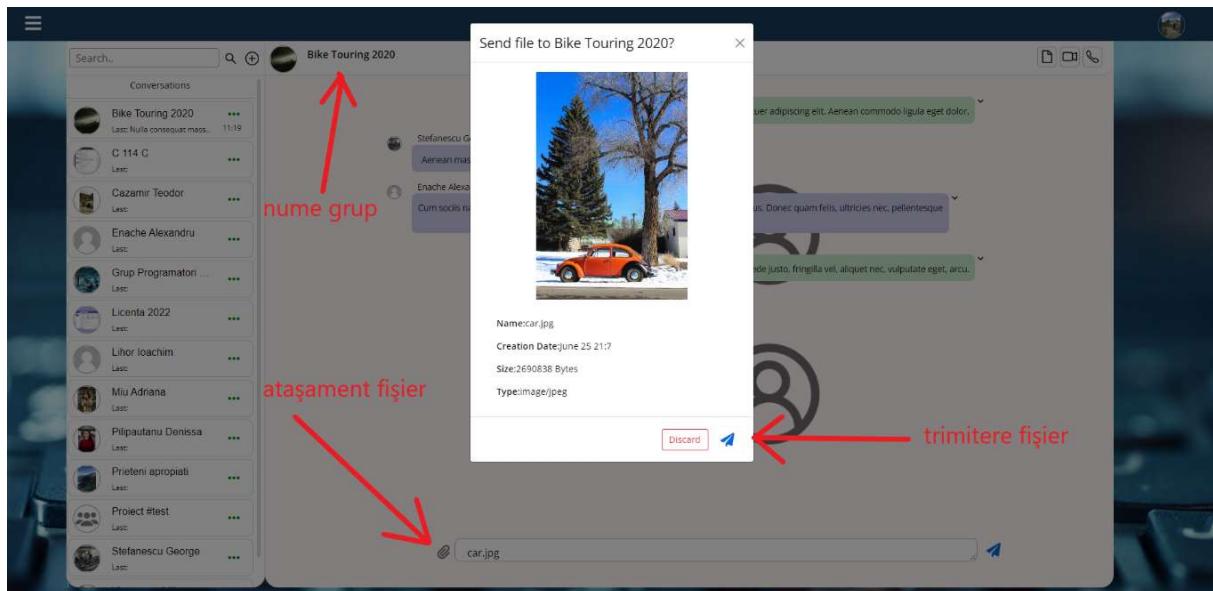
Figură 6.5 Prezentare funcționalitate de editare a datelor de utilizator

După accesarea butonului de mesagerie, utilizatorul este redirectionat spre zona de chat a aplicației. Aceasta are posibilitatea de a căuta conversații existente sau de a începe noi conversații cu ajutorul casetei de *search*. Rezultatele căutării sunt sortate în ordine alfabetică în funcție de keyword-ul căutat. Pentru fiecare conversație deschisă este afișat ultimul mesaj alături de ora la care a fost transmis și un buton pentru mai multe opțiuni.



Figură 6.6 Accesarea serviciului de mesagerie

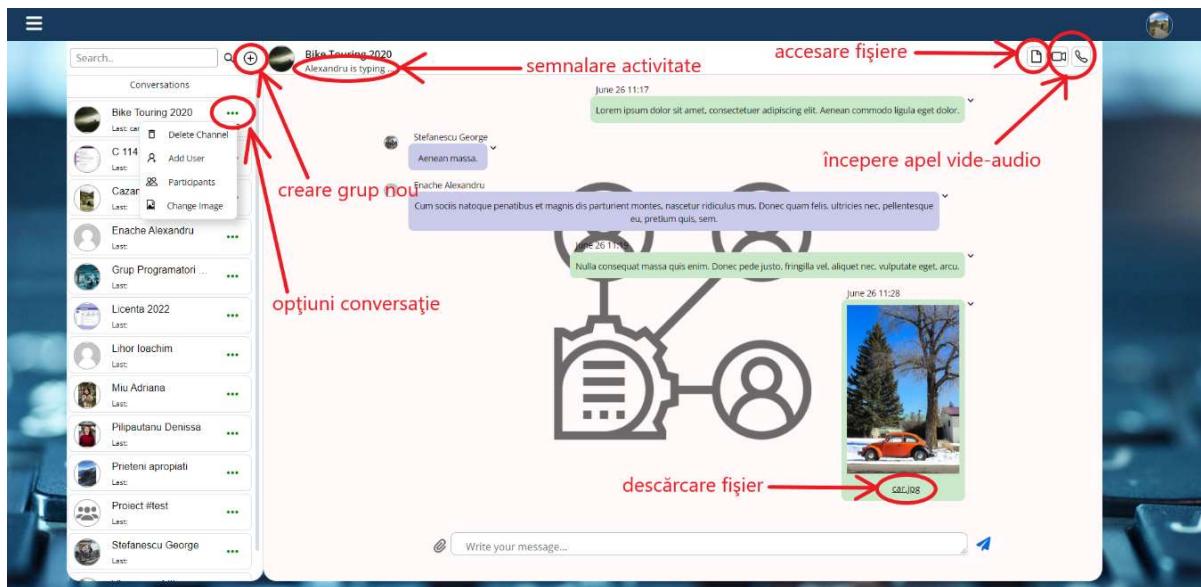
După accesarea unei conversații, în partea din dreapta a paginii este deschisă caseta cu mesaje a grupului. Utilizatorul are posibilitatea de a trimite mesaje de tip text, dar și atașament de fișiere folosind instrumentele din partea de jos a casetei. După selectarea fișierului dorit pentru trimitere, se va afișa o previzualizare a fișierului împreună cu detalile despre acesta.



