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

CUPRINS

SINOPSIS

Problema care se dorește a fi abordată este necesitatea unei aplicații care să gestioneze comunicarea eficientă între membrii diferitelor organizații. Această aplicație are ca scopuri principale utilizarea tehnologiilor actuale pentru a crea un mediu online în care membrii unei organizații pot comunica între ei prin intermediul unui chat, dar și cu întreaga organizație printr-un topic special destinat distribuirii de știri.

Aplicația fiind orientată către un spațiu de muncă, trebuie să ofere adminstratorului organizației diverse instrumente de administrare, cum ar fi: invitația sau înlăturarea unui membru, dar și rapoarte care indică anumite particularități despre membrii sau interacțiunea dintre aceștia. În ceea ce privește publicul țintă, acesta este reprezentat de orice organizație care dorește să comunice.

Pentru dezvoltarea aplicației, am ales să utilizez Node.js ca limbaj de programare și React ca framework de frontend. Aplicația va fi construită pe o bază de date relațională și va include diverse funcționalități, cum ar fi: un sistem de chat, un sistem de topic, un sistem de invitații și înlăturare de membri, precum și diverse rapoarte care să ofere o înțelegere profundă despre membrii și interacțiunile lor.

Cu această aplicație, se dorește îmbunătățirea comunicării și colaborării între membrii organizațiilor, oferind o soluție digitală modernă și ușor de utilizat pentru gestionarea și monitorizarea activităților.

ABSTRACT

The problem that needs to be addressed is the need for an application that efficiently manages communication between members of different organizations. The main purpose of this application is to use current technologies to create an online environment where members of an organization can communicate with each other through a chat, as well as with the entire organization through a special topic for news distribution.

As the application is oriented towards a workplace, it must provide the organization's owner with various administration tools, such as inviting or removing a member, as well as reports that indicate certain details about the members or their interactions. Regarding the target audience, it is represented by any organization that wants to communicate.

For the development of the application, I chose to use Node.js as the programming language and React as the frontend framework. The application will be built on a relational database and will include various functionalities, such as a chat system, a topic system, an invitation and removal system for members, as well as various reports that offer a deep understanding of members and their interactions.

With this application, the goal is to improve communication and collaboration among organization members by providing a modern and easy-to-use digital solution for managing and monitoring activities.

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 eficientizarea proceselor care depind de interacțiunea între mai mulți oameni. În zilele noastre, majoritatea aplicațiilor de comunicare sunt orientate spre entertainment (cum ar fi Facebook și WhatsApp) [1] și nu oferă instrumente de administrare a utilizatorilor cu scopul de a eficientiza procesele dintr-o afacere.

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 distribui știri relevante. De asemenea, aceste aplicații pot genera rapoarte care să monitorizeze activitatea utilizatorilor, cu scopul de a eficientiza sarcinile din mediul de lucru.

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 membri, astfel încât accesul să fie restricționat pentru o anumită categorie de utilizatori. De exemplu, 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, astfel utilizatorul fiind obligat să se conecteze pe cont propriu pe aplicație și nici să asigure 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, 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, astfel fiind la curent cu noile schimbări.

Prin realizarea acestor obiective, aplicația va crește eficiența proceselor de comunicare și colaborare din cadrul organizației, va facilita accesul la informații și va crește gradul de securitate și confidențialitate a acestora.

1.4 Soluția propusă

Pentru implementarea aplicației, am ales să folosesc o arhitectură monolitică [2]. Este o abordare clasică de dezvoltare a aplicațiilor web în care întregul sistem este construit într-o singură unitate. În această arhitectură, baza de date, serverul și partea de client (frontend) sunt toate parte a aceleiași aplicații care rulează pe aceeași mașină, fiind astfel mai simplu de implementat și dezvoltat. Această abordare oferă o performanță rapidă, deoarece comunicarea dintre componente este foarte bună. De asemenea, aceasta dispune de ușurință în depanare și mentenanță.

Frontend-ul va fi realizat prin framework-ul React [3], am ales acest framework deoarece are o comunitate puternică, o documentație excelentă și este eficient 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 este bazat pe ideea de componente, care pot fi reutilizate și combinate pentru a construi interfețe complexe, acest lucru face dezvoltarea mai eficientă și mai rapidă.

Backend-ul va fi realizat folosind tehnologia Node.js [4] deoarece este puternică și versatilă pentru dezvoltarea aplicațiilor web, care oferă o serie de avantaje pentru dezvoltatori, inclusiv scalabilitate, viteza de răspuns și accesul facil 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, deoarece aceasta oferă o performanță bună, o securitate solidă și o serie de instrumente de administrare a bazei de date. În plus, Oracle este una dintre cele mai utilizate baze de date relaționale și oferă 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 accesibila 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 se autentifice folosind codul de acces primit prin e-mail și 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 sau alți utilizatori au fost suspendați sau reactivați. Un utilizator 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 prompt și este ușor de folosit, având o interfață intuitivă și plăcută pentru utilizator. De asemenea, funcționalitățile sunt utile și relevante pentru nevoile utilizatorilor, permițând interacțiunea și colaborarea între aceștia. În ceea ce privește aspectul vizual, aplicația web este atractivă și are o structură clară, încărcându-se rapid.

În figurile [1 - 8], [19 – 27 (Anexe)] sunt prezentate capturi de ecran din aplicație:

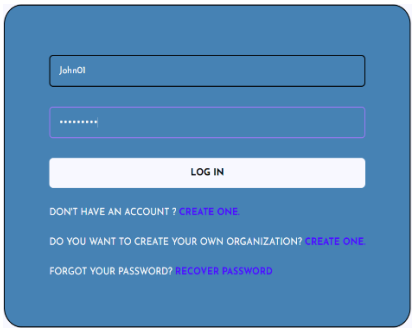


Figura 1: Login

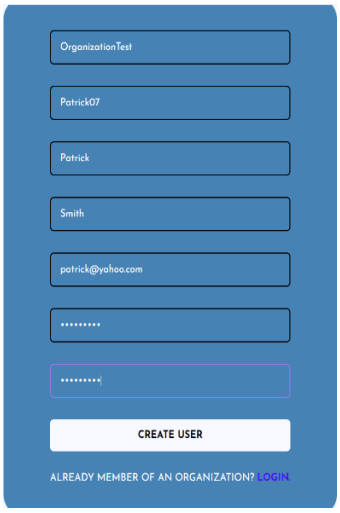


Figura 2: Sign up administrator

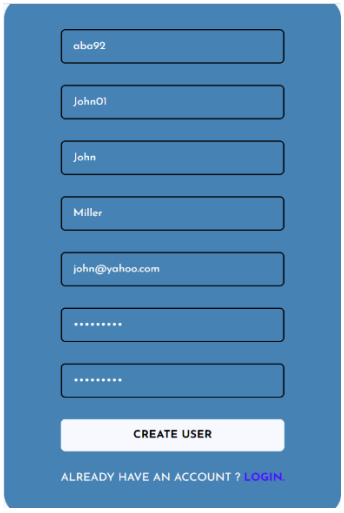


Figura 3: Sign up membru

Figura 4: Formular “Forgot your password”

