MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Chat pentru dezvoltatori software

LUCRARE DE LICENȚĂ

Absolvent: **Radu Bompa**

Coordonator științific:
Prof. Asist. Ing. **Cosmina IVAN**

2019

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN, **Prof. dr. ing. Liviu MICLEA**

DIRECTOR DEPARTAMENT, **Prof. dr. ing. Rodica POTOLEA**

Absolvent: **Radu BOMPA**

Chat pentru dezvoltatori software

1. **Enunțul temei:** *Proiectul propus contribuie la îmbunătățirea comunicării între dezvoltatori în cadrul proiectului, cu accent pe code review și instrumente de versionare (Git).*

2. **Conținutul lucrării:** *Pagina de prezentare, Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză și fundamentare teoretică, proiectare de detaliu și implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bibliografie.*

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanți**:

5. **Data emiterii temei:** 1 noiembrie 2018
6. **Data predării:** 8 iulie 2019

Absolvent: ______________________________

Coordonator științific: ______________________________

Cuprins

Capitolul 1. Introducere – Contextul proiectului 1
1.1. Motivația și contextul lucrării 1
1.2. Conținutul lucrării 1
Capitolul 2. Obiectivele Proiectului 3
2.1. Obiective principale 3
2.2. Obiective secundare 3
Capitolul 3. Studiu Bibliografic 5
3.1. Comunicarea prin mesageria instant într-o echipă de dezvoltatori software 5
Etapele dezvoltării unui proiect și beneficiile oferite de chat-ul nostru 5
3.2. Aplicații de mesagerie existente pe piață 7
3.2.1. Aplicații comerciale 7
3.2.2. Aplicații open-source 10
3.3. Integrare cu medii de control al versiunii 10
3.4. Chatbot 13
Capitolul 4. Analiză și Fundamentare Teoretică 15
4.1. Cerințe funcționale și use-case-uri 15
4.2. Estimarea timpului necesar implementării ....**Error! Bookmark not defined.**
4.3. Cerințe non-funcționale 19
4.4. Cazuri de utilizare 21
4.4.1. Creare cameră de discuții 21
4.4.2. Alternative Flows – creare cameră de discuții 22
4.4.3. Creare și conectare chat cu un repository GitHub 23
4.4.4. Alternative flows – creare și conectare chat cu un repository GitHub 24
4.5. Tehnologii utilizate 25
4.5.1. Protocoale de comunicare 26
4.5.2. Back End 27
4.5.3. Front End 30
4.6. Controlul versiunii 32
4.7. Continuous integration 32
Capitolul 5. Proiectare de Detaliu si Implementare 34
5.1. Structura bazei de date 34
5.1.1. Users 35

5.1.2. Rooms 37
5.1.3. Chats 37
5.1.4. Messages 38
5.1.5. Webhooks 39
5.2. Structura back-end-ului 39
5.3. Structura front-end-ului 41
5.4. Detalierea funcționalităților 43
5.4.1. Autentificarea 43
5.4.2. CRUD pentru camere, chat-uri și mesaje 45
5.4.3. Mesagerie 47
5.4.4. Conectarea cu GitHub 48
Capitolul 6. Testare și Validare 51
Capitolul 7. Manual de Instalare si Utilizare 55
Capitolul 8. Concluzii 64
Bibliografie .... Error! Bookmark not defined.
Anexa 1 – Lista figurilor 66

Capitolul 1. Introducere – Contextul proiectului

1.1. Motivația și contextul lucrării

Zona dezvoltării software este tot mai largă și are parte de o creștere exponențială, cuprinzând un număr tot mai mare de dezvoltatori software, operatori de testare, coordonatori de echipe și alții. În acest mediu, comunicarea eficientă reprezintă o problemă foarte importantă, vitală în materie de dezvoltare, resursele necesare pentru implementarea unui proiect pot fi reduse sau mărite considerabil în funcție de calitatea și eficiența comunicării între părțile implicate.

Există soluții și aplicații care să adreseze această problemă, însă funcționalitatea lor nu este suficientă de multe ori unor nevoi specifice dezvoltatorilor. Unul dintre principalele subiecte de discuție într-o echipă de dezvoltatori este codul și implementarea lui, vizualizarea, adnotarea, marcarea codului ni se par funcționalități absolut necesare într-o aplicație de chat pentru dezvoltatori software. Comunitatea dezvoltatorilor software este numeroasă și consider că un chat care să răspundă cerințelor și așteptărilor câtor mai mulți actori din mediul software, dar care să permită dezvoltarea deschisă, în funcție de nevoile specifice ale unei echipe, este o întreprindere care merită începută și dezvoltată apoi în cadrul comunității de specialitate.

Pornind de la câteva aplicații existente, vom încerca să construim scenarii pe care funcționalitățile existente nu se pliază, apoi să dezvolt o aplicație chat care să răspundă acestor scenarii și să permită orice dezvoltare ulterioară, printr-o implementare modulară, ușor de extins.

Proiectul **ChatDev** se concentrează, în special, pe medii de control al versiunii codului (Git[1]) și pe etapa de code-review a codului, pentru a crea un chat care poate fi legat de un repository, și care poate fi folosit cu maximă eficiență pentru code-review. Consider că e de maximă necesitate o comunicare cât mai facilă în această etapă de dezvoltare, pentru a putea descoperi, înregistra și marca porțiuni din cod care au probleme, care pot fi îmbunătățite, și pentru a menține un istoric al evoluției proiectului dezvoltat, mai mult decât în comentarii și descrierile unui commit. Posibilitatea de a sublinia anumite linii de cod mi s-a părut, de asemenea, o funcționalitate foarte utilă.

Aplicația ChatDev va fi construită open-source, cu ideea dezvoltării ulterioare în minte, pentru a răspunde nevoilor specifice unei echipe și pentru a ține pasul cu dezvoltările tehnologice. Dorim să putem integra aplicația noastră cu diverse instrumente de workflow management, de desenare, de code pen și să oferim posibilitatea de a adăuga noi module cât mai rapid și ușor posibil.

1.2. Conținutul lucrării

Primul capitol prezintă contextul lucrării de licență, precum și scopul acesteia.

A doilea capitol evidențiază obiectivele principale și secundare, acest lucru constând în prezentarea a ce își propune chatul să îmbunătățească fată alte aplicații existente, precum și o scurtă introducere a ***conceptelor de bază***.

[1] https://git.kernel.org/pub/scm/git/git.git/

Capitolul trei este dedicat în totalitate ***studiului bibliografic***, fără de care nu am putea avea o imagine clară pentru înțelegerea deplină a conceptelor care urmează să fie implementate. Aici sunt prezentate ***aplicațiile și platformele existente pe piață***, cu beneficiile și dezavantajele lor, ***principiile de control al versiunii Git***, operațiunile care pot fi efectuate într-un ***repository***, și o descriere în detaliu a ***etapei de code-review*** într-un proiect.

În cel de-**al patrulea capitol**, de analiză și fundamentare teoretică, sunt specificate ***cerințele funcționale și non-funcționale,*** cu câteva exemple semnificative. Totodată, este descrisă ***arhitectura sistemului***, precum și ***tehnologiile și limbajele*** abordate în realizarea aplicației de chat pentru dezvoltatori software.

Capitolul cinci este cel de proiectare și implementare, în care e detaliată ***structura aplicației***, cum este gândită, modalitatea de ***comunicare între componentele sistemului***, precum și cum au fost folosite anumite tehnologii și principii în realizarea sa.

În **al șaselea capitol** sunt descrise ***modalitățile de testare*** ale chatului, care sunt necesare pentru a-i asigura buna funcționare. Testarea s-a realizat prin metoda White Box Testing și prin testarea aplicației de alți utilizatori.

Capitolul șapte este dedicat ***manualului de instalare și de utilizare*** al aplicației. Aici va fi prezentat un manual de instalare pentru tehnologiile folosite, precum și cum trebuie folosită aplicația și flow-ul acesteia.

Ultimul capitol este cel care ne prezintă ***concluziile*** obținute în urma testării precum și modalități de dezvoltare ulterioară, dar și îmbunătățirile care s-ar putea aduce pe viitor.

Capitolul 2. Obiectivele Proiectului

2.1. Obiective principale

Aplicația ChatDev are rolul de a facilita comunicarea în mediul dezvoltării software, între coordonatori echipe, dezvoltatori, testeri și alți membri implicați implicați în proiectul respectiv. Funcționalitatea se concentrează pe partea de ***coding***, ***code-review*** și ***control al versiunii*** codului. Există pe piață câteva soluții de chat pentru dezvoltatori software, iar funcționalitățile cu un grad de necesitate ridicat vor fi implementate în cadrul chatului nostru. Prin intermediul acestui chat, utilizatorii vor putea **să trimită** sub formă de mesaj **porțiuni de cod formatate și subliniate**, și vor putea **lega o conversație la un repository sau un commit** (o anumită versiune din repository), pentru a putea discuta detaliile, problemele și îmbunătățirile implementării.

Un alt obiectiv pe care îl urmărim este acela de a crea o aplicație care să poată fi **ușor extinsă, implementată modularizat**, așa încât dezvoltatorii să poată adăuga funcționalități noi în funcție de necesitățile proiectului sau ale echipei. Aplicația va putea fi **personalizată** la orice nivel, iar funcționalitățile create în comunitate, corect și eficient implementate, și care sunt foarte apreciate, vor putea fi adăugate în versiunea de bază, principală a proiectului.

2.2. Obiective secundare

Există câteva soluții pe piață pentru chat între dezvoltatori software, însă soluțiile cu funcționalitate extinsă nu sunt disponibile open-source, ci doar contra-cost. Ne propunem să creăm ***o aplicație gratuită, disponibilă*** oricui, și care să ofere funcționalitățile aplicațiilor contra-cost, cu posibilitatea extinderii funcționalităților, prin adăugarea de noi module. Chatul se adresează dezvoltatorilor software, dar și studenților din domeniul IT, care pot comunica între ei, în echipe de lucru, sau pot primi indicații de la îndrumători despre implementările proiectelor și temelor.

Unul din obiectivele asupra căruia ne-am concentrat atenția este și cel ***educativ***. Chat-ul nostru va putea fi dezvoltat cu noi funcționalități care pot fi implementate de studenți sau elevi, care pot lucra în echipe sau individual. Pot colabora la implementare cu un profesor, cu care să facă apoi review în aplicație pentru codul implementat. Un scenariu similar poate fi conceptualizat pentru dezvoltatori juniori într-o companie de software.

Chatul va permite ***mesagerie instant clasică***, între doi utilizatori, crearea de ***camere*** cu mai mulți utilizatori, crearea de ***canale*** care vor face broadcast unidirecțional către utilizatori. Pentru fiecare chat, cameră sau canal pot fi create ancore, ***hashtag-uri*** care pot fi trimise ca legături către discuție. În orice discuție vor putea fi ***adăugați sau șterși utilizatori***, în funcție de necesitate.

Chatul permite ***trimiterea de secvențe de cod***, care vor fi ***formatate*** pentru citirea facilă, iar anumite linii din cod vor putea fi ***subliniate***. Vor putea fi trimise mesaje ***multimedia***, ***fișiere***, mesaje ***audio*** sau ***video***. În funcționalitățile ulterioare, vom integra și un modul pentru conferințe audio și video.

Fiecare chat, cameră sau canal poate fi ***atașată*** în momentul creării, ***unui repository*** de platformă de control al versiunii, inițial Git, cu posibilitate de extindere ulterioară. Codul din repository poate fi ***copiat, subliniat și adăugat în chat***. Chatul va fi vizualizat în paralel cu codul din repository, unde pot fi ***vizualizate pull requests, commits*** și pot fi discutate probleme.

Chatul va permite crearea de ***roluri pentru utilizator***, roluri globale în aplicație sau specifice unui chat, camere sau canal. ***Moderatorul*** va putea șterge, modifica sau ascunde mesaje nepotrivite. ***Trusted users*** vor primi acest rol în funcție de calificare pe o anumită tehnologie, și vor putea fi contactați pentru consiliere în anumite discuții. ***Editorul*** va putea face modificări în codul care este distribuit. Atribuirea rolurilor pentru utilizatori se va face pe baza unui ***sistem de reputație***, care folosește două criterii: cât de mult este prezent și interacționează utilizatorul în aplicație, precum și recenziile oferite de ceilalți utilizatori despre calitatea activității din cadrul aplicației. În cadrul unei camere vor putea fi adăugați utilizatori doar cu rol de spectatori, fără posibilitatea de a răspunde.

Aplicația va permite ***marcarea*** unor mesaje ca fiind imporrante, și a le menține în partea de sus a conversației (***pin-to-top***).

Toate conversațiile vor fi stocate, pentru a dispune de posibilitatea de ***editare și ștergere*** a unor mesaje anterioare, și ***căutare*** după conținut. Fișierele, conținutul media și linkurile http vor putea fi ***vizualizate în liste separate***, pentru a facilita accesul rapid la resursele trimise în conversație.

Pentru a adăuga noi module aplicației, vom crea componenta de ***integrare module***, care va permite încărcarea unui nou modul în aplicație. Aplicația va fi open-source, disponibilă pe un repository Git, și se vor putea face modificări și fără a instala un nou modul. Pentru implementarea unui modul, acesta va trebui să folosească o anumită structură de împachetare. Sistemul este cel al plugin-urilor, similar celor de la Wordpress, Joomla sau alte aplicații web.

Capitolul 3. Studiu Bibliografic

Acest capitol va conține o scurtă prezentare a comunicării prin mesageria instant, și necesitățile specifice de comunicare într-o echipă de dezvoltatori software, cu accent pe echipele care lucrează off-site, în locații multiple. Vom descrie câteva etape din ciclul de lucru al unei echipe de dezvoltare software și necesitățile de comunicare specifice fiecărei etape.

Pe lângă aceasta, vom indica o clasificare personală a aplicațiilor care au funcționalități similare, împreună cu o analiză comparativă a funcțiilor oferite de fiecare. Comparația are rolul de a evidenția diferența pe care proiectul nostru încearcă să o aducă, funcționalitățile noi.

Vom încerca să construim imaginea unei aplicații chat ideale, cu toate funcționalitățile de care ar avea nevoie un dezvoltator pentru a lucra eficient. Problemele ridicate, în special de echipele aflate în locații multiple, vor fi adresate pe rând, iar aplicația noastră va oferi funcționalități sau posibilitatea integrării cu alte platforme care să ofere acea funcționalitate. Funcționalitățile care nu vor implementate vor fi incluse la dezvoltări viitoare.

3.1. Comunicarea prin mesageria instant într-o echipă de dezvoltatori software

Sistemele de mesagerie instantanee sunt sisteme care asigură schimbul instantaneu de mesaje de tip text cu una sau chiar mai multe persoane sau calculatoare deodată, interconectate de obicei prin intermediul Internetului. Contrar curierului electronic e-mail, acest mijloc de comunicare afișează mesajele aproape instantaneu, permițând astfel un dialog interactiv (în scris).

Spre deosebire de aplicațiile de mesagerie clasice, o aplicație chat pentru dezvotatori încearcă să răspundă anumitor nevoi specifice dezvoltatorilor și echipelor de software, precum nevoia de a vizualiza cod, a face adnotări, marcaje, conferințe, legături cu repository-uri, oferind funcționalități care să faciliteze etapele de lucru din cadrul unui proiect.

Etapele dezvoltării unui proiect și beneficiile oferite de chat-ul nostru

a. Planificarea proiectului

- Activități administrative și de pornire
- Definirea obiectivelor proiectului
- Obținerea acordului de la managementul superior
- Definirea și revizuirea planului de management al configurațiilor
- Realizarea unei echipe și stabilirea responsabilităților fiecăruia

Comunicarea în etapa de planificare a proiectului presupune multe discuții care s-ar desfășura cu dificultate doar în scris, mai ales în echipele care au membrii în locații multiple. În această etapă, discuțiile nu se desfășoară doar între dezvoltatori și persoane din mediul tehnic, ci și între dezvoltatori și administrație, clienți și administrație, nu în ultimul rând, în echipe mai mici, între dezvoltatori și clienți.

Funcționalitățile oferite de aplicația ChatDev pentru această etapă:

- Camere pentru până la 50 de persoane
- Roluri pentru participanții la chat
- Editare și ștergere mesaje
- Trimitere fișiere, trimitere contacte
- Sondaje de opinie
- Mesaje importante și pin-to-top
- Bot de conversație, cu informații de bază despre o anumită cameră sau un anumit canal

b. Execuția proiectului

- Execuția proiectului după planul propus
- Monitorizarea conformității cu procesele definite
- Analiza defectelor și efectuarea de activități de prevenire a acestora
- Monitorizarea performanțelor la nivel de program
- Efectuarea de review-uri la anumite etape critice și replanificarea unor etape, dacă este necesar
- Monitorizarea progresului proiectului

În cadrul etapei de execuție a proiectului, comunicarea are loc preponderent între membrii echipe de dezvoltare, și sunt necesare instrumente tehnice pentru a facilita lucrul echipei. Aplicația noastră oferă în mod special funcționalități pentru etapa de code-review, în cadrul aplicației va putea fi vizualizat codul din cadrul unui commit, un chat va putea fi atașat unui commit, și se vor putea discuta detaliile și eventualele probleme apărute:

- Code share
- Formatare și subliniere cod
- Markdown în stilul Git
- Code pen – editarea codului și a textului trimis
- Conectarea unui chat la un repository Git
- Căutare avansată, după diferite tipuri de filtre
- Vizualizarea fișierelor și link-urilor trimise într-un chat, într-un hashtag
- Permisiuni utilizatori pe un chat, o cameră, un canal sau un hashtag
- Adăugarea de ancore și hashtaguri către anumite zone din repository
- Sistem de reputație, bazat pe interacțiunea cu aplicația și votul altor participanți

c. Închiderea proiectului

- Analiza datelor post-proiect
- Etapa are loc după ce clientul și-a dat acceptul pentru produsul final
- Se urmărește stabilirea unor concluzii ca urmare a experienței acumulate, pentru a îmbunătăți procesele folosite în viitor
    - Trebuie menținut un istoric al modificărilor
    - Control al versiunii software
    - Rezultă într-un raport de închidere a proiectului

Pentru etapa de finalizare a proiectului aplicația ChatDev oferă câteva funcționalități care considerăm că sunt utile:

- Stocare nelimitată a mesajelor și fișierelor

- Posibilitatea de a atașa o conversație unei altei conversații, prin adăugarea unui hashtag, și vizualizarea grafului conversațiilor conexe
- Istoric și export al conversațiilor pentru arhivare

3.2. Aplicații de mesagerie existente pe piață

Vom realiza o comparație personală a tehnologiilor și soluțiilor existente pe piață în domeniul chat-urilor pentru dezvoltatori software. Vom analiza comparativ prețurile, ce funcționalități oferă, calitatea documentației și istoricul comunității care folosește respectiva aplicație. Am decis să clasificăm aplicațiile în două categorii mari, aplicații comerciale și aplicații open-source, din care face parte și chat-ul nostru pentru dezvoltatori.

3.2.1. Aplicații comerciale

Am ales cele mai utilizate aplicații la ora actuală, și vom compara câteva criterii: costul de utilizare, limita de stocare mesaje, posibilitatea de a realiza apeluri audio și video, integrări cu aplicații third-party, limită stocare fișiere, screen sharing, opțiuni interfață și thread-uri de comunicare.

Cea mai utilizată aplicație de chat colaborativ, conform Spiceworks, este Skype, care are de două mai mulți utilizatori decât Microsoft Teams, care a avut totuși o creștere majoră în ultimii ani, depășind ca număr de utilizatori Slack și Google Hangouts.

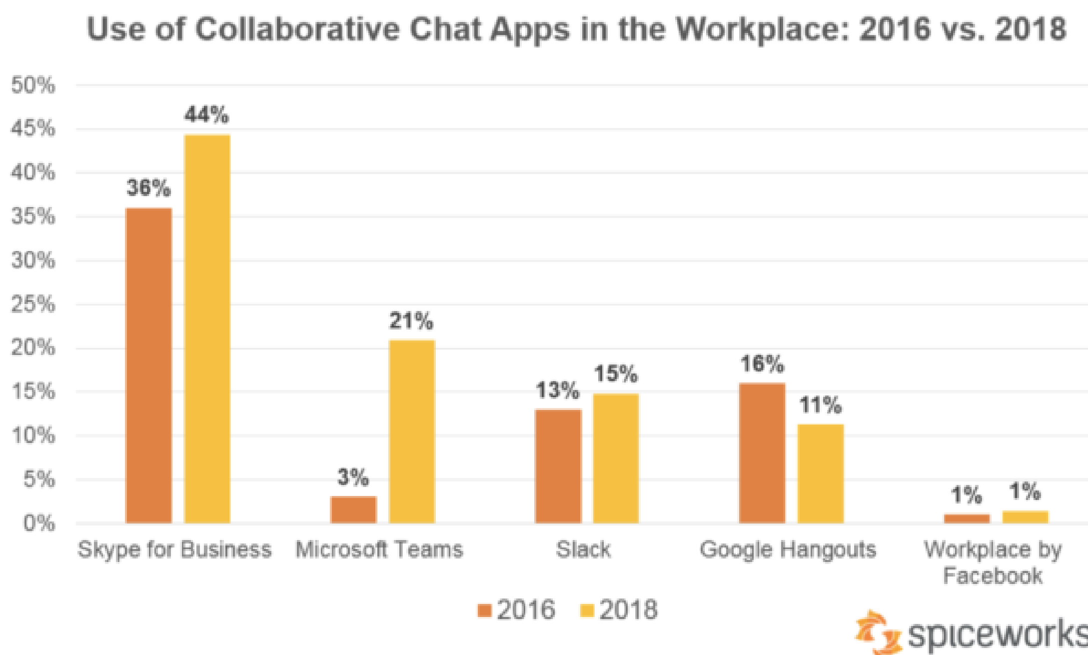


Figura 3.1 Skype este cea mai utilizată aplicație pentru chat colaborativ la momentul actual, urmată de Microsoft Teams, care a crescut mult în ultima perioadă[2]

[2] https://community.spiceworks.com/blog/3157-business-chat-apps-in-2018-top-players-and-adoption-plans

Tabel 3.1 Comparația celor mai utilizate aplicații comerciale de chat colaborativ

Aplicație	**Slack**[3]	**Skype for Business**[4]	**Google Hangouts**[5]	**Microsoft Teams**[6]	**Workplace by Facebook**[7]
Tarif	Freemium 2 pachete cu plată	Gratis	Cu plată	Freemium Două pachete cu plată, trial gratuit pentru ambele, de 30 de zile	Gratis în pachetul standard, cu plată pentru pachetul Premium.
Limită istoric mesaje	10K mesaje în planul Free, nelimitat în planurile Standard și Plus	Nelimitat	Nespecificat	Nelimitat în toate planurile	Nespecificat
Apeluri audio/video	Apeluri video și vocale nelimitate 1:1 în planul Free, conferințe cu până la 15 participanți în planurile plătite	Până la 7 persoane, 1080p, recording	Disponibil via Hangouts Meet, număr maxim de utilizatori – 25 în planurile Basic și Business și 50 în planul Enterprise.	Apeluri audio/video cu până la 80 de oameni într-o întâlnire.	Audio și video (pe mobil și desktop) cu până la 50 oameni
Integrări	Limită de 10 integrări în planul Free, 800+ integrări în planurile plătite	50+ integrări	50+ integrări	180+ aplicații și servicii (în Iulie 2018)	60+ (în Februarie 2019)
Limită stocare fișiere	Planul Free: 5GB stocare pentru o echipă Standard: 10GB per utilizator Plus: 20GB per utilizator	Nelimitat	Basic G Suite oferă 30GB spațiu de stocare, Business and Enterprise oferă stocare nelimitată	Free plan: 2 GB/user and 10GB of shared storage Paid plans: 1 TB per organization	Nelimitat
Screen sharing	Indisponibil în versiunea Free, disponibil în versiunile Standard și Plus	Da	Via Hangouts Meet în toate planurile tarifare	Disponibil în toate planurile	Disponibil în planurile gratuite și premium
Opțiuni culoare interfață	Teme personalizate în sidebar.	Nespecificat	Nespecificat	3 teme (light, dark, high contrast)	Nespecificat
Threaduri conversație	Da	Nu	Da	Da	Nu sunt threaduri în Chat, dar în Groups, Events etc.

