UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Rețea de comunicare în grup

Mădălin Constantin Vasile

Coordonator ştiinţific:

Prof. dr. ing. Alexandru Boicea

BUCUREŞTI

2023

1 CUPRINS

Sinopsis 3

Abstract 4

Mulțumiri 5

1 Introducere 1

1.1 Context 1

1.2 Problema 1

1.3 Obiective 2

1.4 Soluția propusă 2

1.5 Rezultatele obținute 3

1.6 Structura lucrării 5

2 Motivație 6

3 Metode existente 7

4 Soluția propusă 8

4.1 Definirea design-ului 11

4.1.1 Scopul design-ului 11

4.1.2 Identitatea vizuală 11

4.1.3 Structura și organizarea 11

4.2 Functionalitați pentru administrarea echipei 13

4.3 Generarea rapoartelor 13

4.4 Transmiterea mesajelor în timp real 14

4.5 Notificări 15

4.6 Folosirea REST API pentru comunicarea intre client si server 15

4.7 Criptarea parolei în cadrul autentifcării 16

4.8 Setarea avatarelor 17
5 Detalii de implementare 19
5.1 Arhitectura aplicației 19
5.2 Probleme apărute în timpul implementării aplicației 20
5.2.1 Stocarea imaginilor în baza de date NoSQL 20
5.2.2 Folosirea socket.io 21
5.2.3 Notificări 23
5.3 Conectare la baza de date Oracle 25
5.4 Implementarea listei de contacte 26
5.5 Implementarea workspace-ului 28
6 Evaluare 30
7 Concluzii 34
8 Bibliografie 35
9 Anexe 37

SINOPSIS

Problema care urmează să fie abordată este necesitatea unei aplicații de comunicare care să asigure o interacțiune eficientă între membrii diferitelor organizații prin transmiterea de mesaje și generarea de rapoarte și statistici folositoare. În cadrul aplicației, utilizatorii se pot diferenția intre ei prin datele personale cum ar fi numele complet, un nickname dar și prin imaginea de avatar.

Aplicația fiind orientată către zona de munca trebuie să ofere adminstratorului organizației diverse instrumente de administrare cum ar fi: invitarea sau înlăturarea unui membru dar și rapoarte care indică anumite detalii despre membrii sau interacțiunea dintre aceștia. Pentru dezvoltarea aplicației am ales să utilizez Node.js [4] ca pentru backend și React [3] ca framework pentru frontend.

Aplicația va fi disponibilă utilizatorilor prin accesarea unui link, care redirecționează utilizatorul spre pagina de autentificare, de unde poate să-și creeze cont, fie din perspectiva unui administrator, fie din perspectiva unui membru.

În ceea ce privește zona careia ii este adresată aplicația, acesta este reprezentată de orice organizație care dorește să comunice. Rolul acestei aplicații este îmbunătățirea comunicării și colaborării între membrii organizațiilor prin asigurarea transmiterii de informație dar aplicația poate fi folosită si in scopuri de entertainment, crearea unui cont fiind accesibilă oricui dorește.

ABSTRACT

The problem to be addressed is the need for a communication application that ensures effective interaction between members of different organizations by sending messages and generating useful reports and statistics. Within the application, users can differentiate themselves by personal data such as their full name, a nickname, but also by their avatar image.

The application, being oriented towards the work area, must offer the organization administrator various management tools such as: inviting or removing a member, but also reports that indicate certain details about the members or the interaction between them. For the development of the application I chose to use Node.js [4] as the backend and React [3] as the framework for the frontend, the application will be available to users by accessing a link, which redirects the user to the authentication page, from where he can create an account, either from the perspective of an administrator or from the perspective of a member.

As for the area to which the application is addressed it is represented by any organization that wishes to communicate. The role of this application is to improve communication and collaboration between members of organizations by ensuring the transmission of information, but the application can also be used for entertainment purposes, creating an account being accessible to anyone who wants to.

MULȚUMIRI

(opțional) Aici puteți introduce o secțiunea specială de mulțumiri / acknowledgments.

1 INTRODUCERE

1.1 Context

Comunicarea de calitate a fost întotdeauna un obiectiv important pentru desfășurarea de calitate a proceselor care depind de interacțiunea între mai mulți oameni. În zilele noastre majoritatea aplicațiilor de comunicare sunt orientate spre entertainment având ca scop simpla transmiterea a mesajelor [1] și nu oferă instrumente de administrare a utilizatorilor care pot fi folosite în cadrul proceselor dintr-un mediu de afacere. Astfel cu evoluția limbajelor de programare se pot dezvolta aplicații care să răspundă nevoii organizațiilor de a folosi o platformă prin intermediul căreia membrii unei echipe pot fi invitați, pot comunica între ei și pot posta știri relevante. Aceste aplicații pot genera rapoarte care să monitorizeze activitatea utilizatorilor, asigurând in acleasi timp si o comunicare de calitate.

Limbajele actuale de programare dispun de biblioteci și module care permit realizarea obiectivelor menționate anterior într-un mod accesibil pentru orice adminstrator de organizație.

1.2 Problema

Soluțiile folosite pentru comunicarea standard sunt limitate la simplul transfer de mesaje de la un utilizator la altul fără a exista o ierarhizare între membrii astfel încât accesul să fie restricționat pentru o anumită categorie de utilizatori. În aplicațiile clasice de comunicare [1] nu există noțiunea de adminstrator care poate să aibă mai multe drepturi decât un utilizator obișnuit. În același timp o aplicație obișnuită nu poate genera invitații prin care membrii să-și creeze un cont pe aplicație, utilizatorul fiind obligat să se conecteze pe cont propriu pe aplicație lucru care nu asigură o compartimentare față de alte organizații din care poate să facă parte utilizatorul, astfel scurgerea de informații fiind mult mai ușor realizabilă, lucru care nu este recomandat în cadrul unei organizații în care anumite aspecte pot fi confidențiale. Un alt aspect care nu este oferit de o aplicație clasică este generarea de rapoarte în ceea ce privește comunicarea între utilizatori cu scopul eficientizării și prioritizării anumitor task-uri.

1.3 Obiective

Dorim realizarea unei aplicații web pe care o să o numim în continuare “Rețea de comunicare în grup”, care să ruleze în browser, astfel orice adminstrator al unei organizații își poate crea cont folosind procesul de sign in, urmând să invite membrii în organizația sa folosind un meniu accesibil doar adminstratorului. Un viitor utilizator este invitat prin intermediul unui cod unic care este primit pe adresa de e-mail, astfel acesta își poate crea cont introducând codul. După setarea numelui, adresei de e-mail și parolei, user-ul trebuie să-și aleagă un avatar care poate fi ceva standard sau o poză încărcată din calculator. Aplicația permite membrilor să comunice între ei și să primească notificări când se primesc mesaje și să posteze știri pe feed-ul organizației, fiind la curent cu noile schimbări. În cazul adminstratorilor de organizație, aceștia pot genera rapoarte folositoare pentru o cumunicare mai bună și a eficentiza mediul de muncă.

1.4 Soluția propusă

Pentru implementarea aplicației, am ales să folosesc o arhitectură monolitică [2]. Este o abordare de dezvoltare a aplicațiilor web în care toate componentele sistemului fac parte din aceeași unitate. În această arhitectură, baza de date, serverul și partea de client (frontend) sunt parte rulează pe aceeași mașină astfel fiind mai accesibilă dezvoltarea și implementarea. Această abordare oferă o performanță rapidă, deoarece comunicarea dintre componente este foarte bună și cauza simplității arhitecturii, depanarea si mentenanța devin mai usoare.

Frontend-ul va fi realizat prin framework-ul React [3], am ales acest framework deoarece are o comunitate puternică, dispune de documentație și a demonstrat eficiență din punctul de vedere al DOM-ului virtual care permite actualizarea rapidă și eficientă a UI-ului fără a afecta restul paginii web, dar și al reutilizării componentelor. React [3] gestionează foarte bine ideea de componente care sunt folosite pentru a crea interfețe complexe, lucru care face ajută dezvoltarea mai eficientă și mai rapidă.

Backend-ul va fi realizat folosind tehnologia Node.js [4] deoarece este foarte cunoscută, lucru care ajută în cazul problemelor de depanare dar și puternică pentru dezvoltarea aplicațiilor web, care oferă o serie de avantaje pentru dezvoltatori cum ar fi scalabilitate, viteza de răspuns și accesibilitatea la resurse. Legătura dintre frontend și backend se

realizează prin REST API [5] și prin socket.io [6] . De asemenea, am ales să folosesc o bază de date Oracle [7] pentru stocarea datelor din aplicație, aceasta oferă o performanță bună, dispune de o securitate solidă și o comunitatea destul de mare si este una dintre cele mai utilizate baze de date relaționale, oferind suport pentru o varietate de limbaje de programare și platforme.

1.5 Rezultatele obținute

După rularea mai multor teste pe implementarea aplicației, se poate observa că aceasta este capabilă să realizeze obiectivele propuse. Aplicația a fost deployată intr-un mediu cloud pentru a fi accesibilă tuturor utilizatorilor. Aceasta permite unui administrator să-și creeze cont pentru organizația sa și să invite din meniul Team Management noi utilizatori în cadrul comunității sale, tot din același meniu utilizatorii cu un comportament inadecvat pot fi suspendați și reactivați ulterior. Utilizatorii invitați pot să-si creeze cont folosind codul de acces primit prin e-mail, urmand să-și aleagă un avatar care poate fi un avatar standard sau o imagine încărcată. După ce procesul de autentificare este finalizat utilizatorii pot să comunice între ei prin intermediul unui chat, să posteze noutăți pe feed-ul organizației și să primească notificări când primesc mesaje de la alți utilizatori sau cand cineva a fost suspedant sau reactivat pentru ca acesta să-și reactulizeze lista de contacte. Utilizatorul poate să-și recupereze parola prin completarea formularului "Forgot your password", acesta o să primească pe adresa de email un cod de recuperare care va fi folosit pentru noua parolă. Un utilizator suspendat nu mai poate comunica cu ceilalți și nici să posteze pe feed-ul organizației dacă a fost suspendat în timp ce era conectat pe aplicație. După ce acesta părăsește aplicația, nu se mai poate conecta, fiind afișat un mesaj sugestiv. În cazul utilizatorilor suspendați, nici ceilalți utilizatori nu mai pot comunica cu aceștia, căsuța acestora din lista de contacte fiind blocată și marcată cu o culoare sugestivă. Un utilizator reactivat revine la funcționalitățile de care dispunea înainte de a fi suspendat. Administratorul aplicației poate genera rapoarte despre organizație sau despre fiecare utilizator, cu scopul de a eficientiza mediul de muncă.

În ceea ce privește UI-ul, aplicația răspunde este ușor de folosit, având o interfață intuitivă și prietenoasă pentru utilizator.