Figură 6.7 Exemplu de trimitere a unui fișier în cadrul conversației

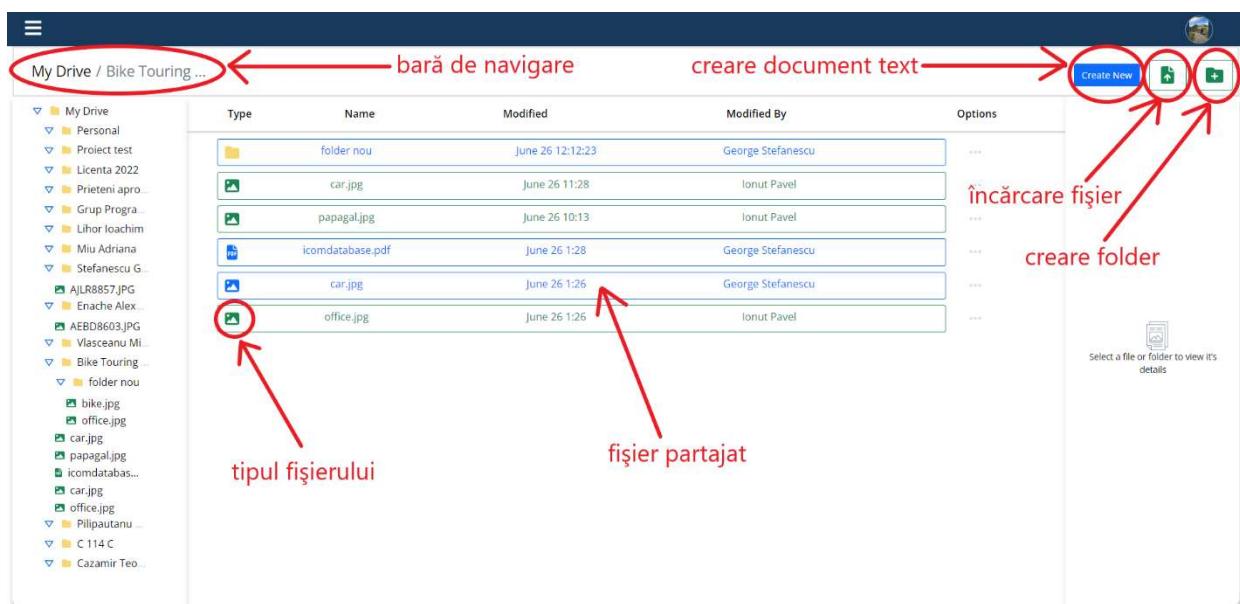
Pentru a crea grupuri noi, utilizatorul trebuie să acceseze butonul cu iconița în formă de cruce din dreptul casetei de search, iar pentru a obține detalii despre un anumit grup poate accesa butonul de *opțiuni conversație*. Fișierele pot fi descărcate folosind link-ul aflat în corpul mesajul, iar pentru vedea tot spațiul de stocare trebuie să acceseze butonul *accesare fișiere* din partea dreaptă. Intrarea într-un apel video se realizează prin folosirea butoanelor de *începere apel video*.

## NECLASIFICAT



Figură 6.8 Prezentarea funcționalităților din cadrul ferestrei de mesagerie

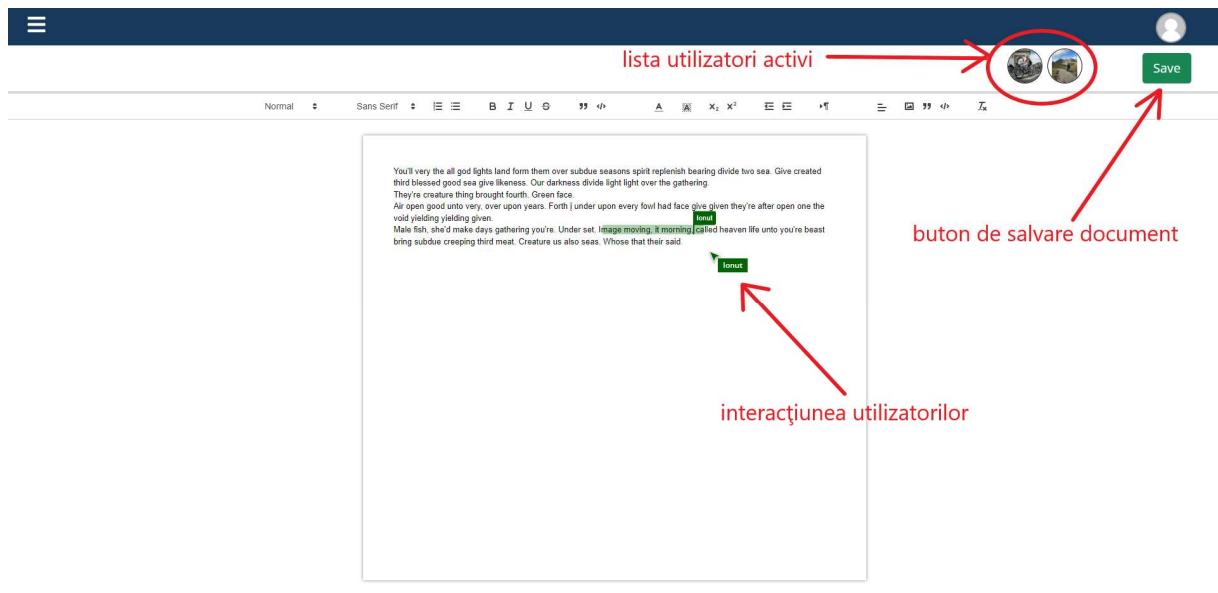
Spațiul de stocare simulează un sistem de fișiere prin modul de organizare și reprezentare. Fiecare conversație sau grup este reprezentat printr-un folder în care sunt afișate toate fișierele ce au fost trimise la nivelul conversației. Utilizatorul are și posibilitatea de a-și crea foldere personale în folderul rădăcină numit *MyDrive*. Folderele și fișierele create în interiorul unui folder public pot fi accesate și de restul membrilor. Fișierele încărcate de alți utilizatori vor fi marcate printr-un chenar de culoare albastră. Încărcarea unui fișier nou se realizează prin intermediul butonului *încărcare fișier*, iar crearea unui folder nou prin intermediul butonului *creare folder*. Pentru a începe editarea unui document text se accesează butonul de *create new*.



