MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Sistem pentru managementul defectelor software
Bug Tracker

LUCRARE DE LICENȚĂ

Absolvent: **Alexandra-Claudia DRAGOȘ**

Coordonator ştiinţific: **Asis. Dr. ing. Cosmina - Daniela IVAN**

2020

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Prenumele NUMELE**

TITLUL LUCRĂRII DE LICENŢĂ

1. **Enunţul temei:** *Aplicație propusă să contribuie la optimizarea procesului de gestiune al defectelor software din cadrul diverselor proiecte*

2. **Conţinutul lucrării:** *Pagina de prezentare, Introducere-Contextul proiectului, Obiectivele Proiectului, Studiu Bibliografic, Analiză și fundamentare teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie, Anexa 1 – Lista figurilor și a tabelelor, Anexa 2 – Glosar de termeni*
3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**:

5. **Data emiterii temei:** 1 noiembrie 2019

6. **Data predării:** 8 iulie 2020

Absolvent: _____

Coordonator ştiinţific: _____

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)_____
_____,
legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării
_____
_____
_____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

_____

Nume, Prenume

_____

Semnătura

Cuprins

Capitolul 1. Introducere – Contextul proiectului

1.1. Contextul proiectului

"*Dacă depanarea este procesul de eliminare a defectelor, atunci programarea trebuie să fie procesul de introducere ale acestora*." (Edsger W. Dijkstra)

Schimbarea reprezintă un proces indispensabil și constant în dezvoltarea produselor software. Un sistem sau elementele componente ale acestuia sunt modernizate, fixate sau înlocuite pentru a oferi servicii mai bune sau o gamă de servicii mai vastă, pentru minimizarea cheltuielilor sau pentru a mări ușurința utilizării sistemului. Deși schimbările aduse proiectelor reprezintă un factor benefic, dacă acestea nu sunt controlate pot să afecteze calitatea sau în situații mai grave să ducă la încheierea contractelor pentru un proiect.

Pentru a se asigura că proiectele în curs de modificare nu se abat de la cerințele clientului, specialiștii în IT care se ocupă cu dezvoltarea produselor software în cauză au nevoie de o modalitate eficientă de gestiune a modificărilor. O soluție pentru acestă problemă este utilizarea unui instrument de urmărire a defectelor unei aplicații (*Bug Tracker*). Mai multe detalii legate de conceptul de *Bug Tracker* sunt prezentate în secțiunea 3.1.1.

Tehnologia și implicit aplicațiile software sunt elemente indispensabile ale societății. Sistemele software au reușit să se răspândească în numeroase domenii: medicină (ex. aplicații pentru vizualizare a informațiilor unui pacient, aplicații care oferă sfaturi pentru menținerea unui stil de viață sănătos și multe altele), bancar (ex. aplicații pentru realizarea tranzacțiilor online), comercial (există o multitudine de aplicații pentru a efectua cumpărături online, de la articole vestimentare și până la vehicule), divertisment (aplicații pentru vizualizarea și descărcarea filemelor și serialelor) ș.a.m.d. Deoarece aceste instrumente software sunt atât de răspândite și utilizate, funcționarea lor defectuoasă afectează un număr ridicat de utilizatori. Toate problemele unui sistem trebuie tratate de către persoanele care oferă mentenanța produselor software.

Conform unor studii realizate de F. Brooks [12], un developer petrece aproximativ 50% din timp realizând depanare și testare. Așadar, acest proces de rezolvare a problemelor implică mult timp și sume mari de bani pe care companiile trebuie să le aloce. Spre exemplu, în anul 2017, conform datelor oferite de tricendis.com, mai bine de 3.5 miliarde de oameni au avut de suferit din cauza unur sisteme care au eșuat, iar companiile au pierdut mai bine de 1,7 trilioane de dolari[1].

Astfel, marile firme doresc să utilizeze o platformă unde pot sa gestioneze și să vizualizeze mai usor și mai rapid probleme care apar fie din cauza numărului mare de utilizatori conectați, a unor cazuri omise la testare sau a diferitelor lucruri neprevăzute care apar și care modifică stabilitatea sistemului.

[1] https://www.tricentis.com/resources/software-fail-watch-5th-edition/

1.2. Motivația

Luând în considerare faptul că procesul de depanare și mentenanță a programelor scrise de developeri este o etapă obligatorie în dezvoltarea aplicațiilor de orice tip, care, mai mult implică numeroase resurse și timp, atrage necesitatea utilizării de instrumente potrivite pentru gestiunea și vizualizarea informațiilor legate de bug-uri.

Aplicațiile deja existente la momentul actual, deși au multe de oferit, pot să fie destul de greu de utilizat, nefiind intuitive sau pur și simplu au o interfață mult prea încărcată sau neatractivă în comparație cu așteptările celor care le utilizează.

De asemenea, în cele mai multe cazuri, platformele existente nu țin cont de rolul utilizatorului și de informațiile pe care aceștia trebuie să le poată vizualiza când vine vorba de interfață.

Motivul pentru care am optat această tematică a fost acela de a integra diverse funcționalități considerate ca fiind absolut necesare și utile într-un ansamblu menit să ofere o aplicație prietenoasă care să vină în ajutorul procesului de management al bug-urilor din diversele proiecte ale unei companii. Datorită experienței mele dobândite în dezvoltarea software, am constatat cât de util este un astfel de sistem de trasare a bug-urilor. Mai mult, am constatat și nevoia de astfel de sisteme gratuite care să vină în sprijinul firmelor mici și medii care sunt la început de drum și care nu au posibilitatea financiară de a utiliza un produs de gestiune a bug-urilor comercial.

Astfel, proiectul propus dorește să ofere o aplicație cu design modern destinată companiilor mici și mijlocii, gratuită care să conțină elementele de bază ale unui sistem de gestiune a bug-urilor.

1.3. Conținutul lucrării

În cele ce urmează va fi realizată o scurtă descriere a capitolelor conținute în această lucrare.

Capitolul 1 – Introducere – În cadrul acestui capitol se decrie contextul aplicației, precum și motivația realizării aplicației de management al bug-urilor.

Capitolul 2 – Obiectivele Proiectului – Acest capitol are scopul de a prezenta tema propriu-zisă a lucrării, precum și obiectivele principale și secundare ale aplicației implementate.

Capitolul 3 – Studiu Bibliografic – Scopul acestui capitol este acela de a furniza informații despre conceptele utilizate în cadrul unui instrument de managemnt al bug-urilor, cât și prezentarea câtorva aplicații deja existente din acest domeniu.

Capitolul 4 – Analiza și Fundamantare Teoretică – În cadrul acestui capitol sunt descrise cerințele sistemului, tehnologiile folosite la implementare și cazurile de utilizare pentru fiecare tip de utilizator.

Capitolul 5 – Proiectare de Detaliu și Implementare – Scopul acestui capitol este acela de a prezenta arhitectura sistemului, precum și modul în care a fost dezvoltat sistemul.