În figurile [1 - 8], [18 – 27 (ANEXE)] sunt prezentate capturi de ecran din aplicație:

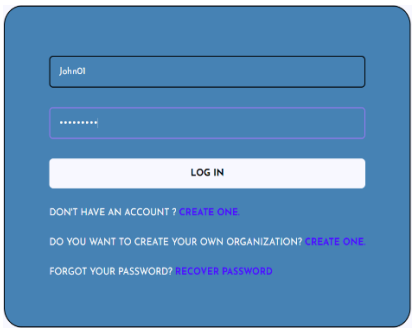


Figura 1: Login

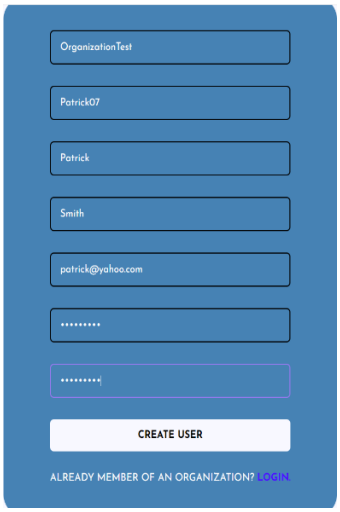


Figura 2: Sign up administrator

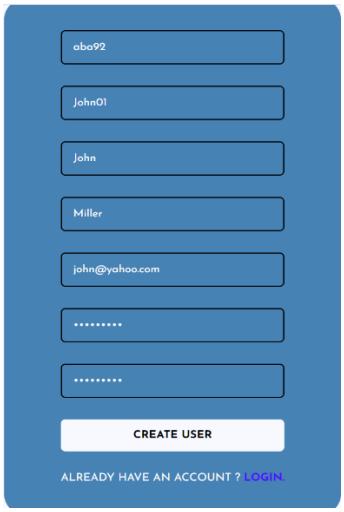


Figura 3: Sign up membru

Figura 4: Formular “Forgot your password”

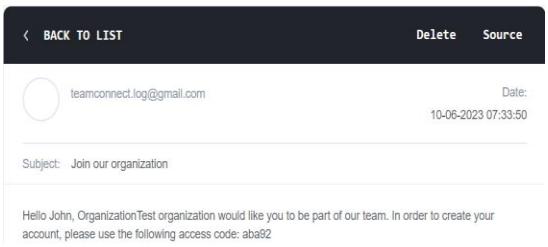


Figura 5: codul de autentficare primit prin mail

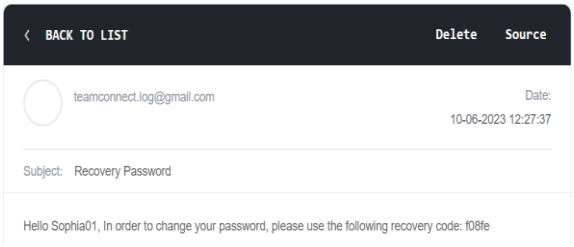


Figura 6: codul de recuperare al parolei

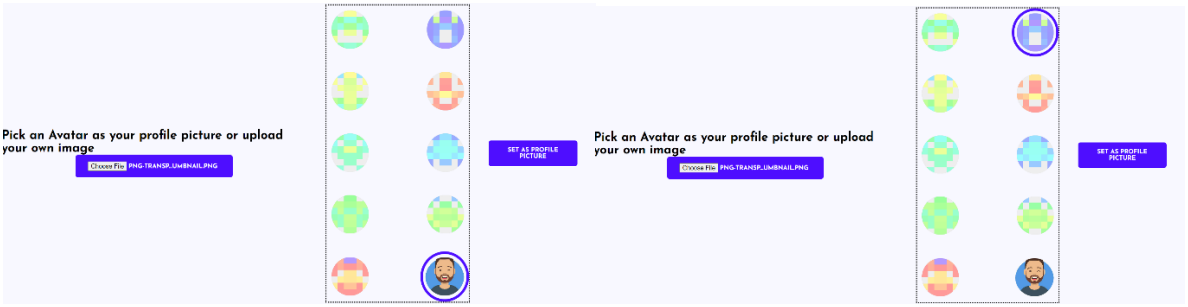


Figura 7a: Avatar incarcat de utilizator

Figura 7b: Avatar generat random

Figura 8 reprezinta workspace-ul din perspectiva unui administrator(meniul TeamManagement deschis 8c sau inchis 8b) respectiv membru 8c

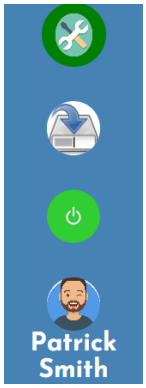


Figura 8a: workspace

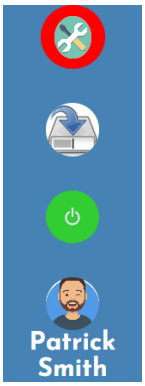


Figura 8b: workspace

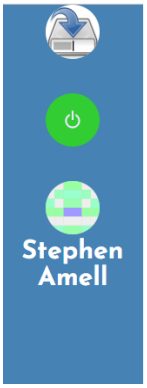


Figura 8c: workspace

1.6 Structura lucrării

Capitolul 2 prezintă motivația din spatele temei alese. Capitolul 3 examinează metodele și tehnologiile existente care pot fi comparate cu tema propusă, se detaliază avantajele si dezavantajele acestora raportate la problema indentificată. În capitolul 4 este prezentată soluția propusă pentru a rezolva problema identificată, fiind explicate conceptele de bază ale acesteia. Capitolul 5 presupune descrierea în detaliu a modului în care soluția prezentată și problemele care au aparut in cadrul implementării. Capitolul 6 efectuează evaluarea soluției propuse, sunt prezentate rezultatele studiilor de caz realizate pentru a testa aplicația si modul prin care aplicația a devenit accesibilă utilizatorilor. Capitolul 7 face o trecere prin restul capitolelor, punând accentul pe evaluare și pe rezultatele obținute.

2 MOTIVAȚIE

Comunicarea în grup a constat foarte mult timp în întâlnirea fizică a mai multor indivizi cu scopul rezolvării unor probleme. În prezent există multe aplicații gratuite destinate comunicării dar majoritatea au scopuri de entertainment. Motivația aplicației prezentate constă intr-o aplicație care să pună la dispoziția unui adminstrator de organizație posibilitatea de aducere a membrilor săi împreună cu scopul cumunicarii eficient si de a eficientiza procesele din mediul său de muncă. În ultimul timp aplicațiile de comunicare la distanță au devenit din ce în ce mai populare deoarece exista nevoia de a comunica și colabora eficient în timp real fara să influențeze locația geografică sau fusul orar. Aplicatiile precum cea prezentată permit adminstratorilor de afaceri și angajaților să comunice rapid și să gestioneze sarcinile și proiectele astfel incat să ia decizii mai bine bune pentru beneficiul tuturor.

Necesitatea acestui proiect a rezultat prin prisma faptului că abordările actuale de comunicare în cadrul business-urilor sunt limitate și anumite funcții importante ale acestora sunt puse la dispoziția adminstratorului doar prin monetizare [8]. Din această cauză, multe setări ajung să fie făcute manual de către adminstrator precum rapoarte și statistici despre membrii organizației sale. Functionalități precum generarea de rapoarte despre membrii și interacțiunea dintre aceștia ajuta foarte mult procesul de gestionare și eficientizare a task-urilor. Metodele prezentate mai sus pot reduce timpul și efortul necesar pentru a obține anumite rezultate astfel incat productivitatea și eficienta să crească.

3 METODE EXISTENTE

Problema comunicării la distanță într-un mod eficient a fost și încă este o problemă deschisă. În prezent, majoritatea soluțiilor în acest sens [1] sunt reprezentate de aplicații de comunicare eficiente, dar care sunt orientate spre entertainment sau care necesită monetizare [8] pentru a accesa totalitatea funcțiilor lucru care poate fi o problemă pentru organizațiile mici. Aceste soluții de comunicare orientate spre entertainment cum ar fi aplicațiile de social media și mesagerie sunt concepute pentru a satisface nevoile de interacțiune socială și divertisment ale utilizatorilor. Acestea oferă funcții atractive dar nu abordează în mod specific nevoile de comunicare în cadrul organizațiilor. Aplicația "Rețea de comunicare în grup" oferă instrumente care facilitează comunicarea în cadrul organizației precum posibilitatea de a izola membrii organizației de restul utilizatorilor aplicației, de a invita doar acei membrii care trebuie să facă parte din organizație și de a genera rapoarte și statistici folositoare în gestionarea task-urilor ulterioare. Un avantaj prezentat de soluțiile existente este determinat de posibilitatea conectării in aplicație prin intermediul altor rețele de socializare, lucru care înca nu este disponbil pe aplicația prezentată. Un alt dezavantaj pentru aplicația prezentată este lipsa unui meniu care să editeze detaliile despre utilzatorul conectat. Am intrebat mai mulți oameni prin intermediul unui formular care este opinia lor despre produsul implementat și despre nevoia unei astfel de aplicații.

Rezultatele formularului de feedback realizat pentru a evalua nevoia unei astfel de aplicații au confirmat importanța și interesul utilizatorilor. Majoritatea participanților au considerat aplicația "Rețea de comunicare în grup" o abordare interesantă care oferă o interfată prietenoasă și au exprimat nevoia de a utiliza o astfel de soluție. O funcționalitate apreciată de utilizatori a fost chat-ul unu la unu care a impresionat prin simplitatea acestuia, astfel aplicația fiind dorită și pentru comunicarea în afara organizațiilor. Feedback-ul obținut susține potențialul aplicației de a satisface nevoile organizațiilor de diferite dimensiuni, organizațiilor non-profit, guvernelor și altor entități care doresc să comunice într-un mod eficient cu membrii lor.

4 SOLUȚIA PROPUSĂ

În această secțiune voi prezenta soluția propusă pentru a rezolva problema prezentată urmând ca detaliile de implementare să fie exemplificate în capitolele următoare. Aceasta va fi construită folosind HTML, CSS și JavaScript în cadrul framework-ului REACT [3] iar partea server va fi dezvoltată cu ajutorul NodeJs [4]. Arhitectura soluției constă în trei părți principale și anume baza de date relațională, partea de frontend si partea de backend. Baza de date folosită este Oracle [7] și are ca scop stocarea informațiile despre utilizatori, mesajele acestora și știrile postate pe feed-ul organizației. Frontend-ul va permite utilizatorilor să gestioneze aceste informații mai exact: să se autentifice, să-și creeze cont ca adminstrator sau ca membru, să-și aleagă un avatar sau o imagine de profil, să trimită mesaje, să posteze anunțuri pe feed-ul organizației sau să genereze rapoarte despre utilizatori și interacțiunea dintre aceștia în cazul adminstratorului.

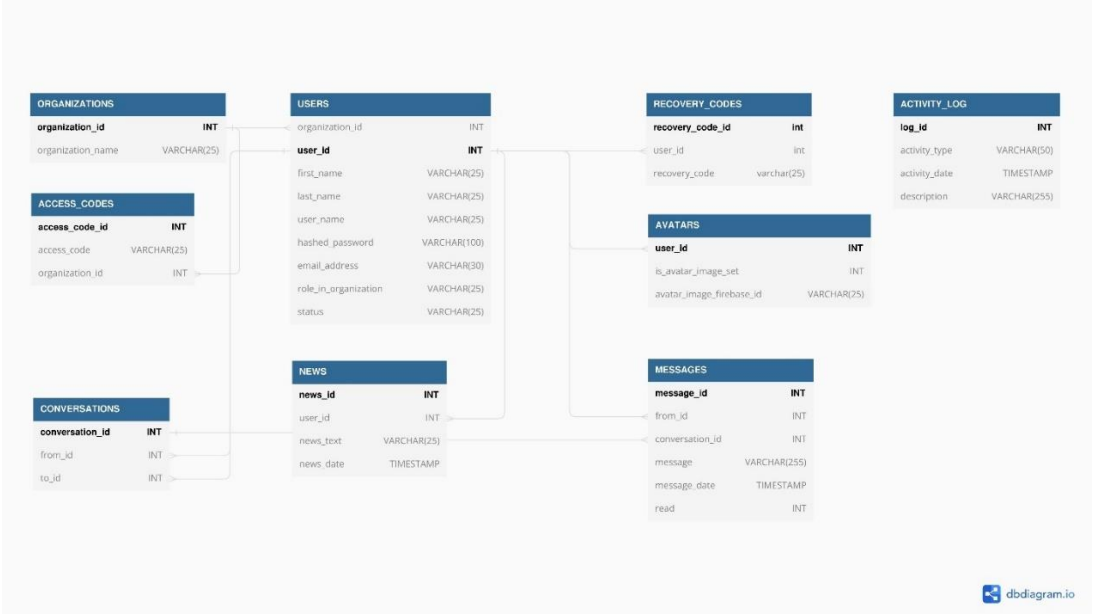


Figura 9: Diagrama entitate-relație (ER) a bazei de date

Aceasta conține nouă entități principale: utilizatorii, organizațiile, avatarele, codurile de acces, codurile de recuperta parolele uitate, știrile, mesajele, conversatiile si log-ul de activitați. Entitatea organizații va stoca informații precum numele organizației și id-ul acesteia. Entitatea de utilizatori va stoca numele, adresa de email, hash-ul parolei obținut prin algoritmul bcrypt, id-ul organizației și rolul în organizație. Entitatea de avatare va conține detalii despre avatarul fiecărui utilizator, precum id-ul imaginii stocate în baza de

date Firebase [14] și un flag care indică dacă imaginea a fost setată sau nu. Entitatea de conversații va fi folosită pentru informatii precum id-ul conversatiei, al utilizatorului care a trimis mesajul si al utilizatorului care a primit mesajul folosite ulterior pentru a identifica conversația din care face parte un mesaj. Entitatea de mesaje va fi folosită pentru informații precum id-ul mesajului, conversatiei din care face parte mesajul, utilizatorului care a trimis mesajul, mesajul și data în care a fost trimis. Entitatea de știri va fi folosită pentru a stoca informații despre fiecare știre postată pe feed-ul organizației precum id-ul utilizatorului care a postat, textul și data în care a fost postată. Entitatea coduri de acces va fi folosită pentru a stoca informații precum id-ul organizației pentru care este generat un cod de acces și codul de acces în sine. Entitatea coduri de recuperare parole uitate va fi folosită pentru a stoca informații precum id-ul utilizatorului pentru care este generat un cod de recuperare și codul de recuperare în sine. Entitatea log-ului de activitate va fi folosită pentru stocarea anumitor activitati importante din cadrul aplicației precum informații precum id-ul log-ului, tipul activității, request HTTP de exemplu, data în care a fost inregistrat log-ul si descrierea acestuia. Pentru prezentarea structurii tabelelor din Figura 9 am folosit dbdiagram.io , o platformă online care permite utilizatorilor să creeze diagrame de baze de date.