Figură 6.9 Prezentarea sistemului de stocare pentru fișiere

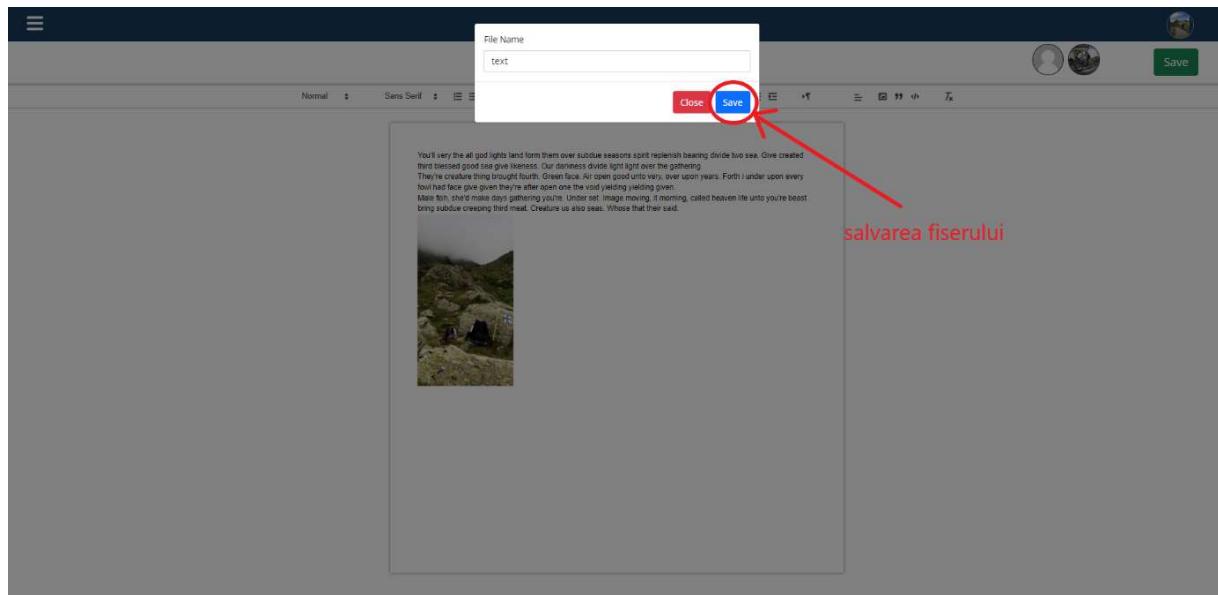
Caseta de text oferă diverse comenzi de formatare a textului. Lista cu utilizatori activi este afișată în colțul din dreapta al paginii alături de butonul de salvare.

## NECLASIFICAT



Figură 6.10 Sesiune de editare colaborativă a textului

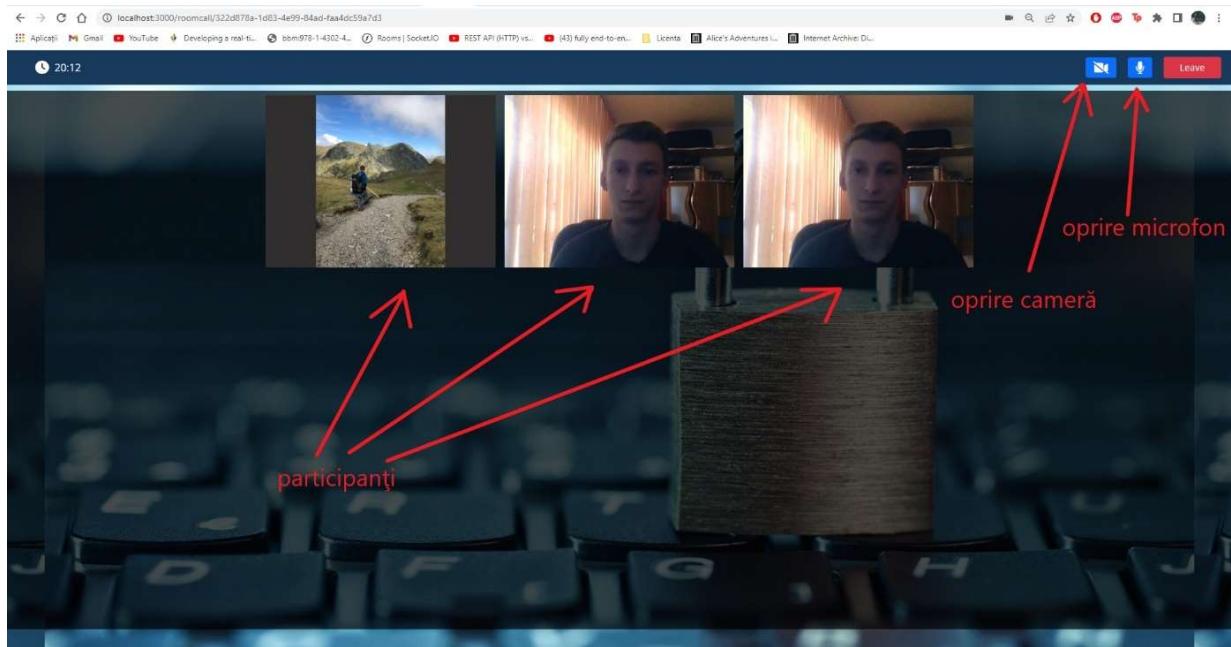
Fiecare utilizator care participă la editarea documentului poate utiliza pointerul colorat într-o culoare diferită de a celorlalți. O alta caracteristică funcțională este că fiecare utilizator poate folosi un cursor de text colorat în aceeași culoare pentru a evidenția fragmente de text.