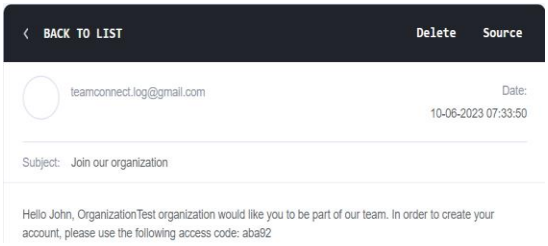


Figura 5: codul de autentficare primit prin mail

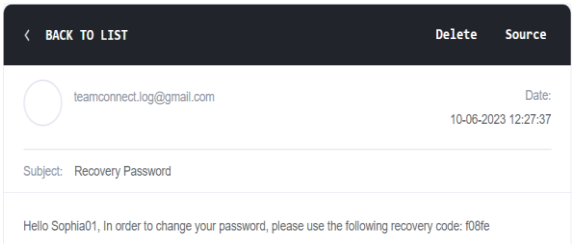


Figura 6: codul de recuperare al parolei

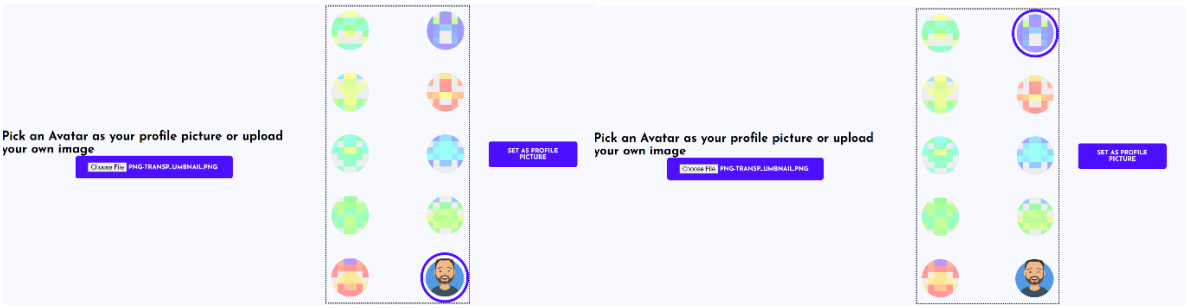


Figura 7a: Avatar incarcat de utilizator

Figura 7b: Avatar generat random

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

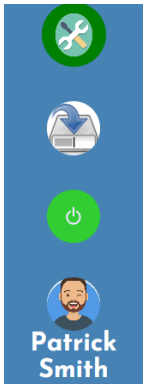


Figura 8a: workspace

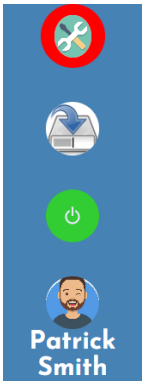


Figura 8b: workspace

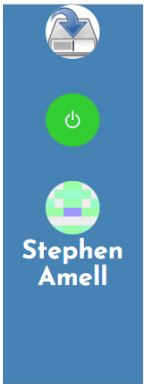


Figura 8c: workspace

1.6 Structura lucrării

Capitolul 2 prezintă contextul și relevanța problemei abordate, argumentând importanța cercetării și identificând lacunele existente în domeniu, Capitolul 3 examinează metodele și tehnologiile existente în domeniul de studiu. Sunt prezentate avantajele și dezavantajele acestora, precum și limitele lor în rezolvarea problemei identificate. În capitolul 4 este prezentată soluția propusă pentru a aborda problema identificată. Sunt explicate principiile și conceptele de bază ale soluției, argumentându-se eficiența și inovația acesteia în comparație cu metodele existente. Capitolul 5 descrie în detaliu modul în care soluția propusă a fost implementată sau poate fi implementată. Sunt prezentate aspecte tehnice și operaționale, precum și resursele necesare pentru implementarea soluției. În capitolul 6 se efectuează evaluarea și analiza soluției propuse. Sunt prezentate rezultatele experimentelor sau studiilor de caz realizate pentru a testa eficiența și performanța soluției, comparându-le

cu obiectivele și cerințele inițiale. Capitolul 7 rezumă concluziile obținute din implementarea și evaluarea soluției propuse. Se subliniază contribuția adusă de soluție, se discută despre posibile îmbunătățiri și se menționează perspectivele pentru dezvoltarea ulterioară a soluției.

2 MOTIVAȚIE

Comunicarea în grup a constat foarte mult timp în întâlnirea fizică a mai multor indivizi cu scopul rezolvării unor probleme. Cu progresul tehnologic, ideea de comunicare la distanță a devenit din ce în ce mai atractivă. În prezent există multe aplicații gratuite destinate comunicării cu scopuri de entertainment, dar se simte lipsa unei aplicații care să pună la dispoziția unui adminstrator de organizație o posibilitate de aducere a membrilor săi împreună, cu scopul eficientizării rezultatelor din mediul respectiv de muncă. În ultimii ani, aplicațiile de comunicare la distanță au devenit din ce în ce mai populare în mediul de afaceri, datorită nevoii de a comunica și colabora eficient în timp real, indiferent de locația geografică sau de fusul orar. Aceste aplicații permit proprietarilor de afaceri și angajaților să comunice rapid și ușor între ei, să gestioneze sarcinile și proiectele și să ia decizii mai bine fundamentate.

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. Integrarea unor modalități de generare de rapoarte despre membrii și interacțiunea dintre aceștia ușurează procesul de gestionare și eficientizare a task-urilor, astfel proprietarii de organizații ar putea lua decizii mai informate și mai bine fundamentate în ceea ce privește gestionarea organizației lor. Prin simplificarea proceselor de comunicare și gestionare a sarcinilor, se poate reduce timpul și efortul necesar pentru a obține rezultatele dorite, ceea ce poate duce la o creștere a productivității și a eficienței.

În general, prin crearea unei astfel de aplicații, se poate îmbunătăți comunicarea și colaborarea între membrii organizației, ceea ce poate duce la o mai mare coeziune și la o îmbunătățire a performanței organizației în ansamblu.

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. 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. Ele oferă funcții atractive, dar adesea nu abordează în mod specific nevoile de comunicare eficientă în cadrul organizațiilor. Pe de altă parte, există și aplicații care necesită o monetizare pentru a accesa toate funcționalitățile. Aceste aplicații gratuite oferă unele facilități de comunicare, dar pentru a beneficia de caracteristici avansate sau pentru a avea acces la opțiuni de administrare și gestionare a comunicării, utilizatorii trebuie să plătească un cost suplimentar. Aceasta poate crea bariere pentru organizațiile mai mici sau cu resurse financiare limitate, limitându-le posibilitățile de a comunica eficient la distanță. TeamConnect ofera instrumente care faciliteaza comunicarea in cadrul organizației precum posibilitatea de a izola membrii organizației de restul utilizatorilor ai aplicației, de a invita doar acei membri care trebuie să facă parte din organizație și de a genera rapoarte și statistici folositoare în gestionarea task-urilor ulterioare.

Rezultatele unui formular de feedback realizat pentru a evalua nevoia unei astfel de aplicații au confirmat importanța și interesul utilizatorilor. Majoritatea participanților au considerat TeamConnect o aplicație interesantă și au exprimat nevoia de a utiliza o astfel de soluție. Feedback-ul a evidențiat aprecierea pentru interfața prietenoasă și ușor de utilizat a aplicației, precum și dorința de a utiliza această platformă atât în echipele de lucru, cât și în comunicarea unu-la-unu.

Prin urmare, se poate concluziona că TeamConnect este o aplicație de comunicare orientată către organizații, care a primit feedback pozitiv din partea utilizatorilor. Aceasta vine în întâmpinarea nevoilor de comunicare eficientă, oferind o interfață prietenoasă și ușor de utilizat. 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, urmând ca detaliile de implementare să fie prezentate în capitolele următoare. Arhitectura soluției se bazează pe o aplicație web cu o interfață utilizator prietenoasă. Aceasta va fi construită folosind HTML, CSS și JavaScript în cadrul framework-ului REACT [3], iar partea server va fi dezvoltată cu ajutorul framework-ului Express în NodeJs [4]. Arhitectura soluției constă din două părți principale: baza de date relațională și aplicația web. Baza de date este Oracle [7] și va fi folosită pentru a stoca informațiile despre utilizatori, mesajele dintre aceștia și știrile postate pe feed-ul organizației, iar aplicația web va permite utilizatorilor să acceseze aceste informații, 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. Diagrama entitate-relație (ER) a bazei de date va avea următoarea structură:

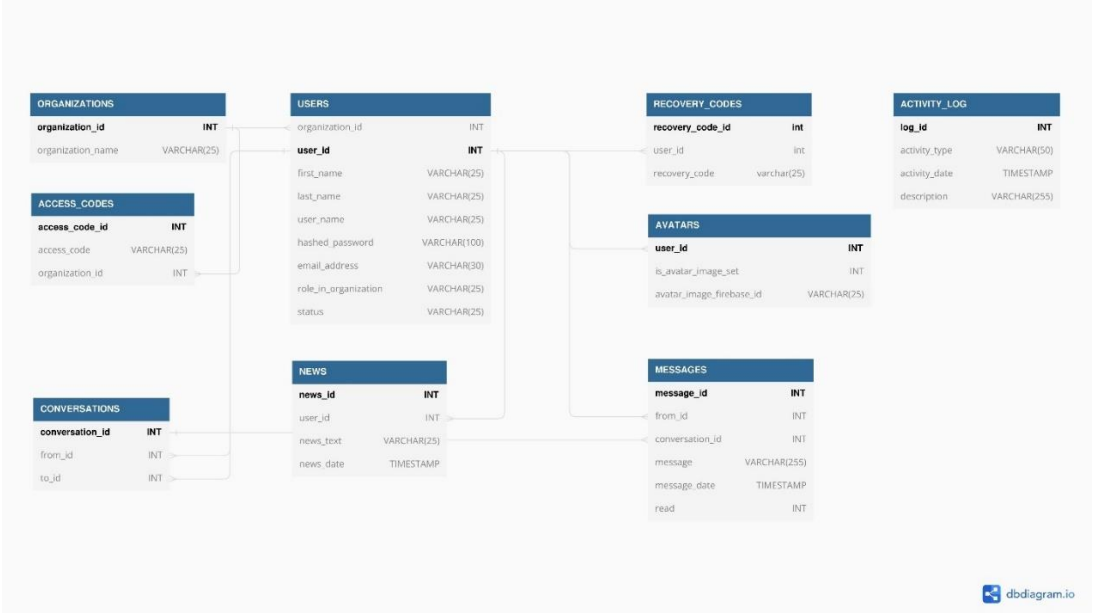


Figura 9: Diagrama entitate-relație (ER) a bazei de date

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 de organizații va stoca informații precum numele organizației și id-ul

acesteia, entitatea de utilizatori va stoca informații precum 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 și un flag care indică dacă imaginea a fost setată sau nu. Entitatea de conversații va fi folosita pentru a stoca informatii precum id-ul conversatiei, id-ul utilizatorului care a trimis mesajul si id-ul utilizatorului care a primit primul mesajul, folosite ulterior pentru a identifica conversația din care face parte un mesaj. Entitatea mesajelor va fi folosită pentru a stoca informații precum id-ul mesajului, id-ul conversatiei din care face parte mesajul, id-ul user-ului care a trimis mesajul, mesajul și data în care a fost trimis. Entitatea știrilor va fi folosită pentru a stoca informații despre fiecare știre postată pe feed-ul organizației, precum id-ul utilizatorului care a postat, știrea 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 recupere 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[1], o platformă online care permite utilizatorilor să creeze diagrame de baze de date într-un mod vizual și interactiv.

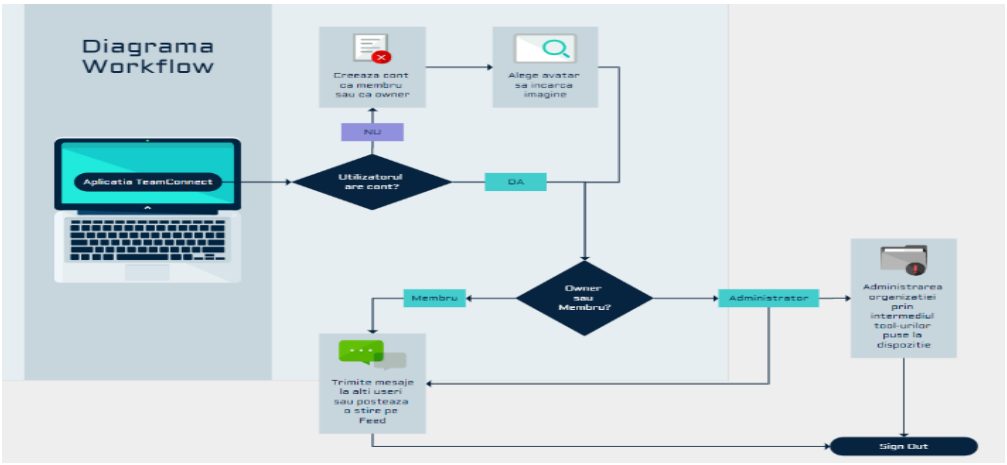


Figura 10: Diagrama workflow arată fluxul de utilizare al aplicației web

[1] dbdiagram.io. Draw Entity-Relationship Diagrams. https://dbdiagram.io/home

După autentificare, utilizatorii vor fi direcționați către pagina principală, unde vor putea trimite mesaje sau, în cazul administratorului, de a gestiona organizația prin invitarea de noi membri sau de a genera rapoarte folositoare. În cazul în care un utilizator este nou, poate să-și creeze cont dacă a primit un link de invitație din partea unui administrator sau să-și creeze propria organizație și să fie administratorul acesteia, urmând să-și invite ulterior membrii. 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, un utilizator poate să trimită mesaje la ceilalți utilizatori. Un utilizator 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.

4.1 Definirea design-ului

4.1.1 Scopul design-ului

Scopul design-ului aplicației de comunicare este de a crea o experiență intuitivă și eficientă pentru utilizatori din toate domeniile, astfel încât aceștia să poată accesa și utiliza funcțiile principale ale aplicației fără dificultate. În special, design-ul ar trebui să faciliteze transmiterea de mesaje, postarea de știri pe feed sau observarea notificărilor, care sunt sarcinile de bază ale aplicației. Astfel, design-ul ar trebui să fie ușor de utilizat și să permită utilizatorilor să interacționeze cu aplicația în mod natural și rapid.

De asemenea, este important ca design-ul să ofere o identitate vizuală atractivă și coerentă pentru aplicație, astfel încât utilizatorii să poată recunoaște și asocia cu ușurință aplicația cu marca și serviciile sale.

În concluzie, design-ul aplicației de comunicare ar trebui să fie conceput pentru a facilita utilizarea și accesul la funcțiile principale ale aplicației, să ofere o identitate vizuală atractivă și coerentă. Prin urmare, utilizatorii pot beneficia de o experiență plăcută și eficientă în utilizarea aplicației de comunicare.

4.1.2 Identitatea vizuală

Identitatea vizuală a aplicației web este construită în jurul culorilor principale și a modului în care acestea sunt utilizate pentru a transmite anumite stări și funcționalități. Pentru a exprima simplitatea, am ales să folosesc o nuanță de alb, #f8f8ff pentru fundalul paginilor de login și chat. Culoarea albă este asociată cu claritatea și ușurință, creând o interfață vizuală ușoară și prietenoasă pentru utilizator.

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 că este în centrul atenției și pentru a ajuta la ușorul recunoaștere a 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, pentru a sugera o încetare temporară a activității sau a permisiunilor de acces.

Butonul de sent pentru mesaje și butonul de refresh pentru notificări sunt în culoarea verde deschis, #32cd32, pentru a sugera succesul și progresul 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 și pe minimalizarea efectelor vizuale, pentru a oferi un aspect curat și modern bazându-se pe principiile design-ului flat. Cele patru secțiuni importante ale aplicației vor fi organizate 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 amplasat 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, cu o paletă de culori restrânsă.

Sectiunea 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 la aplicație.

Sectiunea de feed va fi proiectată în același stil flat minimalist, 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. Paleta de culori folosită va fi restrânsă, pentru a crea o estetică coezivă și echilibrată.

În concluzie, utilizarea design-ului flat pentru aplicația web va simplifica structura și organizarea, oferind utilizatorilor o experiență intuitivă și modernă.