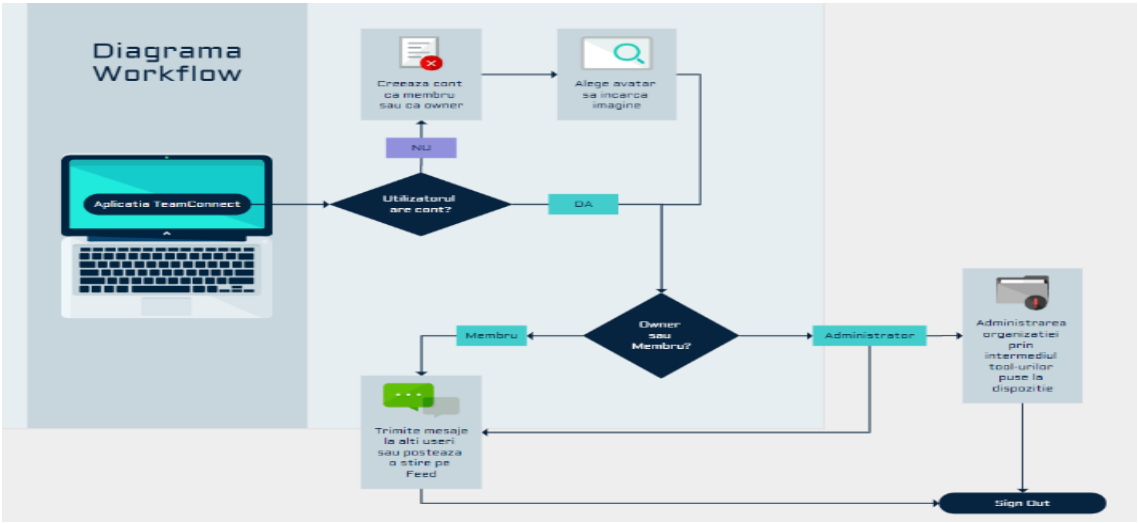


Figura 10: Diagrama workflow arată fluxul de utilizare al aplicației web

După autentificare utilizatorii vor fi direcționați către pagina principală unde vor putea trimite mesaje. În cazul administratorului de a gestiona organizația prin invitarea de noi membri sau de a genera rapoarte folositoare. În cazul unui utilizator nou acesta poate să-și

creeze cont dacă a primit un link de invitație din partea unui administrator sau să creeze propria organizație, fiind administratorul acesteia. În ambele cazuri, un utilizator nou va trece printr-o etapă de personalizare în care își setează un avatar sau o imagine de avatar. În pagina principală a aplicației, utilizator poate să trimită mesaje la ceilalți utilizatori, poate să posteze un mesaj pe feed-ul organizației. În cazul în care dorește să transmită o știre pe care să o vadă toată organizația. Din perspectiva administratorului există posibilitatea de a invita sau de a elimina utilizatori din organizație și de a genera rapoarte despre fiecare utilizator sau despre întreaga organizație. Pentru prezentarea workflow-ului din Figura 10 am folosit infograph.venngage.com[1] , o platformă online care permite utilizatorilor să creeze diverse flowchart-uri.

[1] infograph.venngage.com. Flowchart. https://infograph.venngage.com/templates/diagrams/flowchart. Ultima accesare: 17 iunie 2023

4.1 Definirea design-ului

4.1.1 Scopul design-ului

Scopul design-ului aplicației de comunicare este de a crea o experiență placută pentru utilizatori, acestă aplicație de comunicare trebuie să permită utilizatorilor din toate domeniile să poată să o folosească fară dificultate astfel încât aceștia să poată accesa și utiliza funcționalitățile aplicației. Felul în care aplicația arată trebuie să fie facut în așa fel incăt să faciliteze transmiterea de mesaje, postarea de știri pe feed sau observarea notificărilor, acestea fiind sarcinile de bază ale aplicației.

4.1.2 Identitatea vizuală

Identitatea vizuală a aplicației web este construită în jurul culorilor pe care le-am ales și a modului în care acestea sunt utilizate pentru a transmite anumite stări [18] și funcționalități, astfel incât utlizatorul să poată invața rapid cum se folosește aplicația. Este foarte important ca unele etape din functionalitati să fie intuitive pentru utilizator astfel culorile joacă un rol în dezvoltarea aplicației. Pentru a exprima simplitatea, am ales să folosesc o nuanță de alb, #f8f8ff pentru fundalul paginilor de login și chat. Formularele de sign in, sign up, log out, recovery password și workspace-ul folosesc culoarea #4682b4 pentru a fi ușor de recunoscut și pentru a sugera acțiuni importante pentru utilizatori. Pentru a distinge utilizatorii cu care s-a început o conversație în chat am ales să folosesc culoarea #9a86f3 pentru a semnala unui utilizator selectat și pentru a ajuta la recunoașterea conversațiilor active. Mesajele necitite sunt semnalate prin culoarea roșie, pentru a atrage atenția utilizatorilor asupra unor mesaje noi sau importante. Pentru situațiile în care accesul utilizatorilor este restricționat sau suspendat temporar, am ales să folosesc culoarea #5f9ea0 care sugerează o încetare temporară a activității sau a permisiunilor de acces. Butonul de sent pentru mesaje și butonul de refresh pentru notificări au culoarea verde deschis, #32cd32, pentru a sugera succesul în interacțiunea cu aplicația. În final, am ales să folosesc culoarea #ffffff34 pentru inputul textului, pentru a crea un contrast plăcut între text și fundalul alb și pentru a oferi o experiență de utilizare mai confortabilă.

4.1.3 Structura și organizarea

Aplicația web are structura și organizarea concentrate pe simplificarea elementelor individuale, pentru a oferi un aspect curat și modern bazându-se pe principiile design-ului

flat[2]. Cele patru secțiuni importante ale aplicației vor fi organizate astfel încât să fie într-un mod intuitiv și ușor de navigat pentru utilizator. Secțiunea de workspace va conține informații despre utilizatorul curent, inclusiv numele și avatarul acestuia, butonul de log out va fi pozitionat în partea dreaptă a paginii deasupra de avatar. Pentru adminstratorul aplicației, butonul de Team Management va fi disponibil pentru a gestiona echipa, pozitionat deasupra de butonul de log out. În această secțiune, design-ul flat va fi folosit pentru a simplifica butoanele și meniurile, având un număr restrâns de culori . Secțiunea de contacte va fi proiectată în mod similar, cu un meniu de navigare simplu și ușor de utilizat pentru a selecta un contact cu care să se comunice. Mesajele necitite și utilizatorii suspendați vor fi evidențiați cu culori specifice. Sectiunea de chat/Team Management/Welcome va fi concepută cu un design flat minimalist, cu butoane simple pentru a trimite mesaje, a gestiona echipa și a primi un mesaj de bun venit la conectarea în aplicație. Sectiunea de feed va fi proiectată în același stil , cu un editor de text simplu pentru a posta stiri, iar butonul de refresh va fi folosit pentru a actualiza știrile postate de alți utilizatori.

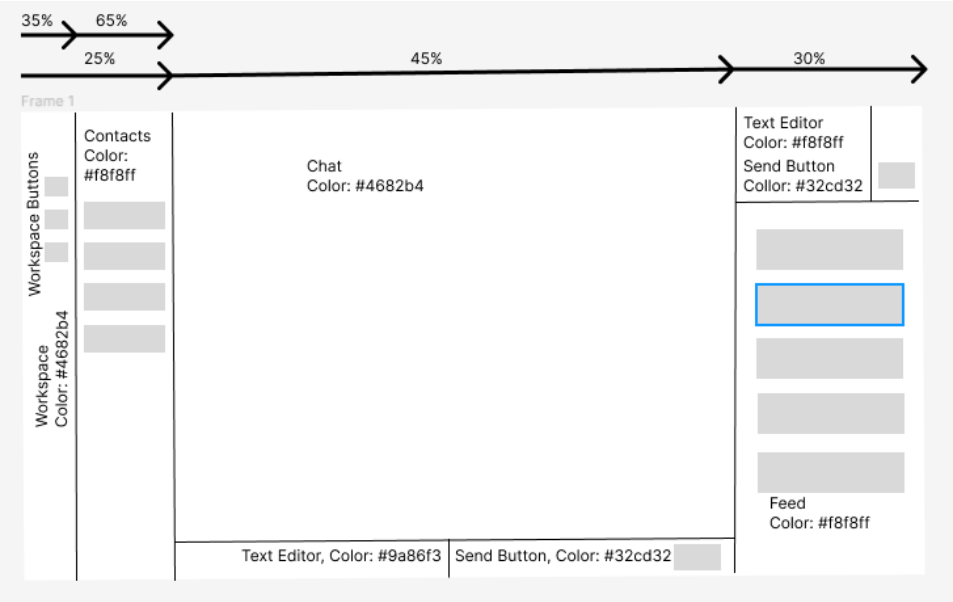


Figura 11: Structura și organizarea aplicației

[2] Flat Design. What is Flat Design? https://www.interaction-design.org/literature/topics/flat-design Ultima accesare: 16 iunie

Pentru prezentarea structurii tabelelor din Figura 11 am folosit figma[3] , o platformă online care permite utilizatorilor să creeze design pentru aplicații web.

4.2 Functionalitați pentru administrarea echipei

Aplicația propusă dispune de funcționalități precum invitarea în cadrul organizației curente a unui nou utilizator prin generarea unui cod de acces unic trimis pe adresa de e-mail a utilizatorului. Practic, este o metodă prin care echipa poate fi populată cu utilizatorii doriți de către administrator. Suspendare acelor utilizatori care, din diferite motive, nu mai trebuie să aibă acces momentan la aplicație. Un utilizator suspendat nu mai poate trimite mesaje pe feed sau către alți utilizatori dacă este suspendat în momentul în care acesta este conectat. După prima delogare, acesta nu mai poate accesa aplicația deloc, fiind notificat la logare că a fost suspendat de administrator. Tot în cazul unui utilizator suspendat, ceilalți utilizatori primesc o notificare că lista de contacte a suferit modificări și observă acest lucru prin caseta utilizatorului respectiv, care își schimbă culoarea, iar la hover mesajul "suspended" este afișat și nu pot să-i trimită mesaje. O altă funcționalitate pentru administrarea echipei este activarea utilizatorilor suspendați. Această funcționalitate permite utilizatorilor suspendați să revină în cadrul aplicației la fel ca în cazul suspendării unui utilizator, ceilalți utilizatori sunt notificați de acest lucru iar accesul la mesaje și postările de pe feed este reluat. Pentru implementarea acestor funcționalități am folosit cereri HTTP apelate din frontend prin intermediul unor butoane pentru a face modificări în baza de date mai exact în tabela Users, coloana Status care este responsabilă de schimbările legate de utilizatori în momentul în care statusul acestora este schimbat și de transmiterea notificărilor prin socket pentru înștiințarea celorlalți utilizatori.