Capitolul 6 – Testare, Validare și Evaluare – Acest capitol are rolul de a prezenta metodele de testare și validare aplicate sistemului implementat, precum și părerea utilizatorilor care au testat platforma.

Capitolul 7 – Manual de Instalare și Utilizare – În acest capitol sunt descriși pașii de instalare și modul de utilizare a produsului final.

Capitolul 8 – Concluzii – Acest capitol conține concluziile bazate pe rezultatele obținute și prezentarea posibileleor dezvoltări ulterioare.

Capitolul 2. Obiectivele Proiectului

În acest capitol se prezintă tema aleasă, realizându-se totodată și descrierea obiectivului principal și a obiectivelor secundare ale proiectului.

2.1. Obiectivele principale

Aplicația pentru gestiuea bug-urilor este menită să ușureze procesul depanării proiectelor prin menținerea informațiilor necesare legate de evoluția și statusul bug-urilor. Un *bug* reprezintă un defect dintr-un program sau sistem software care determină respectivul program să returneze rezultate neașteptate sau incorecte. O explicație mai riguroasă a conceptului de *bug* se regăsește în secțiunea 3.1.1.

Cu ajutorul acestui sistem, utilizatorii au o reprezentare vizuală a muncii lor depuse în etapa depanării, fiindu-le mult mai ușor să se concentreze efectiv pe rezolvarea task-urilor de fixare a bug-urilor(*bugFixing*). Astfel, toți membrii echipei care utilizează această aplicație vor fi la zi cu starea produsului la care lucrează.

2.2. Obiectivele secundare

La ora actuală, pe piață se regăsesc numeroase sisteme asemănătoare cu cel implementat, sisteme care dispun de o mulțime de operații și acțiuni. Aplicația descrisă pe parcursul acestei lucrări dorește să ofere o interfață accesibilă, prietenoasă care să simplifice contactul cu utilizatorii. Acestă aplicație este destinată proiectelor de dimeniuni medii și mici, fiind un instrument de ajutor în activitățile de dezvoltare software ale oricărei companii din mediul IT.

Platforma realizată poate fi folosită de următoarele tipuri de utilizatori: Administrator, Project Manager, Developer și Tester. Acești utilizatori pot să execute diverse acțiuni în funcție de permisiunile asignate rolurilor pe care le dețin:

- un utilizator se poate loga în aplicație și poate accesa paginile aferente drepturilor sale
- Orice tip de utilizator poate să vizualizeze profilul său și se efectueze modificări asupra conținutului acestuia
- Un utilizator poate să selecteze limba în care să fie afișat conținutul aplicației
- Un utilizator are posibilitatea de a vizualiza un istoric al activităților sale desfășurate în cadrul aplicației
- Posibilitatea ca un **administrator**
    - să execute operații de gestiune a utiliazatorilor
    - să adauge diverse permisiuni utilizatorilor existenți în cadrul platformei
- Un **Project Manager**
    - să execute operații de gestiune a proiectelor
    - să adauge/eliminie membrii unui proiect
    - să vizualizeze rapoarte legate de statusul bug-urilor, numărul acestora și severitatea lor

- **Tester**
  - Este singurul care poate modifica statusul unui bug în *CLOSED(*Rezolvat*)*
  - Poate să vizualizeze rapoarte legate de statusul bug-urilor, numărul acestora și severitatea lor
- Orice tip de utilizator implicat în procesul de dezvoltare(**Project Manager, Tester, Developer**) poate să execute următoarele acțiuni:
  - Să vizualizeze bug-urile
  - Să creeze noi bug-uri și să le asigneze unui membru din echipă
  - Să exporte informații unui bug
  - Să editeze informațiile bug-urilor
  - Să adauge atașamente relevante bug-urilor
  - Să adauge comentarii unui bug

Capitolul 3. Studiu Bibliografic

Acest capitol are rolul de a prezenta sisteme similare existente deja pe piață care abordează tematica propusă și care vor fi analizate pentru a le indentifica punctele tari si slabe.

Aceste date au fost colectate cu scopul de a identifica caracteristicile pe care aplicația descrisă în continuarea acestei lucrări să le conțină.

3.1. Introducere în tematica problemei

În cadrul acestei secțiuni vor fi definite și descrise conceptele de: *bug*(defect software), *bug status*, *bug report*(raportul defectelor software) și relațiile dintre aceste elemente.

3.1.1. Necesitatea sistemelor moderne de tracking a bug-urilor

În lucrările [5,10] un bug software(pe scurt “bug”) este definit ca și o eroare, greșeala, defect, caracteristic unui program sau sistem care duce la funcționarea anormală, și uneori terminarea bruscă a programului. Aceste greșeli reprezintă un inconvenientt atât pentru utilizatori, cât și pentru developeri. Majoritatea erorilor apar datorită neatenției sau a lipsei de experință a oamenilor, fie in codul sursă, fie în designul proiectelor. Acest lucru implică alocarea de resurse și timp suplimentar pentru a remedia aceste erori.

Activitățile de fixare a bug-urilor sunt acele procese în care programatorii încep rezolvarea problemelor care au fost descoperite în urma testării, utilizând un status al bug-urilor pentru a se menține o evidență a activiăților desfășurate de programatori, precum și evoluția acestora.[1]

Un bug status este definit cel mai ușor ca fiind o reflectare a stării problemelor identificate într-un program. Tranzițiile între stările posibile pentru un *bug*(defect software), pot fi vizualizate în figura Figura 3.1:

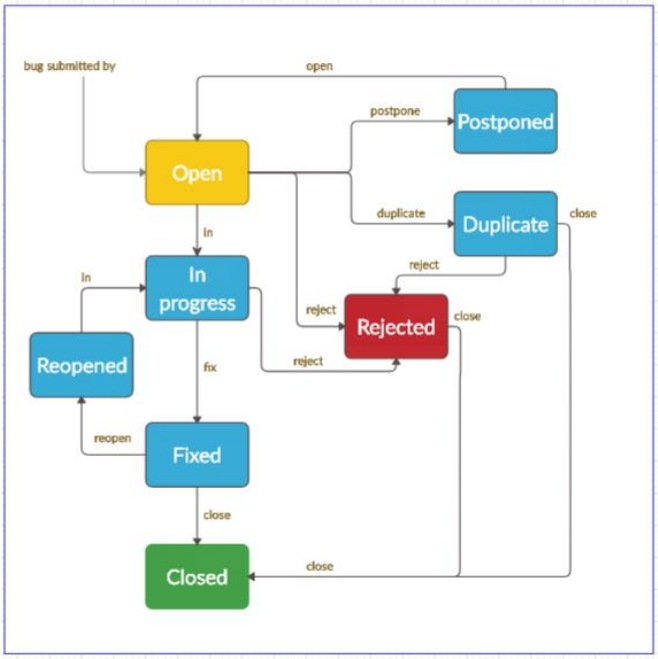


Figura 3.1 *Bug* lifecycle(ciclu de viață) generic

Scopul ciclului de viață al unui defect software prezentat în Figura 3.1 este acela de a coordona cu ușurință modificările de stare ale *bug*-urilor și de a facilita un proces de remediere sistematic a defectelor software. Din Figura 3.1 rezultă că un bug poate să se regăsească în următoarele stări:

- *OPEN* - bug-ul este descoperit și asignat unei persoane din echipa de dezvoltare,
- *IN PROGRESS* - bug-ul se află în proces de fixare
- *DUPLICATE* - bug-ul este o dublură al unuia deja existent
- *REOPENED* - bug-ul persistă și după fixarea de către dezvoltator
- *REJECTED* - bug-ul este respins datorită informațiilor insuficiente sau în situația în care nu este un defect software
- *FIXED* - bug-ul a fost fixat de către dezvoltator
- *CLOSED* - bug-ul a fost analizat, testat și rezolvat de către tester

Un bug report(raportul defectelor software) este instrumentul principal utilizat în rezolvarea bug-urilor. Când un *bug* este identificat, următorul pas spre rezolvarea acestuia este scrierea unui astfel de document detaliat care să ghideze programatorul în procesul de soluționare. În cele mai multe cazuri, aceste rapoarte nu sunt bine întocmite din pricina lipsei de experință a celui care se ocupă de scriere lui.

Problemele care apar în cadrul proiectelor unde managementul bug-urilor se realizează manual duc la desincronizarea echipei deoarece se pot pierde informații utile din cauza dezorganizării și a volumului mare de probleme pe care o aplicație le poate avea.[2] Vizualizarea evoluției activităților de bug fixing poate deveni anevoioasă, ceea ce duce la o gestiune ineficientă și costisitoare a proiectelor. Mai mult, în lucrarea [12], se menționează că aproximativ 50% din timp un programator se ocupă cu soluționarea bug-urilor. Acest lucru duce la creșterea costului de producție și a timpului de efectuare a sarcinilor.

Toate aceste neajunsuri au înclinat balanța spre realizarea unor tool-uri rapide și ușor de folosit pentru a menține o stabilitate în procesul de depanare al diferitelor proiecte. Astfel un bug tracker poate fi definit ca și un sistem software care se utilizează pentru bug reporting a cărui caracteristică principală este aceea de a păstra evidența informațiilor bug-urilor.

În Figura 3.2 este prezentată structura generală a sistemului de *bug report*:

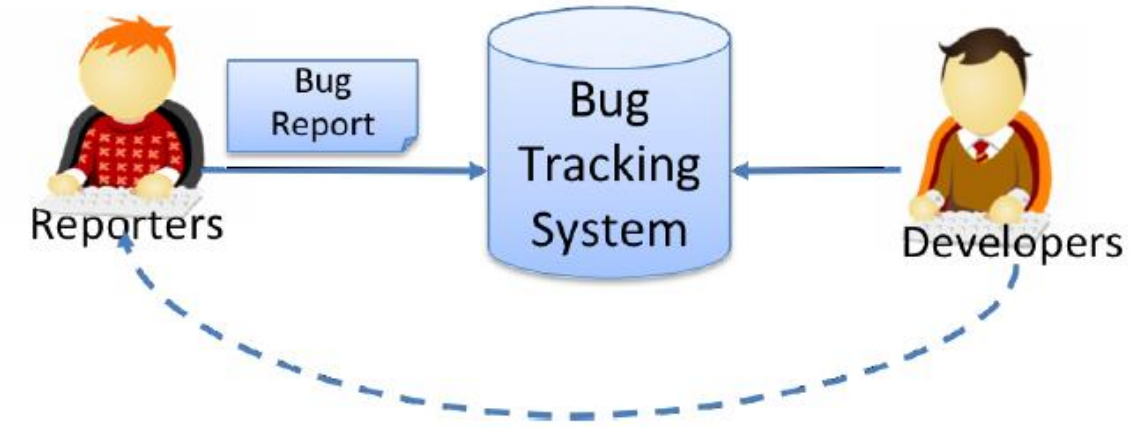


Figura 3.2 Sistemul de bug report [4]

3.1.2. Funcționalitățile unui astfel de sistem

La momentul actual, aplicațiile de trasare ale *bug*-urilor trebuie să se adapteze ușor la cerințele și nevoile utilizatorilor, trebuie să permită oricui următoarele acțiuni de bază:

- Partajarea informațiilor întregii echipe
- Posibilitatea de vizualizare a unui status global asupra bug-urilor unui proiect
- Menținerea unui istoric al modificărilor pentru fiecare bug
- Acordarea de privilegii de executare a unor operații importante de actualizare doar personelor îndreptățite
- Notificarea utilizatorilor atunci când se produc modificări care îi privesc
- Afișarea diverselor statistici legate de starea bug-urilor, a severității lor și a numărului acestora din cadrul unui proiect

Pe lângă aceste funcții primordiale pentru sporirea eficienței procesului de fixare a bug-urilor, majoritatea sistemelor deja existente au introdus și alte acțiuni menite să simplifice și mai mult catalogarea și documentarea bug-urilor, de exemplu adăugarea de atașamente unui bug, aplicarea priorităților unui bug, filtrearea lor ș.a.m.d.

3.1.3. Beneficiile utilizării un bug tracker

Dacă tool-urile existente sunt folosite la adevărata lor capacitate, acestea aduc o multitudine de beneficii:

- Îmbunătățesc calitatea programului – dispunerea de un sistem de tracking potrivit pentru dinamica unei echipe ajută la analiza în detaliu a *bug*-urilor, oferind o privire de ansamblu asupra proiectului.Acest lucru permite implementarea proactivă a măsurilor corecte în timp util. Deoarece se poate vizualiza istoricul problemelor apărute în proiectele anterioare, la apariția aceluiași *bug* într-un nou proiect va fi soluționată într-un timp mult mai scurt.
- Cresc satisfacția utilizatorilor și a clientilor- toți cei implicați sunt conștienți de progresul realizat, fiind informați cu privire la planul de ansamblu al aplicației, fiecare membru lucrând spre atingerea obiectivului comun. Clienții sunt mulțumiți deoarece prin gestiunea eficientă a task-urilor de fixare a problemelor produsul cerut de aceștia este livrat în timp util fără complicații.
- Îmbunătățesc comunicarea între membrii echipei, dar și între echipă și clienți:- datorită mjloacelor de comunicare puse la dispoziție: notificări și comentarii, întreaga echipă este pusă la curent cu ceea ce se întamplă în aplicație, colaborarea desfășurându-se armonios, neexistand neînțelegeri datorită lipsei de informații.
- Ajută la reducerea cheltuielilor și a timpului investit pentru acțiunile de urmărire a defectelor software - deoarece se utilizează un sistem de prioritizare a bug-urilor, asigurandu-se acestora nivele de severitate. Astfel membrii echipei se pot concentra, mai întâi, pe rezolvarea cerințelor mai importante și mai critice.
- Oferă o înțelegere mai profundă a proiectului - dacă o nouă persoană se alătură echipei de dezvoltare și proiectului în curs îi va fi mult mai ușor să se familiarizeze cu cu cerințele și evoluția acestuia
- Pun la dispoziția managerilor de proiect informații cruciale legate de munca membrilor echipei, și a interacțiunilor dintre aceștia