[3] https://www.slack.com
[4] https://www.skype.com/en/business/
[5] https://hangouts.google.com/
[6] https://products.office.com/en-us/microsoft-teams/group-chat-software
[7] https://www.facebook.com/workplace

Fiecare aplicație are unele avantaje și dezavantaje, iar percepția utilizatorilor variază mult în funcție de necesitățile echipei. Aplicațiile pot fi testate gratuit. **Slack** este apreciată pentru creativitate și pentru ușurința utilizării. **Skype for Business** este cea mai utilizată și considerată printre cele mai sigure**, Google Hangouts** este cea eficientă ca costuri și se integrează cu multe servicii Google. **Microsoft Teams** se află într-o creștere semnificativă a numărului de utilizatori în ultimii ani, beneficiind de integrarea cu produsele Microsoft, iar **Workplace by Facebook** a apărut mai recent pe piață, și deși beneficiază de promovarea Facebook, nu este încă atât de utilizată.

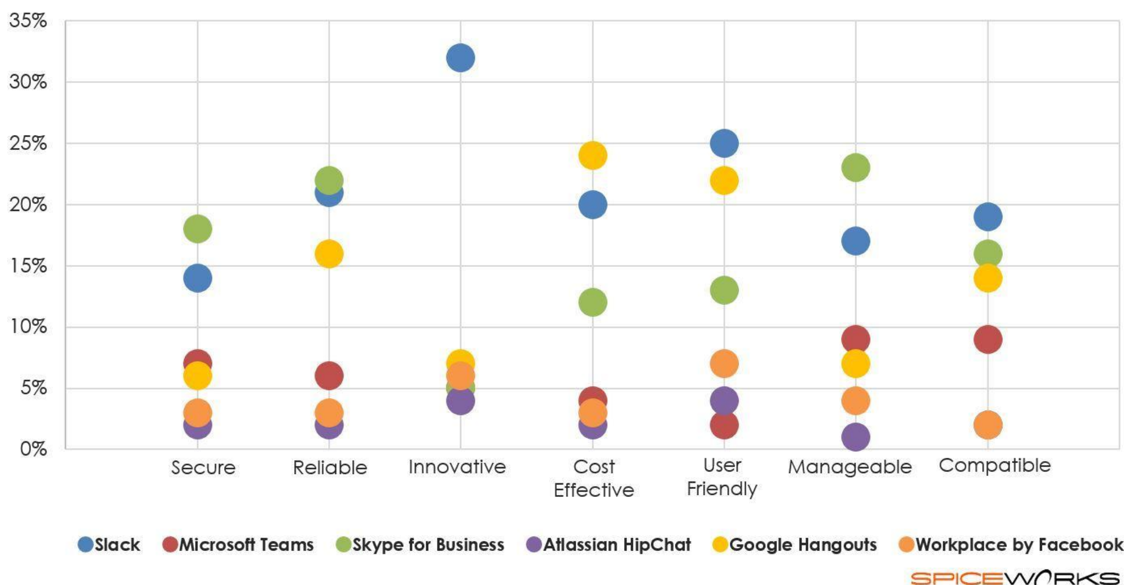


Figura 3.2 Aprecierea utilizatorilor despre aplicațiile chat de colaborare în echipă

Am analizat și alte aplicații chat comerciale, care fie nu au un număr atât larg de utilizatori (Flowdock), fie nu au atât de multe funcționalități de lucru în colaborare, pentru echipe. **Chanty**[8] are multe funcționalități similare Slack, fiind considerată o alternativă ceva mai ieftină. **Flowdock**[9] este o aplicație de chat colaborativă, însă nu este atât de populară și are mai puțini utilizatori. Alte aplicații de chat foarte utilizate, cum ar fi **Messenger**[10]**, Telegram**[11]**, Whatsapp**[12]**, Viber**[13] nu beneficiază de funcționalitățile specifice unui chat colaborativ, fiind mai apropiate ca funcționalitate aplicațiilor chat clasice, deși au multe funcționalități de care nu putea fi vorba acum 10-15 ani (cum ar fi apelul în grup).

[8] https://www.chanty.com
[9]https://www.flowdock.com
[10] https://messenger.com
[11] https://web.telegram.org
[12] https://web.whatsapp.com
[13] http://www.viber.com

3.2.2. Aplicații open-source

3.2.2.1. Gitter[14]

Gitter este o aplicație care seamănă într-o anumită măsură cu Slack, dar este open-source, core aplicației fiind disponibil într-un repository GitHub.

Funcționalități:

- Notificări care sunt trimise pe mobil în pachete, pentru a evita notificările deranjante
- Fișiere media inline
- Vizualizare și abonare la ("starring") camere de chat multiple în același tab de browser
- Link la fișiere individuale în repository-ul git adresat
- Link la GitHub issues (scriind # și numărul issue-ului) în repository-ul atașat, cu detalii la hover.
- Markdown în stilul GitHub în mesajele din chat
- Statusul utilizatorilor activi
- Hovercards pentru utilizatori, în funcție de profilurile și statisticile lor pe Github (număr de urmăritori pe Github, etc.)
- Mesaje grupate după lună, cu posibilitate de căutare
- Conectare de la clienți IRC

3.2.2.2. Alte aplicații open-source

Alte aplicații open-source pe care le-am analizatsunt mai puțin utilizate, sau nu au încă o comunitate de dezvoltare atât de largă. **Matrix**[15] are în centru conceptul de *bridges* și interoperabilitate, prin care oferă posibilitatea de a te conecta cu alte aplicații chat (cum ar fi Slack, IRC, Gitter) și de a comunica cu utilizatorii acestor aplicații, iar **Rocket.chat**[16] care oferă foarte multe posibilități de a personaliza experiența utilizatorului, vizual, prin integrări cu alte aplicații, prin modificarea anumitor componente.

Funcționalitățile aduse de aplicația ChatDev vor fi detaliate în capitolul 5.

3.3. Integrare cu medii de control al versiunii

Aplicația ChatDev va oferi posibilitatea de a conecta un chat de discuție la un repository git de pe GitHub. Utilitatea acestei funcționalități poate fi evidențiată, în special, în etapa de implementare și code-review a unui proiect. Posibilitatea de a discuta și a ancora conversația în activitatea din repository (commit-uri și push-uri, branch, merge), oferă un plus de eficiență în comunicare în cadrul echipei și posibilitatea de a reveni ușor la cele discutate, a le transforma apoi în task-uri din workflow management. Considerăm că ședințele de lucru (SCRUM de exemplu), pot fi eficientizate printr-o aplicație chat care să permită legarea la activitatea din repository-ul proiectului, discuția

[14] https://gitter.im/
[15] http://matrix.org
[16] https://rocket.chat

putând fi folosită ca un desfăşurător pentru activitatea ulterioară, la care se poate reveni cu uşurinţă.

Principii de funcționare ale mediilor de control al versiunii

O componentă a gestionării configurației software, controlul versiunii, cunoscută și sub denumirea de control al reviziei, este gestionarea modificărilor aduse documentelor, programelor software, site-uri web de mari dimensiuni și altor colecții de informații. Modificările sunt de obicei identificate printr-un cod numeric sau din litere, numit "numărul versiunii", "nivelul de revizuire" sau pur și simplu "versiunea". De exemplu, un set inițial de fișiere este "versiunea 1". Când se face prima modificare, setul rezultat este "versiunea 2" și așa mai departe. Fiecare revizuire este asociată cu o marcă de timp și persoana care efectuează schimbarea. Versiunile pot fi comparate, restaurate, iar cu unele tipuri de fișiere, sunt combinate.

Necesitatea unui mod logic de a organiza și de a controla revizuirile a existat pentru aproape atâta timp cât scrisul a existat, dar controlul versiunii a devenit mult mai important și mai complicat odată cu începerea erei computerizate. Numerotarea edițiilor de cărți și a revizuirilor cu specificații sunt exemple care datează din epoca exclusivă a tipăririi. Astăzi, cele mai capabile (dar și complexe) sisteme de control al versiunii sunt cele folosite în dezvoltarea de software, unde o echipă de oameni poate face modificări simultan în aceleași fișiere.

Sistemele de control al versiunilor (VCS) rulează cel mai frecvent ca aplicații independente, dar controlul versiunii este, de asemenea, încorporat în diferite tipuri de software, cum ar fi procesoare de text și foi de calcul, documente web colaborative și în diverse sisteme de gestionare a conținutului. Controlul revizuirii permite abilitatea de a readuce un document la o versiune anterioară, ceea ce este esențial pentru a permite editorilor să urmărească editările reciproce și să corecteze greșelile.

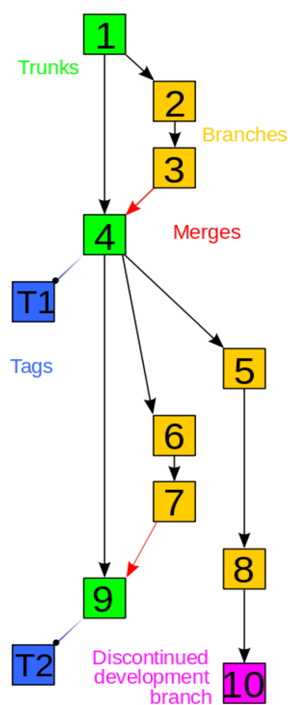


Figura 3.3

Exemplu de istoric de versiune al unui proiect

În ingineria software-ului de calculator, controlul versiunii este orice fel de practică care urmărește și asigură controlul asupra schimbărilor codului sursă. Dezvoltatorii de software folosesc uneori software de control al versiunii pentru a păstra fișierele de documentație și de configurare, precum și codul sursă [1].

Pe măsură ce echipele proiectează, dezvoltă și implementează software-ul, este des întâlnit ca mai multe versiuni ale aceluiași software să fie instalate pe diferite servere și dezvoltatorii software-ului să lucreze simultan pe diferite actualizări. Bug-uri sau caracteristici ale software-ului sunt adesea prezente doar în anumite versiuni (datorită rezolvării unor probleme și a introducerii altora pe măsură ce programul se dezvoltă). Prin urmare, în scopul identificării și remedierii erorilor, este extrem de important să puteți prelua și rula diferite versiuni ale software-ului pentru a determina în ce versiune apare problema.

De asemenea, ar putea fi necesar să se dezvolte simultan două versiuni ale software-ului: de exemplu, în cazul în care o versiune are bug-uri rezolvate, dar nu există noi caracteristici (branch), în timp ce în cealaltă versiune se lucrează la noi funcționalități (trunk).

La cel mai simplu nivel, dezvoltatorii ar putea pur și simplu să păstreze mai multe copii ale diferitelor versiuni ale programului și să le eticheteze corespunzător. Această abordare simplă a fost utilizată în multe proiecte de software mari. Deși această metodă poate funcționa, este ineficientă deoarece multe copii aproape identice ale programului trebuie menținute. Acest lucru necesită multă auto-disciplină din partea dezvoltatorilor și duce adesea la greșeli.

Mai mult, în dezvoltarea de software, în practica juridică și de afaceri și în alte medii, a devenit tot mai frecvent ca un singur document sau un fragment de cod să fie editat de o echipă, ale cărei membri pot fi dispersate din punct de vedere geografic și pot urmări interese diferite sau chiar contradictorii. Controlul sofisticat de revizie care urmărește și contabilizează proprietatea asupra modificărilor aduse documentelor și codului poate fi extrem de util sau chiar indispensabil în astfel de situații.

Luând în considerare toți acești factori, considerăm că o aplicație chat care să fie legată de mediul de control al versiunii ar aduce plus-valoare și eficientizare a lucrului în echipă, mai ales pe proiecte de mari dimensiuni.

Există numeroase servicii de control al versiunii, bazate pe Git (GitHub[17], BitBucket[18], GitLab[19]) sau alte tehnologii (Mercurial, SVN), și aplicația noastră va permite creare de module de integrare cu oricare din acestea, însă pentru versiunea inițială, aplicația va fi conectată cu GitHub.

Funcționarea Git

Git este un sistem distribuit de control al versiunii pentru urmărirea schimbărilor în codul sursă în timpul dezvoltării software-ului. Acesta este conceput pentru coordonarea muncii între programatori, dar poate fi folosit pentru urmărirea modificărilor în orice set de fișiere. Obiectivele sale includ viteza, integritatea datelor și susținerea fluxurilor de lucru distribuite, neliniare.

Ca și în cazul celor mai multe sisteme distribuite de distribuție și spre deosebire de majoritatea sistemelor client-server, fiecare director Git de pe fiecare calculator este un depozit complet cu istorie completă și abilități complete de urmărire a versiunilor, independent de accesul la rețea sau de un server central. Git este un software gratuit și open-source.

Funcționarea GitHub

GitHub este un serviciu de găzduire Git repository, dar care adaugă multe dintre propriile caracteristici. În timp ce Git este un instrument de linie de comandă, GitHub oferă o interfață grafică bazată pe Web. Acesta oferă, de asemenea, controlul accesului și mai multe funcții de colaborare, cum ar fi un wiki și instrumentele de bază de gestionare a sarcinilor pentru fiecare proiect.

[17] https://github.com
[18] https://bitbucket.org
[19] https://gitlab.com/

Principala funcționalitate GitHub este "forking" - copierea unui depozit din contul unui utilizator în altul. Acest lucru vă permite să luați un proiect pe care nu aveți acces la scriere și să îl modificați în contul dvs. propriu. Dacă faceți modificări pe care doriți să le distribuiți, puteți trimite proprietarului original o notificare numită "pull request". Acest utilizator poate apoi, cu un clic pe un buton, să îmbine (merge) modificările găsite în repo-ul vostru cu repo-ul original.

Aceste trei caracteristici - fork, pull request și merge - fac ceea ce face ca GitHub să fie atât de puternic. Chiar și pentru cei care nu ajung să utilizeze interfața GitHub, GitHub poate simplifica gestionarea contribuțiilor, pentru că oferă un loc centralizat în care oamenii pot discuta despre patch-uri (fixuri pentru anumite probleme sau funcționalități noi).

Alte funcționalități oferite de GitHub:

- Documentație, inclusiv fișiere README redactate automat într-o varietate de formate de fișiere asemănătoare Markdown (consultați fișiere README pe GitHub)
- Pull requests (inclusiv feature reuqests) cu etichete, repere, utilizatori asignați și un motor de căutare
- Wikis
- Pull requests cu code-review și comentarii
- Istoric commit-uri
- Grafice: puls, contribuitori, commit-uri, frecvență cod, carte de punch, rețea, membri
- Directorul de integrări
- Diferențe unificate și divizate

În aplicațiile pe care le-am analizat, și care au ca funcționalitate integrarea cu GitHub, integrarea cu mediul de control al versiunii înseamnă că într-o conversație atașată unui repository vor fi afișate mesaje de notificare pentru acțiuni efectuate asupra repository-ului. Însă aplicația noastră își propune să ofere mai mult de atât, și anume posibilitatea de a vedea în paralel chat-ul și pagina de repository, și de a lega conversația de ancore din pagină. Luând în considerare multitudinea de funcționalități oferite de Github, considerăm că discuțiile din chat ar putea beneficia de vizualizarea în paralel a informațiilor de pe o pagină GitHub.

Aplicația noastră va oferi funcționalitatea de conectare a unui chat cu un repository GitHub, care va fi apoi afișat în paralel cu discuția din chat, pe jumătate de ecran (split-screen).

3.4. Chatbot

Un chatbot este un program de calculator care efectuează o conversație prin metode auditive sau textuale. Chatbot-urile sunt de obicei folosite în sistemele de dialog pentru diferite scopuri practice, inclusiv servicii pentru clienți sau achiziții de informații. Unele grupuri de chat utilizează sisteme sofisticate de procesare a limbajului natural, însă multe dintre cele mai simple scanează cuvintele cheie din intrare, apoi trag un răspuns cu cele mai potrivite cuvinte cheie sau cu cel mai asemănător model de formulare dintr-o bază de date.

În afară de chatbots, Conversational AI se referă la utilizarea aplicațiilor de mesagerie, a asistenților pe bază de vorbire și a chatbots pentru a automatiza comunicarea și a crea experiențe personalizate ale clienților la scară.

Există două modalități principale prin care chatbots sunt oferiți vizitatorilor: prin aplicații web sau prin aplicații independente. Astăzi, chatbots sunt folosiți cel mai frecvent în spațiul de servicii pentru clienți, presupunând roluri îndeplinite în mod tradițional de ființe umane, cum ar fi operatorii de și cei care analizează satisfacția clienților.

Chatbots procesează textul prezentat de utilizator (un proces cunoscut sub numele de "parsare"), înainte de a răspunde conform unei serii complexe de algoritmi care interpretează și identifică ceea ce a spus utilizatorul, deduce semnificația și/sau ce își dorește utilizatorul și determină o serie de răspunsuri adecvate bazate pe aceste informații. Unele chatbot-uri oferă o experiență conversațională remarcabil de autentică, în care este foarte dificil de determinat dacă agentul este un bot sau o ființă umană[20].

În versiunea inițială a aplicației noastre am implementat un chatbot simplu care să răspundă câtorva comenzi de bază, prin care să poată oferi informații utilizatorilor care au început să utilizeze pentru prima oară chatul.

---

[20] https://www.wordstream.com/blog/ws/2017/10/04/chatbots

Capitolul 4. Analiză și Fundamentare Teoretică

Acest capitol prezintă metodologia folosită pentru analiza și proiectarea aplicației ChatDev, cerințele funcționale și non-funcționale ale aplicației, cazurile de utilizare, tehnologiile utilizate, controlul versiunii pentru aplicația noastră și instrumentele de continuous integration.

Pentru planificarea și proiectarea aplicației noastre am folosit metoda Agile, care este un set de tehnici de gestionare a proiectelor de dezvoltare software. Este compusă din:

- Abilitatea de a răspunde repede schimbărilor și noilor cerințe.
- Lucrul în echipă, chiar și cu clientul.
- Construirea de software de operare pe baza unei documentații extinse.
- Actorii și interacțiunea acestora prin instrumente.

Considerăm că tehnica s-a potrivit proiectului nostru, deoarece nu am știut toate funcționalitățile încă de la început. Folosind Agile, am reușit să ne concentrăm doar pe caracteristicile care aveau cea mai mare prioritate la momentul respectiv.

4.1. Cerințe funcționale și use-case-uri

Scenariile utilizatorilor sunt una dintre cele mai importante componente de dezvoltare atunci când lucrăm cu metodologia Agile. Un scenariu al unui utilizator este o definiție de nivel înalt a unei cerințe, care conține suficiente informații pentru ca dezvoltatorii să poată produce o estimare rezonabilă a efortului de punere în aplicare a acesteia.

Adunate de la părțile interesate (persoane, grupuri sau organizații interesate de proiect), scenariile ne arată la ce trebuie să lucrăm și din ele putem extrage funcționalitățile aplicației finale. Deoarece lucrăm cu Agile, această listă nu trebuia să fie completă înainte să începem să lucrăm la proiect, dar e de dorit să aibă cel puțin câteva elemente de început, astfel încât să putem stabili prioritățile caracteristice corespunzătoare. La începutul fiecărui sprint, am analizat toate scenariile utilizatorilor, am estimat valoarea pe care au adăugat-o la proiect și cantitatea de timp pe care ar necesita-o fiecare dintre ele, și sortându-le în ordine descrescătoare - plasând scenariile utilizatorilor care au avut cea mai mare valoare adăugată și costul cel mai mic la început.

Valoarea a fost destul de subiectivă. Am acordat cea mai mare prioritate caracteristicilor pe care noi credeau că sunt esențiale pentru platformă (cum ar fi mesajele text instant) sau care au fost foarte legate de subiectul chat-ului. Le-am dat un scor de la 1-10. Costul de timp a fost o estimare a cât de mult credeam că un scenariu individual ar dura pentru a-l pune în aplicare. Măsura a fost efectuată în zile, considerând fiecare zi de lucru ca fiind de 4 ore. Apoi am tradus această valoare după cum urmează: 1-2 zile: 1, 3-4 zile: 2, 5-6 zile: 3, 7-9 zile: 4, 10 zile: 5. Pentru a sorta atât listele de backlog ale proiectului, cât și listele de backlog-uri ale sprintului, ne-am bazat pe un al treilea număr, prioritatea, care a fost pur și simplu rezultatul valorii minus costul de timp.

Cu toate acestea, în unele cazuri, a trebuit să facem excepții din cauza dependenței de anumite scenarii. De exemplu, funcționalitățile de sign in și sign up

trebuiau implementate primul, deoarece am avut nevoie de informații de utilizator pentru a identifica corect proprietarul camerei sau mesajul expeditor.

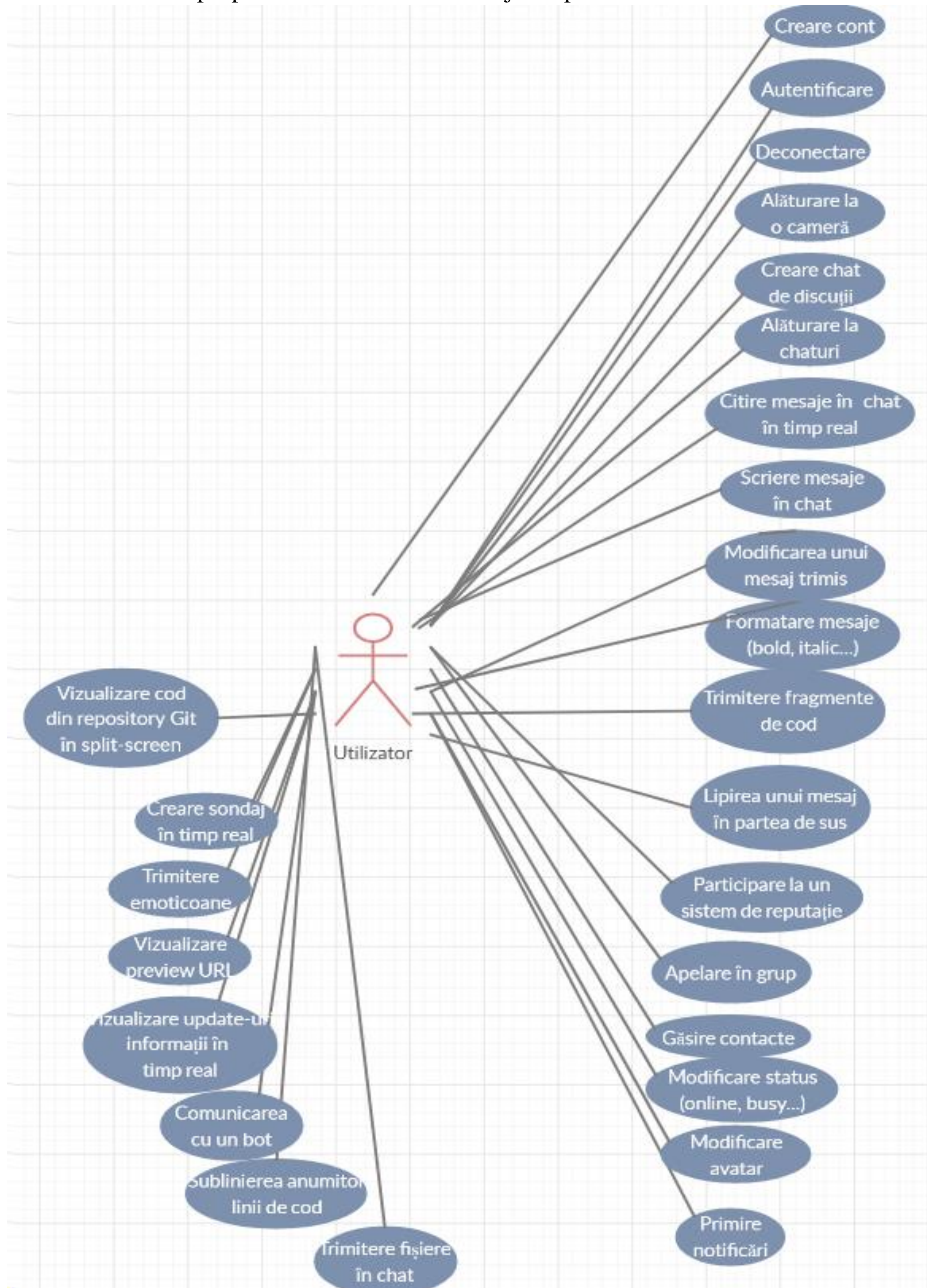


Figura 4.1 Diagrama UML use-case-uri utilizator

Lista noastră inițială de backlog a fost următoarea, împărțită pe actori (roluri utilizator):

Tabel 4.1 Scenarii de utilizare pentru utilizatorul obișnuit

#	Scenariu *utilizator*	Valoare	Timp	Prioritate
1	Autentificare cu un nume unic (care va fi identificator pe perioada întregii utilizări a platformei).	1	4	10*
2	Reconectare.	1	4	10*
3	Deconectare când nu mai am nevoie să folosesc aplicația pe un device.	1	1	10*
4	Alăturare la o cameră creată anterior de un manager.	7	1	10*
5	Creare chaturi de discuție.	6	2	10*
6	Alăturare la chat-urile din camerele în care sunt parte.	6	1	10*
7	Vizualizarea codului din repository-ul Git atașat chat-ului în aceeași fereastră, cu split-screen.	9	3	9
8	Citire mesajele în timp real în chat-ul în care mă aflu.	10	4	8
9	Scriere mesaje care vor fi trimise şi pot fi vizualizate în timp real de ceilalți membri din chat.	10	2	8
10	Formatare mesaje (bold, italice, link-uri, ...).	8	2	7
11	Trimitere fotografii în chat.	8	2	7
12	Trimitere fragmente de cod.	9	3	7
13	Lipirea unui mesaj important în partea de sus a chat-ului.	7	2	6
14	Modificarea unui chat (mesaj trimis de mine).	7	2	6
15	Sublinierea anumite linii într-un fragment de cod.	7	2	6
16	Comunicarea cu un bot de la care să aflu informații de bază despre aplicație	7	2	6
17	Vizualizarea update-urilor în timp real la informații.	6	1	5
18	Vizualizarea descrierii unei pagini către care am primit un URL, înainte să deschid pagina propriu-zisă.	6	2	5
19	Trimitere emoticoane în chat.	6	1	5
20	Trimitere fișiere de orice tip.	7	4	5
21	Creare sondaje în timp real în chat.	6	4	4
22	Primire notificări pe desktop când am primit un mesaj şi nu eram activ pe pagina de chat.	3	1	2
23	Modificare avatar.	2	2	1
24	Modificare status (online, offline, away, busy, ...).	2	2	1
25	Găsire contacte prin conectarea la rețele de socializare.	2	2	1
26	Apelare în grup.	6	20	1
27	Participare la un sistem de reputație (bazat pe activitatea mea în chat şi pe GitHub).	6	15	1

* Prioritate mărită din cauza dependențelor.