4.3 Generarea rapoartelor

Soluția propusă constă în implementarea unei funcționalități speciale pentru administratorul aplicației. Aceaste functionalități îi permit adminstratorului generarea anumitor rapoarte privind activitatea utilizatorilor și cum aceștia interactioneaza intre ei. Pentru a realiza acest lucru am utilizat interogări SQL pentru a face rost de informațiile necesare din baza de date urmând ca apoi acestea să fie prezentate într-un mod grafic în React [3] folosind componente din biblioteca 'chart.js/auto' [9] .

[3] Figma. https://www.figma.com/ . Ultima accesare 17 iunie 2023

Rapoartele puse la dispoziția administratorului sunt următoarele: raportul privind numărul de mesaje trimise pe mai multe intervale orare, raportul privind numărul de mesaje trimise de fiecare utilizator, raportul privind numărul de mesaje primite, raportul privind utilizatorii suspendați și utilizatorii activi, generarea unui fișier CSV cu toți utilizatorii din aplicație.

Aceste rapoarte oferă administratorului posibilitatea de a observa modul în care membrii echipei sale interactionează, determinând astfel procesele din interiorul organizației să funcționeaze mai bine.

4.4 Transmiterea mesajelor în timp real

Pentru transmiterea mesajelor în timp real, am folosit tehnologia de tip socket pusă la dispoziție de biblioteca socket.io [6] pentru NodeJs [4]. Folosirea acestei metode a venit din necesitatea de a găsi o alternativă pentru REST API [5], care necesită un declanșator pentru transmiterea sau verificarea că un alt utilizator a trimis un mesaj. Socket-ul ascultă trei evenimente: adăugarea unui utilizator nou, trimiterea unui mesaj și recepționarea unui mesaj de la alt utilizator. Atunci când evenimentul de adăugare a unui nou utilizator este declanșat, id-ul utilizatorului se adăuga în maparea corespunzătoare alaturi de id-ul socket-ului. Daca evenimentul de trimitere a unui mesaj este declanșat, socket-ul corespunzător utilizatorului destinatar este cautat iar dacă acesta este găsit mesajul este transmis către el.

Daca evenimentul de recepționare a unui mesaj este declanșat, socket-ul corespunzător utilizatorului transmite un mesaj de notificare celui care a trimis mesajul, notificând astfel că mesajul s-a transmis cu succes.

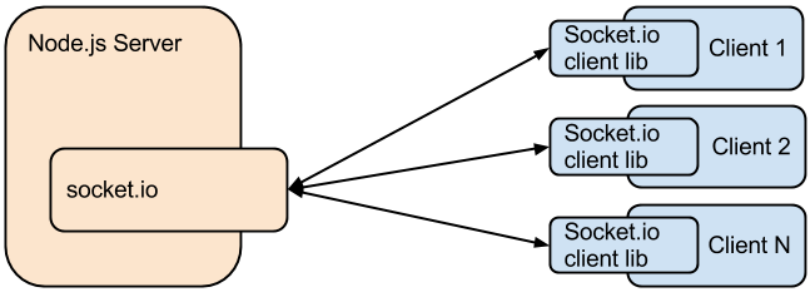


Figura 12: Conexiune de tip socket in NodeJs[4]

[4]JSCRAMBLER. Mixed Signals with Socket.IO and WebRTC. https://jscrambler.com/blog/mixed-signals-with-socket-io-and-webrtc Ultima accesare: 8 iunie 2023

4.5 Notificări

Notificările au pus probleme de la început deoarece, fiind vorba despre o aplicație de comunicare, este important ca schimbările să aibă loc în timp real. Notificările sunt necesare în momentul trimiterii unui mesaj către un utilizator care nu are chat-ul cu respectiva persoană deschis, astfel încât să nu rateze primirea mesajului. Soluția găsită este de a filtra mesajele transmise prin tehnologia de tip socket [6] și de a seta un flag pentru mesajele destinate unui utilizator care nu are chat-ul deschis. În acest fel, frontend-ul poate produce o schimbare în interfață, permițând utilizatorului să observe că a primit o notificare.

De asemenea, notificările sunt necesare și în momentul în care lista de utilizatori suferă modificări. Dacă un membru este suspendat sau reactivat, lista de utilizatori trebuie actualizată astfel încât să evidențieze schimbările. Pentru aceste notificări am folosit socket-uri pentru a transmite un flag de tip alertă, care să producă o alertă în interfața utilizatorului și să îl informeze că pagina trebuie reîncărcată pentru a vedea schimbările.

4.6 Folosirea REST API pentru comunicarea intre client si server

REST API [5] (Representational State Transfer Application Programming Interface) este o metodă de comunicare între servere care permite transmiterea de date folosind cereri HTTP, prin intermediul acestei tehnologii un client trimite o cerere care respecta un anumit standard către un server și acesta ii va răspunde cu date într-un format precum JSON sau XML. Aceste date pot fi informații despre resursele de pe server sau rezultate ale unor operații disponibile pentru o anumită resursă. REST API [5] este metoda principală de comunicare între frontend și backend în aplicația prezentată. Folosind REST API [5] serverele pot să comunice între ele fără să fie necesară folosirea aceluiasi limbaj de programare sau să fie instalate biblioteci speciale, desi REST API [5] dispune de acestă funcționalitate, în cadrul aplicației atât backend-ul cât si frontend-ul o să folosească limbajul JavaScript. Această metodă este folosită pentru a face cereri între cele două componente ale aplicației și pentru a transmite date. Aplicația transmite datele de la frontend către backend prin intermediul unei cereri POST care trimite informații prin intermediul unui corp de mesaj. Acesta este utilizat pentru a trimite datele introduse de utilizator cum ar fi formularul de autentificare, datele unei noi postări, imaginea de avatar, codurile de access pentru utilizatorii noi sau pentru recuperarea parolei. Pe de altă parte cererea GET este utilizată pentru a prelua

informații de la backend, cum ar fi contactele, mesajele din fiecare conversație si din panoul de știri.

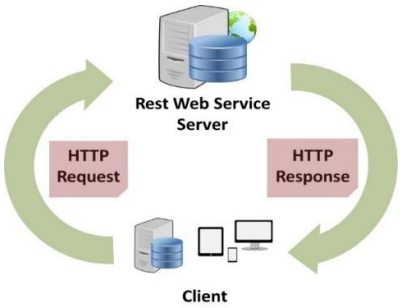


Figura 13: Comunicare folosind REST API[5]

4.7 Criptarea parolei în cadrul autentifcării

În cadrul autentificării am folosit algoritmul bcrypt din biblioteca “bcrypt” [10] din Node.js [4]. Algoritmul bcrypt produce o versiune a hash-ului parolei care poate fi ajustată în funcție de parametrii de cost, astfel atacurile brute-force sunt mai dificile, avantajul acestui algoritm constă în performanța de hash a parolelor, fiind astfel o alegere bună pentru asigurarea securității accesului la date confidențiale. Biblioteca Node.js bcrypt [10] vine cu o interfață simplă de utilizat pentru implementarea algoritmului în cadrul aplicațiilor Node.js [4], fiind astfel o opțiune bună pentru soluții de autentificare în cazul aplicatiilor web.

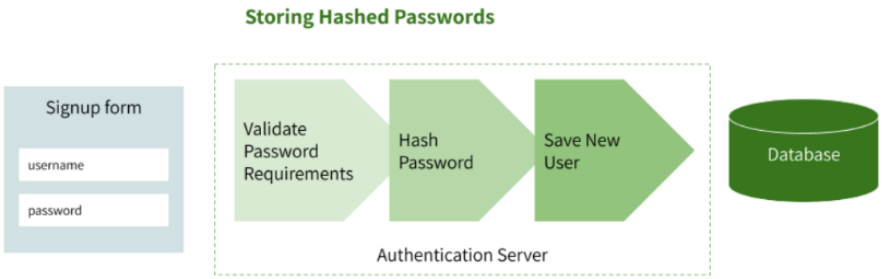


Figura 15: Stocarea parolei criptate in baza de date [11]

[5]Lokesh Gupta. What is an API? https://restfulapi.net/what-is-an-api/ Ultima accesare: 8 iunie 2023

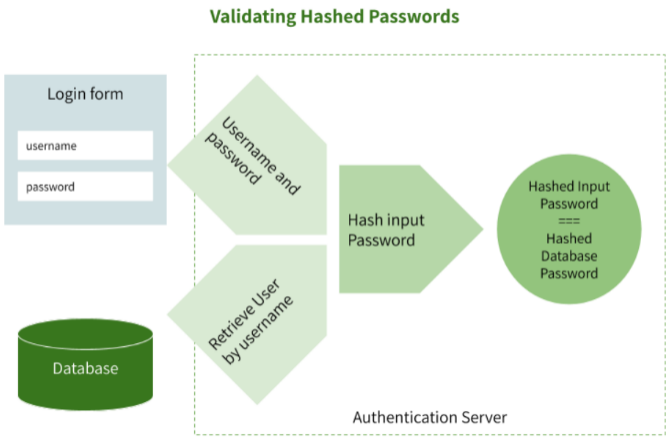


Figura 16: Validarea parolei [11]

4.8 Setarea avatarelor

Pentru o comunicare de calitate, este important ca fiecărui utilizator să-i fie asociată o imagine, pe care o vom numi avatar. În cadrul aplicației, utilizatorul poate alege dintr-un set de avatare generate aleatoriu sau poate încărca propria imagine din calculator. Pentru a genera avatare random am folosit un modul de Node.js [4] numit "github-like-avatar-generator" [12]. Acest modul are la bază un cod JavaScript care definește o funcție numită "generateAvatar", această funcție generează o imagine de avatar aleatorie, utilizând grafica SVG. Funcția returnează un obiect cu trei proprietăți: un șir de caractere codificat în base64 al imaginii SVG generate, un array de culori utilizate în imaginea generată și un element SVG care reprezintă imaginea generată. Funcția generează două culori dominante între 0 și 255, urmând să genereze un array de culori bazat pe aceste culori dominante și câteva variații aleatorii. Dacă se oferă array-ul de culori, se vor folosi acele culori si creează o imagine SVG cu lățimea și înălțimea specificate și o umple cu blocuri ale culorilor generate. În final, encodează imaginea SVG ca un șir de caractere codificat în base64 și o returnează împreună cu culorile generate, ca un array.

Figura 17: Avatare precum celor de pe GitHub [12]

4.9 Generarea codurilor unice pentru autenticitate

În cadrul aplicației a fost important să oferim într-un fel utilizatorilor din extern access la funcționalitățile precum invitarea sau recuperarea parolei fară a risca compromiterea anumitor informații confidențiale. O soluție în acest caz a fost reprezentată de generarea codurilor unice care să fie primite de potențialii utilizatori pe adresa de mail și să fie folosite pentru a verifica autenticitatea. Pentru implementarea acestei soluții a fost nevoie de generarea unui cod unic, stocarea acestui cod în baza de date și folosirea unui serviciu care să transmită mail-ul cu acest cod către o adresa de mail primită de la frontend adică adresa celui care trebuie să folosească codul pentru dovedirea autenticității. După ce potențialul utilizator a completat în interfața codul de acces, acesta primește acces la formularul de creare cont sau la formularul de recuperare parolă, menționez că pentru fiecare scenariu se generează un cod unic (pentru creare cont respectiv recuperare parolă). În momentul în care codul este folosit, se transmite o cerere către backend pentru a da access la acele functionalități ale aplicației și codul este înlaturat din baza de date.

5 DETALII DE IMPLEMENTARE

5.1 Arhitectura aplicației

Arhitectura aplicației este monolitică [2] și funcționează pe un container cu sistemul de operare Oracle Linux Server 7.9 [13] care este o distribuție a sistemului de operare Linux bazată pe Red Hat Compatible Kernel (RHCK) fiind compatibilă cu aceasta și având aceleași caracteristici. Această distribuție mai oferă și integrare cu produse Oracle [17], Oracle Linux Server este optimizată pentru a funcționa în mod ideal cu produsele software Oracle cum ar fi baza de date Oracle [7], pe care o folosesc și în cadrul aplicației.