3.2. Standarde de calitate în sisteme software

Un client oarecare nu are cunoștințele necesare pentru a determina cât de eficient se desfășoară lucrurile într-o companie din mediul IT, nefiind familiarizat cu tehnicile utilizate, cu protocoalele care trebuie urmate, și nici cu modul de organizare la nivel intern al unei astfel de organizații. Pentru a identifica nivelul de maturitate a unei firme, care se refelctă inerent în calitatea produselor pe care firma le livrează, au fost stabilite următoarele standarde pentru managementul calității proceselor realizate în cadrul corporațiilor:

- Modelul CMMI- are rolul de a evalua maturitatea unei firme oferind un framework care descrie componentele unui process software eficace
- Standardele ISO 9000- acestea se referă la calitatea produselor obținute în urma proceselor tehnologice

Orice companie încearcă să optimizeze acțiunile pe care le execută, economisind timp, spațiu, resurse cu scopul de a lucra mai eficient și de a satisface nevoile și cerințele clienților într-un timp cât mai scurt. Cu cât un client este mai mulțumit, cu atât compania are o reputație mai bună și atrage mai multe proiecte.

În continuare se va realiza o scurtă descriere a celor două standarde menționate anterior, specificând și influența lor în activitățile desfășurate de o companie.

3.2.1. Capability Maturity Model Integration (CMMI)

A fost creat pentru a veni în ajutorul îmbunătățirii performanțelor, oferind unei afaceri tot ce are nevoie pentru a-și dezvolta în mod constant produsele și serviciile puse la dispoziția cumpărătorilor. Astfel, încurajează firmele să se axeze pe calitatea rezultatelor pe care le produc, și nu pe cantitatea efectuată.

Oferă ghidare în stabilireaobiectivelor și priorităților, precum și în minimizarea factorilor de risc și creșterea stabilității activităților efectuate.

A fost dezvoltat cu scopul de a aduna mai multe modele de maturitate a firmelor sub același framework. Creat în anul 1991 de către Institutul de Inginerie Software de la Universitatea din Carnegie-Mellon, CMMI pune la dispoziție modele și metode de evaluare, precum și materiale educaționale pentru familiarizare. Toate acestea acoperă ciclul de viață ale unui produs din faza de design, până în faza de implementare, livrare, cât și faza de mentenanță. Acest model este destinat în special urmăririi dezvoltării produselor software, dar poate fi aplicat și la scară largă în diferite întreprinderi și din alte domenii de activitate.

Există două reprezentări a CMMI: *staged(în etape)* și *continuous*(continuu). Fiecare reprezentare permițând companiilor să monitorizeze diferie obiective.

i) CMMI *staged*

Are 5 nivele de maturitate:

1) *Initial*- procesul software fiind caracterizat ca fiind ad-hoc, uneori haotic. În acestă etapă nu există planuri pe termen lung.
2) *Managed(Administrat)*- se încearcă definirea nucleului organizației și a proceselor celor mai utilizate, focalizarea fiind pe selectarea tipului de procese și descrierea acestora
3) *Defined(Definit)*- în această etapă, majoritatea proceselor sunt definite. Ele sunt bine documentate și înțelese. Aici standardele descrierilor proceselor și procedurile standard deja definite sunt adaptate în funcție de proiect.
4) *Quantitatively managed*(Gestionat cantitativ)- faza definirii proceselor este încheiată. Se colectează date legate e nivelul de satisfacție al clienților, date care vor fi utilizate pentru procesul de optimizare.
5) *Optimizing(Optimizare)*- procesele sunt definite și gestionate și optimizate unde este posisibil

ii) CMMI *continuous*

Permite oranizației să își adapteze procesele astfel în așa fel încât să se muleze pe obiectivele companiei.

Oferă 6 nivele de evaluare a proceselor:

1) *Incomplete process*(Proces incomplet)- un proces poate fi sau nu poate fi efectuat parțial. În acest caz unul sau mai multe obiective nu sunt îndeplinite
2) *Performed process* (Proces efectuat) - procesele ating toate obiectivele
3) *Managed process*(Proces gestionat)- procesele sunt executate în conformitate cu regulile stabilite.
4) *Defined process*(Proces definit)- această etapă se concentrează pe atingerea obiectivelor de performanță atât ale proiectului, cât și ale organizației.
5) *Quantitatively managed process* (Proces gestionat cantitativ) - procesele sunt caracterizate ca fiind gestionate cantitativ. Adică ele sunt controlate utilizând metode cantitative.
6) *Optimizing process*(Proces de optimizare)- procesele sunt monitorizate constant și îmbunătățite atunci când este nevoie

În figura de mai jos sunt prezentate succinct cele cinci nivele de maturitate ale CMMI *staged*:

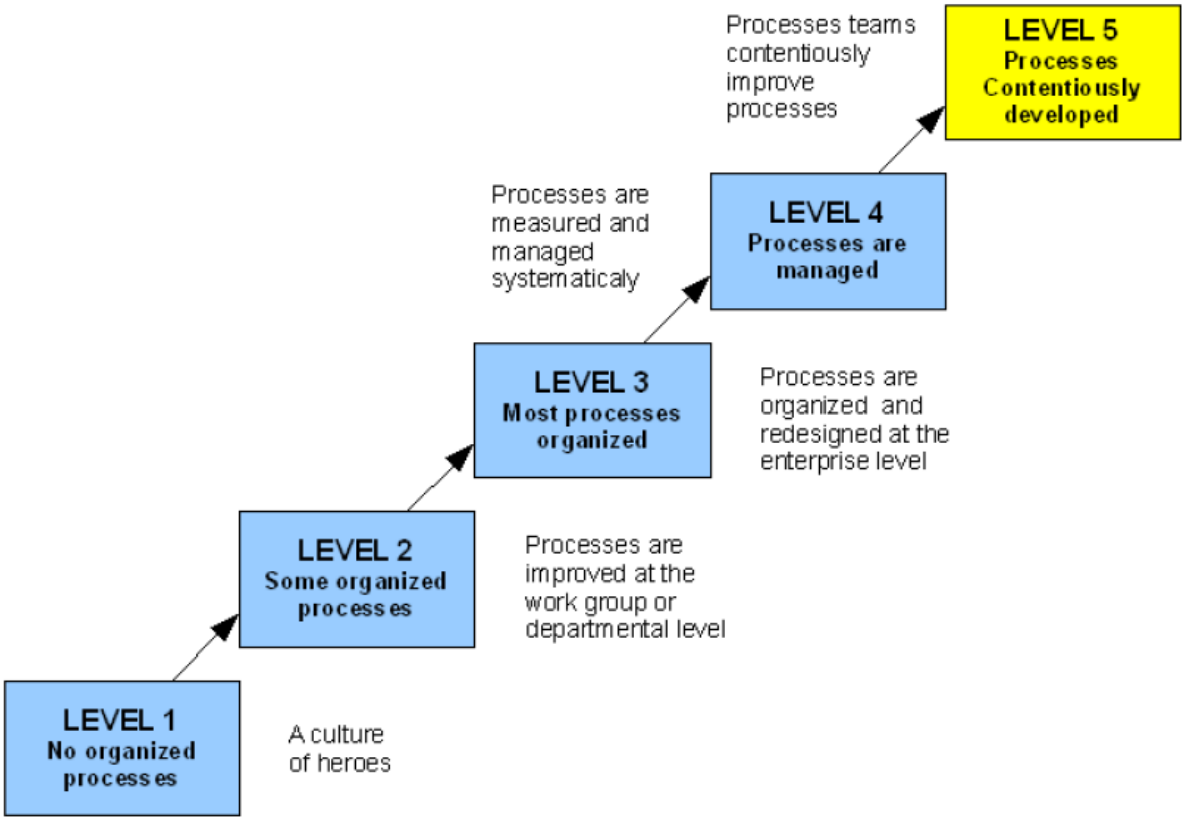


Figura 3.3 Nivelurile de maturitate în CMMI *staged*

(CMMI bazat pe 5 nivele de maturitate) [5]

3.2.2. Standardele ISO(International Organization for Standardization) 9000

Familia ISO 9000 este un set de standarde recunoscute la nivel internațional, utilizate pentru managementul calității.

Elementele asupra cărora se concentrează sunt următoarele[5]:

- Orientarea către client pentru a-i descoperi intențiile și a-i îndeplini cerințele
- Managementul participativ
- Luarea deciziilor bazate pe datele colectate
- Îmbunătățirea continuă
- Tool-uri cantitative pentru îmbunătățirea proceselor
- Creeare de sisteme eficiente de comunicare cu clientul
- Revizuirea în mod regulat a performanțelor obținute
- Documentarea problemelor apărute
- Păstrarea înregistrărilor legate de apariția defectelor unor produse
- Furnizarea de echipamente și resurse adecvate, a dispozitivelor de măsurare și monitorizare

Atât modelul CMMI, cât și Standardele ISO 9000 au un impact major în conceperea de sisteme de tratare a *bug*-urilor calitativ superioare.

3.3. Sisteme existente de management al bug-urilor

Organizarea reprezintă un element de bază în cadrul oricărei companii. Pentru dezvoltarea unui proiect este necesară optimizarea resurselor și structurarea acestora. Cum este menționat în secțiunea 3.1.1 activitățile de fixare a defectelor software reprezintă o componentă crucială și costisitoare în dezvoltarea unui proiect.

Au fost dezvoltate diverse aplicații menite să ușureze procesul de trasare a bug-urilor.În cadrul acestei secțiuni for fi analizate astfel de sisteme atât comerciale, cât și gratuite.

3.3.1. Jira

Un tool des folosit de companiile mari este: **Jira**[2]. Este un produs oferit de Atlassian, care permite bug tracking și utilizarea de tehnici agile in cadrul managementul proiectelor. Este o aplicație scrisă în Java care a fost lansată în 2002.

Denumirea sa a rezultat în urma trunchierii cuvântului Gojira, care în japoneză se traduce ca si Godzilla. Acest nume a fost ales după o porecla a unui sistem de management al bug-urilor utilizat anterior de către Atlassian numit Bugzilla.

Jira este o aplicație comercială, ușor de folosit datorită interfeței prietenoase. Oferă utilizatorilor numeroase acțuni pe care aceștia le pot executa: managemntul task-urilor, prioritizarea problemelor, întocmirea de grafice și multe altele. Totodată dispune de integrarea a numeroase plugin-uri cum ar fi BitBucket, GitLab și GitHub.

Fiind o aplicație comercială, oferă diferite tipuri de pachete în functie de numărul membrilor unei echipe.

[2] https://www.atlassian.com/software/jira/ops

3.3.2. *Clickup*

Un alt tool comercial, folosit des este și **ClickUp**[3]. A fost lansat in 2015 de către compania Mango, care și-a dorit să creeze o aplicație care să vină în ajutorul companiilor pentru o gestiune mai ușoară a proiectelor și a bug-urilor care intervin pe parcursul dezvoltării unei aplicații. Țelul lor este acela de a înlocui toate aplicațiile pentru gestiunea task-urilor, documentele și goal tracking-ul cu o aplicație multifuncțională.

Este foarte ușor de utilizat, design-ul modern fiind foarte atrăgator pentru utilizatori. Acest tool oferă mai mult decât simpla gestiune a task-urilor de orice fel, pune la dispozitie și opțiuni precum calendar, interfață customizabilă, sincronizarea automată cu Outlook etc.

Clichup vine în ajutorul echipelor de toate dimensiunile și din toate domeniile. Mai mult de atât oferă încorporarea platformelor GitHub, GitLab și Slack.

3.3.3. *Monday*

Una dintre aplicațiile cele mai des utilizate în companii pentru managementul bug-urilor, și nu numai, la momentul actual este**: Monday.com**[4]. Este o aplicație ușor de folosit care oferă servicii pentru gestiunea reurselor unei echipe si pentru interacțiunea membrilor, incluzand supervizrea proiectelor, a deadline-urilor si a bug-urilor.

O regăsim atât sub forma unei aplicații web, cât și a unei aplicatți mobile. Produsul a luat naștere în anul 2010, fiind un tool al companiei renumite Wix.com. În februarie 2012 s-a produs separarea de catre WIX.com pentru a deveni un produs al companiei nou înființate daPulse. In 2017 a avut loc redenumirea brand-ului daPulse în Monday.com.

Unul dintre marile benificii este acela că dispune de plugin-uri pentruîncorporarea și sincronizarea cu Gmail, Slack, GitLab, GitHub ș.a.m.d.

Este o aplicație comercială care oferă 4 pachete: Basic, Standard, Pro și Enterprise, costul acestor opțiuni variază însă în funcție de numărul de utilizatori.

3.3.4. *Trello*

Tot un tool pentru gestiunea bug-urilor și nu numai este **Trello**[5]. A fost lansată inițial în anul 2008 de catre Fog Creek Software. Oferă varianta atât mobile, cât și web. Este disponibilă în mai multe limbi, oferă o interfață primitoare și ușor de utilizat. În septembrie 2011 a fost selectată ca una din cele mai bune aplicații de startup de către revista Wired. In 2017 Trello, a devenit membra în familia Atlassian. În prezent, este o aplicație de mare success, având peste 50 de milioane de utilizatori.