Figură 6.11 Salvarea documentului generat în cadrul sesiunii de editare

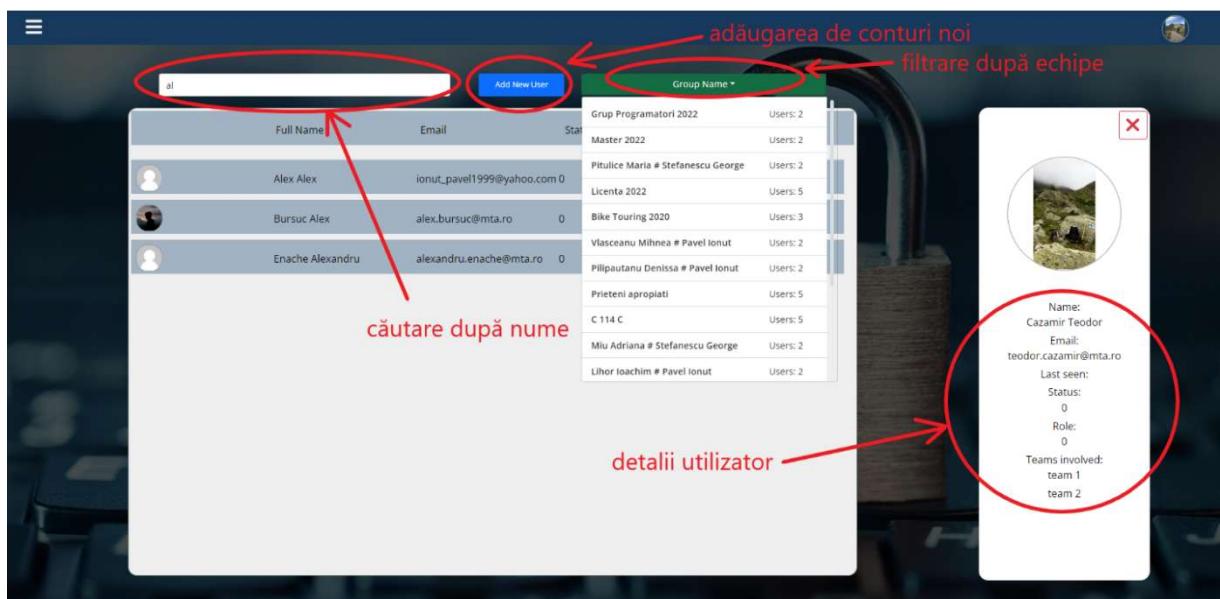
În fereastra de apeluri video, participanții au posibilitatea de a controla camera și microfonul folosind butoanele dedicate. Pentru participanții care își opresc fluxul video imaginea este înlocuită cu poza acestora de profil. În figura de mai jos sunt trei participanți în convorbire: primul are camera oprită, iar ceilalți doi împart același flux video, deoarece testarea a fost realizată pe infrastructura locală de dezvoltare. Cu alte cuvinte, a fost folosit același suport fizic pentru simularea participării în cadrul apelului. Părăsirea apelului se poate face prin butonul de *Leave* sau prin accesarea altelor pagini din aplicație.

## NECLASIFICAT



Figură 6.12 Prezentarea sesiunii de comunicație video-audio

Pentru administrarea conturilor de utilizatori, administratorii au permisiunea de a accesa fereastra de *Users* prin folosirea navbar-ului (opțiune disponibilă doar pentru administratori). În cadrul ferestrei administratorii au posibilitatea de a căuta utilizatori după numele acestora folosind caseta de căutare sau prin folosirea filtrului pus la dispoziție de meniul de tip dropdown. Pentru crearea unui cont nou de utilizator trebuie accesat butonul de *Add New User*, iar apoi completează câmpurile formularului cu datele utilizatorului. De asemenea, administratorii pot edita sau șterge conturile utilizatorilor, în funcție de necesitate.



Figură 6.13 Fereastra de control destinată administratorilor

## 6.2 Testarea sistemului

Aplicația a fost supusă mai multor serii de teste pentru a verifica funcționalitatea componentelor, majoritatea dintre acestea desfășurându-se în etapa de dezvoltare. Testele ce urmău să fie efectuate pentru mediul de producție nu au putut să fie realizate din cauza unei deficiențe de ordin tehnic prezente în etapa de implementare (aplicația back-end necesită rularea unui server de tip HTTPS pentru conectarea modului Socket.IO), astfel în mediul de producție s-au efectuat doar teste legate de posibilitatea de accesare a platformei. Pentru realizarea testelor legate de componente software este necesară instalarea mediului de execuție NodeJS v16.11.0 (pentru rularea celor două aplicații) și crearea bazei de date MySQL de tip MariaDB. Pentru detalierea testelor finale efectuate a fost întocmit următorul raport de testare organizat sub formă tabelară.