Aplicația include trei componente principale:

Pentru interfața grafică folosim framework-ul React [3]. Frontend-ul aplicației rulează pe același container cu celelalte componente ale aplicației, foloseste tehnologia socket.io [6] pentru transmiterea in timp real a mesajelor si notificarilor si comunică cu backend-ul prin intermediul unui API REST [5], utilizând cereri HTTP pentru transferul datelor în format JSON.

Logica aplicației este reprezentată de backend care este implementat folosind tehnologia Node.js [4] care oferă o platformă puternică și stabilă pentru dezvoltarea aplicației, rulează pe același container cu frontend-ul și baza de date, furnizează API-ul Express din Node.js [4] care oferă access la functionalități si prin care sunt procesate datele primite de la frontend și inițilizează socket-ul folosit pentru transmiterea în timp real a mesajelor și notificărilor. Acesta comunică cu frontend-ul prin intermediul REST API [5]. Gestionarea conexiunii cu baza de date Oracle [7] se realizează prin intermediul unui driver specific pentru Node.js [4], care permite executarea de interogări și tranzacții către și dinspre baza de date.

Baza de date permite stocarea persistentă a datelor în cadrul aplicației, aceasta rulează pe același container Linux cu frontend-ul și backend-ul, astfel asigură o arhitectură monolitică [2], conexiunea dintre backend și baza de date se realizează prin intermediul unui driver al bazei de date Oracle [7] specific pentru Node.js [4] care folosește pentru conectare un utilizator al bazei de date cu drepturi depline, asigurând comunicarea și executarea interogărilor sau tranzacțiilor către baza de date.

5.2 Probleme apărute în timpul implementării aplicației

5.2.1 Stocarea imaginilor în baza de date NoSQL

Stocarea imaginilor într-o bază de date NoSQL reprezintă o problemă importantă, deoarece acestea sunt fișiere de dimensiuni mari si devine destul de complicată gestionarea lor într-o bază de date relațională, în special în cazul în care dorim să interacționam din cod cu respectiva bază de date. Firebase [14] oferă o soluție eficientă pentru stocarea și gestionarea imaginilor prin intermediul Firebase Storage. Pentru a rezolva problema stocării imaginilor în baza de date Firebase am folosit SDK-ul Firebase în cadrul aplicației. Am creat o bază de date nouă în consola Firebase [14] și am configurat conexiunea către aceasta în fișierul "firebase.js", exemplificat in Cod 1.1, folosind cheia de autentificare și alte detalii specifice.

Cod 1.1: Conectare la baza de date Firebase

```
const firebaseConfig = {
    apiKey: "<api-key>",
    authDomain: "<auth-domain>",
    projectId: "<project-id>",
    storageBucket: "<storage-bucket>",
    messagingSenderId: "<messaging-sender-id>",
    appId: "<app-id>",
    measurementId: "<measurement-id>"
};
firebase.initializeApp(firebaseConfig);
```

După inițializarea conexiunii către baza de date Firebase [14] am creat o colecție în baza de date pe care am numit-o "UserAvatar" si stochează imaginile de profil ale utilizatorilor. Această colecție este folosită în cadrul endpoint-ului "setavatar" al server-ului pentru a stoca imaginea de profil încărcată de utilizator.

Endpoint-ul "setavatar" este utilizat pentru a actualiza baza de date a server-ului cu informațiile despre imaginea de profil. În acest endpoint prima dată se extrage id-ul utilizatorului și imaginea de profil din request, se observă in Codul 1.2:

Cod 1.2: Parametrii din request

```
const userId = req.params.id;
const avatarImage = req.body.image;
```

Se poate observa că se adaugă imaginea de profil în colecția "UserAvatar" a bazei de date Firebase [14] și se extrage id-ul acesteia pentru a putea asocia această imagine cu utilizatorul corespunzător, Codul 1.3.

Cod 1.3: Încarcare imagine in baza de date Firebase

```
const response = await UserAvatar.add({"avatarImage": avatarImage});
const avatar_image_firebase_id = response._delegate._key.path.segments[1];
```

Se actualizează baza de date a server-ului cu informațiile despre imaginea de profil și se returnează un răspuns către client. În cazul în care imaginea a fost setată cu succes se întoarce un JSON cu flag-ul "isSet" având valoarea true și imaginea de profil in base64 pentru a fi folosite de frontend și a continua procesul de creare de cont. Daca imaginea nu a fost setată cu succes se întoarce un mesaj de eroare, se poate observa în Codul 1.4.

Cod 1.4: Actualizare detalii avatar în baza de date

```
const result = await connection.execute(
    `UPDATE AVATARS SET IS_AVATAR_IMAGE_SET = 1, AVATAR_IMAGE_FIREBASE_ID =
'${avatar_image_firebase_id}' WHERE USER_ID = '${req.params.id}'`
  );

  if (result.rowsAffected === 1) {
    return res.json({
      isSet: true,
      image: avatarImage,
    });
  } else {
    return res.json({ msg: "Error in setting the image", status: false });
  }
```

În acest fel, am reușit să rezolv problema stocării imaginilor de profil ale utilizatorilor într-o bază de date Firebase, având în vedere dificultățile pe care le-am avut cu stocarea de astfel de date în baze de date relaționale.

5.2.2 Folosirea socket.io

O altă problemă pe care am întâmpinat-o în implementarea aplicației a fost transmiterea mesajelor în timp real. Inițial am vrut să folosesc o soluție bazată pe REST API [5], dar după ce am aprofundat problema am realizat că socket.io [6] este o soluție mai bună în această situatie.

Socket.io [6] este mai potrivit pentru trimiterea mesajelor în timp real decât REST API [5] pentru că se bazează pe protocolul HTTP care funcționează pe un model de cerere-răspuns

adică clientul trebuie să trimită o cerere către server. În funcție de mărimea și complexitatea cererii și de încărcarea serverului acest proces poate să dureze si să fie nevoie întotdeuna de un handler care să pornească acțiunea de a cere informatii sau trimite informatii. În schimb socket.io utilizează protocolul WebSockets [15] care permite trimiterea de date în timp real între client și server asigurând o comunicare în timp real fără necesitatea de a trimite cereri. și aștepta răspunsuri. Codul 2.1 reprezintă inițializarea unei instanțe Socket.io.

Cod 2.1: Inițializare socket

```
const io = socket(server_express, {

  cors: {
    origin: "http://ec2-16-16-220-152.eu-north-1.compute.amazonaws.com:3000",
    credentials: true,
  },
});

global.onlineUsers = new Map();
io.on("connection", (socket) => {
  global.chatSocket = socket;
  socket.on("addUser", (userId) => {
    onlineUsers.set(user_id, socket.id);
  });

  socket.on("sendMsg", (input) => {
    socket.to(onlineUsers.get(input.to)).emit("msgRecieve", input);
  });
});
```

În codul de mai sus când un utilizator se conectează la server, se adaugă ID-ul acestuia și ID-ul socket-ului asociat într-o hartă globală a utilizatorilor conectați. Când un utilizator trimite un mesaj se caută ID-ul socket-ului asociat destinatarului și se utilizează metoda socket.to pentru a trimite mesajul direct acestuia.

Această abordare este mai eficientă decât folosirea REST API [5] deoarece este mai rapidă si clientul nu trebuie să facă solicitări repetate către server pentru a obține răspunsuri ceea ce ar consuma multe resurse si nici nu ar fi confortabil pentru utlizator. Codul 2.2 este scris în React [3] și folosește hook-ul useEffect. Când componenta este randată, acest hook este apelat și verifică dacă există un utilizator curent. Dacă utilizatorul există, se realizează o conexiune cu serverul de socket folosind host-ul specificat, urmând ca utilizatorul curent sâ fie adăugat la lista de utilizatori online cu ajutorul evenimentului "addUser".

Cod 2.2: Adăugare utilizator in map-ul socket-ului

```
useEffect(() => {
  socket.current = io(host);
  socket.current.emit("addUser", thisUser.USER_ID);
}, [thisUser]);
```

Aceaste instrucțiuni fac parte din implementarea clientului pentru trimiterea mesajelor folosind socket.io [6]. În momentul în care un utilizator este logat, acesta este adăugat la lista de utilizatori online și poate începe să trimită mesaje către alți utilizatori, Codul 2.3.

Fragmentele de Cod 2.3 si Cod 2.4 sunt folosite pentru a transmite un mesaj de la utilizatorul curent către un alt utilizator în aplicație prin selectarea de către utilizatorul curent a chat-ului dintre cei doi dar și pentru primirea unei mesaj trimis de catre alt user.

Cod 2.3: Transmitere mesaje prin intermediul socket-ului

```
socket.current.emit("sendMsg", {
    to: receiver.USER_ID,
    from: sender.USER_ID,
    message_text
});
```

Cod 2.4: Recepționarea mesajelor prin intermediul socket-ului

```
socket.current.on("msgRecieve", (input) => {
    setArrivalInput({ from_the_same_user: false, message_text: input.msg });
});
```

5.2.3 Notificări

Transmiterea în timp real a notificărilor a fost o altă problemă în procesul de implementare al aplicației. Pentru a rezolva această problemă, am ales să folosesc socket-uri pentru a seta flag-uri în momentul în care un mesaj este transmis către un utilizator care nu are chat-ul deschis.

În Codul 3.1, se inițializează un vector care conține valori booleane (true sau false) pe fiecare poziție, în funcție de faptul dacă utilizatorul de pe respectiva poziție a trimis sau nu un mesaj.

Cod 3.1: Recepționarea notificărilor sub formă de mesaje cu flag de tip alert

```
if (socket.current) {
    socket.current.on("msgRecieve", (input) => {
      // daca se primeste o alerta atunci se seteaza variabila
      // corespunzatoare
      if(input.flag === "alert") {
          setAlert(true);
          setMsgAlert(input.msg);
      } else {

     // se pune un flag pentru fiecare utilizator care a trimis un mesaj pentru a
     // a fi afisate in interfata sub forma de notificari
      for (let elem in contacts) {
            if(contacts[elem].USER_ID === input.from) {
              notifiedContacts[elem] = true;
            }
      }
      setArray(notifiedContacts);
   }
}
```

În Codul 3.2, un contact este afișat în interfața utilizatorului fie ca selectat (clasa CSS “selected”), fie ca suspendat (clasa CSS “suspendedUser”), fie cu mesaje necitite sub formă de notificări (clasa CSS “unReadMessages”).

Codul 3.2: Afișare contact

```
className={`contact ${index === currentSelected ? "selected" : (contact.STATUS
=== 'SUSPENDED' ? "suspendedUser" : (array[index] === true ? "unReadMessages" :
""))
}`}
```

În Codul 3.3, prin intermediul socket-ului se trimite o alertă către toți utilizatorii (cu excepția adminstratorului, care nu trebuie să primească notificare) pentru a notifica că utilizatorul selectat a fost activat.

Codul 3.3: Trimitere flag-ul alert, cu msg = “other” când se generează o notificare

```
let flag = "alert";
let msg = "other";
for (let elem in toBeNotifiedContacts) {
  socket.current.emit("sendMsg", {
    to: parseInt(toBeNotifiedContacts[elem].USER_ID),
    from: contactToBeUpdated.USER_ID,
    msg,
    flag
  });
}
```

În Codul 3.4, prin intermediul socket-ului se trimite o alertă către contactul care trebuie suspendat și către ceilalți utilizatori (cu excepția adminstratorului, care nu trebuie să primească notificarea). Astfel, utilizatorii care primesc notificarea pot afișa schimbările în interfața grafică (frontend).