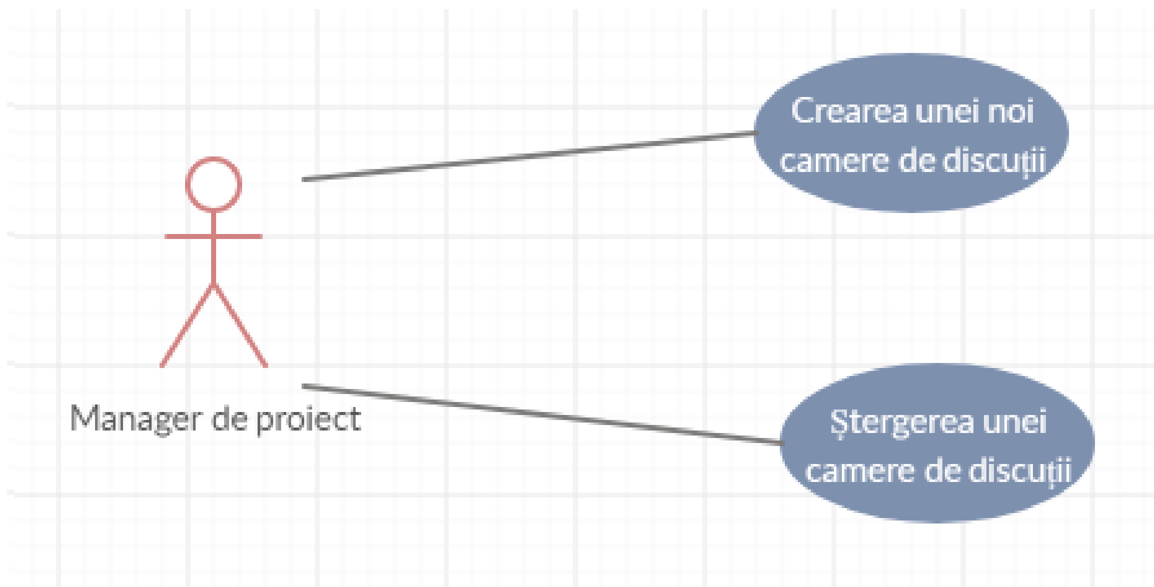


Figura 4.2 Diagrama UML use-case-uri manager de proiect

Tabel 4.2 Scenarii de utilizare pentru utilizatorul manager de proiect

#	**Scenariu *manager proiect***	**Valoare**	**Timp**	**Prioritate**
1	Crearea unei camere noi de discuții pentru un proiect.	5	2	10*
2	Șterge unei camere de chat.	4	2	3

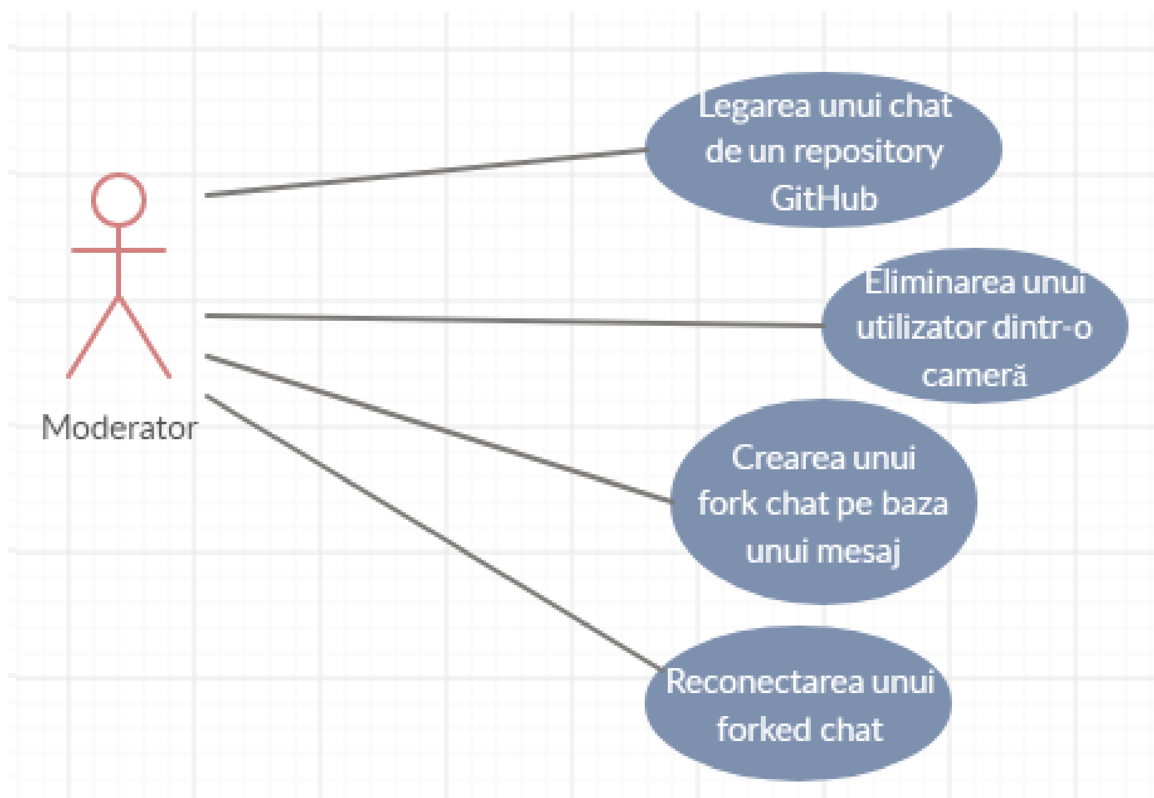


Figura 4.3 Diagrama UML use-case-uri moderator

Tabel 4.3 Scenarii de utilizare pentru utilizatorul moderator

#	Scenariu *moderator*	Valoare	Timp	Prioritate
1	Legarea unui chat de un repository GitHub, așa încât membrii chat-ului să vadă modificările în timp real (pull requests, issues, commits, comentarii, ...).	9	12	7
2	Eliminarea unui utilizator dintr-o cameră.	6	2	5
3	Crearea unui fork în chat bazat pe un anumit mesaj.	8	4	6
4	Reconectarea unui chat care a fost desprins, când discuția s-a încheiat.	7	4	5

Tabel 4.4 Scenarii de utilizare pentru utilizatorul developer

#	Scenariu *developer*	Valoare	Timp	Prioritate
1	Accesarea publică a API-ului proiectului, și crearea unor aplicații bazate pe aceasta.	9	30	4

Mai târziu, când implementam proiectul, am descoperit câteva cerințe care lipsesc şi le-am adăugat. Posibilitatea de a extinde specificaţiile şi a adapta cerinţele proiectului este motivul principal pentru care folosim dezvoltarea Agile.

Noul tabel de cerințe arată după cum urmează:

Tabel 4.5 Scenarii adăugate pentru utilizatorul obișnuit

#	Scenariu *utilizator*	Valoare	Timp	Prioritate
1	Accesarea unei pagini care să ducă spre chat-urile existente.	8	2	7
2	Notarea MarkDown în mesaje, care să includă cel puțin bold, italice, sublinieri, tabele și link-uri.	8	2	7
3	Auto-scroll a mesajelor și nu scroll manual la mesajele nou primite când containerul de chat e full.	7	1	6
4	Redimensionare automată a căsuței de introducere text dacă textul e prea lung și nu încape în căsuța standard.	7	1	6
5	Reconectare automată la chat dacă se pierde conexiunea.	5	3	3
6	Nu vreau ca toate conversațiile să fie încărcate de la început, ci conversațiile mai vechi să fie încărcate doar când sunt cerute.	4	3	2

4.2. Cerințe non-funcționale

Utilizabilitatea

Utilizabilitatea reprezintă măsura uşurinţei cu care un sistem poate fi învăţat şi utilizat, a siguranţei pe care o inspiră, a eficienţei şi eficacităţii, a atitudinii utilizatorilor în raport cu acel sistem. Printre atributele utilizabilităţii se numära uşurinţa de învăţare a aplicaţiei de către utilizatorii noi, rapiditatea cu care se familiarizează cu ea, eficienţa cu care utilizatorii avansaţi se descurcă, predictibilitatea sistemului precum și multe altele.

Aplicaţia propusă are o interfaţă simplistă, nu e încărcată, astfel încât utilizatorii să găsească cu uşurinţă informaţiile de care au nevoie. Controalele sunt uşor de observat, cu imagini și descrieri cât mai sugestive, oferind productivitate sporită, odată ce utilizatorul e obișnuit cu aplicația.

Securitatea

Securitatea aplicațiilor web reprezintă totalitatea măsurilor luate pentru a mări siguranța protejării datelor, prin căutarea, rezolvarea și prevenirea vulnerabilităților de securitate.

Deoarece aplicația implică trimiterea de date, informații, documente, datele utilizatorilor, ale proiectelor în care organizația e implicată trebuie protejate. Astfel, fiecare utilizator trebuie să fie autentificat pentru a putea folosi aplicația. Acest lucru se face pe baza unui token, unic pe sesiune pentru un utilizator, existând și o perioadă de expirare a acestuia, pentru a nu putea fi folosit de către cei din exterior. Fără acest token, request-urile către server nu vor putea fi realizate. Mesajele vor fi încriptate end-to-end pentru a spori securizarea conversațiilor.

Extensibilitatea

Extensibilitatea unui sistem este acea caracteristică prin care se care determină în ce măsură poate fi extins sau reimplementat în diverse moduri.

Întrucât aplicația a fost realizată pe layere, integrarea unor funcționalități noi nu impune o dificultate majoră, întrucât nu modifică comportamentul deja existent.

Performanța

Performanța ca și indicator de calitate reprezintă o măsură care definește fie volumul de procesări pe care o aplicație trebuie să îl poată face pe unitatea de timp sau termenul care trebuie respectat pentru finalizarea corectă a unei aplicații.

Timpul de răspuns la request-uri depinde de cantitatea de informații care este livrată și de complexitatea care se află în spatele prelucrării datelor, oscilând de la 5 milisecunde, în cazul operațiilor de trimitere mesaje până la câteva secunde, în cazul trimiterii unui fișier sau a conectării unui chat cu un repository.

4.3. Cazuri de utilizare

4.3.1. Creare cameră de discuții

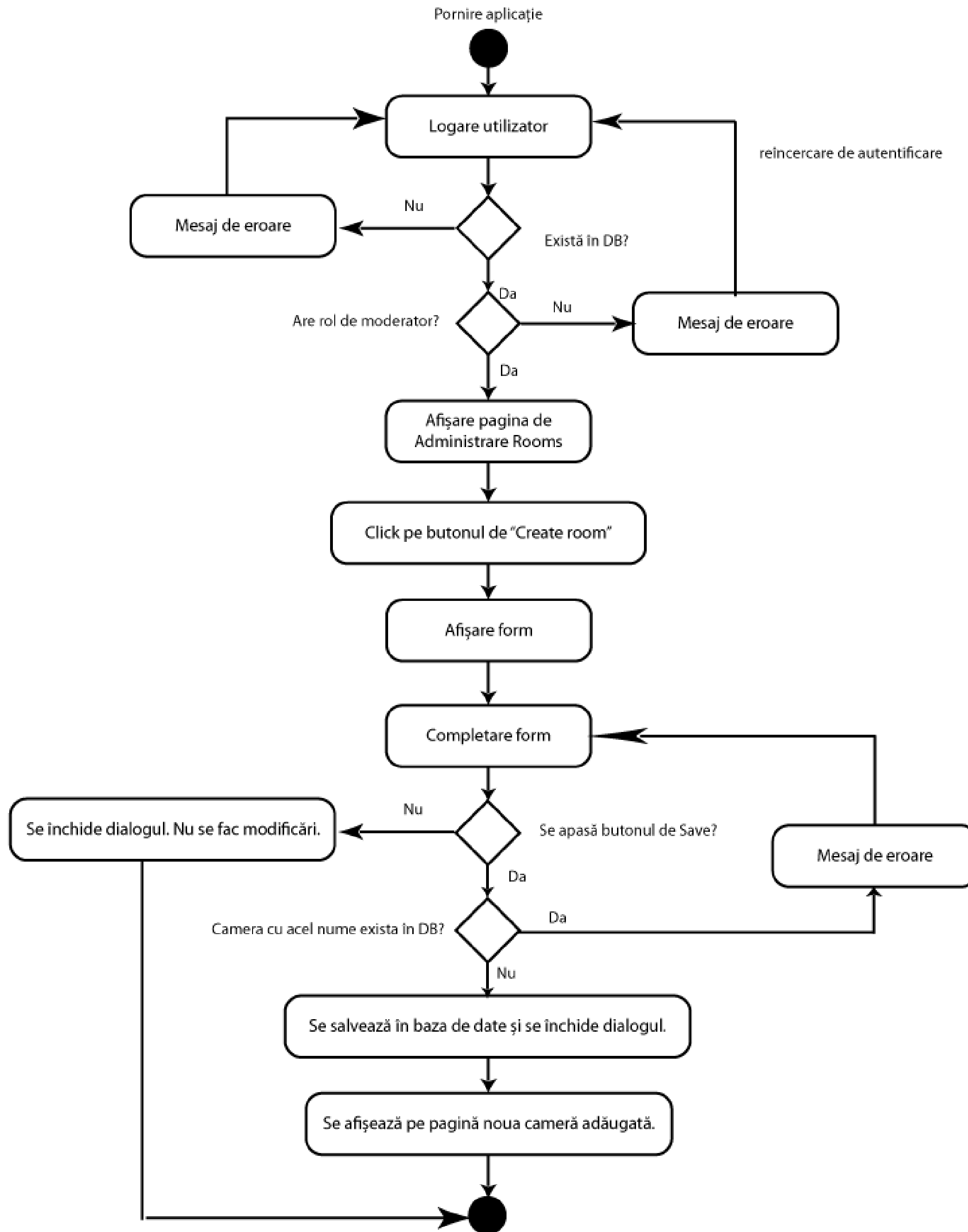


Figura 4.4 Diagrama use-case pentru crearea unei camere de către un moderator

Actor Principal: Moderator

Precondiții:

- Persoana care folosește platforma trebuie sa fie autentificată și de asemenea să aibă autorizație de moderator
- Moderatorul trebuie să se afle pe pagina de “Administrare Rooms”

Postcondiții:

- Camera este adăugată în baza de date și apare în lista din interfață.
- Cameră cu acel nume exista deja în baza de date și în interfața apare un mesaj de eroare.

Use case Start

1. Sistemul verifică dacă utilizatorul există în baza de date
2. Dacă acesta exista are rolul de moderator, este afișată fereastra de creare cameră
3. Sistemul pune la dispoziție lista camerelor existente
4. Moderatorul apasă pe butonul de “Create room”
5. Apare un form, cu câmpurile care trebuie completate
6. Moderatorul completează câmpurile
7. Sistemul verifica corectitudinea câmpurilor introduse
8. Butonul de Save este enabled și poate fi apăsat
9. Sistemul verifică daca camera cu acel nume există deja în baza de date. Se salvează datele

Use-Case End: Noua cameră este salvată în baza de date.

4.3.2. Alternative Flows – creare cameră de discuții

1. Utilizatorul care dorește să se autentifice nu există în baza de date **Alternative Flow Starts**

a. Este afișat un mesaj de eroare

b. Datele trebuie reintroduse

Alternative Flow Ends : Utilizatorul se loghează

2. Utilizatorul nu are rol de moderator **Alternative Flow Starts**

a. Utilizatorul este atenționat că nu poate accesa acea funcționalitate.

Alternative Flow Ends : Pentru a adaugă un utilizator trebuie să se autentifice cu rol corespunzător

7. Câmpurile sunt introduse eronat **Alternative Flow Starts**

a. Utilizatorul este notificat vizual, cu mesaje corespunzătoare în interfața vizuala

b. Utilizatorul reintroduce datele

c. Se revine la pasul 6

9. Camera cu acel nume există deja în baza de date **Alternative Flow Starts**

a. Sistemul transmite un mesaj de eroare spunând că o cameră cu acel nume există în baza de date

b. Utilizatorul verifică datele introduse

c. Se revine la pasul 6

4.3.3. Creare și conectare chat cu un repository GitHub

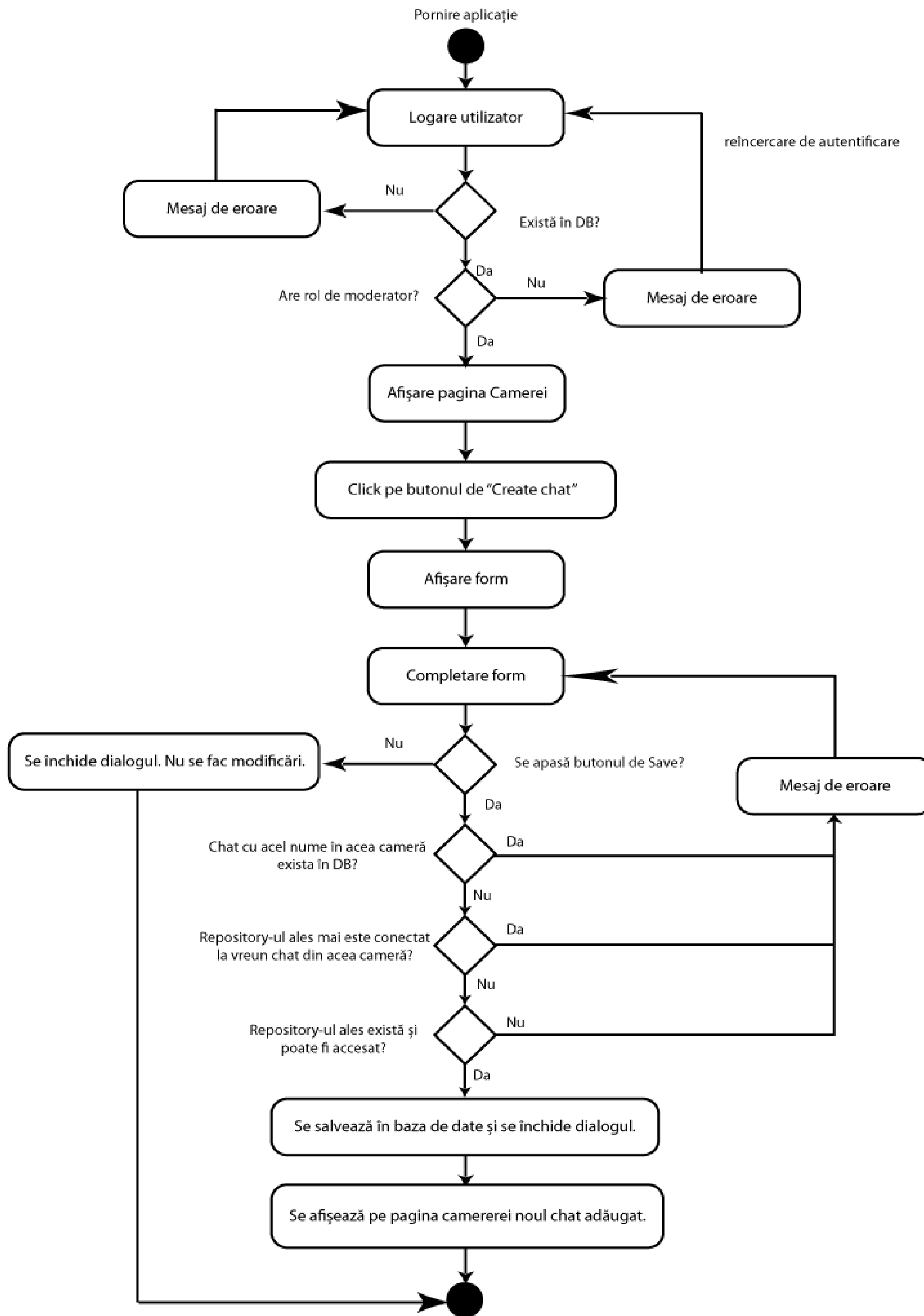


Figura 4.5 Diagrama use-case pentru crearea unui chat într-o cameră de către un moderator, chat conectat la un repository GitHub

Actor Principal: Moderator

Precondiții:

- Persoana care folosește platforma trebuie sa fie autentificată și de asemenea să aibă autorizație de moderator
- Moderatorul trebuie să se afle pe pagina unei camere
- Repository-ul cu care se va face conectarea trebuie să existe

Postcondiții:

- Chatul este adăugată în baza de date, aparținând camerei respective și apare în lista din interfață. Chatul este conectat cu repository-ul ales, care va fi afișat în split-screen.
- Chatul cu acel nume exista deja în acea cameră și în interfața apare un mesaj de eroare.
- Repository-ul ales nu există sau nu poate fi accesat și în interfață apare un mesaj de eroare

Use case Start

1 Sistemul verifică dacă utilizatorul există în baza de date

2 Dacă acesta exista are rolul de moderator, este afișată fereastra de creare chat

3. Apare un form, cu câmpurile care trebuie completate

4. Moderatorul completează câmpurile

5. Sistemul verifica corectitudinea câmpurilor introduse. Daca datele sunt corecte se trece la pasul 8

6. Butonul de Save este enabled și poate fi apăsat.

7. Sistemul verifică dacă chatul cu acel nume în acea cameră există deja în baza de date.

8. Sistemul verifică dacă repository-ul ales nu este deja conectat la un chat din acea cameră.

9. Sistemul verifică dacă repository-ul ales există și poate fi accesat. Dacă da, se salvează datele.

Use-Case End: Noul chat din acea cameră este salvată în baza de date, cu conexiunea la repository-ul Git.

4.3.4. Alternative flows – creare și conectare chat cu un repository GitHub

1. Utilizatorul care dorește să se autentifice nu există în baza de date **Alternative Flow Starts**

a. Este afișat un mesaj de eroare

b. Datele trebuie reintroduse

Alternative Flow Ends : Utilizatorul se loghează

2. Utilizatorul nu are rol de moderator **Alternative Flow Starts**

a. Utilizatorul este atenționat că nu poate accesa acea funcționalitate.