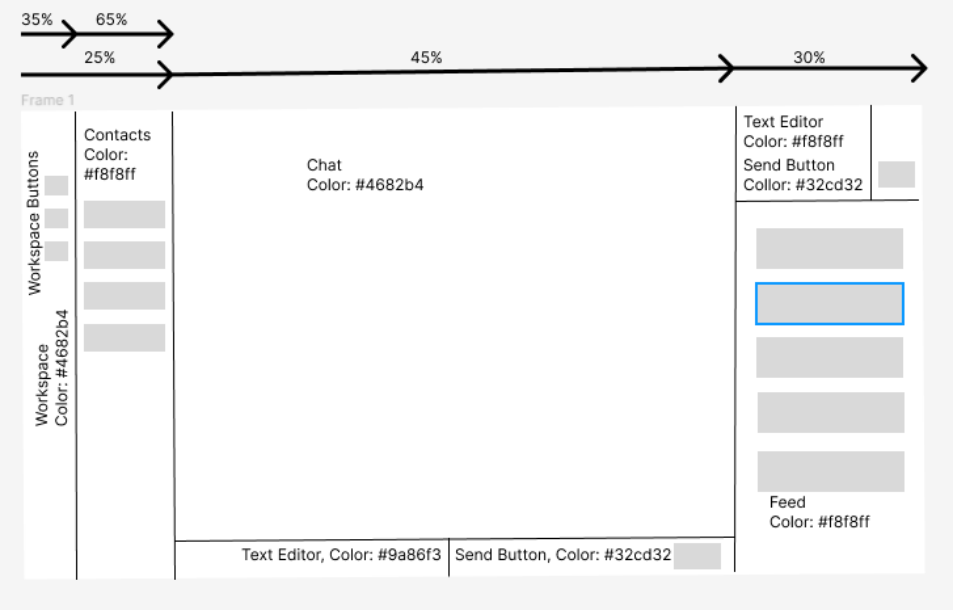


Figura 11: Structura și organizarea aplicației

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. Dezactivarea 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 provoca 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, care îi permite generarea anumitor rapoarte privind activitatea utilizatorului. Pentru a realiza acest lucru, am utilizat interogări SQL pentru a extrage informațiile relevante din baza de date, iar apoi am folosit biblioteca 'chart.js/auto' [9] pentru a le prezenta într-un mod grafic și ușor de înțeles.

Rapoartele puse la dispoziția administratorului sunt următoarele:

1. Raportul privind numărul de mesaje trimise pe mai multe intervale orare: Această funcționalitate permite administratorului să genereze rapid un raport care furnizează

numărul de mesaje trimise în acel interval specific de către utilizatorii aplicației din cadrul aceleiași organizații. Această caracteristică oferă administratorului o imagine clară asupra activității de comunicare din aplicație, permitându-i să ia decizii informate.

2. Raportul privind numărul de mesaje trimise de fiecare utilizator: Această funcționalitate permite administratorului să selecteze un utilizator specific și să genereze un raport care afișează numărul total de mesaje trimise de acel utilizator. Acest raport ajută la monitorizarea și evaluarea implicării fiecărui utilizator în comunicarea prin aplicație.
3. Raportul privind numărul de mesaje primite: Această caracteristică permite generarea unui raport care evidențiază numărul total de mesaje primite de la alți utilizatori. Prin analizarea acestui raport, administratorul poate evalua volumul total de comunicare și poate identifica eventualele tendințe sau modele care pot fi utilizate în îmbunătățirea experienței utilizatorilor.
4. Raportul privind utilizatorii suspendați și utilizatorii activi: Această funcționalitate oferă administratorului o imagine clară asupra stării utilizatorilor din aplicație. Raportul furnizează informații despre numărul total de utilizatori suspendați și numărul total de utilizatori activi. Acesta este util pentru monitorizarea și gestionarea stării conturilor utilizatorilor, oferind administratorului posibilitatea de a lua măsuri corespunzătoare pentru a menține un mediu sigur și funcțional în aplicație.
5. Generarea unui fișier CSV cu toți utilizatorii din aplicație: Această funcționalitate permite administratorului să genereze un fișier CSV care conține informații despre toți utilizatorii din aplicație. Fișierul CSV include detalii precum nume, prenume, nume de utilizator, adresă de email, rolul în organizație și statusul fie activ, fie suspendat. Acest raport cuprinzător facilitează administrarea și organizarea datelor despre utilizatori într-un format ușor de utilizat și de partajat.

În ansamblu, aceste rapoarte și funcționalități suplimentare oferă administratorului un set puternic de instrumente pentru a analiza și gestiona eficient comunicarea și utilizatorii din aplicație. Prin intermediul acestor rapoarte, administratorul poate lua decizii informate, îmbunătăți experiența utilizatorilor și asigura buna funcționare a aplicației.

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 este adăugat în maparea corespunzătoare împreună cu id-ul socket-ului. Atunci când evenimentul de trimitere a unui mesaj este declanșat, este căutat socket-ul corespunzător utilizatorului destinatar, iar dacă acesta este găsit, mesajul este transmis către socket-ul respectiv.

Evenimentul de recepționare a mesajelor este declanșat atunci când un mesaj este primit de la un alt utilizator prin intermediul socket-ului. Acest eveniment este emis de către socket-ul corespunzător destinatarului mesajului în urma recepționării mesajului de către server și este ascultat de către clientul care a trimis mesajul, astfel încât acesta poate fi notificat atunci când mesajul a fost livrat cu succes.

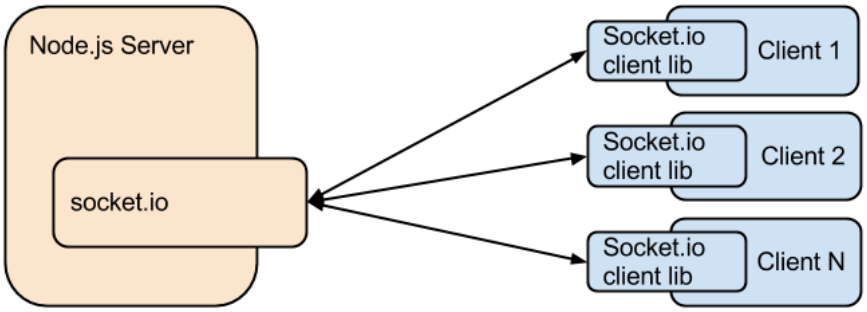


Figura 12: Conexiune de tip socket in NodeJs[2]

[2]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

Folosirea REST API [5] (Representational State Transfer Application Programming Interface) este o metodă comună de comunicare între servere, care implică transmiterea de date prin intermediul cererilor HTTP (HTTP requests).

Prin intermediul REST API, un client poate trimite o cerere către un server, care va răspunde cu date într-un format standardizat, precum JSON (JavaScript Object Notation) sau XML (eXtensible Markup Language). Aceste date pot fi, de exemplu, informații despre resursele de pe server sau operațiile disponibile pentru o anumită resursă.

Folosind REST API, serverele pot comunica între ele fără a fi necesar să împărtășească aceeași limbaj de programare sau să fie instalate biblioteci speciale. Acest lucru face posibilă integrarea cu ușurință a serviciilor externe și dezvoltarea aplicațiilor care folosesc date din diverse surse.

În cadrul aplicației, REST API este metoda principală de comunicare între frontend(client) și backend(server). 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 un formular de autentificare sau datele unei noi postări.

Pe de altă parte, cererea GET este utilizată pentru a prelua informații de la backend, cum ar fi contactele, mesajele din fiecare conversație sau setările din panoul de știri. Aceste date sunt returnate într-un format specificat în cadrul răspunsului primit de la server.