Codul 3.3: Trimitere flag-ul alert, cu msg “same” când se generează o notificare

```
let flag = "alert";
let msg = "same"
socket.current.emit("sendMsg", {
  to: parseInt(selectedOption),
  from: contactToBeUpdated.USER_ID,
  msg,
  flag
});
msg = "other";
for (let elem in toBeNotifiedContacts) {
  if (parseInt(selectedOption) !== toBeNotifiedContacts[elem].USER_ID) {
    socket.current.emit("sendMsg", {
      to: parseInt(toBeNotifiedContacts[elem].USER_ID),
      from: contactToBeUpdated.USER_ID,
      msg,
      flag
    });
  }
}
```

Dacă mesajul este “same”, atunci utilizatorul care primește notificarea este cel suspendat și trebuie să părăsească sesiunea curentă. Dacă mesajul este “other”, atunci utilizatorii care primesc notificarea trebuie să reactualizeze lista de contacte prin intermediul butonului "reloadPage".

Codul 3.4: Afișarea în interfata a unei notificari cu flag de tip alert

```
    <div>
      {msg === "same" ? (
        <div className="alert" style={styles.alert}>
          <p>Your account has been suspended. Please leave the page</p>
        </div>
      ) : (<div className="alert" style={styles.alert}>
            <p>The contacts have been updated. Please reload the page.</p>
            <button style={styles.button} onClick={reloadPage}>Reload
Page</button>
          </div>)
      }
    </div>
```

5.3 Conectare la baza de date Oracle

Codul 5.1 utilizeaza driver-ul Oracle Database[6] pentru Node.js [4] pentru a stabili o conexiune la baza de date, codul rulează într-o funcție async care așteptă ca o promisiune să fie îndeplinită. Metoda getConnection() este folosită pentru a returna un obiect de tip Connection aceasta primește un obiect ca parametru, care conține proprietățile de configurare pentru conexiune: user (Numele de utilizator pentru contul de bază de date utilizat pentru autentificare), password (Parola pentru contul de bază de date),

[6] Node.js node-oracledb version 6. https://oracle.github.io/node-oracledb/. Ultima accesare: 16 iunie 2023

connectString (Șirul de conexiune pentru baza de date, care specifică gazda și numele serviciului. În acest caz, se conectează la baza de date care rulează pe localhost și utilizează numele serviciului XE).

Obiectul rezultat Connection este utilizat pentru a efectua operații de bază de date, cum ar fi executarea de interogări SQL sau tranzacții.

Cod 5.1 Inițializare conexiune OracleDB

```
connection = await oracledb.getConnection( {
    user          : myuser,
    password      : mypw,
    connectString : "localhost/XE"
});
```

5.4 Implementarea listei de contacte

În cadrul oricărei aplicații de comunicare este foarte important să existe un spațiu în care să se regăsească o listă cu utilizatorii care pot să interacționeze cu utilizatorul curent pe care o să o numim în continuare "Lista de contacte".

În interfața, "Lista de contacte" reprezintă 65% din spațiul de lucru și este reprezentată printr-o succesiune de informații în care fiecare element este reprezentat sub forma unui "box" care permite să se facă click pe acesta, deschizând astfel prin intermediul unei variabile un chat cu utilizatorul pe care s-a făcut click.

Codul 6.1 (ANEXE) afișează o listă de contacte și oferă interacțiune cu acestea. În detaliu:

- Map-ul contacts.map iterează prin fiecare contact din lista de contacte.
- Dacă contactul curent nu aparține organizației selectate, nu este afișat (return null).
- Variabilele isSuspendedUser, isSelected și hasUnreadMessages sunt stabilite în funcție de starea curentă a contactului și a selecției utilizatorului.
- La apăsarea pe elementul div de contact se apelează funcția changeCurrentChat având indexul contactului și obiectul contactului ca argumente.
- În fiecare element div de contact sunt afișate avatarul si username-ul

Codul 6.2 definește un endpoint pentru a obține o listă cu toți utilizatorii din baza de date care sunt diferiți de utilizatorul curent împreună cu informațiile lor și informațiile despre avatarul utilizatorului.

- app.get('/api/auth/allusers/:id', async (req, res) => { ... } definește un endpoint GET care va fi accesat prin URL-ul /api/auth/allusers/:id. Endpoint-ul primește un parametru id, reprezentând ID-ul utilizatorului curent.
- const result = await connection.execute(...) definește o interogare SQL care extrage numele de utilizator, prenumele, numele de familie, ID-ul organizației, adresa de e-mail, ID-ul utilizatorului, statusul și ID-ul imaginii de avatar al utilizatorului din tabelele USERS și AVATARS pentru toți utilizatorii cu excepția celui curent.
- const snapshot = await UserAvatar.get(); obține o referință către colecția de imagini de avatar stocate în baza de date Firebase, folosind metoda get() a obiectului UserAvatar.
- const list = snapshot.docs.map((doc) => ({ id: doc.id, ...doc.data() })); obține o listă cu toate documentele din colecția de imagini de avatar

Codul 6.2: Endpoint-ul request-ului care obține lista cu toți utilizatorii

```
app.get('/api/auth/allusers/:id', async (req, res) => {

    const userId = req.params.id;

    const result = await connection.execute(
      `SELECT A.USER_NAME, A.FIRST_NAME, A.LAST_NAME, A.ORGANIZATION_ID,
A.EMAIL_ADDRESS, A.USER_ID, A.STATUS,
       B.AVATAR_IMAGE_FIREBASE_ID FROM USERS A, AVATARS B WHERE A.USER_ID =
B.USER_ID AND A.USER_ID != '${userId}'`
    );

    const snapshot = await UserAvatar.get();
    const list = snapshot.docs.map((doc) => ({ id: doc.id, ...doc.data() }));

    for(elem in result.rows) {
      let firebaseID = result.rows[elem].AVATAR_IMAGE_FIREBASE_ID;

      for (l in list) {
        if (list[l].id === firebaseID) {
          result.rows[elem].AVATAR_IMAGE = list[l].avatarImage
        }
      }
    }

    return(res.json(result.rows));
})
```

5.5 Implementarea codurilor unice pentru autenticitate

Codurile de acces sunt folosite în cazul recuperării parolei uitate dar și în cazul când un utilizator dorește să-și creeze cont. În momentul în care un administrator invita un nou membru, se generează un cod unic care este trimis pe adresa de mail a potențialului utilizator, Codul 6.2 (ANEXE) exemplifică procesul de generare a unui cod de acces, adăugarea acestuia în baza de date, pentru a putea fi comparat cu cel primit de utilizator. În acest caz, codul de access depinde de id-ul organizației, astfel că utilizator este direcționat direct spre organizația sa în momentul inițializării contului, asigurând confidețialitate și securitate. Pentru trimiterea de email-uri din Node.js [4] am folosit biblioteca nodemailer [16], care folosește un cont de Gmail. Procesul este asemănător și pentru recuperarea parolei uitate dar codul de recuperare depinde de id-ul utilizatorului, astfel parola schimbată este asociată utilizatorului care trebuie.

5.6 Implementarea workspace-ului

Workspace-ul este reprezentat de butoanele TeamManagement, SaveState, detaliile despre utilizator (avatarul utilizatorului, numele și prenumele) și Lista de Contacte. TeamManagement este un buton accesibil doar administratorului și oferă acces la anumite meniuri destinate gestionării organizației, cum ar fi invitarea unui nou membru în organizație, suspendarea unui membru din diverse motive, activarea unui membru și butoane dedicate generării de anumite rapoarte, cum ar fi numărul de mesaje trimise în intervalele orare de la 8 la 16, numărul de mesaje trimise sau primite de fiecare utilizator, numărul de utilizatori activi sau suspendați și posibilitatea de descărcare a unui CSV cu informații despre toți utilizatorii.

Codul 6.3 evidentiază modul în care informațiile necesare raportului cu numarul de mesaje primite sunt extrase pentru a fi trimise către frontend.

Codul 6.3: extrage din baza de date informatiile necesare pentru un raport

```
app.get('/api/messages/allusersreceivedmessages/:id', async (req, res) => {

   const orgId = req.params.id;

   const result = await connection.execute(
    `select COUNT(B.message) AS NRmessages, A.user_id, A.first_name,
A.last_name, A.organization_id from messages B, users A, conversations C where
(A.user_id = C.to_id or A.user_id = C.from_id) and A.user_id != B.from_id and
C.conversation_id = B.conversation_id and A.organization_id = '${orgId}' group by
A.user_id, A.first_name, A.last_name, A.organization_id`
    );

   console.log(result);

   return(res.json(result.rows));
})
```

Butonul TeamManagement setează valoarea unei variabile la 1, indicând că trebuie afișată o altă componentă în locul chat-ului, reprezentând meniul de administrare a echipei. Butonul SaveState apelează o funcție care face un request de tip POST spre HTTP, notificând astfel serverul că modificările trebuie reținute și în baza de date. Imaginea de profil a utilizatorului este preluată din localStorage și afișată împreună cu numele și prenumele.

Butoanele din meniul TeamManagement oferă funcționalitățile pentru administrarea echipei, funcționalități precum InviteMember, SuspendMember, ActivateMember permit modificarea informațiilor despre utilizatori, prin adăugarea unor membrii noi sau schimbarea statusului acestora. Funcționalitatea de InviteMember presupune invitarea în organizație a unui nou membru, prin trimiterea pe adresa de email a acestuia a unui cod de acces unic.

6 Evaluare

Înainte de implementarea finală a aplicației web de comunicare, am efectuat mai multe teste pentru a evalua performanța și funcționalitatea acesteia. Am utilizat Lighthouse[7] care o extensie pentru browserul Chrome dezvoltată de Google, pentru a obține metrici clare privind performanța aplicației și pentru a le compara cu alte soluții existente.

În urma utilizării Ligthhouse pe aplicația prezentată și pe alte doua aplicații existente pe piață am creat tabelul de mai jos:

Tabela 1: Performanțele aplicației

Metrica	First Contentful Paint	Largest Contentful Paint	Total Blocking Time	Cumulative Layout Shift	Speed Index
Aplicația prezentată	0.4s	21.8s	50ms	0.451	1.1s
Instagram	0.6s	3.7s	250ms	0	3.0s
Facebook	1.5s	3.7s	740ms	0.027	3.0s

O explicație a fiecărei metrici este prezentată mai jos:

- First Contentful Paint (FCP)[8]– reprezintă timpul necesar pentru a afișa primul element în cadrul aplicației. Aplicația prezentată are o încărcare inițială rapidă care o situează mai bine decat celelalte aplicații din comparație.
- Largest Contentful Paint (LCP)[9] - reprezintă timpul necesar pentru a afișa cel mai mare element în cadrul aplicației. Aplicația prezentată are timpul LCP de 21.8s mai mare decât Instagram (3.7s) și Facebook (3.7s), ceea sugerează probleme de încarcare a conținutului.

[7] Documentation Lightouse. https://developer.chrome.com/docs/lighthouse/overview/ Ultima accesare: 8 iunie 2023

[8] Documentation Lightouse. First Contentful Paint. https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint/ Ultima accesare: 16 iunie 2023

[9] Documentation Lightouse. Largest Contenful Paint https://developer.chrome.com/docs/lighthouse/performance/lighthouse-largest-contentful-paint/ Ultima accesare: 16 iunie 2023

- Total Blocking Time (TBT)[10] - reprezintă timpul total în care aplicația este blocată și utilizatorii nu pot interacționa cu aceasta. Aplicația prezentată are TBT de doar 50ms care o situează mai bine decat celelalte aplicații din comparație.
- Cumulative Layout Shift (CLS)[11] - reprezintă gradul de stabilitate al elementelor din aplicației în timpul încărcării. Aplicația prezentată este in dezavantaj deoarece are un CLS de 0.451, care este mai mare decât Instagram (0) și Facebook (0.027)
- Speed Index[12] - reprezintă viteza percepută de utilizatori în timpul încărcării aplicației. Aplicația prezentată are un Speed Index de 1.1s, ceea ce o face mai rapidă în comparație cu Instagram(0) și Facebook(0.027)

În urma metricilor de mai sus putem spune că un avantaj al aplicației prezentate este dat de simplitate, având din acest motiv sunt valori mai bune în ceea ce privește unele aspecte, dar și dezavantaje în încărcarea anumitor componente cu dimensiunea mai mare, cum ar fi lista de contacte.