Alternative Flow Ends : Pentru a adaugă un utilizator trebuie să se autentifice cu rol corespunzător

5. Câmpurile sunt introduse eronat **Alternative Flow Starts**

a. Utilizatorul este notificat vizual, cu mesaje corespunzătoare în interfața vizuala

b. Utilizatorul reintroduce datele
c. Se revine la pasul 4
7. Chatul cu acel nume în acea cameră există deja în baza de date **Alternative Flow Starts**
a. Sistemul transmite un mesaj de eroare spunând că un chat cu acel nume în acea cameră există în baza de date
b. Utilizatorul verifică datele introduse
c. Se revine la pasul 4
Alternative Flow Ends : Pentru a crea un chat, utilizatorul introduce un nume unic în cameră.
8. Există deja un chat în acea cameră conectat la repository-ul ales **Alternative Flow Starts**
a. Utilizatorul este atenționat că repository-ul ales este deja conectat la un chat din acea cameră.
b. Utilizatorul verifică datele introduse
c. Se revine la pasul 4
Alternative Flow Ends : Pentru a conecta un chat dintr-o cameră la un repository, utilizatorul trebuie să aleagă repository-ul corespunzător sau să folosească un chat deja existent
9. Repository-ul ales nu există sau nu poate fi accesat **Alternative Flow Starts**
a. Utilizatorul este atenționat că repository-ul ales nu este disponibil.
b. Utilizatorul verifică datele introduse
c. Se revine la pasul 4

4.4. Tehnologii utilizate

Arhitectura aplicației este client-server, formată din front-end și back-end, ambele având propriile dependențe stabilite (biblioteci și framework-uri).

Front-end-ul este nivelul de prezentare pe care utilizatorul final îl vede când intră pe site-ul aplicației. Partea din spate (back-end) oferă toate datele și o parte a logicii și rulează în spatele scenei.

Modelul client-server este o structură de aplicație distribuită care împarte sarcini sau încărcări de lucru între furnizorii unei resurse sau servicii, numite servere și solicitanți de servicii numiți clienți. Deseori, clienții și serverele comunică printr-o rețea de calculatoare pe hardware separat, dar atât clientul, cât și serverul pot locui în același sistem. O gazdă de server rulează unul sau mai multe programe server care împărtășesc resursele lor clienților. Un client nu împărtășește nici una dintre resursele sale, ci solicită o funcție de conținut sau de serviciu a unui server. Prin urmare, clienții inițiază sesiuni de comunicare cu servere care așteaptă cererile primite. Exemple de aplicații informatice care utilizează modelul client-server sunt E-mail, imprimare în rețea și World Wide Web [2].

În general, un serviciu este o abstractizare a resurselor de calcul și un client nu trebuie să fie preocupat de modul în care serverul funcționează în timp ce îndeplinește cererea și transmite răspunsul. Clientul trebuie doar să înțeleagă răspunsul bazat pe

protocolul binecunoscut al aplicației, adică conținutul și formatarea datelor pentru serviciul solicitat.

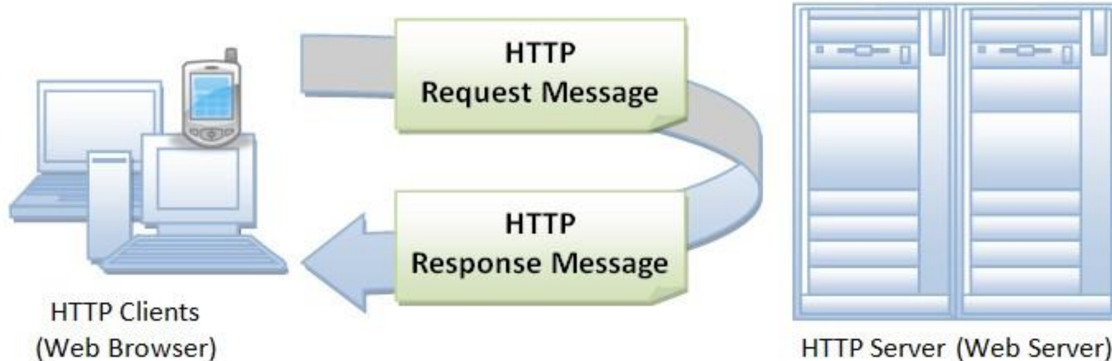


Figura 4.6 Aplicație client-server care comunică prin HTTP[21]

Clienții și serverele fac schimb de mesaje într-un model de mesagerie cerere-răspuns. Clientul trimite o cerere, iar serverul returnează un răspuns. Acest schimb de mesaje este un exemplu de comunicare între procese. Pentru a comunica, computerele trebuie să aibă un limbaj comun și trebuie să respecte regulile, astfel încât atât clientul cât și serverul să știe la ce să se aștepte. Limba și regulile de comunicare sunt definite într-un protocol de comunicații. Toate protocoalele client-server funcționează în stratul de aplicație. Protocolul nivelului de aplicație definește tiparele de bază ale dialogului. Pentru a formaliza și mai mult schimbul de date, serverul poate implementa o interfață de programare a aplicațiilor (API). API este un strat de abstractizare pentru accesarea unui serviciu. Prin limitarea comunicării la un anumit format de conținut, facilitează parsarea. Prin abstractizarea accesului, facilitează schimbul de date între platforme.

4.4.1. Tehnologii de comunicare

Pentru majoritatea aplicațiilor web, AJAX prin HTTP este alegerea cea mai ușoară, deoarece e un protocol stabil și larg suportat. Totuși, pentru aplicația noastră, nu aceasta ar fi fost alegerea cea mai potrivită. Avem nevoie de o metodă rapidă de a trimite și primi mesaje în timp real (chiar dacă aceasta nu se aplică la orice situație).

Pentru mesaje, există câteva protocoale de comunicare disponibile pentru web. Cele mai populare sunt AJAX, WebSockets și WebRTC. **AJAX** este o abordare lentă. Nu numai din cauza anteturilor care trebuie trimise în fiecare cerere, dar, și mai important, pentru că nu există nicio modalitate de **a primi notificări** (push) despre mesaje noi într-o cameră de chat. Dacă am utiliza AJAX, ar trebui **să solicităm** (pull) noi mesaje de la server la fiecare câteva secunde, ceea ce ar duce la apariția unor mesaje noi care au nevoie de câteva secunde pentru a apărea pe ecran, și numeroase cereri redundante generate.

WebSockets[22] reprezintă o abordare mai bună. Conexiunile WebSockets pot dura până la câteva secunde, însă, mulțumită canalului de comunicare full duplex, mesajele pot fi schimbate rapid (cu o medie de întârziere de câteva milisecunde pe mesaj). De

[21] https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
[22] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

asemenea, clientul și serverul pot primi notificări prin intermediul aceluiași canal de comunicare, ceea ce înseamnă că, spre deosebire de AJAX, clientul nu trebuie să trimită serverului o cerere pentru a prelua mesaje noi, ci mai degrabă așteaptă ca serverul să le trimită [3].

WebRTC[23] este noul protocol de comunicare disponibil pentru cele mai moderne browsere (Chrome, Firefox și Opera). Este proiectat pentru înaltă performanță, de înaltă calitate, comunicarea datelor video, audio și arbitrare. WebRTC nu necesită niciun server ca proxy pentru schimbul de date, altul decât serverul de semnalizare care este necesar pentru partajarea metadatelor de rețea și media (adesea realizate prin WebSockets). Faptul că datele de flux pot fi schimbate direct între clienți înseamnă adesea mesaje mai rapide și mai puțină încărcare pe server.

WebRTC poate rula pe TCP și UDP, dar rulează adesea cu UDP în mod implicit. Deși UDP poate duce la pierderea de pachete, acesta oferă o performanță mai bună, care poate produce un apel vocal sau video mai fluid, iar pierderea câtorva cadre în timpul apelurilor video nu e un impediment atât de mare.

Având în vedere avantajele și dezavantajele celor trei tehnologii, am decis să folosim WebSockets pentru mesageria în timp real, ceea ce ne garantează livrarea de pachete (spre deosebire de cadrele dintr-un apel video, nu dorim să pierdem niciun mesaj text) o bună compatibilitate și o tehnologie populară și bine documentată.

Vom folosi WebRTC atât pentru apeluri vocale, cât și pentru apeluri video, care fac parte din implementările viitoare. WebRTC ar face o conversație mai fluidă în majoritatea cazurilor, datorită timpului de livrare mai rapid al pachetelor. Când vine vorba de pachetele de voce sau video pierdute, atât timp cât calitatea conversației nu e afectată, acestea pot fi neglijate.

Când vine vorba de solicitări aleatorii, cum ar fi autentificarea, crearea camerei sau listarea, AJAX este o opțiune bună. Nu necesită o conexiune permanentă la server, ceea ce duce la o utilizare mai redusă a energiei atât pentru client, cât și pentru server, iar timpii de răspuns ai cererilor sunt decente. Niciunul dintre aceste tipuri de solicitări nu necesită un răspuns extrem de rapid.

Mai mult, solicitările AJAX sunt atât de populare încât versiunile celor mai recente browsere oferă deja API-uri de nivel înalt, precum și bibliotecile vechi de nivel scăzut, ceea ce face facilă implementarea unor cereri pentru un JSON, de exemplu.

4.4.2. Back End

"Back end" se referă la nivelurile logice și de date care rulează pe partea serverului. În cazul nostru, back-end-ul se asigură că datele introduse prin client (front end) sunt valide. Deoarece front-end-ul poate fi evitat sau ușor manipulat (codul sursă este disponibil pentru utilizatorul final) va trebui să ne asigurăm că toate cererile pe care le primim sunt verificate mai întâi de către server: URI solicitat este acceptat, utilizatorul are permisiunile corespunzătoare, parametrii sunt valizi, etc.

Dacă datele de solicitare sunt valide, procedăm adesea pentru a executa o anumită logică, însoțită de una sau mai multe accesări ale bazei de date.

[23] https://webrtc.org/

4.4.2.1. Business Logic

Aplicația noastră se bazează pe I/O. A fost necesar să folosim un mediu de programare care să fie capabil să gestioneze foarte multe solicitări pe secundă, mai mult decât una specializată în manipularea sarcinilor intensive ale procesorului.

Dintre opțiunile care ne stăteau la dispoziție, alegerea finală a fost între PHP, Python, Java, Go sau Node.js. Aceste limbaje dispun de documentație solidă disponibilă, și au fost deja testate pe scară largă de mulți utilizatori și dezvoltatori. Cea mai în vogă tehnologie în momentul dezvoltării aplicației a fost Node.js, care a fost adecvată pentru manipularea cererilor de I/O, prin prelucrarea asincron într-un singur thread [4].

Alegerea finală a fost Node.js[24], nu doar din cauza performanței, ci luând în considerare și timpul de implementare necesar pentru aplicația noastră cu această tehnologie, comparativ cu alte limbaje, cum ar fi Java, care e mult mai verbose și necesită mai multe linii de cod pentru a implementa aceeași funcționalitate. Pentru dezvoltarea web, am folosi Express, care utilizează puterea Node.js pentru a face conținutul web chiar și mai rapid de implementat.

O alternativă fezabilă la Node.js ar fi Go, care devine populară în zilele noastre, fiind oarecum mai rapidă pe I/O decât Node.js, cu subrutinele Go, și performanță fără îndoială mai bună la calculele intense (deși în aplicația noastră aceasta nu era o cerință obligatorie).

Cu toate acestea, Go ar fi însemnat o viteză mai mică de dezvoltare. Anumite biblioteci nu erau implementate, tehnologia nu e atât de matură ca Node.js și gestionarea greoaie a JSON a făcut-o mai puțin de dorit pentru aplicația noastră (deoarece clientul JavaScript ar folosi JSON tot timpul).

Standardele ES6/ES2017 diferă de clasicul Vanilla JavaScript, în sensul că au câteva funcții și utilitare de limbaj deja existente, ceea ce face codul mai ușor de citit, mai rapid la implementare și reduce nevoia de a face uz de bibliotecile externe pentru cele mai comune operațiuni. De exemplu, Promises în loc de callbacks sau clase în loc de funcții, chiar dacă acestea sunt doar artificii.

Câteva framework-uri/biblioteci demne de mențiune pe care le folosim pentru dezvoltarea platformei sunt:

Express[25]: Un framework Node.js care face posibilă dezvoltarea web mai rapidă. Abstractizează cea mai mare parte a complexității din spatele serverului web și acționează ca un route handler HTTP. De asemenea, poate genera view-uri de interfață (un fel de șabloane HTML cu variabile), dar vom folosi aplicația front end pentru aceasta. Folosind Express, suntem capabili să ne concentrăm mai degrabă pe logica din spatele fiecărei cereri, decât pe cererea în sine.

Mongoose[26]: O bibliotecă de nivel înalt MongoDB, care mapează obiecte pe modelele de baze de date, care ulterior vor fi datele din interiorul colecțiilor noastre, se ocupă de inserări, actualizări și ștergeri, precum și validarea fiecărui câmp.

Passport[27]: o bibliotecă de autentificare construită special pentru Node.js. Utilizând diferite module de conectare (un modul per furnizor), ascunde toată

[24] https://nodejs.org/
[25] https://expressjs.com/
[26] https://mongoosejs.com/

complexitatea din spatele OAuth, OAuth2 și OpenID. Passport se angajează să notifice dezvoltatorul în același mod, indiferent de metoda de autentificare pe care au ales-o. Folosim Passport pentru a gestiona autentificarea GitHub și Google, precum și autentificarea locală (email + parolă).

Sinon[28]: o bibliotecă vastă de testare care are un set de utilități utile: spies, stubs și mocks. În timpul testelor noastre, de multe ori simțim nevoia de a ști dacă o anumită funcție a fost apelată, a fost apelată cu parametrii potriviți, sau chiar pentru a fabrica date de intrare pentru a ne asigura că testăm doar ceea ce dorim.

Socket.io[29]: o bibliotecă JavaScript care gestionează conexiunile WebSocket. Abstractizează cea mai mare parte a complexității din spatele WebSockets și oferă, de asemenea, metode alternative care funcționează fără o configurație specială. Socket.io are grijă de actualizările în timp real în aplicația noastră, cum ar fi trimiterea sau primirea mesajelor.

4.4.2.2. Stocarea datelor

Considerăm că **NoSQL**[30] este viitorul în stocarea datelor. Prin urmare, nu am ezitat să alegem să folosim doar baze de date NoSQL.

De ce alegem baze de date NoSQL în locul celor tradiționale SQL?

- Sunt mai flexibile: poți accesa date imbricate fără a trebui să efectuezi niciun Join.
- Sunt mai rapide: datele imbricate sunt stocate în același loc și pot fi consultate fără nici o interogare suplimentară.
- Scalează mai bine la distribuirea datelor pe noduri diferite.
- Există multe tipuri de baze de date NoSQL care se potrivesc pentru diferite tipuri de proiecte, cum ar fi Key-Value pentru sesiuni sau bazate pe documente pentru date complexe.

La început, aveam de gând să folosim doar MongoDB, dar mai târziu am realizat că ar fi o idee bună să avem și Redis pentru a mapa cheile de sesiune cu identificatorii de utilizatori.

MongoDB[31] este o bază de date fără schemă, orientată pe documente. Ne dă posibilitatea de a stoca cu ușurință date complexe și a le recupera imediat, fără nici o interogare suplimentară. Deși datele sunt întotdeauna stocate pe disc, este foarte rapidă și foarte scalabilă.

Folosim MongoDB pentru a stoca orice date persistente, cum ar fi detaliile și preferințele utilizatorului, informații despre camere și mesaje de chat. Detalierea modelării documentelor MongoDB se găsește capitolul de implementare.

Redis[32] este o structură de date cheie-valoare, care utilizează memoria de stocare pentru a efectua căutări pentru o cheie dată foarte rapid. Interogările sunt efectuate mai

[27] http://www.passportjs.org/
[28] https://sinonjs.org/
[29] https://socket.io/
[30] https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL
[31] https://www.mongodb.com/
[32] https://redis.io/

rapid decât în MongoDB, dar există, de asemenea, un risc mai mare de a pierde date și nu pot fi procesate valori complexe (cum ar fi documentele imbricate).

Deși Redis are performanțe mai bune decât MongoDB, nu ne putem baza pe Redis pentru date critice. Din acest motiv, îl folosim doar pentru a stoca sesiunile de utilizatori, care în caz de pierdere, ar însemna doar că utilizatorul ar trebui să se autentifice din nou pentru a continua să folosească platformă. Cu toate acestea, ne așteptăm la un câștig de performanță destul de observabil atunci când site-ul este la capacitatea sa maximă, deoarece datele din sesiune vor fi cerute în fiecare cerere către aplicație.

4.4.3. Front End

Partea de client și interfața utilizator au fost implementate folosind principiile Single Page Application (SPA[33]), care aduc dinamic datele, în timp ce utilizatorul navighează pe site, evitând să reîmprospăteze întreaga pagină ori de câte ori utilizatorul completează un formular sau navighează în altă parte a site-ului.

Îmbunătățirea experienței utilizatorului pe care un SPA o poate obține față de un site tradițional este foarte semnificativă. Este adevărat că adesea este nevoie de mai mult timp pentru încărcare pentru prima dată, datorită descărcării unui set de fișiere JavaScript mai mare, dar odată încărcată întârzierea dintre operațiuni este minimă, ceea ce duce la o utilizare mai eficace a site-ului de către utilizator și la o utilizare mai mică a lățimii de bandă în cele mai multe cazuri [5].

Implementarea unei aplicații scalabile pentru o SPA, utilizând doar Vanilla JavaScript ar necesita mult timp, deoarece biblioteca nu are nici o utilitate la nivel înalt care simplifică dezvoltarea unui astfel de tip de pagină, cum ar fi un renderer HTML de nivel înalt care să permită construirea dinamică a elementelor, storage sau router. Prin urmare, a fost necesară folosirea unui framework/unei biblioteci întreținute și documentate activ pentru a implementa aplicația noastră.

La momentul respectiv, decizia a fost între Angular, React și Vue. Atât Angular, cât și React erau întreținute de corporații puternice, Google și respectiv Facebook, așa că am făcut o scurtă analiză a documentației lor și la recenziile dezvoltatorilor înainte de a lua decizia noastră finală.

Varianta aleasă, **React[34]**, este o bibliotecă puternică, cu un ecosistem bine dezvoltat (multe utilități au fost create pentru a fi utilizate cu React). Este promovată datorită performanțelor sale în ce privește viteza și consumul redus de memorie, care este util în special în dezvoltarea pentru dispozitive mobile. În plus, există o documentație bogată pe site-ul său oficial și pe internet. Principalele caracteristici ale bibliotecii sunt: structura de arbore, elemente DOM custom (React creează un DOM virtual în memorie), transmiterea informațiilor prin proprietățile componentelor [6].

- **Structura de arbore**

O pagină React începe întotdeauna cu o singură componentă rădăcină (nod de arbore) exportat într-un element HTML existent deja pe pagină. Fiecare componentă poate avea unul sau mai mulți copii (composition).

[33] https://www.cmswire.com/digital-experience/what-is-a-single-page-application/
[34] https://reactjs.org/

- **Elemente DOM custom**

React nu funcționează direct cu componente HTML. În schimb, folosește o componentă (care deseori are aceleași nume) care va fi transpusă mai târziu în componenta HTML.

- **Informațiile sunt transferate în arbore prin proprietățile componentelor**

Este posibil ca o componentă rădăcină să nu aibă nevoie de acces la informații externe, deoarece din punct de vedere tehnic conține întreaga aplicație.

Cu toate acestea, informațiile pot fi necesare pentru componentele copii. De exemplu, o componentă care este responsabilă pentru afișarea unei casete text generice de introducere împreună cu o etichetă va trebui să știe numele.

- **Informațiile sunt transferate în sus pe arbore prin referințele funcțiilor**

La un moment dat, părintele ar putea avea nevoie să afle despre informațiile care s-au schimbat la un copil, astfel încât să reacționeze într-un fel sau altul. De exemplu, în fragmentul de mai sus, ar putea fi necesar să cunoaștem când valoarea a fost schimbată la intrare, astfel încât să poată mai târziu procesa informațiile din formular.

React, spre deosebire de Angular, este doar V(iew) în arhitectura MVC. Prin urmare, a fost nevoie de librării suplimentare pentru a acoperi componentele lipsă, pentru a realiza funcționalitățile proiectului nostru.

Cele mai relevante biblioteci pe care le folosim în partea de client a aplicației noastre sunt:

Babel[35]: Unii utilizatori care folosesc chat-ul nostru ar putea să folosească versiuni vechi de browser, care suportă puțin sau deloc caracteristicile ES6 / ES2017. Pentru a vă asigura că toate browserele pot înțelege codul, folosim Babel, care transformă JavaScript-ul modern în cod JavaScript pe care majoritatea browserelor îl pot înțelege.

Redux[36]: Un spațiu de stocare în memorie pentru JavaScript. Se salvează stările aplicației, care, în alți termeni, sunt datele pe care aplicația noastră le utilizează de-a lungul timpului. Redux evită să transfere date în sus și în jos pe arborele React, deoarece Redux stochează totul într-un singur loc care poate fi accesat oricând. Este, de asemenea, modular, ceea ce îl face ideal pentru aplicația noastră, deoarece ajută la scalabilitate.

Enzyme[37]: O bibliotecă care facilitează testarea componentelor React. Această bibliotecă a fost construită pentru React și ne permite să simulăm componente ca și cum ar fi create în DOM. Este adesea folosită pentru a testa butoanele de manipulare sau elemente care ar trebui să apară doar în anumite condiții.

Sinon: O bibliotecă de testare JavaScript. Deoarece nu este specifică doar Node.js, o folosim și pe front-end pentru a face mai ușoară testarea.

Express: Aplicațiile SPA nu au conținut în fișierul HTML care este trimis clientului. Tot conținutul (și logica) se află în interiorul uneia sau mai multor bucăți JavaScript pe care le conține site-ul, ceea ce înseamnă că browserele trebuie să descarce aceste fișiere și să le execute înainte de a putea afișa ceva util în ecranul utilizatorului. Asta poate să dureze câteva secunde, în funcție de dimensiunile principalelor bucăți JavaScript.

[35] https://babeljs.io/
[36] https://redux.js.org/
[37] https://airbnb.io/enzyme/

Prin redarea primei pagini solicitate pe server, putem răspunde înapoi cu HTML preîncărcat (și chiar CSS), și chiar dacă navigarea va fi foarte limitată, în timp ce fișierele sunt încă în descărcare utilizatorii vor putea să citească conținutul acelei pagini ca și cum pagina ar fi fost complet încărcată.

În afară de aceasta, unele motoare de căutare sunt limitate la citirea HTML. Deoarece toate conținutul nostru se află în fișierele JavaScript, acestea nu vor vedea conținut. Prin urmare, nu vor lua în considerare niciunul dintre headerele sau alte metadate atunci când îl indexează [7].

O combinație de ReactDOMServer și Node.js ne permite să returnăm un rendered view al oricărei pagini de pe site-ul nostru, împreună cu endpoint-urile pentru bucățile JavaScript.

4.5. Controlul versiunii

Un sistem de control al versiunii poate fi util dezvoltatorilor, chiar și atunci când lucrează pe cont propriu, deoarece ne permite să ne reluăm etapele istoricului dezvoltării fiecărui fișier și fiecărei componente din proiect pentru a vedea ce a cauzat o eroare într-o anumită funcționalitate, să lucrăm la diferite caracteristici în același moment și să facem revert/merge în codul sursă original fără dificultăți, să urmărim cum a evoluat proiectul în timp și așa mai departe.

Pentru implementare vom folosi Git, nu numai pentru că este cea mai populară și mai folosită versiune sistemul de control, dar și pentru că o parte din proiectul nostru a fost integrarea cu GitHub, iar GitHub lucrează cu Git.

Din același motiv ca mai sus, am ales ca GitHub să fie repository-ul remote pentru codul nostru.

4.6. Continuous integration

Serviciile de integrare continuă sunt software-uri automate care rulează imediat ce o nouă versiunea este urcată într-un repository și trimite abonaților la repository informații permanente despre starea noii versiuni.

Popularitatea GitHub a dus la dezvoltarea multor servicii integrate care pot fi integrate cu acesta: servicii de integrare continuă (Travis, CircleCI, Appveyor), verificatori de dependență (David-DM, Greenkeeper, Dependency CI), verificatori ai calității codului (CodeClimate, Codacy), code coverage (Codecov, combinezoane) și multe altele.

Folosim câteva dintre acestea:

Travis[38] și **CircleCI**[39]: două din cele mai importante servicii pentru aplicația noastră, sunt ambele sisteme de integrare continuă care asigură faptul că aplicația noastră face *build* cu succes și toate testele trec.

CircleCI a fost adăugat spre sfârșitul proiectului și este responsabil și pentru ambalarea codului sursă într-un container Docker și implementarea acestuia într-un

[38] https://travis-ci.org/
[39] https://circleci.com

serviciu bazat pe cloud. Am descris acest lucru în detaliu în secțiunea de implementare a aplicației.