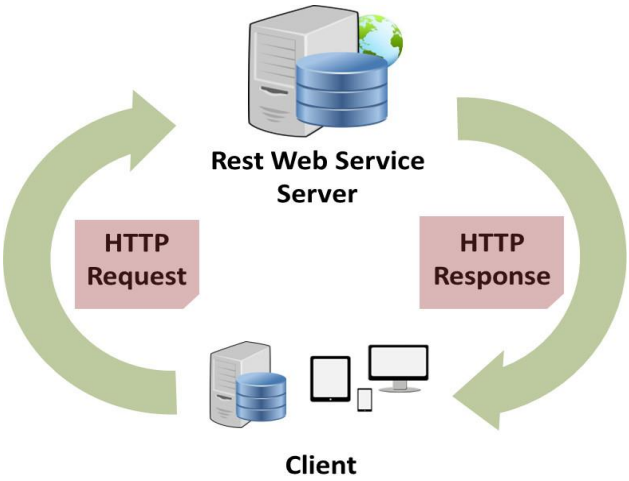


Figura 13: Comunicare folosind REST API[3]

4.7 Criptarea parolei în cadrul autentifcării

În cadrul autentificării se folosește un algoritm bcrypt pus la dispoziție de biblioteca “bcrypt” [10] din Node.js [4] . Avantajul pus la dispoziție de algoritmul bcrypt în cadrul autentificării este faptul că este o metodă puternică de hash parolă, ceea ce înseamnă că poate proteja parolele utilizatorilor împotriva atacurilor de tip brute-force. Algoritmul bcrypt [6] oferă o funcționalitate suplimentară pentru a face dificilă construirea de tabele de hash sau de arbori pentru atacurile de tip rainbow table. În plus, bcrypt este capabil să producă o versiune a hash-ului parolei, care poate fi ajustată în funcție de parametrii de cost, făcând

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

astfel atacurile brute-force mai dificile. Mai mult, biblioteca Node.js bcrypt vine cu o interfață simplă de utilizat pentru implementarea algoritmului în cadrul aplicațiilor Node.js.

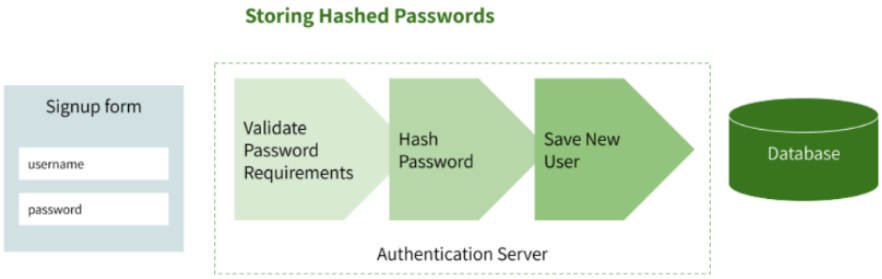


Figura 15: Stocarea parolei criptate in baza de date [11]

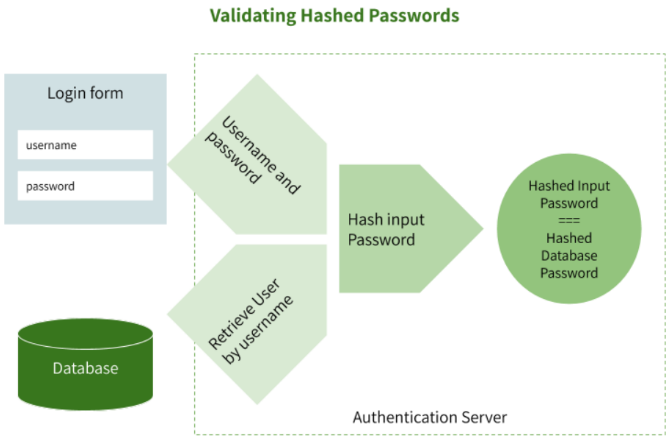


Figura 16: Validarea parolei [11]

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, folosind un modul de Node.js [4] numit "github-like-avatar-generator" [12], sau poate încărca propria imagine din calculator.

Acest cod JavaScript definește o funcție numită "generateAvatar", care generează o imagine de avatar aleatorie, utilizând grafica SVG. Funcția primește un obiect ca argument, cu trei proprietăți opționale: numărul de blocuri care vor fi folosite în avatar, lățimea avatarului și un array de culori care să fie utilizate în loc să se genereze altele noi.

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 (disponibil doar într-un mediu de browser).

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. Apoi, 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]

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]. Oracle Linux Server 7.9 este o distribuție a sistemului de operare Linux, dezvoltată și oferită de Oracle Corporation. Este bazată pe Red Hat Enterprise Linux (RHEL) fiind compatibilă cu aceasta și având aceleași caracteristici de stabilitate și securitate. Această distribuție mai oferă și integrare cu produse Oracle, Oracle Linux Server este optimizată pentru a funcționa în mod ideal cu produsele software Oracle, cum ar fi baza de date Oracle [7] sau aplicațiile Oracle Middleware. Acest lucru facilitează implementarea și administrarea soluțiilor Oracle.

Aplicația include următoarele componente trei componente principale:

Frontend-ul este implementat folosind framework-ul React [3], care oferă un mod eficient de construire a interfețelor utilizator (UI), rulează pe același container cu celelalte componente ale aplicației, foloseste tehnologie de tip socket pentru transmiterea in timp real al mesajelor si notificarilor si comunică cu backend-ul prin intermediul unui API REST [5], utilizând cereri HTTP pentru transferul datelor în format JSON.

Beneficiază de performanța rapidă și actualizări eficiente ale UI-ului datorită utilizării DOM-ului virtual și reutilizării componentelor în React [3]. Interacționează cu utilizatorii prin intermediul browserului, oferind o interfață intuitivă și prietenoasă.

Backend-ul este implementat folosind tehnologia Node.js [4], care oferă o platformă puternică și versatilă pentru dezvoltarea aplicațiilor web, rulează pe același container cu frontend-ul și baza de date, furnizează API-uri, susține logica pentru a manipula și procesa datele primite de la frontend, inițilizeaza socket-ul folosit pentru transmiterea în timp real a mesajelor și notificărilor.

Comunică cu frontend-ul prin intermediul API-urilor REST, asigurând transferul și manipularea datelor între frontend și backend. Gestionarea conexiunii cu baza de date Oracle [7] se realizează prin intermediul unui driver specific pentru Node.js, care permite executarea de interogări și tranzacții către și dinspre baza de date.

Baza de date, folosește Oracle ca sistem de gestiune a bazelor de date (SGBD), rulează pe același container Linux cu frontend-ul și backend-ul, asigurând astfel o arhitectură monolitică, conexiunea dintre backend și baza de date se realizează prin intermediul unui driver Oracle specific pentru Node.js, 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 complexă, deoarece acestea sunt fișiere de dimensiuni mari, care pot fi dificil de gestionat într-o bază de date relațională. În această situație, 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 noastre. Am creat o bază de date nouă în consola Firebase 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, am creat o colecție în baza de date, numită "UserAvatar", pentru a stoca imaginile de profil ale utilizatorilor. Această colecție este folosită în cadrul endpoint-ului "setavatar" al server-ului nostru, pentru a stoca imaginea de profil uploadată de utilizator.

Endpoint-ul "setavatar" este apoi utilizat pentru a actualiza baza de date a server-ului cu informațiile despre imaginea de profil. În acest endpoint, prima dată extragem 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;
```

Apoi, adăugăm imaginea de profil în colecția "UserAvatar" a bazei de date Firebase și extragem id-ul acesteia, pentru a putea lega 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];
```

În final, actualizăm baza de date a server-ului cu informațiile despre imaginea de profil și întoarcem un răspuns către client. În cazul în care imaginea a fost setată cu succes, vom întoarce un JSON cu flag-ul "isSet" setat la true și imaginea de profil. În caz contrar, vom întoarce un JSON cu un mesaj de eroare, se observă in Codul 1.4.

Cod 1.4: Actualizare detalii avatar in 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 rezolvăm problema stocării imaginilor de profil ale utilizatorilor într-o bază de date Firebase, având în vedere dificultățile asociate 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, dar după ce am aprofundat problema am realizat că socket.io [6] este o soluție mai bună.

Socket.IO este mai potrivit pentru trimiterea mesajelor în timp real decât REST API [5] pentru că REST API se bazează pe protocolul HTTP, care funcționează într-un model de cerere-răspuns. Adică, clientul trebuie să trimită o cerere către server, serverul procesează acea cerere și apoi trimite un răspuns înapoi la client. Acest proces poate dura o perioadă de timp, în funcție de mărimea și complexitatea cererii și de încărcarea serverului.