Pentru asigurarea faptului ca aplicația rulează pe orice platforma si pentru a facilita procesul de testare am creat un script care folosind tehnologii precum Docker instalează aplicația pe orice sisteme de operare, de asemenea, am utilizat serviciul Amazon Web Services[13] pentru a implementa aplicația în mediul de cloud și am realizat un set de teste manual pentru a verifica funcționarea corectă a aplicației în producție, atât în ceea ce privește performanța, cât și funcționalitatea.

Pașii pe care i-am luat în considerare în procesul de deployment sunt următorii:

Pasul 1: Crearea instanței EC2 pe Amazon Web Services (AWS). Pentru a pune aplicația la dispoziția utilizatorilor am creat o instanță EC2 pe platforma Amazon Web Services. Am ales o configurație t3.xlarge și sistemul de operare Ubuntu. De asemenea, am generat o cheie SSH pentru a permite conexiunea securizată la instanță.

---

[10] Documentation Lightouse. Total Blocking Time https://developer.chrome.com/docs/lighthouse/performance/lighthouse-total-blocking-time/ Ultima accesare: 16 iunie 2023

[11] Cumulative Layout Shift (CLS). https://web.dev/i18n/en/cls/ Ultima accesare: 16 iunie 2023

[12] Documentation Lightouse. Speed Index. https://developer.chrome.com/docs/lighthouse/performance/speed-index/ Ultima accesare: 16 iunie 2023

[13]Shyamli Jha. What Is AWS? https://www.simplilearn.com/tutorials/aws-tutorial/what-is-aws Ultima accesare 8 iunie 2023

Pasul 2: Pentru a permite comunicarea corectă între frontend și backend, am configurat regulile de securitate pentru instanța EC2. Folosind meniul Security Groups și am adăugat reguli pentru fiecare port utilizat de frontend și backend.

Pasul 3: Pentru transferul fisierelor necesare pe instanța EC2 am folosit SSH.

Pasul 4: Utilizarea bazei de date Oracle [7] în aplicație. Am instalat containerul Docker pentru Oracle Database Enterprise Edition. Acest pas implică descarcărea imaginii de Docker si maparea porturilor necesare pentru a accesa aplicația din exterior:

Pasul 5: Crearea unui utilizator cu drepturi depline în baza de date. Am folosit comenzile SQL pentru a crea un utilizator în baza de date cu drepturi depline. Acest utilizator va fi utilizat de aplicație pentru accesul la baza de date.

Pasul 6: Actualizarea adresei host pentru backend în frontend. În fisierul de configurare al aplicației am setat valoarea pentru host care inițial era localhost la adresa IP publica a instanței EC2, astfel aplicația este accesibilă pentru utilizatori prin folosirea link-ului pus la dispoziție de Amazon Web Services.

Pentru evaluare aplicației am creat un formular la care au raspuns mai mulți oameni:

Întrebarile sunt urmatoarele: Considerați că aplicația “Rețea de comunicare în grup” este o alegere bună pentru comunicare și colaborare?, Cât de ușor v-a fost să creați un cont și să vă alăturați unei organizații?, Cum ați evalua funcționalitatea chat-ului direct unu la unu din aplicație?, Cât de mult v-a plăcut să utilizați feed-ul organizației și interacțiunea cu acesta?, Vă rugăm să evaluați caracteristicile de generare random a avatarului și opțiunile de alegere a imaginii de profil disponibile, Cât de mult doriți să vedeți în viitor caracteristici sau funcționalități suplimentare în aplicație?, Cât de accesibil este procesul de generarea a rapoartelor despre utilizatori? Cum a fost experiența dvs. în colaborarea cu administratorii organizației? Cum ați resimțit procesul de suspendare și reactivare a membrilor? În ce măsură ați recomanda aplicația altor utilizatori?

Tabela 2: Întrebari formular feedback

Nr. Intrebare	Raspuns 1	Raspuns 2	Raspuns 3	Raspuns 4	Raspuns 5
1	0%	0%	5.3%	26.3%	68.4%
2	0%	0%	0%	36.8%	63.2%
3	0%	0%	5.3%	36.8%	57.9%
4	0%	0%	0%	52.6%	47.4%
5	0%	0%	5.3%	52.6%	42.1%
6	0%	0%	0%	21.1%	78.9%
7	10.5	0%	5.3%	26.3%	57.9%
8	0%	0%	0%	42.1%	57.9%
9	0%	0%	0%	36.8%	63.2%
10	0%	0%	0%	21.1%	78.9%

Feedback-ului indică o percepție generală favorabilă a aplicației “Rețea de comunicare în grup” în ceea ce privește comunicarea și colaborarea, cu aprecieri în special pentru chat-ul direct, feed-ul organizației și caracteristicile de generare a avatarului. Există și aspecte care pot fi imbunatățite cum ar fi procesele de creare a contului, generare a rapoartelor și colaborare cu administratorii organizației.

În ceea ce privește procesul de creare a contului și alăturarea la o organizație se poate realiza un proces echivalent care să presupună mai puțini pasi cum ar fi autentificarea prin intermediul conturilor de social media sau prin intermediul adresei de e-mail.

7 CONCLUZII

Lucrarea de licență prezintă dezvoltare unei aplicații web pentru Desktop având ca scop eficientizarea proceselor de colaborare și comunicare din cadrul unei organizații. Aplicatia presupune comunicarea intre membrii unei organizatii si generarea de rapoarte folositoare care pot eficientiza procesele din cadrul acesteia.

Prin intermediul acestui proiect au fost aprofundate mai multe tehnologii și instrumente relevante pentru domeniu astfel încât să se asigure o dezvoltare eficientă a aplicației,

O altă realizare importantă este dezvoltarea aplicației web în sine care dispune de numeroase funcționalități menite să faciliteze comunicarea și colaborarea între membrii organizației, functionalități care au fost verificate de mai multi utilizatori, avand feedback in mare parte pozitiv.

Un alt aspect important al lucrării este deployment-ul aplicației într-un mediu cloud, mai exact folosind serviciul AWS (Amazon Web Services), lucru care a permis accesul utilizatorilor la aplicație, fara sa fie nevoie de anumiți pași pentru crearea unui setup.

O realizare pe care o consider folositoarea a fost gasirea si apoi investigarea cu un tool prin care am realizat analize comparative a performanțelor, astfel s-a putut evalua eficiența și calitatea aplicației dezvoltate în comparație cu alte soluții similare de pe piață, oferind o perspectivă clară asupra avantajelor pe care poate să le aducă aplicația “Rețea de comunicare in grup” în cadrul unei organizații.

În concluzie, lucrarea “Rețea de comunicare în grup” a presupus dezvoltarea unei aplicații web pentru Desktop care să aducă administratorilor de organizații de orice dimensiune o soluție gratuită pentru comunicare și care să eficientizeze procesele in cadrul echipei lor. Realizările principale ale acestei lucrări constau în tehnologiile moderne folosite pentru dezvoltarea unei aplicații cu funcționalități diverse având posibilitatea ca acestea să fie testate de utilizatori printr-o manieră accesibilă care să nu necesite etape tehnice. O altă realizare este reprezentată de deployment-ul aplicației într-un mediu cloud si folosirea de tool-uri de investigare pentru compararea performanțelor obținute cu alte soluții existente. Aceste rezultate demonstrează importanța și beneficiile aduse de aplicația dezvoltată în cadrul organizațiilor.

8 BIBLIOGRAFIE

[1] Kelly Lyons. 28 Top Social Media Platforms Worldwide. https://www.semrush.com/blog/most-popular-social-media-platforms/ Ultima accesare: 8 iunie 2023

[2] Rahul Awati. Monolithic architecture for software explained. https://www.techtarget.com/whatis/definition/monolithic-architecture Ultima accesare: 9 iunie 2023

[3] React. Getting Started. https://legacy.reactjs.org/docs/getting-started.html Ultima accesare: 8 iunie 2023

[4] Node.js®. Cross-platform JavaScript runtime environment https://nodejs.org/en. Ultima accesare: 10 iunie 2023

[5] What is a REST API? https://www.ibm.com/topics/rest-apis. Ultima accesare: 10 iunie 2023

[6] Socket.IO. Get Started. https://socket.io/get-started/chat Ultima accesare: 8 iunie 2023

[7] Oracle. Tehnologii Oracle Database https://www.oracle.com/ro/database/technologies/ Ultima accesare: 8 iunie

[8] Vartika Kashyap. 17 Best Team Chat Apps (To Use in 2023): Who's Here to Stay?. https://www.proofhub.com/articles/team-chat-apps Ultima accesare: 8 iunie 2023

[9] @openapplus/react-auto-chart. Welcome to react-auto-chart https://www.npmjs.com/package/@openapplus/react-auto-chart Ultima accesare: 8 iunie

[10] Nathan Sebhastian. Hashing passwords in NodeJS with bcrypt library. https://sebhastian.com/bcrypt-node/?utm_content=cmp-true Ultima accesare: 9 iunie 2023

[11] Karl Hughes. https://developer.okta.com/blog/2021/03/05/ultimate-guide-to-password-hashing-in-okta Ultima accesare: 8 iunie 2023

[12] Github-like random avatar generator. https://www.npmjs.com/package/github-like-avatar-generator?activeTab=readme Ultima accesare: 8 iunie 2023

[13] Oracle® Linux 7. Release Notes for Oracle Linux 7.9 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/ Ultima accesare: 8 iunie 2023

[14] Firebase Realtime Database. NoSQL cloud database. https://firebase.google.com/docs/database. Ultima accesare: 10 iunie 2023

[15] WebSockets. The WebSocket API. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. Ultima accesare: 10 iunie 2023

[16] Nodemailer. Powered by EmailEngine. https://nodemailer.com/about/ Ultima accesare: 9 iunie 2023

[17] Oracle. Oracle Linux. https://docs.oracle.com/en/operating-systems/oracle-linux/ Ultima accesare: 21 iunie 2023

[18] Cameron Chapman. Color Theory for Designers. https://www.smashingmagazine.com/2010/01/color-theory-for-designers-part-1-the-meaning-of-color/ Ultima accesare: 23 iunie 2023

9 ANEXE

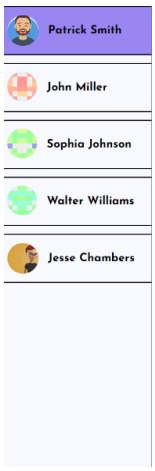
Codul 6.1: Reprezentarea listei de contacte in React

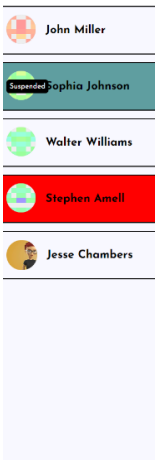
```
<div className="contacts">
  {contacts.map((contact, index) => {
    if (contact.ORGANIZATION_ID !== currentIdOrganization) {
      return null;
    }

    const isSuspendedUser = contact.STATUS === 'SUSPENDED';
    const isSelected = index === currentSelected;
    const hasUnreadMessages = array[index] === true;

    const contactClassNames = `contact ${isSelected ? "selected" : ""}
${isSuspendedUser ? "suspendedUser" : ""} ${hasUnreadMessages ? "unReadMessages"
: ""}`;

    return (
      <div
        key={contact.USER_ID}
        className={contactClassNames}
        onClick={() => changeCurrentChat(index, contact)}
      >
        <div className={contact.AVATAR_IMAGE.startsWith("data:image/jpeg;base64")
? "image-preview" : "avatar"}>
          <img
            src={contact.AVATAR_IMAGE.startsWith("${contact.AVATAR_IMAGE}`}
            alt=""
          />
        </div>
        <div className="username">
          <h3>{`${contact.FIRST_NAME} ${contact.LAST_NAME}`}</h3>
        </div>
      </div>
    );
  })}
</div>
```





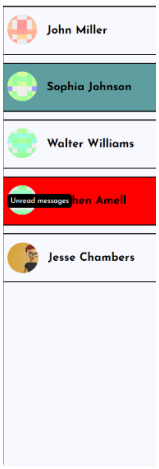


Figura 18a: utilizator selectat Figura 18b: utilizator suspendat Figura 18c: mesaje necitite