David-DM[40]: Supraveghează dependențele NPM ale proiectului nostru și ne avertizează ori de câte ori o dependența este obsolete sau deprecated. O dependență obsolete sau deprecated poate cauza probleme grave care vor fi corectate mult mai târziu în timp, așa că este recomandat să țineți dependențele mereu la curent.

Codecov[41]: raportează starea curentă de acoperire a testelor și calculează o acoperire procentuală. Acesta oferă o interfață web pentru a vedea cum se schimbă procentajul total în timp, precum și procente individuale pentru fiecare fișier din proiect.

CodeClimate[42]: Analizează fișierele proiectului și dă un rating de 1-4 în funcție de calitatea fiecărui. De exemplu, valoarea totală se diminuează dacă vreuna din fișierele de cod au scoruri prea mici sau aveți variabile sau funcții neutilizate. Prin utilizarea interfeței lor web, puteți verifica fiecare dintre aceste greșeli de calitate și le puteți corecta pentru a vă face codul mai curat și mai ușor de citit.

[40] https://david-dm.org/
[41] https://codecov.io/
[42] https://codeclimate.com/

Capitolul 5. Proiectare de Detaliu și Implementare

Cel mai des utilizat pattern arhitectural și, de asemenea, cel ales în realizarea acestei aplicații este cel structurat pe layere, adică pe mai multe nivele. Componentele acestei arhitecturi sunt organizate în layere orizontale, fiecare având un anumit rol bine definit. Ca și alte aplicații care folosesc această abordare, și aceasta are cele 3 layere de bază: presentation, business și database [8].

Clientul, care conține modulele de React, comunica cu serverul prin intermediul protocolului HTTP, întrucât, deși organizată pe layere, aplicația are la baza principiile client-server. Serverul interceptează cererile în nivelul de Rest Controller, le procesează la nivel de business logic, și operează cu baza de date prin intermediul layerului de persistență. Comunicarea este bidirecțională, iar după terminarea acestor operații, clientul primește un răspuns fie cu mesaje de eroare, fie cu datele necesae. Cea mai importantă caracteristică a acestei arhitecturi este separarea responsabilităților dintre componente, întrucât fiecare se ocupă de o anumită funcționalitate, care corespunde layerului din care face parte. Datorită acestei separări a comportamentului, este mult mai ușor de gestionat, de adăugat anumite funcționalități, fiind totodată mai ușor de dezvoltat, testat și menținut o astfel de aplicație bazată pe nivele. Fiecare nivel este independent față de celelalte, nefiind conștiente de funcționalitatea celorlalte, iar când o schimbare apare, aceasta afectează de obicei, doar nivelul din care face parte.

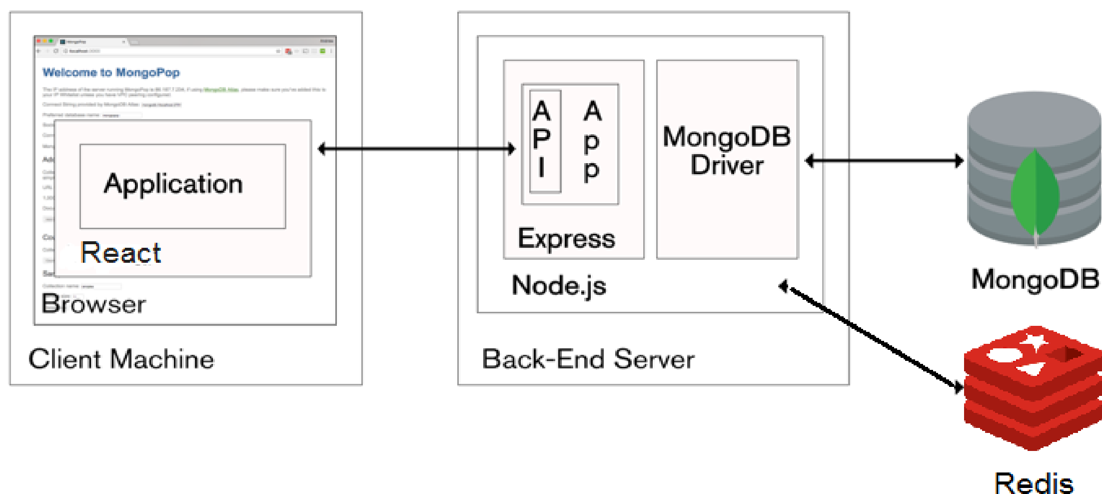


Figura 5.1 Arhitectura generală a aplicației

5.1. Structura bazei de date

Un aspect definitoriu al oricărei aplicații care necesită persistența datelor este structura bazei sale de date. Proiectarea bazei de date poate varia în funcție de numeroși factori, cum ar fi raportul de citiri/scrieri sau valorile pe care utilizatorul este probabil să le solicite cel mai mult. Acest lucru se datorează faptului că, în calitate de dezvoltatori ai întregii aplicații, dorim ca baza de date să aibă cea mai bună performanță, ceea ce poate fi adesea realizat prin concentrarea optimizărilor asupra acțiunilor cele mai comune.

Ne-am concentrat pe baza de date MongoDB, care este cea mai complexă structură de stocare a datelor și cea care stochează cele mai multe date.

Structura noastră de date Redis se limitează la maparea sesiunilor cu identificatorii de utilizator, ambele de tip text. Acesta este modul în care funcționează o solicitare web: interogările Node.js trimite un query către Redis utilizând identificatorul sesiunii de utilizator și al contului pentru a determina dacă utilizatorul este autentifica. Dacă se găsește un identificator de cont, Node.js interoghează MongoDB pentru a afla restul informațiilor despre utilizator.

Baza de date MongoDB stochează: informații despre utilizatori, camere, chaturi și mesaje. Proiectul nostru final de baze de date a ajuns să aibă patru colecții diferite: utilizatori, camere, chaturi și mesaje. Deși MongoDB nu are o schemă, prin utilizarea bibliotecii Mongoose pe Node.js, am putut, de asemenea, să definim o schemă flexibilă pentru fiecare colecție. O schemă restricționează conținutul unei colecții într-un format cunoscut, ne salvează de validarea structurii datelor înainte sau după ce a fost introdusă în baza de date [9].

5.1.1. Users

Colecția utilizatorilor stochează informațiile care fac referire la autentificarea și la datele lor personale. Camerele, chaturile și mesajele lor ar trebui să fie stocate în colecții separate, dar toate vor avea o referință către utilizatorul care este owner.

Câmpurile schemei:

- _id: identificator
- username: afișat pe interfață
- email: adresa de e-mail
- password: parola încriptată
- passwordResetToken: token pentru resetarea parolei
- passwordResetExpires: data expirării pentru tokenul de resetare parolă
- github: id-ul profilului de GitHub
- google: id-ul profilului de Google
- tokens: lista token-urilor serviciilor conectate
    - kind: numele serviciuui (i.e. github)
    - accessToken: token de acces oferit de serviciu
- profile: detalii personale
    - name: numele complet
    - gender: genul
    - location: locația
    - website: URL către site-ul sau blogul personal
    - picture: avatar URL.
- updatedAt: ultima modificare
- createdAt: data creării

Colecția de utilizatori este indexată pe câmpurile: _id, username, email, github și google. Acestea acoperă cele mai multe căutări, ceea ce se întâmplă cel mai des: utilizatorii sunt queried de foarte multe ori, deși informațiile acestora se schimbă foarte rar.

Cu toate că nu am specificat, unele câmpuri sunt obligatorii în schemă, în timp ce altele pot fi lăsate nedefinite. Toate aceste specificații, inclusiv fiecare validare a câmpurilor, au fost setate în Mongoose, fie sub formă de configurație, fie prin funcții.

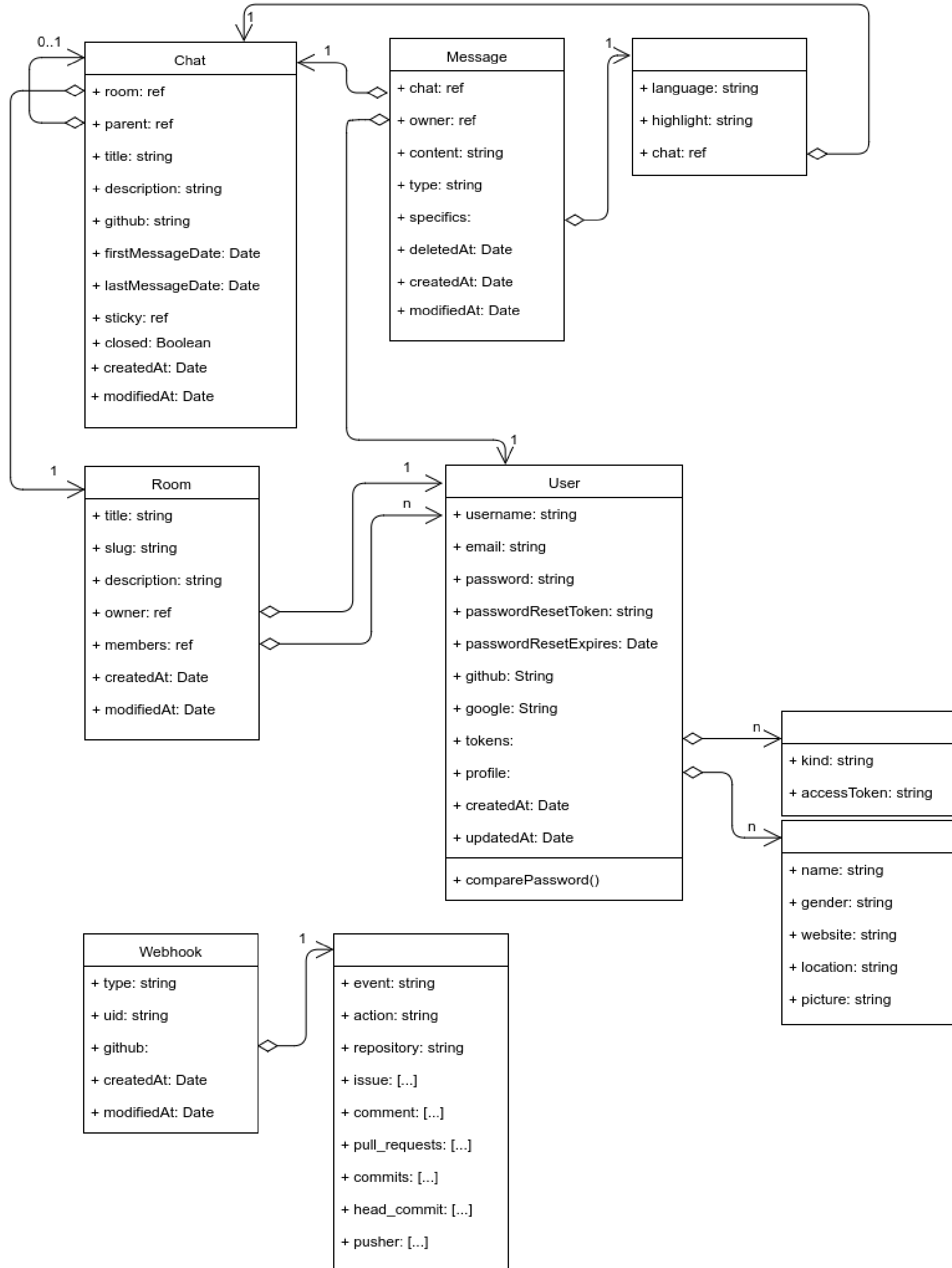


Figura 5.2 Structura bazei de date MongoDB

5.1.2. Rooms

Camerele sunt entități care pot fi folosite pentru clasificare sau pentru a stabili roluri în aplicație.

Câmpurile schemei:

- _id: identificator
- title: numele camerei
- slug: identificator pentru URL-ul camerei
- description: o descriere a camerei
- owner: _id-ul utilizatorului care e proprietar
- isPrivate: dacă o cameră e publică sau privată
- members: mulțime de _id utilizatori care sunt membri în acea cameră
- updatedAt: data modificării
- createdAt: data creării

Stocăm o referință la membrii unei camere, deoarece nu așteptăm mai mult de câteva sute de utilizatori pe cameră și nu sunt, de asemenea, o entitate clară în sine și spațiul pe disc pe care îl ocupă aceste referințe nu este semnificativ.

Atunci când proiectăm colecții MongoDB, întotdeauna trebuie să ținem cont de faptul că dimensiunea maximă pe document este de 4MB (16MB în cele mai recente versiuni[43]).

Celălalt câmp pe care îl stocăm prin referință este proprietarul camerei. Motivul pentru care nu încorporăm utilizatorul, în acest caz, nu este dimensiunea, ci mai degrabă faptul că datele profilului de utilizator ar putea fi actualizate în mod frecvent, ceea ce ar însemna necesitatea de a actualiza toate camerele pe care le deține, și nu doar User-ul corespunzător.

Camerele sunt indexate pe: _id, slug, proprietar și membri. La început, ne-am gândit că _id și slug ar fi suficiente deoarece acoperă cele mai frecvente căutări: utilizatorii care se referă la un chat prin identificatorul său sau introducând printr-o adresă URL directă (caz în care căutarea ar fi făcută prin adresa URL a slugului). Cu toate acestea, mai târziu am realizat că utilizatorii ar putea dori adesea să caute chat-uri pe care le dețin sau sunt membri, motiv pentru care am creat doi indecși suplimentari pentru a acoperi proprietarul și membrii.

5.1.3. Chats

Chat-urile aparțin unei camere, și sunt locul în care se transmit efectiv mesajele între utilizatori.

Câmpurile schemei:

- _id: identificator
- room: identificatorul camerei căreia chatul îi aparține
- title: numele chatului
- description: o scurtă descriere
- github: numele repository-ului GitHub, luat în considerare când se creează un chat conectat la un repository

[43] https://docs.mongodb.com/manual/reference/limits/

- firstMessageAt: data primului mesaj trimis. Câmpul este utilizat pentru a determina dacă utilizatorul a afișat deja toate mesajele din chat.
- lastMessageAt: data ultimului mesaj trimis.
- updatedAt: data ultimei modificări
- createdAt: data creării

Indexăm Chats pe: _id, and room. _id este folosit oricând cineva dorește să intre într-un chat, în timp ce room facilitează căutarea mai rapidă a chat-urilor dintr-o cameră.

5.1.4. Messages

Putem rezuma nevoile mesajelor noastre după cum urmează:

- Abilitatea de a stoca sute de mesaje pe oră.
- Abilitatea de a prelua mii de mesaje pe oră.
- Abilitatea de a prelua mesajele în ordine cronologică (mai întâi cele mai recente).

Anticipăm mai multe operații de citire decât de salvare, datorită faptului că unii dintre colegii de discuții pot cere mesajele mai recente de mai multe ori și, în timp ce un mesaj este stocat doar o singură dată, este posibil ca mai mulți membri să îl citească de mai multe ori. Astfel, am vrut să proiectăm o schemă de colectare care favorizează citirile, față de scrieri.

Mai mult, nu am vrea niciodată să preluăm toate mesajele odată. Nu numai că le-ar fi imposibil ca utilizatorul să le citească pe toate, dar, de asemenea, nu am putea să manipulăm încărcarea dacă am efectua o cerere pentru un chat cu multe mesaje.

Câmpurile schemei:

- _id: identificator
- chat: identificatorul chatului de care mesajul aparține
- owner: identificatorul utilizatorului care a trimis mesajul
- content: conținutul text
- type: tipul conținutului
    - language: tipul limbajului, dacă mesajul e de tip cod
    - highlight: linii care să fie subliniate, dacă mesajul e de tip cod
    - chat: referință către Chat-ul părinte, dacă mesajul e de tip fork
- deletedAt: data ștergerii. Conținutul va fi șters, însă utilizatorii vor putea vedea că un mesaj a fost trimis la acel moment.
- updatedAt: data ultimei modificări
- createdAt: data creării

Această structură a mesajelor, deși relativ simplă, a fost testată și e funcțională pentru până la 1.000.000 de conexiuni concurente.

Am indexat mesajele cu _id și chat + createAt. Primul index ajută la căutarea unui mesaj specific, în timp ce cel de-al doilea index compus ajută la căutarea mesajelor anterioare. Am compus data creării cu chatul pentru a filtra numai mesajele care aparțin unui anumit chat, deoarece nu vom fi niciodată interesați în amesteca mesajele din chat-uri.

5.1.5. Webhooks

În timpul implementării aplicației, am ajuns la punctul în care trebuia să stocăm date de la GitHub WebHooks, care sunt necesare pentru conectarea cu un repository GitHub și primirea informațiilor despre operațiile efectuate.

GitHub WebHooks trimite doar informațiile o singură dată pentru un repository, așa că a trebuit să ne asigurăm că datele au fost stocate astfel încât orice chat conectat la acel depozit GitHub să poată accesa actualizările GitHub. De asemenea, informațiile trebuiau să poată fi preluate și stocate rapid, deoarece actualizările GitHub sunt numeroase și am vrut să notificăm utilizatorii chatului în timp real [10].

Acest caz seamănă cu cel al mesajelor, deși de data aceasta ne-am ocupa de mesajele trimise de roboți.

Câmpurile schemei:

- type: tipul webhook-ului, în cazul în care avem mai mult de un provider
- uid: identificator unic al mesajului webhook
- github: câmpuri specifice GitHub, cum ar fi numele repository-ului și acțiunea
- updatedAt: data ultimei modificări
- createdAt: data creării

Am indexat webhook-urile pe _id și _id + github.repository, similar cu mesajele. Cu toate acestea, va trebui să creăm mai mulți indecși dacă avem mai mulți furnizori, deoarece _id + github.repository se potrivește doar pentru cele GitHub.

5.2. Structura back-end-ului

Partea de back-end a aplicației se bazează pe Node.js, combinat cu Express, care împreună pun la dispoziția clientului diverse date. Deoarece clientul nu face parte din back-end, pentru transmiterea datelor acestea trebuie serializate, de preferință într-un format larg răspândit.

În mod tradițional, XML ar fi fost calea de urmat, dar am ales ca JSON să fie formatul nostru de transport de date din următoarele motive:

- Avem de-a face cu JavaScript tot timpul, iar structurile obiectului JavaScript sunt foarte asemănătoare cu JSON și pot fi citite și transformate fără efort.
- JSON câștigă din ce în ce mai multă popularitate în contextul web.
- Express poate lucra by default cu JSON.
- Companiile precum Oracle recomandă utilizarea JSON față de XML[44].

Așa cum menționăm în capitolul despre tehnologii, am decis să folosim framework-ul Express, astfel încât tot nucleul nostru este construit în jurul acestuia. Express este responsabil pentru gestionarea sesiunilor de utilizatori, schimbul de informații cu bazele noastre de date și prelucrarea cererilor HTTP (parsarea datelor cum ar fi x-www-formurlencoded în obiecte JavaScript, executarea anumitor controlere atunci când un route este găsit și returnarea unei valori).

Express este capabil să pornească de la sine un server web (cu toată logica în el), ceea ce a fost suficient inițial, însă, mai apoi, am transferat această responsabilitate unei

[44] https://blogs.oracle.com/xmlorb/entry/ analysis_of_json_use_cases

biblioteci third-party (http), care ne-a permis să executăm atât API-ul AJAX, cât și WebSockets pe același port.

Folosim Socket.io pentru a lucra cu WebSockets, iar Socket.io are acces la datele de sesiune și se execută cu biblioteca http (deja implementată în Express). În cele din urmă, pachetul http pornește atât Express, cât și Socket.io pe un singur port, ceea ce este avantajos atunci când se evită restricțiile CORS[45], fie în curs de dezvoltare, fie în producție.

Structura de directoare pentru server a fost gândită așa încât să fie scalabilă, și în stadiile incipiente arăta astfel:

```
|-- bin
|-- config
|-- data
|-- data-session
|-- docs
|-- node_modules
‘-- src
    |-- controllers
    |-- helpers
    |-- middleware
    |-- models
    |-- routes
    ‘-- sockets
```

În versiunile ulterioare a crescut într-o oarecare măsură, dar, din fericire, schimbările au rămas minime.

bin: fișiere batch sau alte executabile

config: fișiere de configurare ale utilizatorilor, care pot rămâne neschimbate, cum ar fi setările bazei de date sau cheile aplicației

data și **data-session**: unde MongoDB și Redis își vor stoca datele

node_modules: modulele npm

src: conține toate logica aplicațiilor

src/controllers: request handlers

src/helpers: utilitare care pot fi utilizate oriunde în aplicație, care vor fi utilizate în general de mai multe ori

src/middleware: framework-ul Express al Node.js poate folosi middleware pentru a administra cererile, înainte de a ajunge la un controler

src/models: modele de baze de **src/sockets**: handler de conectare la socket (inclusiv event senders și listeners)

[45] CORS: Cross-Origin Resource Sharing https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

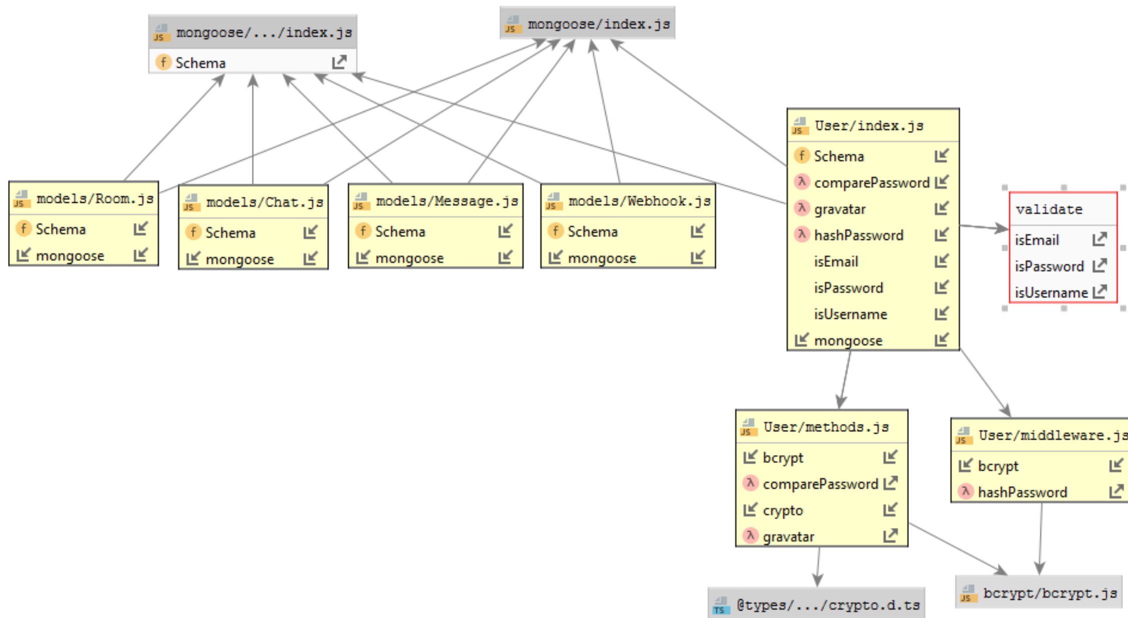


Figura 5.3 Dependențe între modele în back-end

5.3. Structura front-end-ului

Partea de front-end a aplicației a fost implementată folosind React, însă React nu oferă toate funcționalitățile necesare implementării complete, așa încât am creat un ecosistem de dependențe, care au în centru React.

Structura inițială a directoarelor proiectului, pe partea de front-end:

|-- dist

|-- src

   |-- components

   |-- containers

   |-- helpers

   |-- redux

|-- redux

         |-- middleware

         '-- modules

|-- routes

|-- styles

'-- tests

'-- webpack

dist: fișierele destinate distribuției, generate în cea mai mare parte de pachetul de module webpack. Acesta conține un index.html, care va fi punctul de pornire pentru oricine accesează site-ul nostru și pachetele JavaScript care conțin logica aplicației (inclusiv bibliotecile externe necesare).

src: conține toate logica aplicației.

src/components: componente care nu interacționează cu Redux.

src/containers: conține componente conectate, componente care interacționează cu spațiul de stocare Redux.

src/helpers: cod general JavaScript care nu se potrivește în altă parte (utilitare). În prezent, stocarea propriilor implementări de Promise bazate pe AJAX Fetching și WebSockets Client, URLPreviewer și trimitere fișiere.

src/redux: fișierele Redux.

src/redux/middleware: enhancement-uri pentru Redux, cum ar fi preluarea directă de la API.

src/redux/modules: reducer-ele Redux, unul pentru fiecare subiect distinct de pe site.

src/routes: rute de aplicare legate de componentele lor corespunzătoare.

src/styles: conține stilurile globale ale aplicației și variabile SCSS.

src/tests: configurarea testelor globale.

webpack: fișiere de configurare Webpack, care conțin cel puțin 2 fișiere de configurare: dev și prod. **Dev** generează fișiere bundle.js reîncărcate, abia optimizate, dar care se construiesc foarte repede, iar **Prod** generează pachete minimizate, pentru a se încadra într-un mediu de producție.