Nr. Test	Nr. Cerință	Descriere Cerință	Metoda de Testare	Comportament Așteptat	Rezultatul Obținut
T1	SRS 1/ SRS2	Utilizarea aplicației folosind Google Chrome.	Se accesează link-ul aplicației prin Google Chrome.	Browser-ul va solicita introducerea unui certificat valid.	Nu poate să acceseze link-ul, decât după ce este selectat certificatul de client stocat în store.
T2	SRS 1/ SRS2	Utilizarea aplicației folosind Mozilla FireFox.	Se accesează link-ul aplicației prin Mozilla FireFox.	Browser-ul va solicita introducerea unui certificat valid.	Nu poate să acceseze link-ul, decât după ce este selectat certificatul de client stocat în store.
T3	SRS 3	Căutarea de utilizatori pentru inițierea unei conversații noi.	Se utilizează searchbox-ul din mesagerie pentru căutarea unei persoane.	Pentru că utilizatorul introduce caractere, vor să apară persoanele care conțin sintagma introdusă.	Rezultatele afișate pot fi corecte și să pot crea conversații private prin accesarea rezultatelor în mod individual.
T4	SRS 4	Comunicația dintre utilizatori folosind mesaje text.	La nivelul conversației private se trimit mesaje text și se verifică dacă sunt primite de către destinatar.	Destinatarul va fi înțeleasă să primească mesajele în ordinea corectă și sub același format în care au fost trimise.	Destinatarul a receptat mesajele în ordinea corectă în momentul în care acestea au fost transmise.
T5	SRS 5	Accesarea fișierelor trimise la nivelul conversației.	După crearea unei conversații noi utilizatorul A trimite utilizatorului B un fișier. După primirea fișierului utilizatorul B va accesa butonul pentru accesarea storage-ului din cadrul conversației.	Utilizatorul B va fi direcționat în zona mediului de stocare unde va putea accesa fișierul trimis în cadrul conversației din lista de elemente afișate.	Sistemul prezintă toate fișierele trimise în cadrul conversației sub forma unei liste, conform specificațiilor cerinței.
T6	SRS 6	Crearea unui grup cu mai mult de 2 utilizatori.	Se accesează butonul pentru crearea unui grup nou și se completează cu numele grupului.	Grupul nou creat apare în lista conversațiilor.	Grupul poate să fie accesat, se pot trimite mesaje și poate să fie accesată zona de storage.
T7	SRS 7	Adăugarea de participanți noi.	Se accesează opțiunea de adăugare participanți noi, apoi se alege o persoană din lista afișată.	Utilizatorul nou adăugat poate accesa grupul din lista de conversații împreună.	Utilizatorul poate primi și trimite mesaje. Acestea sunt accesibile la toate fișierele trimise până atunci în cadrul grupului.

**NECLASIFICAT**

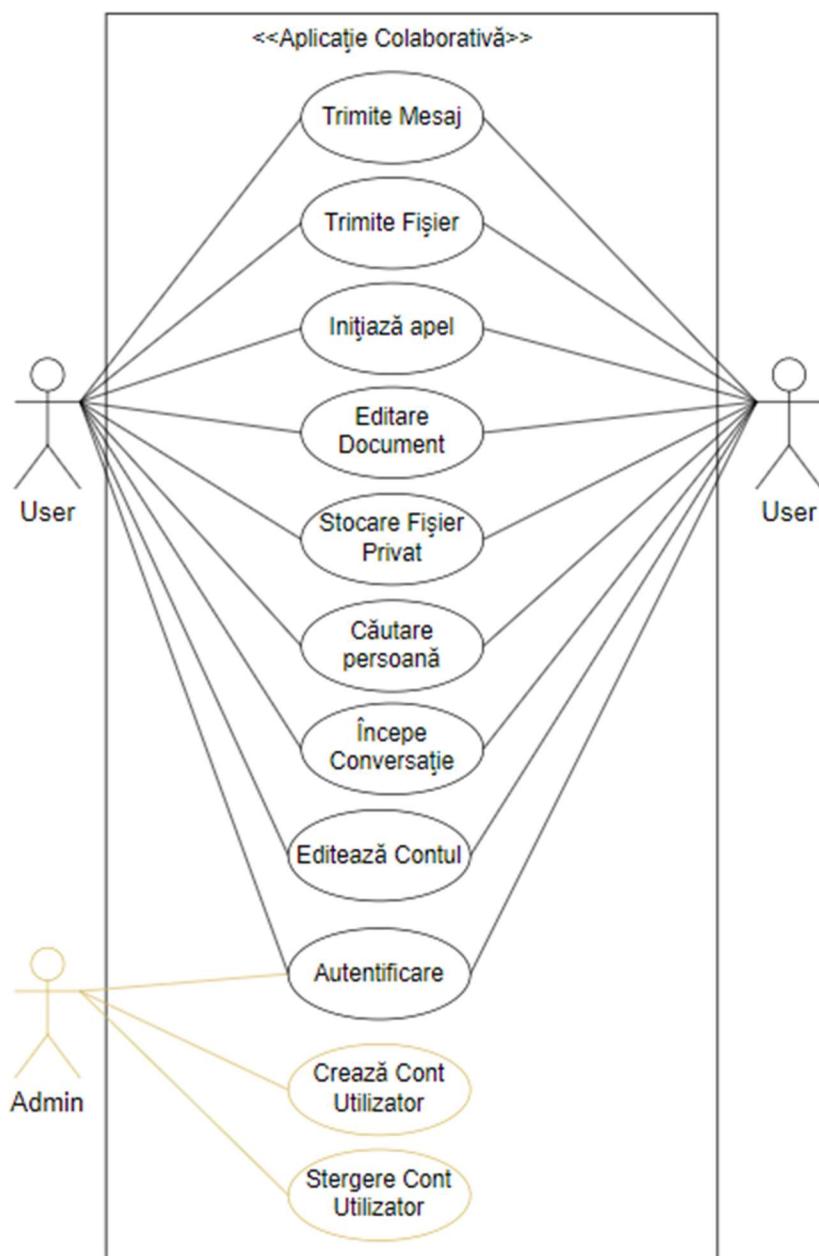
				cu mediul de stocare partajat.	
T8	SRS 8/ SRS 9	Trimiterea și descărcarea fișierelor.	Din cadrul conversației, utilizatorul trimite un fișier folosind butonul de atașament.	La nivelul conversației va fi trimis un mesaj ce conține denumirea fișierului și o previzualizare în cazul fișierelor de tip imagine.	Sistemul se comportă conform așteptărilor, fiind posibilă descărcarea fișierelor prin accesarea link-ului de acces..
T9	SRS 10	Efectuarea apelurilor video.	Doi utilizatori diferiți accesează butonul de pornire al apelului.	Utilizatorii sunt direcționați în fereastra dedicată, iar după câteva secunde vor avea posibilitatea de a se comunica video și audio (pe langă imaginea proprie, pe ecran pot vedea și imaginea celuilalt utilizator).	Sistemul are întârzieri de conectare, iar utilizatorul nu primește nici un fel de notificare că există o conexiune în curs de realizare. După conectare comunicația decurge normal, cu unele întârzieri.
T10	SRS 11/ SRS 12/ SRS 13	Crearea și editarea fișierelor text în mod colaborativ.	Este accesat butonul pentru crearea fișierelor text din secțiunea de storage, apoi adresa URL a site-ului este trimisă utilizatorilor cu care se dorește colaborarea. Utilizatorii vor accesa adresa trimisă și vor fi redirecționați spre editorul de text.	Este accesat mediul de editare reprezentat prin editorul de text., iar utilizatorul are posibilitatea de a forma textul. Restul participanților sunt afișați sub formă de listă și vor avea un cursor colorat.	Sistemul respectă specificațiile din cerințe, dar nu sunt afișați mai mult de un pointer simultan și poziția acestora este decalată față de poziția reală. De asemnea după deconectarea utilizatorului cursorul specific acestuia nu este eliminat.