Pune la dispoziția celor care optează pentru utilizarea ei diverse feature-uri: de la filtrare de date, functionalități drag&drop, organizare ușoară a card-urilor utilizând etichete, până la notificări atât live, cât și prin intermediul e-mail-ului etc.

Dispune de trei opțiuni de plată, oferind o versiune gratis, dar care este limitată din punctul de vedere a funcționalităților incluse. Se adresează atât firmelor mici, cât și celor medii/mari.

[3] https://clickup.com/features
[4] https://monday.com/product/
[5] https://trello.com/guide/feature-deep-dive

3.3.5. The Bug Track

Tot o aplicație online comercială pentru gestiune bug-urilor este și **The Bug Track**[6]. A fost lansată în cadrul companiei Wukong Design. A fost creată pentru a oferi servicii de tracking a proiectelor în special pentru utilizatorii de Google Apps. Este ușor de utilizat și oferă multe feature-uri pentru gestiunea bug-urilor: asignare bug-uri, vizualizare istoric, precum și vizualizarea rapidă a bug-urilor existente într-un proiect sun formă tabelară, export date în format .csv etc.

Și această aplicație pune la dispoziția utilizatorilor trei planuri de tarifare îm funcție de numărul de utilizatori și a numărului de proiecte. Oferă o versiune trial de 30 de zile și un demo pentru familiarizarea cu tool-ul.

3.3.6. Bugzilla

Un alt sistem des utilizat de companii pentru gestiunea bug-urilor este și **Bugzilla**[7]. Este o aplicație open-source, lansată în anul 1998 cu scopul de a fi utilizată de mozilla.org, aceasta înlocuind sistemul intern de la acea perioadă. Inițial scrisă în TCL, s-a decis ulterior migrarea spre Perl. Bugzilla este un sistem bazat pe web, care necesită instalare pe server-ul celui care o utilizează.

În ciuda faptului că este o aplicație gratuită, pune la dispozitie o mulțime de caracteristici: de la filtre de căutare, setare de priorități bug-urilor, adăugare atașamente, vizualizarea rapidă a bug-urilor sub forma unui tabel, la multiplele versiuni lingvistice în care este disponibilă, până la rapoarte și statistici, notificări prin e-mail, detecția automată al bug-urilor duplicate și multe altele.

3.3.7. Mantis

O alta aplicație in conformitate cu tema abordată este: **Mantis**[8]. Este un program open-source pentru identificarea problemelor din cadrul unui proiect, reușind să mențină echilibrul între simplitate și putere. Utilizatorii pot cu ușurință să își gestioneze proiectele în timp ce colaborează cu alți colegi de breaslă și clienți.

Mantis a fost lansată în 2000 de către Knzaburo Ito, iar din iulie 2012 GitHub a devenit locul de unde se poate lua codul sursă a acestui tool. Acest software a fost scris în PHP, utilizând și protocoale GNU.

Deși este o aplicație open-sorce, ea vine la pachet cu o versiune web, dar și cu o versiune mobile ceea ce o face și mai atractivă pemtru utilizatori.

Deoarece mulți utilizatori doreau să faciliteze rapid de funcționalitățile puse la dispoziție de MantisBT fără a mai fi neceară configurarea server-ului web, efectuarea copiilor de siguranță și actualizărilor, dezoltatorii au lansat MantisHub- un serviciu al MantisBugTracker care poate fi accesat online și care beneficiază de toate caracteristicile deținute de varianta originală: support pentru mai multe limbi, bug history, grafice etc. Însă există o variantă gratis de doar 30 de zile, ulterior un utilizator putând să opteze pentru unul dintre planurile de tarifare existente.

[6] http://www.thebugtrack.com/tour
[7] https://www.bugzilla.org/about/
[8] https://www.mantisbt.org/

3.3.8. RedMine

Redmine[9] este o alta aplicație pentru managementul bug-urilor. Dezvoltată de Jean-Phelippe Lang, a fost lansată în iunie 2006, ultima sa versiune fiind din octombrie 2019.

A fost scrisă in Ruby on Rails și este open-source. Poate fi utilizată pe orice platformă și cu orice baza de date.

Unele din caracteristicile RedMine sunt: integrarea calendarul Grantt, diagrame și statistici, suport pentru mai multe limbi, sistem flexibil de urmărire a bug-urilor și multe altele. Interfața afișată este neintuitivă pentru utilizatorii neexperimentați, bug-urile sunt afișate în tabele asupra cărora se pot aplica filtre de căutare și sortare personalizate.

Un principal impediment cu care se confruntă utilizatorii este mentenața și înțelegerea manipulării unor acțiuni din cadrul aplicației.

3.3.9. FogBugz

O alta aplicație des intâlnită este : **FogBugz**[10]. Este un tool creat pentru a veni în ajutorul echipelor din cadrul companiilor oferindu-le o platformă de management a proiectelor. A fost lansată în 2000, ca fiind parte a companiei Fog Creek Software..

FogBugs este un sistem de urmărire cu un motor de cătare puternic care permite căutarea rapidă a diverselor informații legate de un caz(un bug este denumit în cazul aplicației *case*), fiind compatibil cu apropae toate sistemele de operare existente.

Printre caracteristicile acestui sistem se numără și: crearea rapidă a cazurilor, tool integrat de gestiune a timpului, realizarea automată a backup-urilor și actualizărilor, suport pentru e-mail, suport pentru metodologii Agile etc.

Este un tool comercial care nu ține cont de dimensiunea echipei care îl folosește, ci de perioada de timp pentru care se face plata licenței.

3.3.10. BugZero

Un sistem de gestiune comercial a bug-urilor este și **BugZero WebSina**[11]. Este un sistem software care permite unei firme sa urmarească progresul unei probleme apărute în cadrul unor proiecte interne. Este o aplicație flexibilă și scalabilă care poate fi configurată în funcție de modul unic de lucru al unei companii.

Este un produs web care este acceptat atât de sistemele de operare: Windows, Unix și Mac OS, care acceptă diverse sisteme de baze de date.

Din interfața simplistă se pot identifica diverse caracteristici: suport pentru mai multe limbi, adăugare de atașamente unui bug, integrare mail, export de informație în format .csv și multe altele.

În Tabel 3.1 și Tabel 3.2 este realizată o comparație între sistemul propriu și sistemele similare descrise în secțiunile anterioare. Ca și criterii de comparație au fost utilizate funcționalitățile de bază identificate .