Funcționalitățile care au implementarea preponderent pe partea de front end sunt cele care țin de afișarea mesajelor:

- **preview pentru URL-uri în mesaje**: a fost realizat folosind un plugin[46] React modificat, care cere câteva informații de bază de pe site-ul al cărui URL a fost trimis în chat (titlul, descrierea, favicon-ul, adresele sursă ale imaginilor)
- **lipirea unui mesaj** în partea de sus a conversației (pin-to-top): mesajul care este marcat ca sticky este reținut în documentul chat-ului, și am creat o nouă componentă React, ScrollContainer, care ia o parte din responsabilitățile de afișare a istoricului chat-ului. Mesajele sticly sunt comunicate în timp real prin WebSockets. Structura componentelor React pentru o cameră de chat arată astfel:

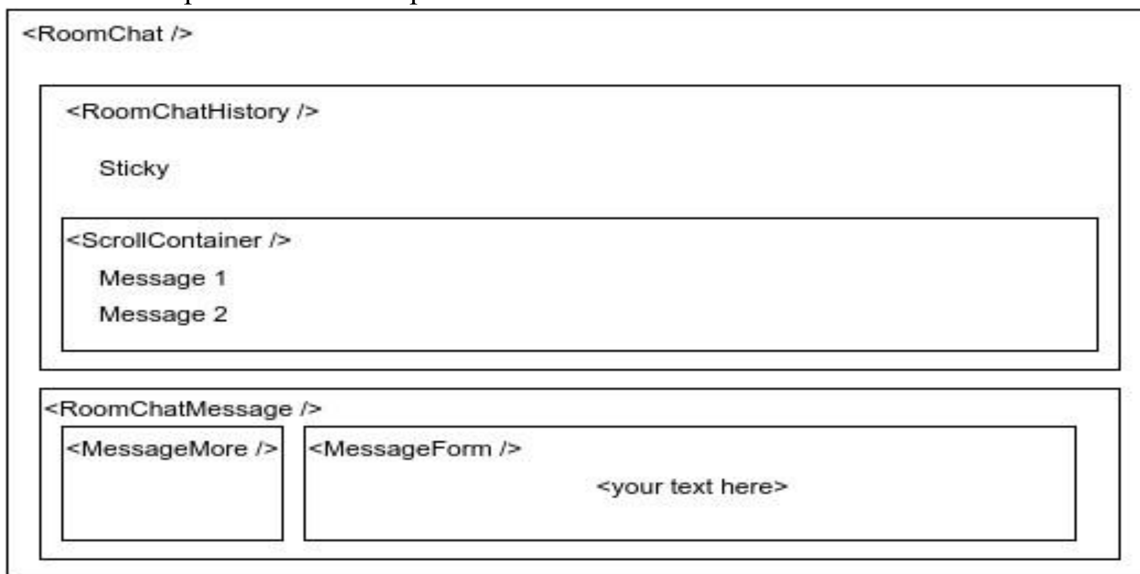


Figura 5.4 Componente React pentru funcționalitatea Sticky Messages

[46] https://www.npmjs.com/package/react-tiny-link

- **trimiterea de fragmente de cod:** pentru formatarea și sublinierea anumitor linii am folosit plugin-ul PrismJS, păstrând formatul folosit de plugin, așa încât conversia necesară să fie minimă
- **trimiterea de fișiere (imagini și raw):** pentru încărcarea fișierelor am folosit un serviciu cloud, așa încât să nu stocăm local fișierele. Componenta **cloudinary** pune la dispoziție un serviciu de stocare imagini și fișiere raw cu posibilitatea de editare a fișierelor imagine. În aplicația noastră, fișierele sunt încărcate în cloudinary, iar afișareaîn chat se face folosind plugin-ul de URL preview, cu o configurare specifică pentru imagini.

5.4. Detalierea funcționalităților

Funcționalitățile aplicației ChatDev au fost implementate în ordinea priorității, care a fost stabilită bazat pe cât de legată era de conceptul de chat colaborativ. Am pus de asemenea accent pe funcționalitățile de care depindea flow-ul central al aplicației (accesul utilizatorilor, creare de camere și chat-uri). Am ales câteva funcționalități complexe, care implică atât partea de front end, cât și cea de back end, pentru a le detalia implementarea.

5.4.1. Autentificarea

Pe partea de server, aceasta implică crearea a câtorva rute noi pentru a gestiona conectarea, înregistrarea (numai pentru autentificarea locală) și deconectarea, precum și strategiile adecvate pentru gestionarea fiecăruia dintre acești furnizori.

Pentru autentificarea locală am configurat următoarele două rute:

- /auth/signup
- /auth/signin

Acestea se ocupă de înregistrarea și de datele formularului de autentificare, respectiv. Pentru autentificările OAuth, avem rutele:

/auth/github
/auth/github/callback
/auth/google
/auth/google/callback

Sunt necesare două endpoint-uri pentru fiecare autentificare OAuth. Primul este cel de cerere, care va aduce utilizatorul pe pagina de autentificare a furnizorului, iar callback-ul este adresa URL de retur la care furnizorul va întoarce utilizatorul, după ce a terminat autentificarea, împreună cu token-urile de autentificare.

Este de remarcat că, din moment ce gestionăm toată serverul de autentificare, chiar și callback-urile, nu returnăm JSON pe niciuna dintre rutele OAuth. Ca excepție, facem apel la redirecționări către paginile clientului atât în momentul în care a reușit autentificarea, cât și în momentul erorii (fie din cauza unei probleme din partea noastră, fie din cauza refuzului utilizatorului de a ne acorda permisiuni pe pagina furnizorului).

Am folosit cookie-uri pe client pentru a informa serverul despre utilizatorul care s-a autentificat.

Aplicația noastră de client ar avea nevoie, de asemenea, de un anumit endpoint în care să se verifice dacă tokenul memorat curent a fost valabil și să cunoască utilizatorul

care se află în spatele acestuia pentru a afișa utilizatorul curent conectat și pentru a activa anumite caracteristici disponibile doar utilizatorilor. Am folosit utilizatori /whoami pentru asta.

Pentru logica de autentificare, am folosit Passport în cea mai mare parte. Passport ascunde toată complexitatea din spatele OAuth și OAuth2 într-un API generic care funcționează în același mod pentru sute de furnizori. De asemenea, am folosit-o pentru a ne ocupa de autentificarea noastră locală, chiar dacă nu era strict necesară, doar din motive de coerență.

Considerăm că conectarea contului este foarte importantă pentru a îmbunătăți experiența utilizatorului pe site-ul nostru, dat fiind că folosim cheia API furnizor pentru a colecta informații suplimentare în timp. Din acest motiv, încercăm să conectăm contul de utilizator cu cât mai mulți furnizori posibil, dacă aceasta este o posibilitate. În prezent, un utilizator care și-a creat contul prin intermediul formularului web poate ajunge la conturile Google și GitHub conectate.

Validarea datelor nu e foarte strictă: e-mailul local trebuie să fie o adresă de e-mail validă, iar parola trebuie să aibă cel puțin 4 caractere.

Deoarece Redux stochează date în memorie, va fi gol de fiecare dată când un utilizator intră pe pagina noastră. Pentru a prelua din nou toate datele de la utilizator, vom folosi ruta API users/whoami pentru a reîncărca spațiul nostru de stocare Redux odată ce aplicația a pornit.

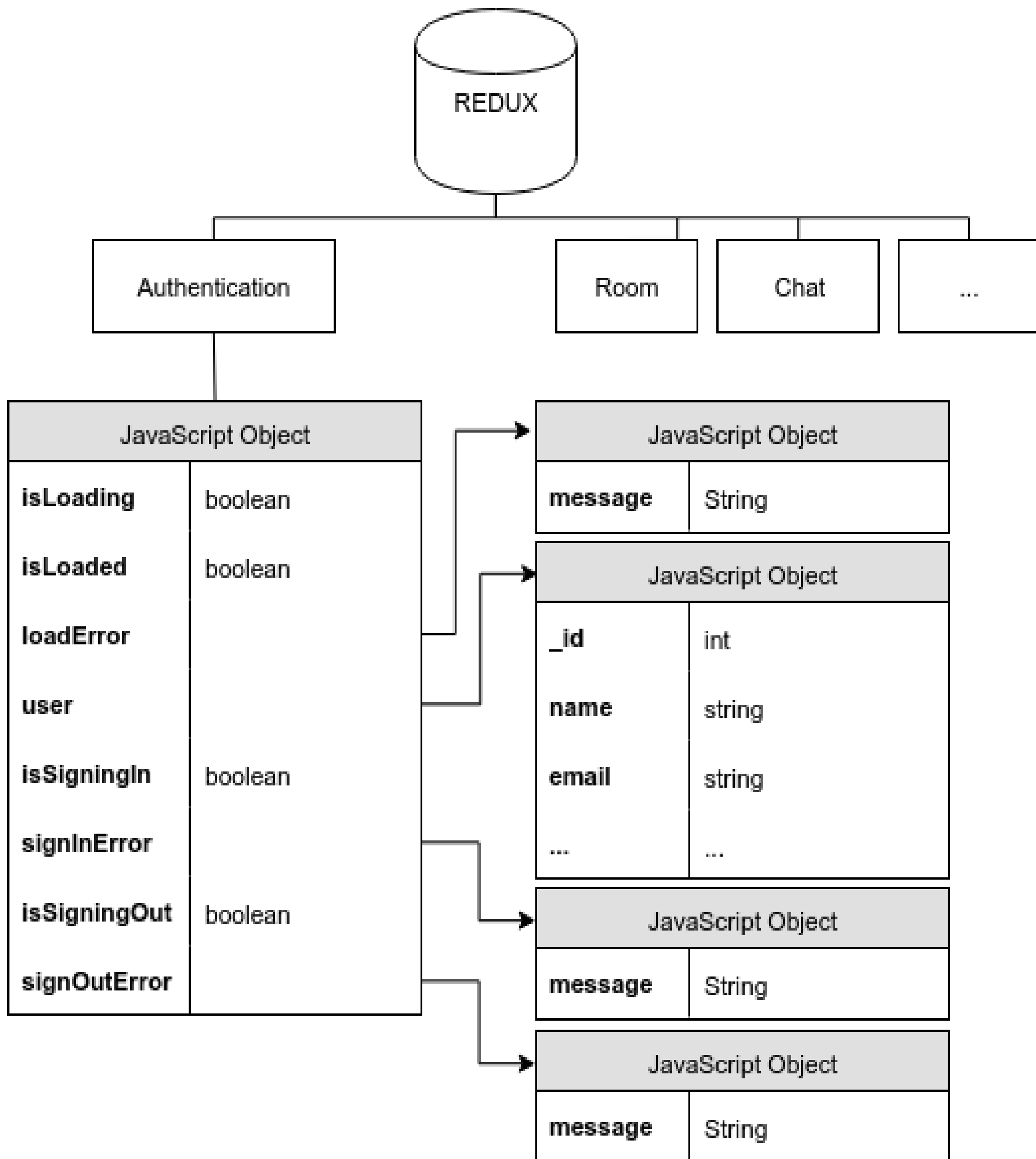


Figura 5.5 Diagrama modulului de autentificare Redux

5.4.2. Operații pentru camere, chat-uri și mesaje

Pentru a face querying în mod eficient, am abstractizat cele mai frecvente operațiuni, în special cele de validare (pe care le vom folosi ulterior): verificare dacă utilizatorul este conectat, cine e owner, camerele, chat-urile și mesajele existente etc.

Endpoint-urile aplicației pentru fiecare ramură sunt următoarele:

5.4.2.1. Camere

Tabel 5.1 Rute pentru camere

Metoda	Route	Descriere
GET	/rooms	Lista camerelor
POST	/rooms	Creează o cameră nouă
GET	/rooms/search	Filtrare camere folosind parametri opționali de query: _id, slug sau title.
GET	/rooms/_id	Vezi o cameră cu un anumit identificator.
PATCH	/rooms/:_id	Modifică o cameră existentă cu parametri care sunt valizi
POST	/rooms/:_id/join	Alătura-te unei anumite camere identificată prin id. Este necesar un utilizator autentificat.
POST	/rooms/:_id/leave	Părăsește o cameră identificată prin id. Este necesar un utilizator autentificat.
DELETE	/rooms/:_id	Șterge o anumită cameră, identificată prin id. E necesar ca utilizatorul autentificat să fie owner-ul camerei.
GET	/rooms/:_id/chats	Lista de chaturi dintr-o anumită cameră.
POST	/rooms/:_id/chats	Creează un nou chat în camera identificată prin id. E necesar ca utilizatorul autentificat să fie owner-ul camerei

5.4.2.2. Chat-uri

Tabel 5.2 Rute pentru chaturi

Metoda	Route	Descriere
GET	/chats/:_id	Vezi un anumit chat, identificat printr-un id.
PATCH	/chats/:_id	Modifică un anumit chat, identificat printr-un id, cu câmpuri valide.
DELETE	/chats/:_id	Șterge un anumit chat, identificat printr-un id.
POST	/chats/:_id/fork	Fă fork la un chat.
POST	/chats/:_id/fork-merge	Lipește un chat care fusese forked la chatul original.
POST	/chats/:_id/fork-upgrade	Mută fork-ul unui chat..
GET	/chats/:_id/messages	Vezi mesajele dintr-un chat identificat printr-un id.
POST	/chats/:_id/messages	Creează un mesaj într-un chat identificat printr-un id.

5.4.2.3. Mesaje

Tabel 5.3 Rute pentru mesaje

Metoda	Route	Descriere
PATCH	/messages/:_id	Modifică conținutul unui mesaj identificat prin id. E necesar ca utilizatorul autentificat să fie owner-ul mesajului
DELETE	/messages/:_id	Șterge un mesaj identificat prin id. E necesar ca utilizatorul autentificat să fie owner-ul mesajului

5.4.3. Mesagerie

Pentru implementarea componentei de mesagerie am folosit **WebSockets**, care fac posibilă menținerea unei conexiuni în timp real, și oferă latență redusă și conexiune bidirecțională între client și server. Conexiunea persistentă bidirecțională este ceea ce face posibilă recepționarea mesajelor serverului în orice moment. Se evită, de asemenea, trimiterea oricăror antete HTTP redundante, care pot face o diferență semnificativă în utilizarea lățimii de bandă a aplicației.

Pe partea de ***server***, **Socket.io**, biblioteca WebSockets de nivel înalt pe care o folosim, ascultă mereu noi conexiuni. Când folosim WebSockets, considerăm că utilizatorul va avea deja un cont pe site și că va fi deja conectat. În caz contrar, conexiunea la socket este respinsă.

Pentru mesaje, am declarat următoarele evenimente:

Tabel 5.4 Lista evenimentelor declanșate pentru mesaje

Event	**Data**	**Descriere**
Disconnect		Închide conexiunea socket.
EnterRoom	Slug-ul camerei	Cerere de a se alătura unei camere, care va abona utilizatorul la noile mesaje primite.
SendMessage	Obiect care conține chatId and conținutul mesajului.	Trimite noul mesaj către chatul dat.
ReceiveMessage	Obiectul Message	Când un mesaj este trimis într-un chat dintr-un cameră, el este emis către toți membrii acelei camere.

Un lucru de remarcat aici este că, spre deosebire de AJAX, WebSockets nu așteaptă un răspuns, ori de câte ori trimite date. Utilizarea WebSockets-ului brut ar fi un dezavantaj mare, clientul nu știe niciodată dacă vreuna dintre solicitările sale a eșuat din cauza validării de la server (de exemplu, atunci când a fost trimis un corp de mesaj gol). Din fericire, Socket.io oferă așa numitele " acknowledgments", un tip de răspunsuri asemănătoare AJAX, prin care serverul poate returna orice tip de date evenimentelor.

Vom folosi Socket.io și pentru partea de ***client***, în acest caz, implementarea clientului (Socket.io-client).

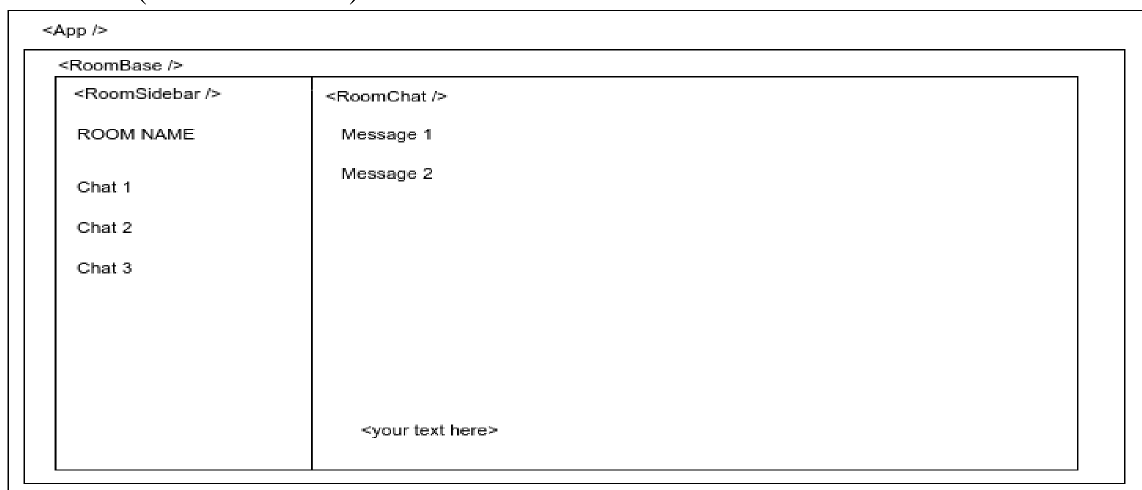


Figura 5.6 Componente React într-o cameră (scenariul de succes)

Pentru stocarea stărilor Socket.io, am folosit soluția middleware. Avantajele acestei metode sunt că acțiunile Redux nu trebuie să știe despre biblioteca WebSockets și despre configurația specifică pe care o execută. Tot ceea ce au de expediat este conținutul pe care doresc să-l citească sau să scrie de pe server, iar middleware-ul va face restul.

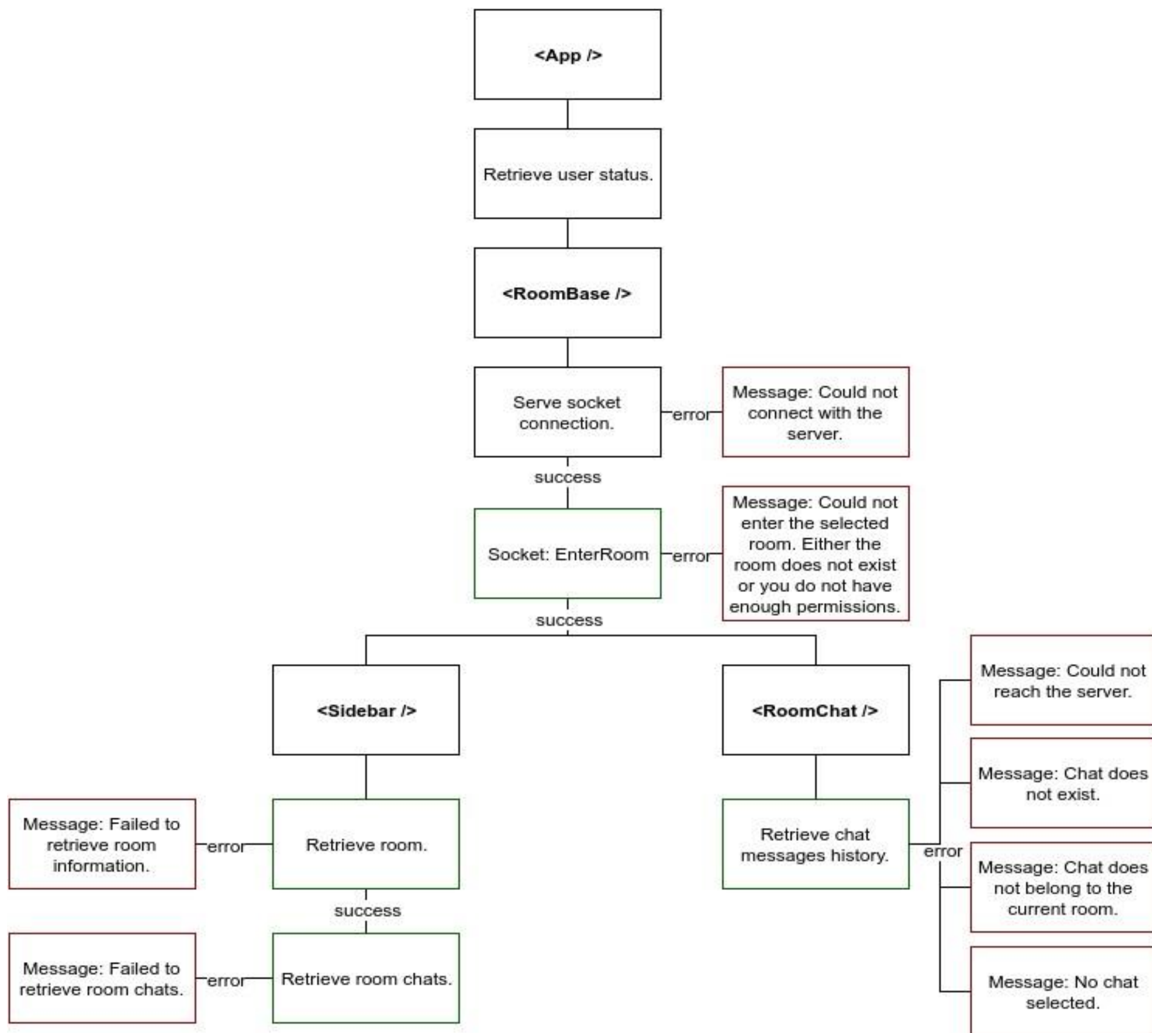


Figura 5.7 Diagrama de activitate React pentru o cameră

5.4.4. Conectarea cu GitHub

Funcționalitatea inițială a fost să obținem activitatea utilizatorului în cadrul unui depozit GitHub în timp real, cel puțin push-uri și pull requests. Apoi, vom afișa aceste informații ca notificări ale GitHub pe un chat care este legat de acel repository. Acest lucru evită necesitatea de a porni GitHub numai pentru a verifica actualizările și, de asemenea, permite o anumită interacțiune cu acestea.

Endpoint-urile GitHub pentru commit-uri, pull requests și issues:

- /repos/:owner/:repo/commits
- /repos/:owner/:repo/pulls
- /repos/:owner/:repo/issues

GitHub WebHooks permite abonarea la anumite evenimente care se întâmplă într-un repository sau organizație. În calitate de dezvoltatori, trebuie doar să avem o rutăpe care să ascultăm apariția unor actualizări și ne vor trimite toate noile date în timp real.

Endpoint-ul la care se trimit request-urile către WebHooks:

- /webhooks/github

GitHub ne poate trimite o gamă largă de date/evenimente, pe care, la început, este puțin probabil să o putem procesa în mod corespunzător. Deoarece GitHub numește fiecare tip de solicitare, putem filtra unul pe care nu îl acceptăm. În prezent, acceptăm următoarele evenimente: issue_comment, issues, pull_request și push.

În afară de abonare, oferim și un endpoint pentru API pentru ca clientul să poată prelua toată activitatea pe care a raportat-o serverului nostru API într-un depozit GitHub:

- /webhooks/github/:repositoryUser/:repositoryName

5.4.5. Vizualizare GitHub în split-screen

Funcționalitatea care diferențiază cel mai mult aplicația noastră de alte chat-uri colaborative este posibilitatea de a vedea un repository GitHub în aceeași pagină cu chat-ul care s-a conectat la acel repository. Analizând desfășurarea unei discuții din cadrul unei ședințe (SCRUM), dar mai ales în cadrul efectuării unui code-review, putem constata că o parte semnificativă a timpului este alocată găsirii porțiunilor de cod semnificative de către toți participanții la discuție, apoi de transmitere (prin copy-paste, screen-shoturi) a informațiilor care necesită modificări, și apoi transformarea discuției în task-uri de workflow management. Toate aceste operațiuni necesită folosirea unor instrumente multiple, în programe și ferestre diferite, și se irosește mult timp pe operațiuni care ar putea fi mult eficientizate dacă ar putea fi efectuate într-o singură fereastră, într-o singură aplicație.