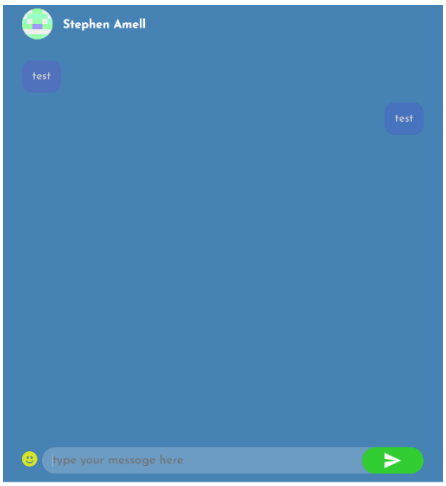


Figura 19: Chat

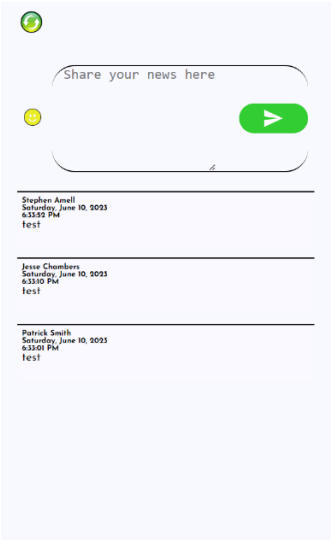


Figura 20: feed-ul organizației

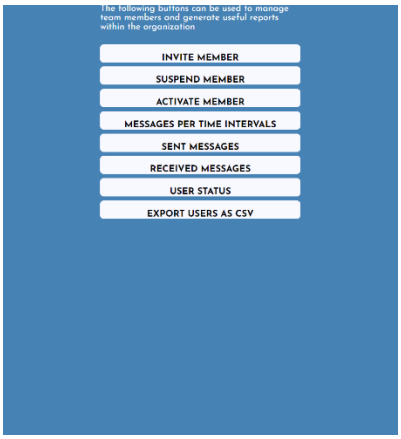


Figura 21: TeamManagement

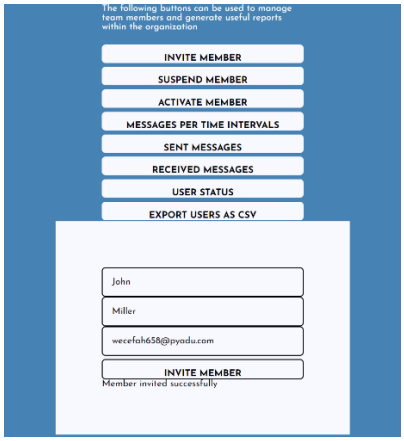


Figura 22: InviteMember

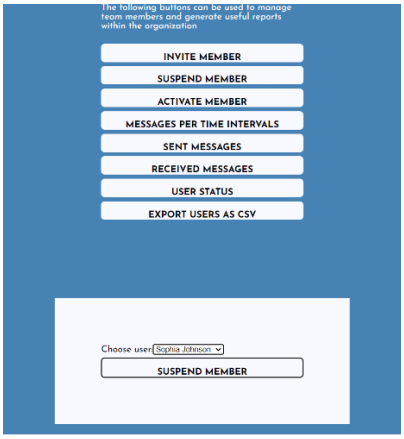
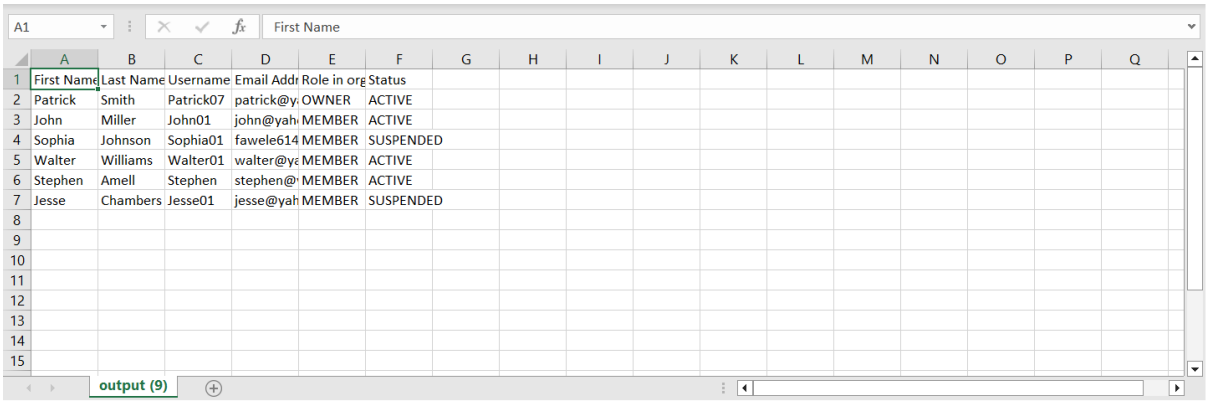


Figura 23: SuspendMember



	A	B	C	D	E	F
1	First Name	Last Name	Username	Email Addr	Role in org	Status
2	Patrick	Smith	Patrick07	patrick@y	OWNER	ACTIVE
3	John	Miller	John01	john@yah	MEMBER	ACTIVE
4	Sophia	Johnson	Sophia01	fawele614	MEMBER	SUSPENDED
5	Walter	Williams	Walter01	walter@ya	MEMBER	ACTIVE
6	Stephen	Amell	Stephen	stephen@	MEMBER	ACTIVE
7	Jesse	Chambers	Jesse01	jesse@yah	MEMBER	SUSPENDED

Figura 24: Raport CSV generat din aplicație

Figura 25: Raport numarul de mesaje trimise în intervale de timp

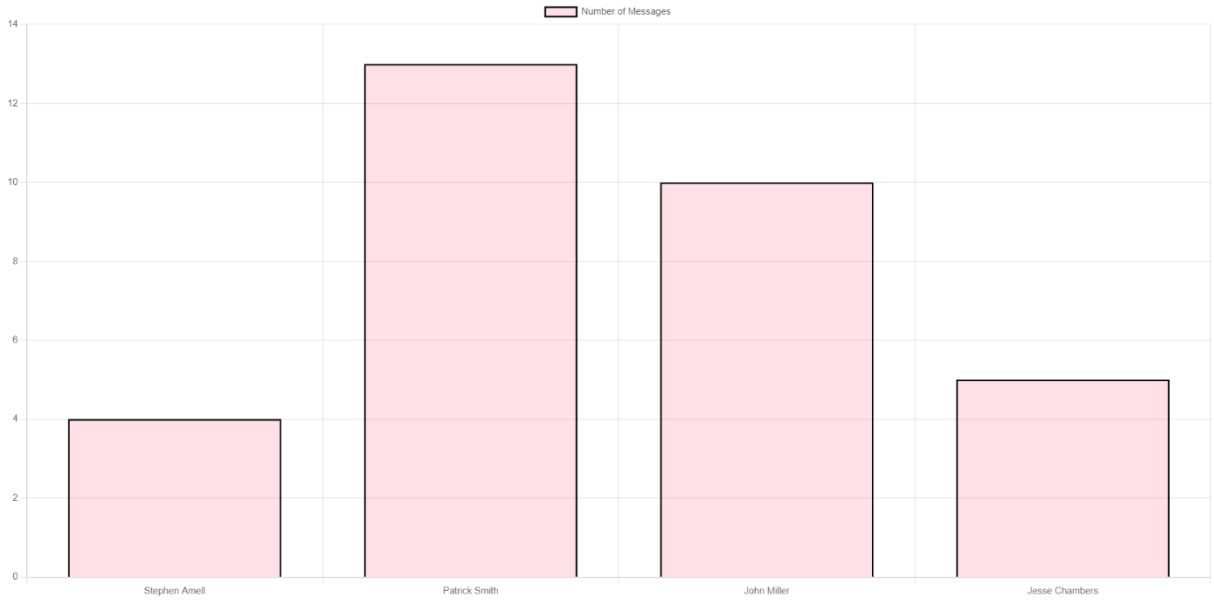


Figura 26: Raport cu numarul de mesaje trimise de fiecare utilizator din organizație

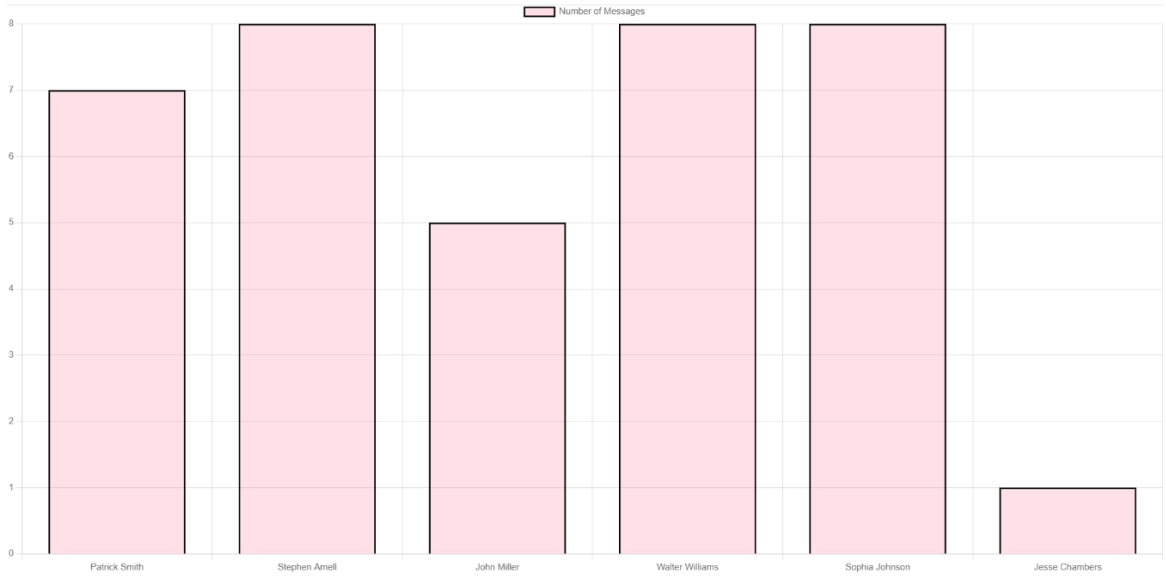


Figura 27: Raport cu numarul de mesaje primite de fiecare utilizator din organizație

Codul 6.2: exemplifică procesul de generare a unui cod de acces

```
        app.post('/api/auth/inviteMember', async (req, res) => {
          const resultUsers = await connection.execute(
            `SELECT * FROM USERS WHERE USER_ID = '${req.body.ownerId}'`
          );
          let user = resultUsers.rows[0];
          const organization_id = user.ORGANIZATION_ID;
          const resultOrganization = await connection.execute(
            `SELECT * FROM ORGANIZATIONS WHERE ORGANIZATION_ID =
'${organization_id}'`
          );
          const organization_name = resultOrganization.rows[0].ORGANIZATION_NAME;
          let transporter = nodemailer.createTransport({
            service: 'gmail',
            auth: {
                user: 'teamconnect.log@gmail.com',
                pass: 'ygvdprsinjwxeaup'
            }
          });
          const accessCode = generateRandomString(5);
          let mailOptions = {
            from: 'teamconnect.log@gmail.com', // sender address
            to: req.body.mail, // list of receivers
            subject: 'Join our organization', // Subject line
            text: 'Hello ' + req.body.firstname + ", " + organization_name + "
organization would like you to be part of our team. In order to create your
account, please use the following access code: " + accessCode // plain text body
          };
          transporter.sendMail(mailOptions, (error, info) => {
            if (error) {
                console.log(error);
                return res.json({ msg: "Error in sending the message", status:
false })
              } else {
                console.log('Message sent: %s', info.messageId);
            }
          });
          console.log(accessCode);
          const result1 = await connection.execute(
            `SELECT MAX(ACCESS_CODE_ID) FROM ACCESS_CODES`
          );
          let idAccessCode = 1;
          if (result1.rows[0]["MAX(ACCESS_CODE_ID)"]) {
            idAccessCode = result1.rows[0]["MAX(ACCESS_CODE_ID)"] + 1;
          }
```