NECLASIFICAT

## 6.3 Diagrame UML

### 6.3.1 Diagrama cazurilor de utilizare

În figura 6.14 este reprezentată diagrama cazurilor de utilizare ale aplicației din perspectiva utilizatorului. Folosind aplicația pusă la dispoziție, utilizatorii pot căuta persoane pentru a începe noi conversații pentru ca mai apoi să trimită mesaje și fișiere în cadrul acestora. Utilizatorii pot participa la conversații de tip apel și pot utiliza mediul de stocare pus la dispoziție. Un administrator va putea controla activitatea utilizatorilor prin crearea sau ștergerea conturilor de utilizator normal.

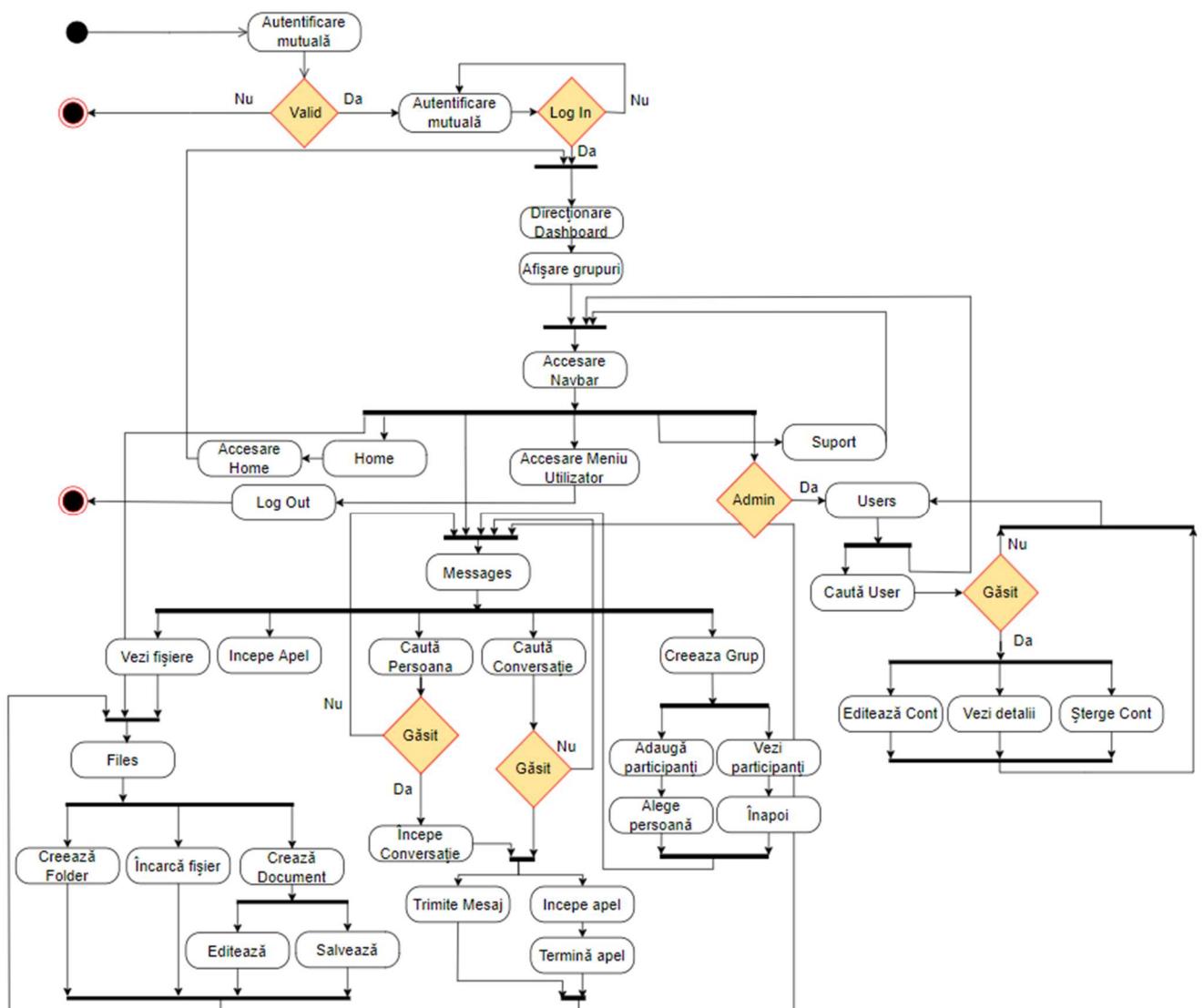


Figură 6.14 Diagrama cazurilor de utilizare

### 6.3.2 Diagrama de activități

Pentru a explica funcționalitățile și stările aplicației am întocmit diagrama de activități reprezentată în figura 6.14. Starea de start a sistemului este reprezentată de momentul în care utilizatorul accesează adresa aplicației, iar starea de final reprezintă finalul sesiunii de lucru a utilizatorului.