În urma acestei analize, câteva funcționalități s-au evidențiat în mod special prin utilitate:

- Posibilitatea de a vedea în același ecran și chat-ul de discuție și repository-ul conect
- Selectarea de fragmente de cod și trimiterea lor direct în chat, cu un singur click, păstrându-se formatarea
- Posibilitatea de a face screenshot pe chat, la care să poată fi adăugate adnotări și care să poată fi trimis direct în chat
- Crearea de ancore în pagina repository-ului care să poată fi trimise ca link-uri în chat
- Posibilitatea de a crea un task în urma discuției, direct din chat, folosind secvențele de cod, screenshoturile, anchorele trimise

Pentru a implementa aceste funcții în aplicație, trebuia să putem încorpora o pagină GitHub în pagina unui chat. GitHub nu permite încorporarea paginilor sale, răspunzând cu header-ul `X-Frame-Options: deny`, care blochează posibilitatea de embed. Până când aplicația va deveni publică și vom obține acordul GitHub pentru încorporarea paginilor, a fost necesar să găsim o altă soluție.

Componenta **X-Frame-Bypass** este un Web Component, mai specific un Customized Built-in Element, care extinde un IFrame ca să treacă de headerul de răspuns *X-Frame-Options:*

deny/sameorigin. În mod normal, aceste headere previn încorporaerea unei pagini web în elementul <iframe>, dar X-Frame-Bypass folosește un proxy CORS pentru a permite asta[47].

Am integrat această componentă în partea de front end a aplicației noastre și am putut astfel să încorporăm pagini GitHub în chat-ul nostru.

După ce frame-ul repository-ului a fost încorporat, am adăugat peste acest cadru butoanele pentru funcționalitățile menționate anterior:

- La selectare text din frame-ul GitHub este activat butonul de copiere text în chat
- Buton pentru a face un screenshot cu întregul frame sau o porțiune din acest și a trimite fotografia în chat
- Buton pentru a face o ancoră/legătură cu o anumită zonă a paginii care să fie trimis în chat
- Buton de creare a unui task, care deschide un formular de creare task, în care pot fi copiate informațiile din chat

Frame-ul repository-ului poate fi ascuns, afișat sau redimensionat în funcție de necesitatea apărută în timpul discuției.

[47] https://github.com/niutech/x-frame-bypass

Capitolul 6. Testare și Validare

Testarea codului sursă, cunoscută sub numele de testare dinamică, este o tehnică de testare a software-ului analizând comportamentul dinamic al codului. Prin testarea codului sursă micșorăm posibilitatea unor erori după ce modificăm părți din acesta, chiar dacă acestea nu sunt direct legate de modificările pe care le-am comis recent.

Testele sunt o condiție prealabilă pentru refactorizarea în condiții de siguranță, deoarece aceasta adesea implică modificarea multor secvențe de cod, folosite în multe părți ale aplicației. Dependențele, parametrii și chiar implementările funcțiilor s-ar putea schimba, ceea ce ar putea duce la nefuncționarea aplicației așa cum ar trebui, chiar dacă aplicația se compilează fără erori.

Există zeci de tipuri de test pentru a ne asigura că aplicația funcționează pe deplin conform așteptărilor. Ne vom concentra pe testele Unitary, Integration și E2E, care sunt cele mai frecvente și care s-au dovedit a fi suficiente pentru majoritatea proiectelor. O distribuție de teste eficientă se consideră a fi de 70% unit, 20% integration și 10% end-to-end[48].

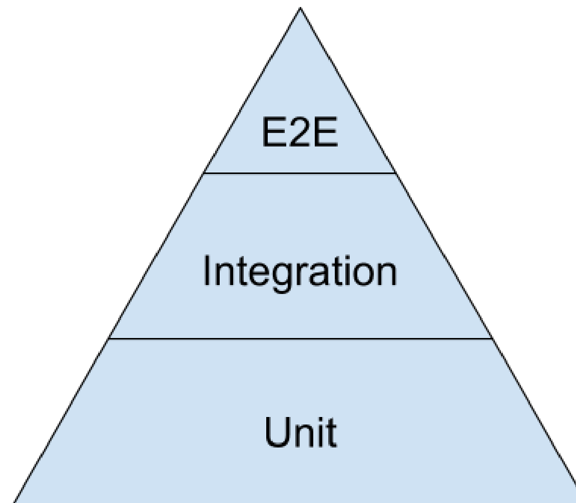


Figura 6.1 Echilibrul între teste unit, integration și end-to-end, conform Google

6.1. Unit testing

Testele din front end validează că diferitele componente care fac parte din client lucrează conform așteptărilor; astfel încât să funcționeze bine pentru utilizatori când API-ul rulează.

O aplicație React-Redux-Router este configurată de mai multe componente, dar ne-am concentrat testele asupra Redux: reducers, acțiuni, handlers și views.

Reducers

Redux, stocarea în memorie pe partea de client, este un reducer. Aceasta generează o stare nouă după procesarea unui obiect recunoscut. Ce am testat aici este că, dat fiind un anumit obiect pentru reducer-ul Redux, noua stare generată (și returnată) este cea pe care o așteptam.

[48] https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

În unele cazuri, este în interesul nostru ca starea inițială a reducer-ului să fie *empty*, în altele am dorit ca reducer-ul să aibă o anumită stare pentru a testa cazurile de *edge*.

Acțiuni

Acțiunile generează obiecte noi pentru procesarea reducer-ului, care generează noi stări Redux. Layer-ul de acțiuni între controllere și Redux este folosit pentru a masca intrările Redux în funcții simple.

Unit testing pentru acțiuni a însemnat crearea unui stub pentru dispatcher-ele care trimiteau cererea către reducer. Am putut apoi să ne asigurăm că, având în vedere anumiți parametri setați în acțiune, se generează obiectul așteptat pentru reducer.

În plus, am fi putut stabili teste de integration pentru a ne asigura că, având în vedere anumiți parametri funcția de acțiune, starea finală a reducer-ului ar fi corectă.

Handlers

View-urile randate (pagina HTML pe care utilizatorul îl vede pe ecran) au mai multe tipuri de handlere, de exemplu handler-ul de click.

Enzyme, biblioteca de testare JavaScript pentru React, poate simula acțiunile utilizatorilor prin reproducerea conținutului React. Apoi putem crea spies sau stubs cu funcția handler pentru a ne asigura că este apelată.

Uneori, este de asemenea util să testăm și logica handler-ului.

Views

Views ajung să fie paginile HTML pe care le vede utilizatorul pe ecran. Deși testarea interfeței poate părea uneori trivială, în unele cazuri este necesară, cum ar fi atunci când anumite conținuturi ar trebui afișate doar cu o anumită stare (e.g.: să se afişeze butonul "Sign in" atunci când utilizatorul nu este conectat). Enzyme ne poate ajuta și cu astfel de teste.

Testele pentru back end constau în asigurarea faptului că partea de business logic gestionează, stochează și returnează corect răspunsurile JSON către utilizator (prin AJAX sau WebSockets). Arhitectura noastră expresă este împărțită în 4 părți: modele, servicii, controllere și router. Fiecare dintre ele, cu excepția router-ului care era prea simplu, are un set propriu de unit teste. În timpul testelor noastre, am creat spies, stubs și mocks, pe care le-am explicat în timpul analizei diferitelor teste.

Modele

Modelele Mongoose constituie nucleul aplicației noastre. Ele definesc structura MongoDB și impun tipurile de date pe care le va suporta. Datorită faptului că Mongoose ne furnizează acest strat deasupra bazei de date şi îl face complet izolat de implementarea bazei de date, modelele de testare a unităţilor au fost simplificate.

O schemă Mongoose (care este definiția unui Model) poate fi împărțită în 4 părți: tipuri de date, validatori de date, middleware (care pot fi atașate înainte sau după salvarea/actualizarea datelor) și metode.

Următoarele reprezintă un fragment al unit testului pentru validarea numelui de utilizator:

```
import { expect } from 'chai';
import { isUsername, isPassword } from '../validate';
describe('Model: User (validate)', () => { it('should be invalid if username length
is not between 5-20', () => {
        expect(isUsername('x')).to.be.false;
        expect(isUsername('x'.repeat(21))).to.be.false;
        expect(isUsername('x'.repeat(5))).to.be.true;
});
```

Servicii

Serviciile folosesc deseori datele modelelor, și putem folosi stubs pentru a furniza un răspuns predictibil la baza de date, ceea ce face testele deplin unitare.

Controllere

Controllerele procesează inputul routerului și apelează serviciile corespunzătoare, făcând transformările corespunzătoare la acea intrare.

Unit testele sunt similare celor de la servicii, cu excepția faptului că în acest caz, stub-urile vor fi făcute pe servicii, nu pe modele.

Router

Structura MVC a aplicației a permis crearea unui router simplu, iar testarea s-a făcut doar pe cele două handlere ale routerului.

Rutele noastre de router au arătat astfel:

```
router.get('/users/whoami', c(user.whoami, req => [req.user]));
```

6.2. Integration

Testarea integrării (uneori numită integrare și testare, abreviată I & T) este faza de testare software în care modulele software individuale sunt îmbinate și testate ca un grup. Testarea integrării este efectuată pentru a evalua conformitatea unui sistem sau a unei componente cu cerințe funcționale specificate. Apare după testarea unității și înainte de testarea validării. Testarea de integrare ia module de intrare care au fost testate unitar, le grupează în agregate mai mari, aplică testele definite într-un plan de testare a integrării acestor agregate și oferă ca ieșire sistemul integrat gata pentru testarea sistemului.

Pentru a executa aceste teste, o instanță a serverului este pornită. Fiecare dintre aceste teste va trimite una sau mai multe cereri către acest server și va examina răspunsul, simulând comportamentul unui utilizator (sau al clientului nostru React). Pentru testele noastre AJAX am folosit SuperAgent, care este o bibliotecă care rulează pe platforma Node.js și se comportă la fel ca și browserul nativ Fetch.

6.3. E2E

End-to-end este o metodologie utilizată pentru a testa dacă fluxul unei aplicații funcționează așa cum a fost proiectat de la început până la sfârșit. Scopul efectuării testelor de tip end-to-end este identificarea dependențelor de sistem și asigurarea transmiterii informațiilor corecte între diferitele componente ale sistemului.

Endpoint-urile aplicației au fost testate folosind o combinație de teste de integration și E2E. Endpoint-urile sunt diferitele rute la care utilizatorul poate trimite cereri și așteaptă adesea un răspuns (fie o eroare sau un mesaj de succes cu date).

Instanței serverului de test i-am atașat o instanță falsă (mock) pentru MongoDB și Redis, care sunt utilizate de server pentru citirea / scrierea datelor.

Obiectul Mock este o implementare simplă a unui obiect "real". Ele sunt folosite atunci când nu suntem interesați de testarea implementării reale, ci doar de funcționalitate. Mocks oferă răspunsuri previzibile și exacte care evită să depindă de obiecte de la terțe părți, asupra cărora nu avem suficient control, care pot avea erori sau pot să nu returneze răspunsuri previzibile.

În cazul nostru, am folosit mocked storages în memorie pentru a ne asigura că bazele noastre de date reale nu au fost niciodată modificate și că au fost ușor de curățat după fiecare test.

Unul dintre testele noastre de autentificare E2E arată după cum urmează:

```
function signin(email = ’demo@example.com’, password = ’password’) {
       return new Promise((resolve, reject) => {
              request.post(‚${server}/auth/signin’)
                     .send({ email, password })
                     .end((err, res) => {
                            if (err) return reject(err);
                                   return resolve(res);
}); }); }
it(’should log in with the right credentials’, (done) => {
       chain.then(() => signin())
              .then(() => { request
              .get(‘${server}/users/whoami’)
              .end((err, res) => { expect(res.status).to.equal(200);
              done();
}); }); });
```

În unele cazuri, răspunsurile aplicației nu sunt suficiente pentru a determina că operația este executată în mod corespunzător și, prin urmare, trebuie să importăm anumite părți ale codului sursă pentru a verifica dacă datele sunt stocate așa cum ar trebui. De exemplu, pentru a verifica dacă timestamp-urile utilizatorului sunt stocate corect, trebuie să căutăm utilizatorul interogând modelul utilizatorului imediat după finalizarea solicitării AJAX.

Capitolul 7. Manual de Instalare si Utilizare

7.1. Instalarea aplicației

Aplicația a fost concepută pentru a fi un serviciu cloud pe care utilizatorii să îl poată accesa oricând fără să instaleze vreun software ei înșiși. Vom descrie totuși și procedura de instalare, întrucât aplicația a fost gândită să fie open-source, disponibilă oricărui dezvoltator software care ar vrea să aducă îmbunătățiri sau funcționalități noi.

Rularea back-end-ului

Cerințe inițiale:

- Node.js (v6+)
- MongoDB
- Redis

Împachetarea componentelor se face folosind Node și comanda

```
npm install
```

Pornirea MongoDB și Redis se poate face cu acest script: (dacă nu rulează deja)

```
scripts/startdb.sh
```

Pornirea back-end-ului pentru production:

```
npm run start-prod-backend
```

Pornirea back-end-ului pentru development:

```
npm run start-dev-backend
```

Rularea front-end-ului

Pornirea front-end-ului pentru production:

```
npm run start-prod-frontend
```

Pornirea front-end-ului pentru development:

```
npm run start-dev-frontend
```

Aplicația poate fi deployată și folosind **Docker**[49].

Docker este o platformă open-source pentru dezvoltatori și administratori de sistem pentru a construi, a expedia și a distribui aplicații.

Docker funcționează cu "containere" software. Un container poate stoca orice tip de software cu tot ce trebuie să ruleze: dependențe, configurații și alte instrumente.

Prin utilizarea unui container Docker, putem executa tot codul sursă cu setările de producție și bazele de date cu o singură comandă.

[49] https://www.docker.com/

Docker Compose[50] poate gestiona mai multe containere Docker. Fiecare dintre ele poate comunica între ele și are un set de configurație partajată, în timp ce dependențele runtime sunt izolate între ele.

Avantajul oferit de Docker Compose este faptul că multe proiecte open source, cum ar fi MongoDB sau Redis, oferă deja containerele Docker preconfigurate. Nu trebuie făcută nicio configurație personalizată pentru un proiect care este deja pe DockerHub, altul decât transmiterea valorilor de comandă corespunzătoare la pornirea lor.

Singurul software specific pe care proiectul îl necesită este o versiune recentă Node.js. Configurarea a constat în extinderea unui Docker Official Image de Node pe LTS, care să copieze toate fișierele sursă din containerul Docker și să instaleze dependențele NPM. Containerul web a expus portul 3000, portul pe care Node.js îl pornește în mod implicit, pe care mai apoi l-am legat la un proxy pe portul 80 folosind Docker Compose.

```
FROM node:boron
RUN mkdir -p /app
WORKDIR /app
COPY . /app
RUN npm install
EXPOSE 3000
CMD ["npm", "start"]
```

După configurarea Docker Compose, am integrat aplicația deployată cu repository-ul GitHub, așa încât în momentul încărcării unei noi versiuni pe GitHub aceasta să fie adusă și în Docker, care să repornească cu ultima versiune.

```
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
(cd "$DIR/.."
ssh $SSH_USERNAME@$SSH_HOSTNAME -o StrictHostKeyChecking=no <<-EOF
cd  $SSH_PROJECT_FOLDER  git  pull  docker-compose  pull  docker-compose
stop docker-compose rm -f
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d
EOF)
```

Configurări

Principalul fișier de configurare pentru aplicație se găsește la `config/default.js`.

Pentru a porni aplicația este necesar un fișier `config/production.js` sau `config/development.js`, pentru producție, respectiv dezvoltare, care să arate în modul următor:

[50] https://docs.docker.com/compose/

```
module.exports = {
  passport: {
    github: {
      clientID: 'Github client ID',
      clientSecret: 'GitHub secret key',
      callbackURL: 'http://mydomain:3000/api/auth/github/callback',
    },
    google: {
      clientID: 'Google client ID',
      clientSecret: 'Google secret key',
      callbackURL: 'http://mydomain:3000/api/auth/google/callback',
    }}}
```

7.2. Utilizarea aplicației

7.2.1. Pagina de start

Pagina de start oferă posibilitatea accesului la formularele de autentificare Sign In sau Sign Up.

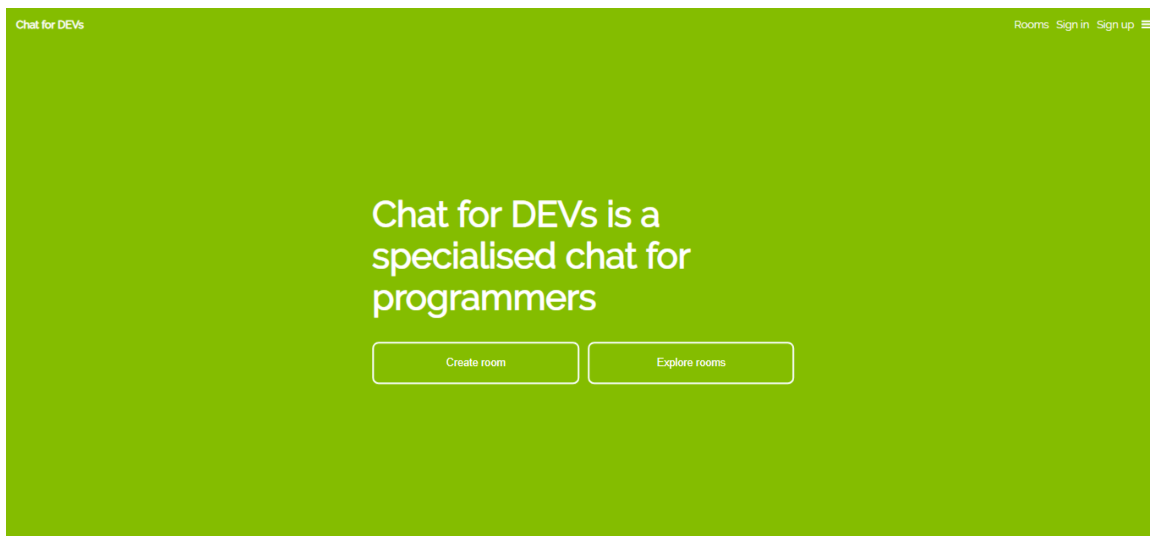


Figura 7.1 Pagina de start a aplicației ChatDev

7.2.2. Crearea contului și autentificarea

Aplicația permite crearea unui cont local, dar și conectarea folosind contul de Google sau GitHub.

În cazul în care utilizatorul care se autentifică are rol de administrator, acesta poate crea noi camere. Utilizatorul cu rol de moderator poate crea un chat într-o cameră.

Figura 7.2 Formular de autentificare

Figura 7.3 Formular de creare cont

7.2.3. Crearea unei camere

Utilizatorii autentificați pot să vadă lista camerelor existente și să se alăture unei camere la care au acces (publică).

Figura 7.4 Lista camerelor existente în aplicație

Utilizatorii care au rol de administrator pot crea o cameră nouă, care va fi apoi afișată în lista camerelor.

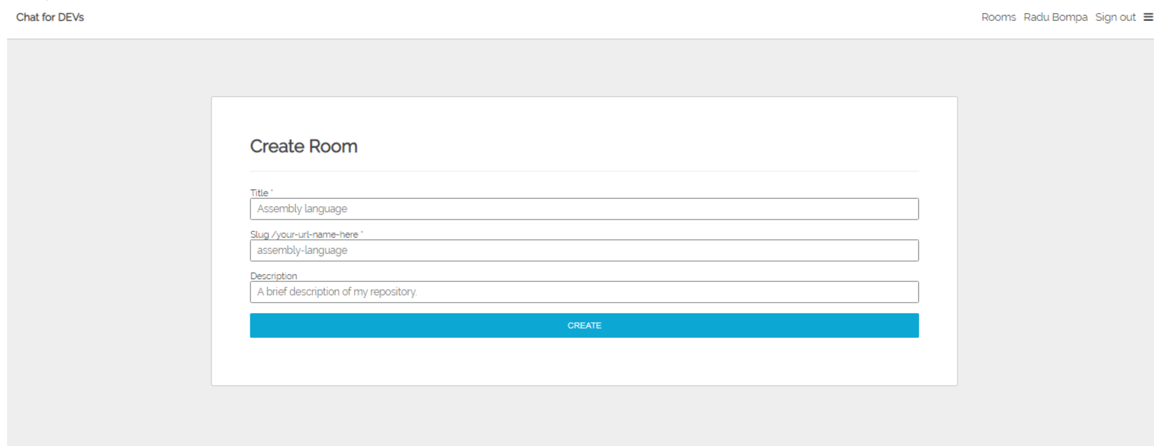


Figura 7.5 Formular de creare a unei camere

După ce o cameră a fost creată, utilizatorii să i se alăture, iar moderatorii pot crea chat-uri în această cameră.

7.2.4. Crearea unui chat

În cadrul unei camere existente, utilizatorii pot să vadă lista chat-urilor existente și se pot alătura discuției.

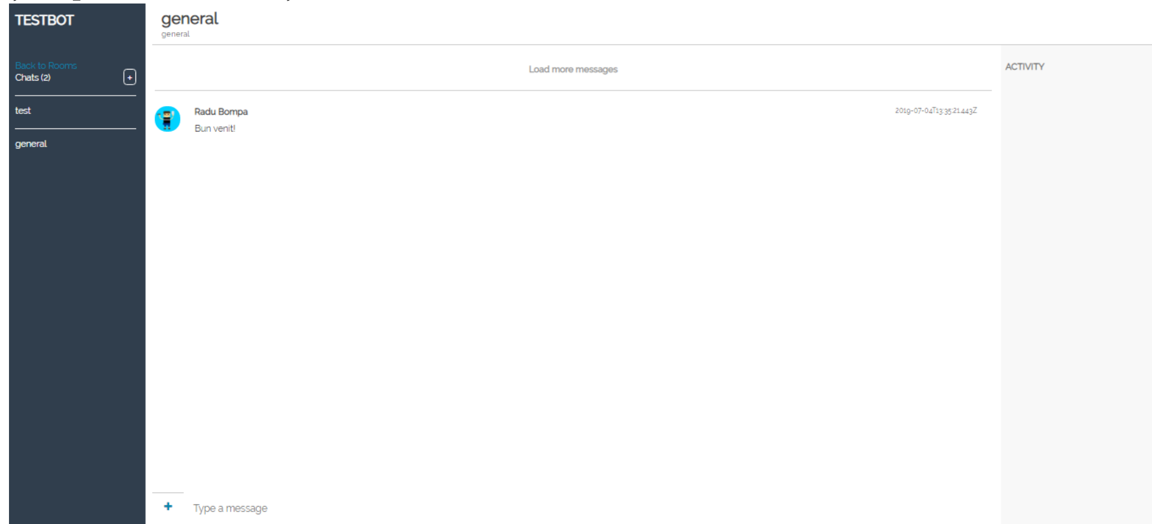


Figura 7.6 Pagina de vizualizare a unei camere, cu lista de chat-uri, spațiul de discuții și coloana de activitate GitHub

Utilizatorii care au rol de moderator pot crea o cameră nouă, apăsând butonul de (+) din sidebar-ul din stânga.

Create Chat

Title *

General

Description

A brief description of the chat

Github repository

octocat/Hello_world

CREATE

Figura 7.7 Formular de creare a unui chat

În câmpul GitHub repository, care este opțional, este ales repository-ul pe care vrem să îl conectăm la noul chat creat.

După crearea unui nou chat, utilizatorii se pot alătura discuției.

7.2.5. Trimiterea de fragmente de cod

Pe lângă mesajele de tip text, utilizatorii pot trimite într-un chat fragmente de cod, formatate specific limbajului în care e scris, și cu posibilitatea de a sublinia anumite linii.