[9] https://www.redmine.org/
[10] https://www.fogbugz.com/FeaturesTour
[11] http://www.websina.com/

Caracteristici	Jira	Click up	Monday	Trello	The Bug Track	Sistemul dezvoltat
Suport multilingvist	✓	✗	✓ (limitată)	✓	✗	✓
Sortare	✓	✓	✓	✓	✓	✓
Căutare	✓	✓	✓	✓	✓ (limitată)	✓
Notificări	✓	✓	✓	✓	✗	✓
Email	✓	✓	✓	✓	✗	✓
Atașamente	✓	✓	✓	✓	✓	✓
Import bugs	✓	✓	✓	✓	✗	✗
Export bugs	✓	✓	✓	✓	✓	✓
Istoric	✓	✓	✓	✓	✓	✓
PieCharts	✓	✓	✓	✓	✗	✓
Comentarii	✓	✓	✓	✓	✗	✓
Ușor de utilizat	✓	✓	✓	✓	✓	✓
Gratis	✗	✗	✗	✗	✗	✓

Tabel 3.1 Partea I: Comparație între sistemul dezvoltat și cele existente

Caracteristici	Mantis	FogBugz	Bugzilla	Redmine	Bug Zero	Sistemul dezvoltat
Suport multilingvist	✓	✓	✓ (limitată)	✓	✓	✓
Sortare	✓	✓	✓	✓	✓	✓
Căutare	✓	✓	✓	✓	✓	✓
Notificări	✓	✓	✓	✗	✗	✓
Email	✓	✓	✓	✓	✓	✓
Atașamente	✓	✗	✓	✓	✓	✓
Import bugs	✓	✗	✓	✓	✗	✗
Export bugs	✓	✓	✓	✓	✓	✓
Istoric	✓	✓	✓	✓	✗	✓
PieCharts	✓	✓	✓	✗	✓	✓
Comentarii	✓	✓	✓	✓		✓
Ușor de utilizat	✗	✓	✗	✗	✗	✓
Gratis	✗	✗	✓	✓	✗	✓

Tabel 3.2 Partea a II-a: Comparație între sistemul dezvoltat și cele existente

Din Tabelul 3.1 și 3.2 se poate constata faptul că sistemele gratuite deși dispun de multe funcționalități, la capitolul interfață grafică nu excelează.

Capitolul 4. Analiză şi Fundamentare Teoretică

4.1. Cerințele sistemului

4.1.1. Cerințe funcționale

Pentru a înțelege mai bine funcționaliățile sistemului ales pentru implementare, în această secțiune se vor prezenta cerințele funcționale. Cerințele funcționale definesc ceea cum să se comporte sistemull în funcție de datele de intrare pe care le primește.

În tabelul 4.1 sunt descrise toate cerințele funcționale ale aplicației, precum și utilizatorii care pot executa și au dreptul să realizeze acele acțiuni.

Cerință	Administrator	Project Manger	Developer	Tester
Login/logout/ Modificare limba afișată	✓	✓	✓	✓
Editare informații personale	✓	✓	✓	✓
Ștergere utilizatori	✓	✗	✗	✗
Adăugare utilizatori	✓	✗	✗	✗
Adăugare permisiuni utilizatori	✓	✗	✗	✗
Vizualizare listă utilizatori	✓	✓	✓	✓
Vizualizare grafice	✗	✓	✓	✗
Vizualizare listei de bug-uri	✗	✓	✗	✓
Adăugare Proiect	✗	✓	✗	✗
Ștergere Proiect	✗	✓	✗	✗
Editare Proiect	✗	✓	✗	✗
Adăugare membru proiect	✗	✓	✗	✗
Ștergere membru proiect	✗	✓	✗	✗
Asignare bug	✗	✓	✓	✓
Editare bug	✗	✓	✓	✓
Vizualizare detalii bug	✗	✓	✓	✓
Vizualizarea istoric bug	✗	✓	✓	✓
Descărcarea detaliilor unui bug in format pdf și csv	✗	✓	✓	✓
Adăugare comentarii bug	✗	✓	✓	✓
Inchiderea unui bug	✗	✗	✗	✓

Tabel 4.1 Cerințele funcționale ale sistemului propus

4.1.2. Cerințe non-funcționale

În acest subcapitol vor fi detaliate acele cerințe care au un aport major în proiectarea oricărei aplicații, deci și a sistemului de gestiune a bug-urilor dezvoltat. Aceste cerințe se numesc cerințe non-funcționale și ele au rolul de a specifica atributele de calitate ale sistemelor prin impunerea unor constrângeri (constrângeri legate de tehnologiile utilizate în implementare, arhitectura sistemului, modul de structurare al logicii ș.a.m.d) cu scopul de a obține sisteme calitativ superioare.

Utilizabilitatea

Reprezintă capacitatea unui sistem de a a fi înțeles și utilizat cu ușurință de către utilizatori, fără alocarea unei cuante de timp semnificative pentru procesul de învățare al sistemului. Orice persoană care utilizează acea aplicație trebuie să poată să pregătească cu ușurintă date de intrare sau să interpreteze ieșirile respectivului sistem fără impedimente.

Tool-ul propus are o interfață intuitivă, simplistă, astfel încât utilizatorii să poată regăsi rapid informațiile de care au nevoie. Elementele componente ale paginilor sunt plasate strategic, captând atenția asupra componentelor imporatnte(ex: câmpuri de due date). Mai mult de atât, paleta de culori aleasă este sugestivă, indicând însemnătatea elementelor.

Realizarea unei interfețe în care obiectivele celui care o folosesc sunt atinse rapid este necesară, căci utilizatorii tind să utilizeze aplicații cu design plăcut și pragmatic în detrimentul unei aplicații cu multe funcționați greu de deslușit.

Securitatea

Este o proprietate deosebit de importantă, care presupune protecția sistemului împotriva atacurilor rău intenționate și păstrarea confidențialității datelor cu conținut sensibil. Acest lucru se asigură prin filtrarea utilizatorilor care pot citi și scrie date, acest lucru realizându-se prin autentificarea și verificarea permisiunilor asociate lor. Mai mult, trebuie asigurată și integritatea datelor acest lucru ralizându-se în aplicația aleasă prin verificarea permisiunilor unei peroane de a modifica anumite date.

Pentru ca o persoană să poată utiliza platforma, aceasta, mai întâi va trebui să se autentifice, moment în care se va genera un token unic pentru sesiunea curentă fără de care request-urile către aplicația server nu vor pute fi excutate cu succes. Pentru o siguranță mai sporită, token-ul va avea o durată de viață limitată pentru evitarea posibilelor atacuri. La nivel de aplicație Spring Boot există încă un nivel de securitate: cel oferit de Spring Security(utilizează *JWT*(JSON Web Token) și criptarea parolei) și de utilizarea protocolului HTTPS.

Extensibilitatea

Reprezintă capacitatea unui sistem de a fi completat ulterior cu alte functionalități și nivelul de efort necesar pentru implementarea extensiei respective. Este o caracteristică fundamentală pentru orice proiect. Arhitectura aplicației țintă permite adăugarea de funcționalități noi datorită structurii pe layere a componentelor, a organizării elementelor în pachete și prin utilizarea interfețelor între diferitele nivele ale aplicației.

Performanța

Reprezintă factorul cheie în indicarea calității unui sistem. Măsoară rapiditatea și consecvența aplicației pentru livrarea răspunsului corespunzător cererii solicitate de către utilizator, precum și randamentul cu care se realizează volumul de procesare. Timpul de răspuns este influențat de cantitatea de informație care este furnizată, de modul în care sunt transmise cererile, de numărul de request-uri la baza de date, precum și de complexitatea operațiilor efectuate pentru filtrarea informațiilor cerute.