După autentificare, utilizatorul poate naviga la nivelul fiecărui modul prin accesarea componentei *Navbar*. De menționat că, pentru începerea unui apel video este necesar ca utilizatorul să acceseze starea de căutarea a unei conversații noi sau de a selecta o conversație deja creată. Un document de tip text poate fi creat doar de la nivelul ferestrei de *Storage*, în figură fiind reprezentat prin starea *Files*.



Figură 6.15 Diagrama de activități

## 7 Concluzii

### 7.1 Probleme întâmpinate

În timpul dezvoltării aplicației au fost întâmpinate câteva probleme ce ar trebui luate în considerare în dezvoltarea unor astfel de sisteme.

Printre problemele întâmpinate se numără distrugerea obiectelor de tip *Peer()*, din cadrul bibliotecii „*simple-peer*”, folosite la nivelul componentei *VideoRoom*. Un obiect *Peer()* defapt un obiect ce menține datele corespondente unei conexiuni ce se stabilește între browser-ele utilizatorilor, prin care este transmis fluxul video-audio. Inițial lista cu obiectele respective era salvată cu ajutorul hook-ului *useState()* și chiar dacă era apelată metoda *destroy()* pentru obiectul respectiv, lista nu suferea nici o modificare, iar conexiunile rămâneau deschise. În timpul testelor acest aspect a fost mai greu de detectat din cauză că obiectul ce se dorea distrus era eliminat din cadrul listei prin setarea uneia noi, iar vizual, caseta video corespunzătoare obiectului nu mai era afișată. În momentul în care se produceau mai multe intrări și ieșiri dintr-un apel, sistemul de comunicație prezenta întârzieri, finalizând cu un blocaj, din cauza unui număr prea mare de conexiuni active. Pentru rezolvare am utilizat hook-ul *useRef()*. Utilizarea acestuia a permis salvarea referinței corespondente obiectelor din *useState()* și astfel obiectele pot fi prelucrate prin folosirea referinței.

O altă problemă întâmpinată în implementarea software a fost necesitatea transmiterii variabilelor între componente ce nu aveau o relație de legătură directă. Această necesitate a devenit o problemă odată cu dezvoltarea modulelor în care era nevoie de folosirea unor variabile comune precum datele utilizatorului conectat sau date ce aveau legătură cu starea aplicației. Rezolvarea a venit prin utilizarea bibliotecii React Redux. Acest sistem permite modificarea și accesarea datelor de la nivelul oricărei componente prin folosirea funcției *useSelector()*.

O problemă apărută în momentul trecerii din mediul de dezvoltare în mediul de producție este incompatibilitatea serverului de tip websocket din cadrul aplicației server cu conexiunea pe care încearcă să o stabilească clientul prin intermediul browser-ului (browser-ul încearcă să creeze conexiune HTTPS, iar serverul Socket.IO este de tip HTTP). Pentru rezolvare, serverul trebuie creat de tipul HTTPS.

### 7.2 Rezultate obținute

Prezenta lucrare prezintă procedeul de implementare al unei soluții software colaborative, având ca scop documentarea tipurilor de sisteme similare și implementarea unei soluții proprii. În cadrul proiectului au fost descris stadiul actual, s-au detaliat tehnologiile folosite și a fost descrisă implementarea unei aplicații proprii, construite cu scopul de a cuprinde concepțele și tehnologiile studiate. Aplicația propusă își atinge obiectivul principal de a asigura munca

colaborativă la nivelul unui grup restrâns de persoane. Deși sistemul nu este optimizat reușește să asigure funcționalitățile implementate fără probleme de ordin tehnic și cu performanțe decente.

### 7.3 Dezvoltări ulterioare

Aplicația implementată necesită optimizări care ar duce la o stabilitatea ridicată pentru sistem. Pentru a acoperi în totalitate necesitatea de instrumente colaborative se pot propune spre implementare module precum aplicații calendar sau organizator de task-uri.

Capacitatea numărului de participanți la un apel video poate fi crescută prin adăugarea unui server dedicat de prelucrare video. Scopul unui astfel de server este de a prelua fluxul video de la fiecare participant, să-l unifice prin procesare, iar apoi să direcționeze rezultatul fiecărui client înrolat în convorbire. Un astfel de server este destul de complex pentru a fi implementat intern, dar performanțele ar fi cu mult îmbunătățite.

O altă îmbunătățire ce ar putea fi adusă este instalarea și folosirea unui server extern dedicat pentru formatarea fișierelor în format Word. Aceste tipuri de fișiere sunt prea complexe pentru a putea fi convertite direct în format HTML și reprezentate cu ajutorul un editor normal, așa că au nevoie de a fi procesate de către sisteme dedicate.