Pe de altă parte, socket.io utilizează protocolul WebSockets [15], care permite trimiterea de date în timp real între client și server. Acest lucru permite o comunicare bidirecțională instantanee între client și server, fără necesitatea de a trimite cereri și aștepta răspunsuri.

În plus, socket.io are funcționalități încorporate pentru gestionarea conexiunilor și comunicarea între diferitele părți ale aplicației, ceea ce face implementarea mai ușoară și mai eficientă decât prin intermediul unui REST API.

În concluzie, socket.io este o opțiune mai bună decât REST API pentru trimiterea mesajelor în timp real, deoarece este mai rapid, mai eficient și mai ușor de implementat.

Codul 2.1 reprezinta inițializarea unei instante Socket.io prin intermediul careia putem crea o conexiune bidirecțională între client și server, ceea ce permite schimbul de mesaje în timp real între utilizatori.

Cod 2.1: Inițializare socket

```
const io = socket(server, {
  cors: {
    origin: "http://localhost:3000",
    credentials: true,
  },
});

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

  socket.on("send-msg", (data) => {
    const sendUserSocket = onlineUsers.get(data.to);
    if (sendUserSocket) {
      socket.to(sendUserSocket).emit("msg-recieve", data);
    }
  });
});
```

Spre deosebire de REST API, care funcționează prin solicitarea de resurse și returnarea de răspunsuri, Socket.io utilizează un model push, în care serverul poate trimite date clientului fără ca acesta să solicite aceste date în mod explicit. Acest lucru face ca comunicarea în timp real să fie mult mai eficientă decât în cazul folosirii REST API.

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

Această abordare este mult mai rapidă și mai eficientă decât folosirea REST API, unde clientul ar trebui să facă solicitări repetate către server pentru a obține actualizări, ceea ce ar consuma multe resurse și ar fi mult mai lent în timp real. În plus, Socket.io are și alte funcții utile, cum ar fi gestionarea automată a conexiunilor și posibilitatea de a gestiona în mod eficient mai mulți utilizatori simultan.

Codul 2.2 este scris în React și folosește hook-ul useEffect. Când componenta este randată, acest hook este apelat și verifică dacă există un utilizator curent. Dacă acesta există, atunci se realizează o conexiune cu serverul de socket folosind host-ul specificat. De asemenea, utilizatorul curent este adăugat la lista de utilizatori online cu ajutorul evenimentului "add-user".

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

```
useEffect(() => {
  if (currentUser) {
    socket.current = io(host);
    socket.current.emit("add-user", currentUser.USER_ID);
  }
}, [currentUser]);
```

Aceaste instrucțiuni fac parte din implementarea clientului pentru trimiterea mesajelor folosind socket.io. 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 catre un alt utilizator desemnat in aplicație prin selectarea de catre

utilizatorul curent a chat-ului dintre cei doi dar si pentru primirea unei mesaj trimit de catre alt user.

Cod 2.3: Transmitere mesaje prin intermediul socket-ului

```
socket.current.emit("send-msg", {
  to: currentChat.USER_ID,
  from: data.USER_ID,
  msg
});
```

Cod 2.4: Recepționarea mesajelor prin intermediul socket-ului

```
if (socket.current) {
  socket.current.on("msg-recieve", (data) => {
    setArrivalMessage({ fromSelf: false, MESSAGE: data.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("msg-recieve", (data) => {
      // daca se primeste o alerta atunci se seteaza variabila
      // corespunzatoare
      if(data.flag === "alert") {
          setAlert(true);
          setMsgAlert(data.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 === data.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("send-msg", {
    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("send-msg", {
  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("send-msg", {
      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 pentru Node.js [4] pentru a stabili o conexiune la baza de date.

Codul rulează într-o funcție async care poate aștepta ca o promisiune să fie îndeplinită. Metoda getConnection() returnează o promisiune care rezolvă la un obiect Connection după o conexiune de succes.

Metoda getConnection() 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.
- 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 poate fi 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"
});
```

Codul 5.2 face parte din fișierul de configurare tnsnames.ora pentru baza de date Oracle Database, conține informații de configurare pentru conexiunea, inclusiv numele serviciului, adresa IP, portul, protocoalele și tipul de conectare.

Cele trei servicii definite sunt:

- XE: Acest serviciu utilizează protocolul TCP și rulează pe gazda (DESKTOP-H8HNR2D in acest caz) la portul 1521. Conexiunea la acest serviciu utilizează un server dedicat.
- LISTENER_XE: Acest serviciu este responsabil pentru ascultarea cererilor de conectare la serviciul XE și este configurat să ruleze pe aceeași gazdă și port ca și serviciul XE.
- ORACLR_CONNECTION_DATA: Acest serviciu este utilizat pentru a conecta la procesul CLRExtProc și este configurat să utilizeze protocolul IPC cu cheia EXTPROC1521.

Aceste servicii pot fi utilizate pentru a configura conexiunile la bazele de date Oracle [7], fie prin utilizarea unui client de bază de date, fie prin intermediul unui driver de bază de date pentru o limbă de programare specifică. În codul 5.1 anterior, conexiunea este realizată la serviciul XE utilizând driver-ul Oracle Database pentru Node.js (oracledb module).

Codul 5.2 Configurare fisier tnsnames.ora pentru a permite conectarea la baza de date

```
XE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = DESKTOP-H8HNR2D)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = 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-un array 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:

- Este afișat un element HTML div cu clasa contacts.
- Map-ul contacts.map iterează prin fiecare contact din array-ul contacts.
- 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.
- Variabila contactClassNames este construită pe baza valorilor de mai sus și se adaugă la proprietatea className a elementului HTML div de afișat.
- La apăsarea pe elementul div de contact, se apelează funcția changeCurrentChat cu indexul contactului și obiectul contactului ca argumente.
- În fiecare element div de contact, sunt afișate următoarele elemente:
    - div cu clasa avatar sau image-preview, în funcție de tipul imaginii. Dacă imaginea este în format base64, este afișată direct, altfel se adaugă un prefix data URI pentru a fi o sursă validă.
    - div cu clasa username și afișează numele de familie și prenumele utilizatorului contactului.
- La final, lista completă de elemente div pentru fiecare contact este afișată în interiorul elementului div cu clasa contacts.

Codul 6.2 definește un endpoint pentru a obține o listă cu toți utilizatorii din baza de date care nu sunt utilizatorul curent, împreună cu informațiile lor de autentificare 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 userId = req.params.id; extrage valoarea ID-ului utilizatorului curent din obiectul req.params.
- const result = await connection.execute(...) definește o interogare SQL care selectează numele de utilizator, prenumele, numele de familie, ID-ul organizației, adresa de e-mail, ID-ul utilizatorului, starea de autentificare și ID-ul imaginii de avatar al utilizatorului din tabelele USERS și AVATARS pentru toți utilizatorii, cu excepția celui curent. Interogarea este executată folosind obiectul connection primit prin argument la începutul programului.
- const snapshot = await UserAvatar.get(); obține o referință către colecția de imagini de avatar stocate în baza de date Firebase Firestore, 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 și le transformă într-un array de obiecte cu un câmp id și toate datele documentului.
- for(elem in result.rows) { ... } parcurge fiecare rând din rezultatul interogării SQL și atribuie valoarea imaginii de avatar corespunzătoare utilizatorului curent din lista de imagini de avatar.
- let firebaseID = result.rows[elem].AVATAR_IMAGE_FIREBASE_ID; extrage ID-ul imaginii de avatar al utilizatorului curent.
- for (l in list) { ... } parcurge fiecare obiect din lista de imagini de avatar și verifică dacă ID-ul său corespunde ID-ului imaginii de avatar al utilizatorului curent.
- if (list[l].id === firebaseID) { ... } dacă ID-urile corespund, adaugă URL-ul imaginii de avatar la rândul curent din rezultatul interogării SQL.
- return(res.json(result.rows)); returnează rezultatul interogării SQL în format JSON.

Codul 6.1: 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 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.

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. Pentru trimiterea de email-uri din Node.js am folosit biblioteca nodemailer [16], care folosește un cont de Gmail, Codul 6.5.

Codul 6.5 afișează inițializarea unei instanțe de transporter nodemailer.