Create Snippet

Language

JavaScript ▾

Code

```
1  export function createMessage(userId, chatId, values) {
2    const sanitizedContent = validator.trim(values.content);
3
4    if (validator.isEmpty(values.content)) {
5      throw 'Message cannot be empty.';
6    }
7
8    const newMessage = new Message();
9    newMessage.owner = userId;
10   newMessage.chat = chatId;
11   newMessage.content = sanitizedContent;
12   newMessage.type = values.type;
13   newMessage.specifics = values.specifics;
14
15   return newMessage.save();
16 }
```

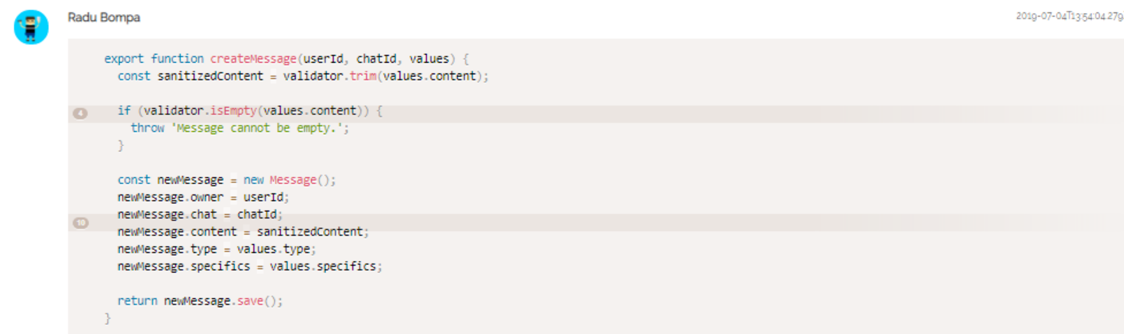
Highlights

4, 10

CREATE

Figura 7.8 Formular de trimitere a unui fragment de cod

După ce este apăsat butonul de Create, mesajul este transmis cu formatarea aleasă.



\+ Type a message

Figura 7.9 Afişarea în chat a unui fragment de cod formatat şi cu sublinieri

7.2.6. Trimiterea unui link

La trimiterea în chat a unui mesaj care conține un URL este afișată o căsuță de preview a URL-ului.

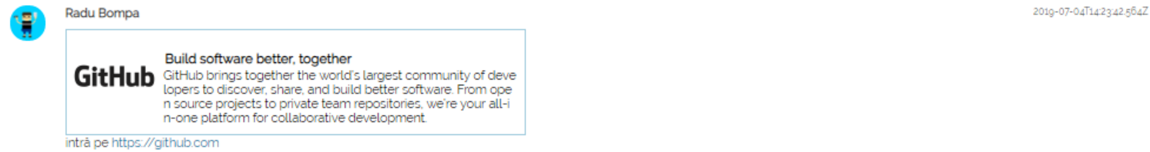


Figura 7.10 Afișarea unor date de preview pentru un link trimis în chat

7.2.7. Trimiterea unui fișier

Trimiterea unui fișier se face apăsând butonul (+) din stânga căsuței de scriere mesaje.

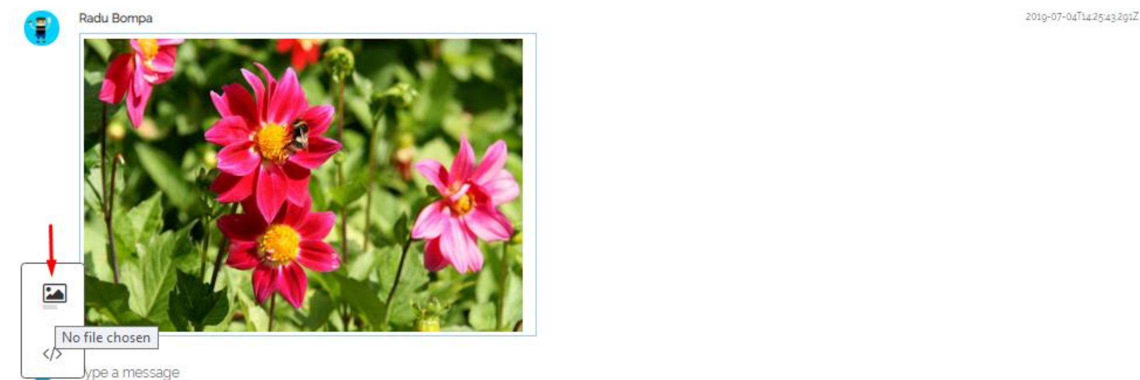


Figura 7.11 Trimiterea și afișarea în chat a unui fișier (imagine)

7.2.8. Comunicarea cu Chatbot-ul

Comunicarea cu chatbot-ul se face folosind eticheta @bot, urmată de interogarea adresată bot-ului.

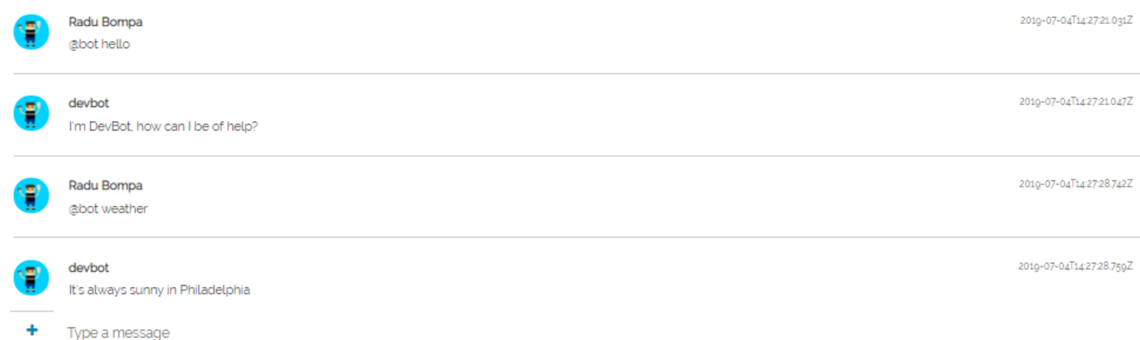


Figura 7.12 Comunicarea cu chatbot-ul aplicației

7.2.9. Mesaje sticky (pin-to-top)

Mesajele importante pot fi lipite în partea de sus a unei conversații, prin apăsarea pe butonul de pin-to-top.

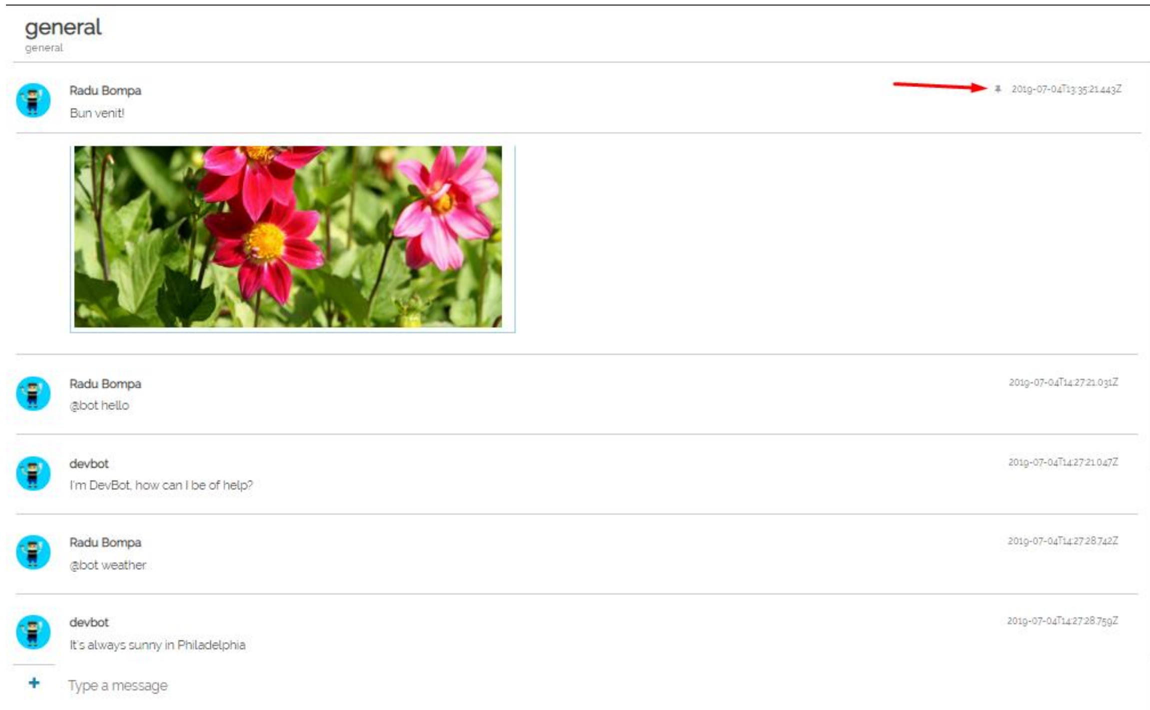


Figura 7.13 Lipirea unui mesaj important în partea de sus a conversației (pin-to-top)

7.2.10. Vizualizare activitate GitHub

Într-un chat care a fost conectat la un repository GitHub, în tab-ul de Activity poate fi vizualizată activitatea efectuată în repository.

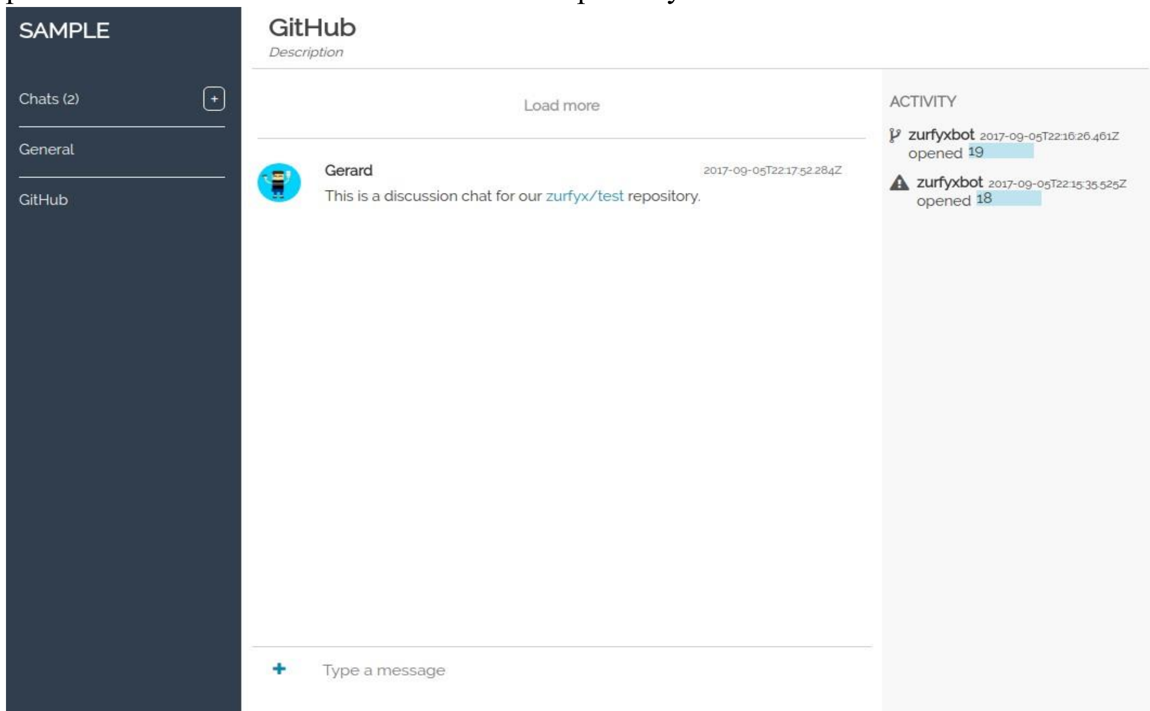


Figure 7.14 Vizualizarea activității din GitHub pentru un chat conectat la un repository

7.2.11. Vizualizare repository GitHub în split-screen

Un chat care a fost conectat la un repository oferă posibilitatea de a vedea pagina repository-ului în același ecran cu chat-ul, pentru a eficientiza discuția asupra conținutului repository-ului.

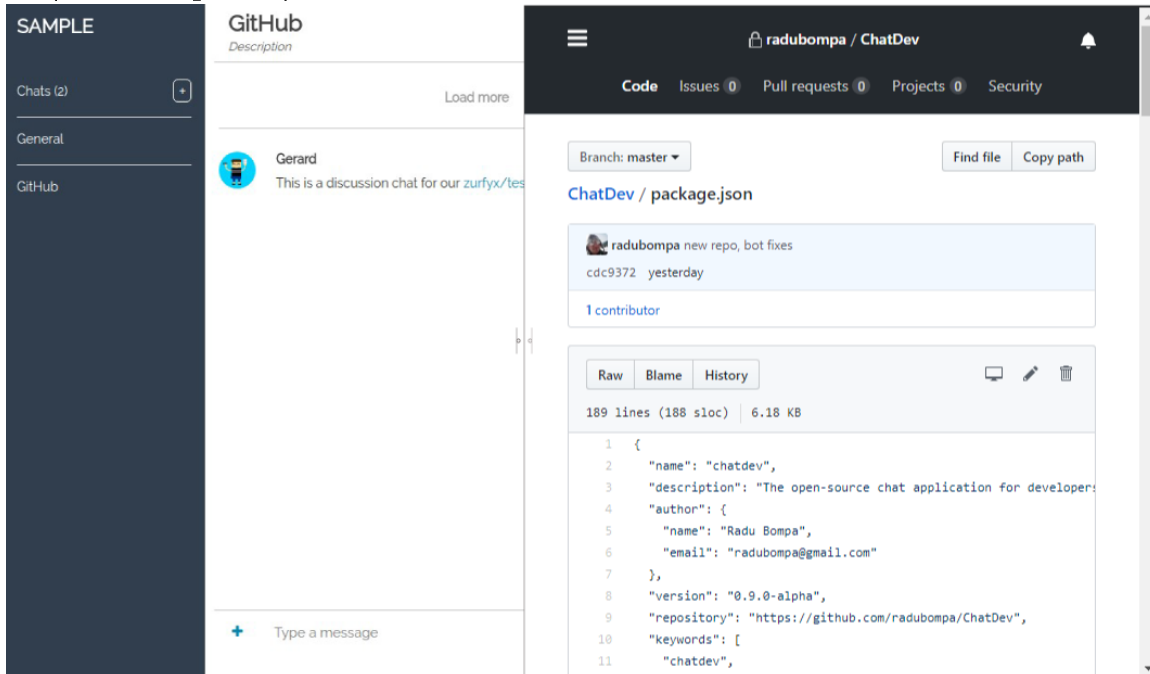


Figura 7.15 Vizualizarea în split-screen a repository-ului conectat la chat

Peste frame-ul care conține pagina GitHub se află butoane de interacțiune cu această pagină, copiere text formatat în chat, screenshot, ancoră, creare task.

Capitolul 8. Concluzii

8.1. Analiza rezultatelor obținute

Aplicația ChatDev este începutul implementării unei aplicații chat colaborativ cu tehnologii actuale și funcționalități utile dezvoltatorilor software. În versiunea actuală am implementat, chiar dacă pe alocuri doar demonstrativ, funcționalități care pot fi utile în cadrul unui spațiu de muncă al dezvoltatorilor software.

În implementare am integrat mult componente, Node.js, React, Express, MongoDB, Redis, Socket.io, și altele, iar rezultatul obținut are performanțe bune chiar și la o încărcare mare.

Platforma este ușor de înțeles și învățat de către utilizatori, deoarece denumirile, dar și imaginile folosite sunt sugestive. Navigarea în cadrul aplicației se face cu multă ușurință, tranziția dintre elemente având un curs logic.

Platforma este ușor de extins, datorită organizării pe layere, adăugarea unor componente noi, este facilitată datorită designului adecvat.

Utilizatorii care au testat aplicația au fost în medie încântați de aplicație. Funcționalitățile noi aduse au fost considerate utile, deși necesită multe optimizări.

8.2. Dezvoltări ulterioare

Aplicația este concepută pentru a fi dezvoltată, în funcție de nevoile unei echipe de dezvoltatori software sau de cerințele pieței. Câteva funcționalități trebuie însă implementate pentru ca aplicația să poată fi în mod real folosită într-o echipă de lucru

- **Audio și video conferințe**, o funcționalitate necesară pentru orice echipă ai cărei membri sunt distribuiți în diferite locație.
- Integrarea cu aplicații de **desen** și creare **scheme grafice** poate ușura partea de proiectare și analiză a proiectelor.
- Integrarea cu aplicații care oferă instrumente de **workflow management** (ex: Apache Taverna), pentru a putea crea, asigna task-uri și a administra evoluția implementării din chat-ul echipei. După această integrare, un chat va putea fi atașat unui task din workflow, cu ancore pentru anumite etape din rezolvarea task-ului
- **Screen sharing** – posibilitatea de a vedea ecranul în colaborare, combinată cu vizualizarea repository-ului GitHub în split-screen pot reprezenta un bonus mare de eficientizare a lucrului în echipă.

Bibliografie

[1] D. E. W. G. Blischak JD, „A Quick Introduction to Version Control with Git and GitHub,” *PLoS Comput Biol,* vol. 12(1), nr. e1004668, 2016.

[2] S. Sulyman, „Client-Server Model,” *IOSR Journal of Computer Engineering,* vol. 16, nr. 10.9790/0661-16195771, pp. 57-71, 2014.

[3] K. K. A. G. M. S. Panagiotakis, „Architecture for Real Time Communications over the Web,” *International Journal of Web Engineering,* vol. 2, nr. 1, pp. 1-8, 2013.

[4] „Node.js online documentation,” Node.js, 2015. [Interactiv]. Available: https://nodejs.org/en/docs/. [Accesat 2018].

[5] V. &. O. O. &. B. Solovei, „The difference between developing single page application and traditional web application based on mechatronics robot laboratory onaft application,” *Автоматизація технологічних і бізнес-процесів,* vol. 10, nr. 10.15673/atbp.v10i1.874, 2018.

[6] „React framework online documentation,” Facebook, 2017. [Interactiv]. Available: https://reactjs.org/docs/. [Accesat 2019].

[7] T. Ştefănuț, Proiectarea Interfețelor Utilizator - note de curs.

[8] M. Dînșoreanu, Proiectarea sistemelor - note de curs.

[9] „MongoDB online documentation,” MongoDB Inc., 2015. [Interactiv]. Available: https://docs.mongodb.com/. [Accesat 2018].

[10] „GitHub Webhooks Docs,” Microsoft, 2018. [Interactiv]. Available: https://developer.github.com/webhooks/. [Accesat 2019].

Anexa 1 – Lista figurilor

Figura 3.1 Comparația procentajelor utilizării aplicațiilor chat colaborative cele mai utilizate…..7
Figura 3.2 Aprecierea utilizatorilor despre aplicațiile chat de colaborare în echipă…………..…9
Figura 3.3. Exemplu de istoric de versiune al unui proiect…………………………………...11
Figura 4.1 Diagrama UML use-case-uri utilizator……………………………………………16
Figura 4.2 Diagrama UML use-case-uri manager de proiect…………………………………18
Figura 4.3 Diagrama UML use-case-uri moderator…………………………………………..18
Figura 4.4 Diagrama use-case pentru crearea unei camere de către un moderator………………21
Figura 4.5 Diagrama use-case pentru crearea unui chat într-o cameră de către un moderator, chat conectat la un repository GitHub…………………………………………………...23
Figura 4.6 Aplicație client-server care comunică prin HTTP…………………………………26
Figura 5.1 Arhitectura generală a aplicației……………………………………………...…34
Figura 5.2 Structura bazei de date MongoDB………………………………………………..36
Figura 5.3 Dependențe între modele în back-end……………………………………………40
Figura 5.4 Componente React pentru funcționalitatea Sticky Messages……………………..42
Figura 5.5. Diagrama modulului de autentificare Redux……………………………………45
Figura 5.6 Componente React într-o cameră (scenariul de succes)…………………………..47
Figura 5.7 Diagrama de activitate React pentru o cameră……………………………………48
Figura 6.1 Echilibrul între teste unit, integration și end-to-end, conform Google…………....51
Figura 7.1 Pagina de start a aplicației ChatDev………………………………………………57
Figura 7.2 Formular de autentificare…………………………………………………………58
Figura 7.3 Formular de creare cont..…………………………………………………………58
Figura 7.4 Lista camerelor existente în aplicație..……………………………………………58
Figura 7.5 Formular de creare a unei camere...………………………………………………58
Figura 7.6 Pagina de vizualizare a unei camere, cu lista de chat-uri, spațiul de discuții și coloana de activitate GitHub……….………………………………………………………...59
Figura 7.7 Formular de creare a unui chat...…………………………………………………59
Figura 7.8 Formular de trimitere a unui fragment de cod……………………………………60
Figura 7.9 Afișarea în chat a unui fragment de cod formatat și cu sublinieri.…………………..60
Figura 7.10 Afișarea unor date de preview pentru un link trimis în chat………………………61
Figura 7.11 Trimiterea și afișarea în chat a unui fișier (imagine)………………………………61
Figura 7.12 Comunicarea cu chatbot-ul aplicației……………………………………………61
Figura 7.13 Lipirea unui mesaj important în partea de sus a conversației (pin-to-top)…………..………………………………………………………………………62
Figura 7.14 Vizualizarea activității din GitHub pentru un chat conectat la un repository……..62
Figura 7.15 Vizualizarea în split-screen a repository-ului conectat la chat……………………63

Anexa 2 – Lista tabelelor

Tabel 3.1 Comparația celor mai utilizate aplicații comerciale de chat colaborativ……………..8
Tabel 4.1 Scenarii de utilizare pentru utilizatorul obișnuit……………………………………..17
Tabel 4.2 Scenarii de utilizare pentru utilizatorul manager de proiect…………………………18
Tabel 4.3 Scenarii de utilizare pentru utilizatorul moderator…………………………………….19
Tabel 4.4 Scenarii de utilizare pentru utilizatorul developer……………………………………19
Tabel 4.5 Scenarii adăugate pentru utilizatorul obișnuit………………………………………….19
Tabel 5.1 Rute pentru camere……………………………………………………………………..46
Tabel 5.2 Rute pentru chaturi……………………………………………………………………..46
Tabel 5.3 Rute pentru mesaje……………………………………………………………………..46
Tabel 5.4 Lista evenimentelor declanșate pentru mesaje..……………………………………..47

Anexa 3 – Glosar de termeni

Termen	Definiție
SPA	Single Page Application – o aplicație web în care se aduc dinamic datele, în timp ce utilizatorul navighează pe site, evitând să se reîmprospăteze întreaga pagină ori de câte ori utilizatorul completează un formular sau navighează în altă parte a site-ului
Webhooks (GitHub)	O modalitate de a primi notificări pe un site extern când anumite acțiuni sunt efectuate pe un repository GitHub.
Sticky messages (pin-to-top)	Mesaje importante care sunt atașate (în partea de sus a) unei conversații pentru a fi mereu vizibile.
Snippet	Fragment de cod

Anexa 4 - Estimarea timpului necesar implementării

Pe baza scenariilor inițiale ale utilizatorilor, am scris următoarea cronologie de implementare. Am construit-o luând în considerare prioritățile de backlog și fiecare dintre timpii de dezvoltare ale caracteristicilor, pentru a face o aproximare a ceea ce s-ar putea face într-un semestru.

Săptămâna	Task-uri
0-4	Raportul inițial cu obiectivele aplicației, planificarea timpului și alte date de început. Crearea șablonului aplicației, care va constitui baza implementării ulterioare.
5	Înregistrare și autentificare utilizatori.
6	Implementarea camerelor de chat publice, la care orice utilizator se poate alătura.
7	Dezvoltarea unui chat de grup de bază într-o cameră, în care utilizatorii pot comunica. Va asigura următoarele funcționalități: mesagerie instant și comunicarea de grup.
8	Notificări pe desktop și pe site, status utilizatori, trimitere de emoji, MarkDown și formatare HTML limitată.
9	Implementarea rolurilor de cameră.
10	Posibilitatea de a lipi un mesaj deasupra conversației, subliniere cod, editare cod, formatare în funcție de limbajul în care e scris.
11	Documentație. Revizuirea implementării de până acum, logica și design-ul, și reluarea unor aspecte care au fost amânate anterior.
11	Chat dedicat unui repository Github: commits, requests, pulls. Split screen cu pagina de repository.
12	Bifurcarea discuțiilor. Utilizatorii pot bifurca un chat bazat pe un comentariu, issue, commit, pull request. După încheiere, chat-ul poate fi merged.
13	Reputație bazată pe activitatea utilizatorilor (în chat și pe GitHub), cum ar fi numărul de commit-uri, push requests.
14	Camere private și invitația administratorilor de a te alătura unei camere.
15	Deployment-ul serverului, al bazei de date și aplicației client pe un server în cloud.
16	Documentație.
17	Documentație.