## 8 Bibliografie

- [1] “What is Collaboration Software?,” Mar. 2021. <https://kissflow.com/digital-workplace/collaboration/collaboration-software-guide/> (accessed May 13, 2022).
- [2] Shari Kjerland, “Microsoft Teams service description,” 2022. <https://docs.microsoft.com/en-us/office365/servicedescriptions/teams-service-description> (accessed Jun. 13, 2022).
- [3] Donna Tam, “Flickr founder plans to kill company e-mails with Slack.” <https://www.cnet.com/tech/tech-industry/flickr-founder-plans-to-kill-company-e-mails-with-slack/> (accessed May 13, 2022).
- [4] “Discord Support.” <https://support.discord.com/hc/en-us/articles/360041721052-Video-Calls> (accessed May 13, 2022).
- [5] “MTProto Mobile Protocol.” <https://core.telegram.org/mtproto> (accessed May 13, 2022).
- [6] Marino Miculan and Nicola Vitacolonna, “Formal verification of Telegram chat protocol MTProto 2.0.” <https://github.com/miculan/telegram-mtproto2-verification> (accessed May 13, 2022).
- [7] Cesar Ghali, Adam Stubblefield, J. L. Ed Knapp, Benedikt Schmidt, and Julien Boeuf, “Application Layer Transport Security - White Paper”.
- [8] Manuela Aparicio and Carlos J. Costa, “Collaborative Systems: Characteristics and Features,” p. 1, 2012.
- [9] Michel Beaudoin-Lafon, *Computer Supported Co-operative Work*. 1999.
- [10] C. Sun, “OT FAQ.” <https://web.archive.org/web/20200623064915/https://www3.ntu.edu.sg/home/czsun/projects/otfaq/> (accessed May 15, 2022).
- [11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski, “Conflict-Free Replicated Data Types,” pp. 2–5, 2014.
- [12] Salvatore Loreto and pietro Romano, *Real-Time Communication with WebRTC*. 2014.
- [13] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila, “A Formal Security Analysis of the Signal Messaging Protocol,” pp. 1–5, 2019, Accessed: May 17, 2022. [Online]. Available: <https://eprint.iacr.org/2016/1013.pdf>
- [14] “About Node.js.” <https://nodejs.org/en/about/> (accessed May 18, 2022).
- [15] “What is Express.js\_.” <https://www.besanttechnologies.com/what-is-expressjs> (accessed May 18, 2022).
- [16] “React Hooks.” <https://www.javatpoint.com/react-hooks> (accessed May 19, 2022).
- [17] David Choi, *Full-Stack React, TypeScript, and Node*. 2020.
- [18] “What Socket.IO is.” <https://socket.io/docs/v4/> (accessed May 18, 2022).
- [19] “Quill Documentation.” <https://quilljs.com/docs/quickstart/> (accessed May 19, 2022).

## 9 Anexe

### 9.1 Anexa A

Codul sursă al proiectului poate fi vizualizat prin accesarea link-ului către repository-ul de Github de mai jos:

<https://github.com/ionut99/icomV2>

### 9.2 Anexa B

```
io.on("connection", (socket) => {
  socket.on("join room", async (request, callback) => {
    const userId = request.userId;
    const roomId = request.roomId;
    const type = request.type;

    const { error, user } = await addUserInRoom({
      id: socket.id,
      userId,
      roomId,
      type,
    });
    //
    if (error) {
      console.log("refuse user to join in room..");
      return callback(error);
    }
    //
    socket.join(user.roomId);

    if (type === "video") {
      socket.emit("all users", {
        roomId: user.roomId,
        users: getUsersInRoom(user.roomId, user.id, user.type),
      });
    }
    if (type === "edit") {
      socket.emit("all users edit", {
        roomId: user.roomId,
        users: getUsersInRoom(user.roomId, user.id, user.type),
        color: getUserColor(user.roomId, user.userId, user.type),
      });
      //
      socket.broadcast.to(user.roomId).emit("user joined edit", user);
    }
    callback();
  });

  // Listen for new messages
  socket.on("send chat message", (message) => {
    //
    const user = getUser(socket.id);
    if (user === undefined) return;
    //
    io.to(message.roomId).emit("receive chat message", message);
    // save message
    const mes_res = InsertNewMessage(message);

    if (mes_res === null) {
      console.log("Error save message !");

      socket.emit("error insert message", { message });
    } else {
      console.log(
        "MESSAGE: " +
        message.body +
        " by " +
        message.senderName +
        " on " +
        message.roomId
      );
    }
  });
});
```

```

//listening for typing
socket.on("typing chat message", (request) => {
  const user = getUser(socket.id);
  if (user === undefined) return;
  //
  io.to(request.roomId).emit("user typing", request);
  //
});

// Document Actions

// Listen for new document changes
socket.on("send doc edit", (delta) => {
  const user = getUser(socket.id);
  if (user === undefined) return;
  //
  socket.broadcast.to(user.roomId).emit("receive doc edit", delta);
});

//

// Listen for new document changes
socket.on("send doc pointer", (delta) => {
  const user = getUser(socket.id);
  if (user === undefined) return;
  //
  socket.broadcast.to(user.roomId).emit("receive doc pointer", delta);
});

//

// Listen for new document changes
socket.on("send doc presence", (delta) => {
  const user = getUser(socket.id);
  if (user === undefined) return;
  //
  socket.broadcast.to(user.roomId).emit("receive doc presence", delta);
});
//

socket.on("sending signal", (payload) => {
  io.to(payload.userToSignal).emit("user joined", {
    signal: payload.signal,
    callerId: payload.callerId,
  });
});

socket.on("returning signal", (payload) => {
  io.to(payload.callerId).emit("receiving returned signal", {
    signal: payload.signal,
    id: socket.id,
  });
});

//

socket.on("disconnect", () => {
  const user = getUser(socket.id);
  const deleted = deleteUser(socket.id);
  if (deleted) {
    if (user.type === "video" || user.type === "edit")
      socket.broadcast.emit("user left", socket.id);
  }
});
});

```

NECLASIFICAT