```
let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
      user: 'teamconnect.log@gmail.com',
      pass: 'ygvdprsinjwxeaup'
  }
});
```

6 Evaluare

Înainte de implementarea finală a aplicației web de comunicare, am efectuat o serie de teste pentru a evalua performanța și funcționalitatea acesteia. Am utilizat Lighthouse[4], 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 create tabelul de mai jos:

Tabela 1: Performanțele aplicației

Metrica	First Contentful Paint	Largest Contentful Paint	Total Blocking Time	Cumulative Layout Shift	Speed Index
TeamConnect	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) - Acesta măsoară timpul necesar pentru a afișa primul element de conținut vizibil în cadrul aplicației. Cu cât FCP este mai mic, cu atât aplicația se încarcă mai rapid și utilizatorii pot începe să interacționeze mai repede cu conținutul. Avantajul TeamConnect constă în faptul că are un timp de FCP de doar 0.4s, ceea ce indică o încărcare inițială rapidă și o experiență receptivă pentru utilizator
- Largest Contentful Paint (LCP) - Această metrică măsoară timpul necesar pentru a afișa cel mai mare element de conținut vizibil în cadrul aplicației. Cu cât LCP este mai mic, cu atât utilizatorii vor vedea mai repede conținutul principal al aplicației. În cazul TeamConnect, timpul LCP de 21.8s este mai mare în comparație cu Instagram (3.7s) și Facebook (3.7s), ceea ce poate indica întârzieri semnificative în încărcarea conținutului de dimensiuni mari în aplicație

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

- Total Blocking Time (TBT) - Această metrică măsoară timpul total în care aplicația este blocată și utilizatorii nu pot interacționa cu aceasta din cauza sarcinilor de procesare intensivă. Cu cât TBT este mai mic, cu atât aplicația este mai receptivă la interacțiunea utilizatorului. Avantajul TeamConnect constă într-un TBT de doar 50ms, ceea ce indică faptul că sarcinile de procesare nu perturbă semnificativ experiența utilizatorului în timpul interacțiunii
- Cumulative Layout Shift (CLS) - Această metrică măsoară gradul de stabilitate al elementelor din layoutul aplicației în timpul încărcării. Un CLS scăzut indică faptul că elementele rămân relativ stabile, fără să se miște sau să cauzeze schimbări bruște în interfață. Dezavantajul TeamConnect constă într-un CLS de 0.451, care este mai mare decât Instagram (0) și Facebook (0.027), ceea ce poate indica o stabilitate mai scăzută a layoutului și o experiență mai puțin coerentă pentru utilizatori.
- Speed Index - Această metrică măsoară viteza percepută de utilizatori în timpul încărcării aplicației. Cu cât Speed Index este mai mic, cu atât aplicația este percepută ca fiind mai rapidă. Avantajul TeamConnect constă într-un Speed Index de 1.1s, ceea ce indică faptul că utilizatorii percep aplicația ca fiind mai rapidă în comparație cu Instagram și Facebook, care au un Speed Index de 3.0s

În urma metricilor de mai sus putem spune că un avantaj al aplicației TeamConnect 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 a facilita procesul de testare, am creat un script care utilizeaza Docker pentru a instala și rula aplicația pe sistemele de operare Windows, Linux și macOS. Acest lucru ne-a asigurat că putem verifica consistent comportamentul aplicației pe diferite platforme. De asemenea, am utilizat serviciul Amazon Web Services[5] 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 in considerare in procesul de deployment sunt următorii:

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

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 am selectat sistemul de operare Ubuntu. De asemenea, am generat o cheie SSH pentru a permite conexiunea securizată la instanță.

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: Pentru a utiliza baza de date Oracle în aplicația noastră, 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 Oracle 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.

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

membrilor în aplicația TeamConnect? In ce măsură ați recomanda aplicația TeamConnect 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 TeamConnect 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, totuși, o cerere evidentă pentru îmbunătățiri și funcționalități suplimentare. De asemenea, procesele de creare a contului, generare a rapoartelor și colaborare cu administratorii organizației pot fi îmbunătățite în funcție de feedback-ul utilizatorilor.

În ceea ce privește procesul de creare a contului și alăturare la o organizație, se poate realiza o simplificare a fluxului de înregistrare. Aceasta ar putea implica reducerea numărului de pași necesari și cerințele de completare a informațiilor. De asemenea, se poate lua în considerare implementarea unor opțiuni de autentificare alternative, cum ar fi autentificarea prin intermediul conturilor de social media sau prin intermediul adresei de e-mail.

7 CONCLUZII

Lucrarea de licență evidențiază realizările și rezultatele obținute în cadrul proiectului de dezvoltare a unei aplicații web pentru Desktop, care are ca scop eficientizarea proceselor și colaborarea într-o organizație.

Una dintre principalele realizări ale acestei lucrări constă în tehnologiile moderne utilizate în dezvoltarea aplicațiilor web. Prin intermediul acestui proiect, s-a explorat și utilizat un set divers de tehnologii și instrumente relevante pentru domeniu, astfel încât să se asigure o dezvoltare corespunzătoare și 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. Prin intermediul aplicației, administratorii pot gestiona procesul de invitație a membrilor în organizație, utilizând un formular de invitație, iar membrii invitați își pot crea conturi utilizând un cod de acces primit prin e-mail. Aceasta oferă un sistem simplu și sigur pentru a adăuga noi membri și a facilita comunicarea în organizație.

De asemenea, un alt aspect important al lucrării este deployment-ul aplicației într-un mediu cloud, mai exact folosind serviciul AWS (Amazon Web Services). Aceasta permite accesibilitatea aplicației de către utilizatori și o scalabilitate mai mare în funcție de nevoile organizației. Utilizarea unui mediu cloud oferă, de asemenea, flexibilitate și ușurință în gestionarea și actualizarea aplicației.

Un alt obiectiv al lucrării a fost de a permite utilizarea aplicației de către publicul larg. Prin intermediul interfeței user-friendly și a funcționalităților diversificate, aplicația poate fi utilizată de orice persoană interesată, oferindu-le o modalitate eficientă de comunicare și colaborare în cadrul organizației.

Un aspect important în evaluarea acestei lucrări este compararea performanțelor obținute cu alte aplicații existente. Prin realizarea unei analize comparative a performanțelor, s-a putut evalua eficiența și calitatea aplicației dezvoltate în comparație cu alte soluții similare de pe piață. Aceasta oferă o perspectivă clară asupra contribuției și avantajelor aduse de aplicația dezvoltată în cadrul organizației respective.

În concluzie, lucrarea de licență a prezentat dezvoltarea unei aplicații web pentru Desktop, având ca scop eficientizarea proceselor și colaborarea într-o organizație. Realizările principale ale lucrării au constat în tehnologiile moderne, dezvoltarea unei aplicații cu funcționalități diverse, deployment-ul aplicației într-un mediu cloud, posibilitatea utilizării aplicației de către public și compararea performanțelor obținute cu alte soluții existente. Aceste rezultate demonstrează importanța și beneficiile aduse de aplicația dezvoltată în cadrul organizației.

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

9 ANEXE

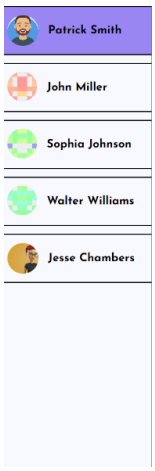
Codul 6.1: Reprezentarea listei de contacte in React

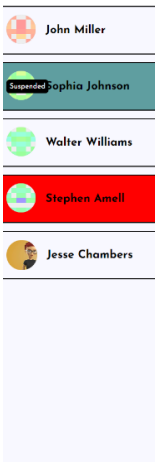
```
<div className="contacts">
  {contacts.map((contact, index) => {
    if (contact.ORGANIZATION_ID !== currentIdOrganization) {
      return null;
    }

    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>
```





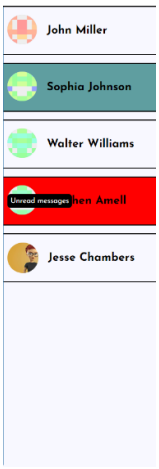


Figura 19a: utilizator selectat Figura 19b: utilizator suspendat Figura 19c: mesaje necitite

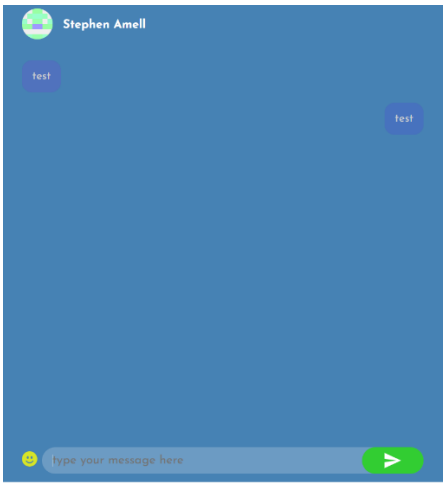


Figura 20: Chat

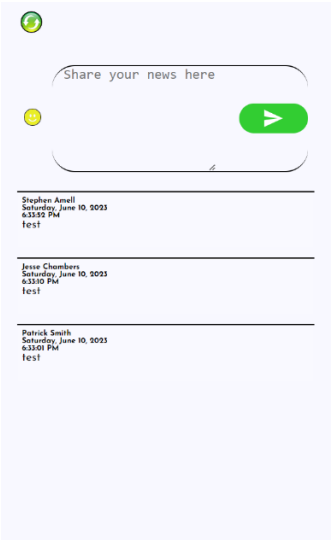


Figura 21: feed-ul organizației

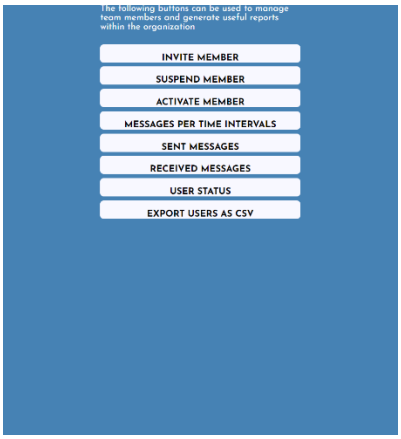


Figura 22: TeamManagement

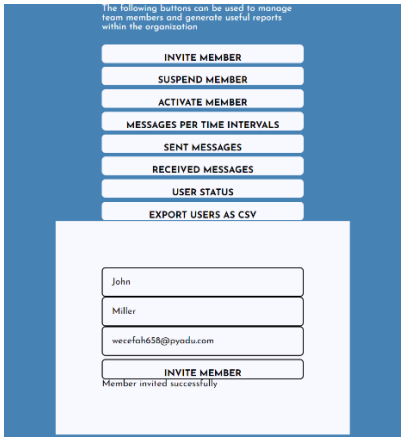


Figura 23: InviteMember

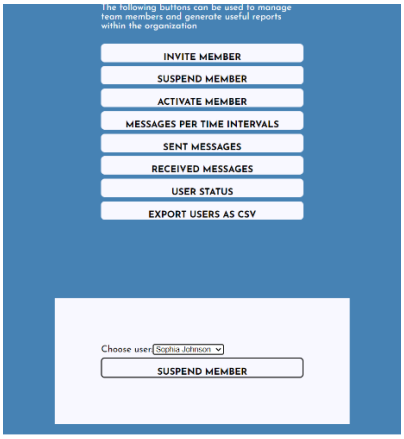
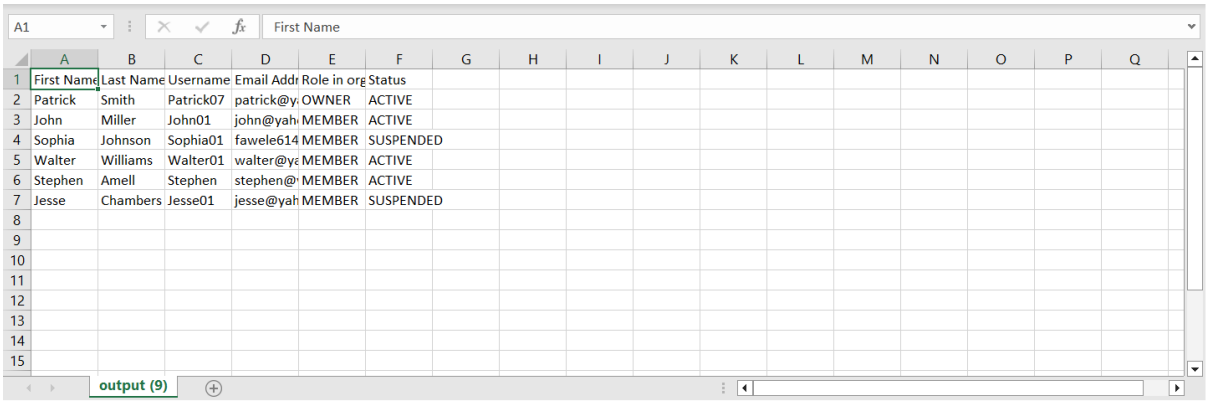


Figura 24: SuspendMember



	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 25: Raport CSV generat din aplicație

Figura 26: Raport numarul de mesaje trimise în intervale de timp

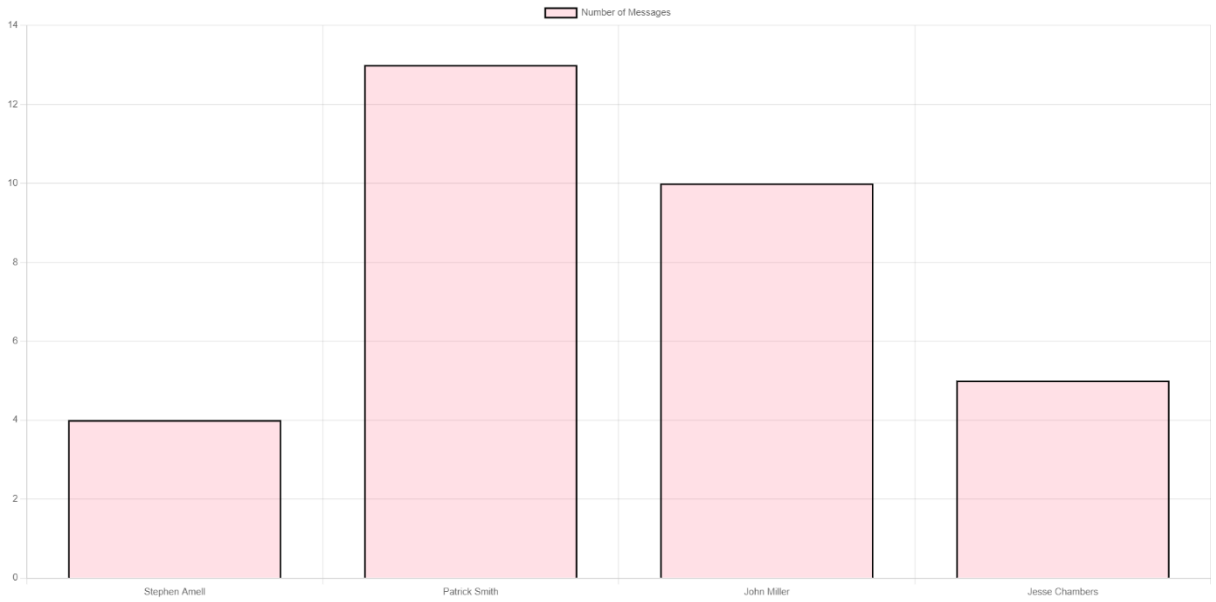


Figura 27: Raport numarul de mesaje trimise de fiecare utilizator din organizație