Sistemul realizat are un timp de răspuns scurt (până în 3 secunde[12] , aceasta fiind limita maximă a răspunsului acceptabil, aceste valori sunt doar aproximări și sunt influențate de tipul de utilizatori, domeniul de activitate adresat ș.a.m.d) datorită utilizării eficiente a resurselor disponibile și prin transmiterea între diferitele nivele ale aplicație doar a informațiilor necesare și prin reducerea numărului de cereri către server prin filtrarea și combinarea datelor de la nivelul cel mai inferior nivel: baza de date(prin utilizarea operațiilor *JOIN*=combinarea rîndurilor dintre două sau mai multe tabele pe baza unei coloane înrudită)

Scalabilitatea

Este definită ca fiind calitatea unui sistem de a fi disponibil și funcțional în condițiile majorării numărului de utilizatori care interacționează cu respectivul sistem, numărul de cereri care necesită prelucrare crescând astfel considerabil.

Aplicația de gestiune a bug-urilor prin utilizarea interogărilor cât mai optimizate, filtrarea datelor cât de mult posibil la nivelul bazei de date, evitarea executării mai multor interogări pentru o singură acțiune solicitată de utilizator astfel reducând supraîncărcarea server-ului. Sistemul dezvoltat, fiind dedicat companiilor mici și mijlocii, își propune să susțină până la 200 de utilizatori. Aplicația dezvoltată poate fi ulterior scalată vertical sau orizontal fără a implica modificări majore la nivelul arhitecturii de bază.

Scalarea pe orizontală are loc atunci când sunt adăugate mai multe elemente în același nivel al aplicației care să lucreze în paralel. Scalarea verticală are loc atunci când se adaugă mai multe resurse unui nod sau atunci când se optimizează algortimii implementați.

Pentru dezvoltări ulterioare, scalabilitatea aplicației dezvoltate poate fi îmbunătățită utilizând următoarele tehnici: sporirea resurselor (memorie, capacitatea de procesare și stocare) și îmbunătățirea programului scris pentru a permite sistemului satisfacerea nevoilor tuturor clienților.

4.2. Cazuri de utilizare

Cazurile de utilizare reprezintă o listă de pași(acțiuni) pe care un actor(utilizator) le execută, acesta interacționând cu un sistem pentru a-și atinge un obiectiv.

Rolul acestui subcapitol este acela de modela toate scenariile pe care un utilizator le poate întalni, în funcție de rolul și permisiunile sale, în aplicație.

[12] https://www.quicksprout.com/website-speed/

4.2.1. Actorii aplicației

Sistemul implementat admite patru tipuri de utilizatori

1. Administratorul - acesta se ocupă de managementul tuturor user-ilor din aplicație
2. Project Manager – este utilizatorul care are rolul de a crea noi proiecte și de a gestiona persoanele care se ocupă de un anumit proiect, precum și monitorizarea activităților realizate de ceilalți participanți
3. Developer – se ocupă de executarea operațiile care îi sunt adresate, este utilizatorul cu cele mai puține drepturi
4. Tester – se ocupă de validarea acțiunilor de bug fixing realizate de developeri

Fiecare tip de utilizator are asociate și un set de permisiuni. Pemisiunile disponibile în cadrul aplicației sunt:

- Super_user_operations – deținătorul acestei permisiuni poate executa operații CRUD asupra utilizatorilor
- User_operations – necesară pentru editarea informațiilor contului propriu
- Permission_operations – permite adăugarea/ștergerea unor permisiuni
- Bug_operations – executarea operațiilor CRUD asupra bug-urilor
- Extended_bug_operations – export informații bug-uri, vizualizare statistici
- Close_bug- permisiune necesară pentru setarea unui bug pe *CLOSED*

În Tabel 4.2 se pot vizualiza legăturile dintre rolul unui utilizator și permisiunile inițiale asociate:

Permisiuni	Administrator	Project Manager	Developer	Tester
Super_user_operations	✓	✗	✗	✗
User_operations	✓	✓	✓	✓
Permission_operations	✓	✗	✗	✗
Bug_operations	✗	✓	✓	✓
Extended_bug_operations	✗	✓	✓	✓
Close	✗	✗	✗	✓

Tabel 4.2 Permisiunile asociate rolurilor

În Figura 4.1, Figura 4.2, Figura 4.3, și Figura 4.4 se regăsesc toate funcțiile care pot fi exercitate de tipurile de utilizatori menționate anterior:

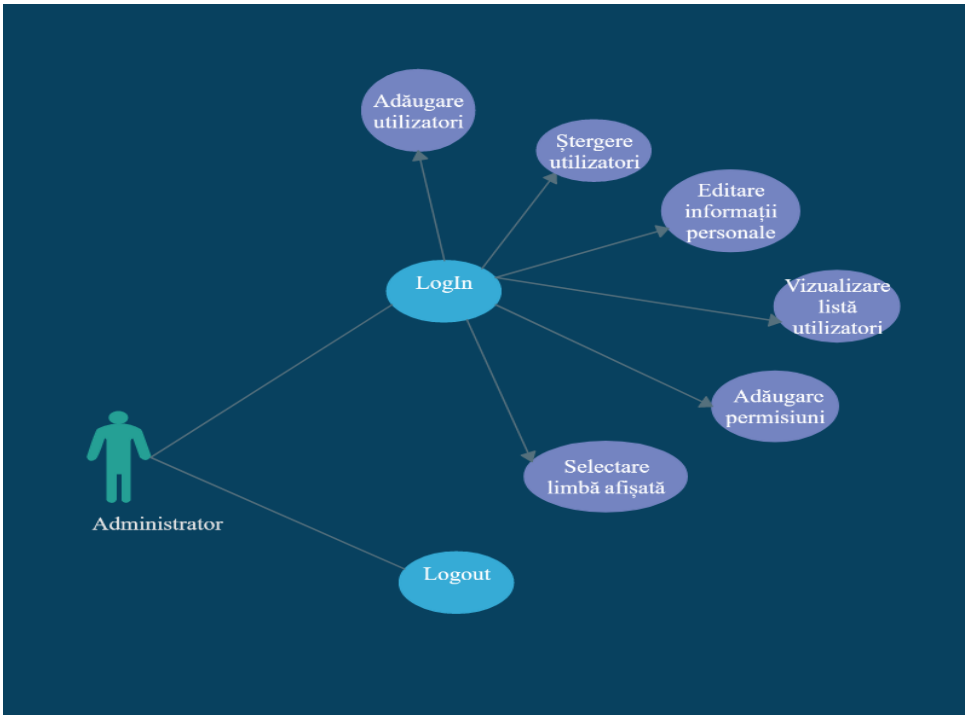


Figura 4.2 Cazuri de utilizare administrator

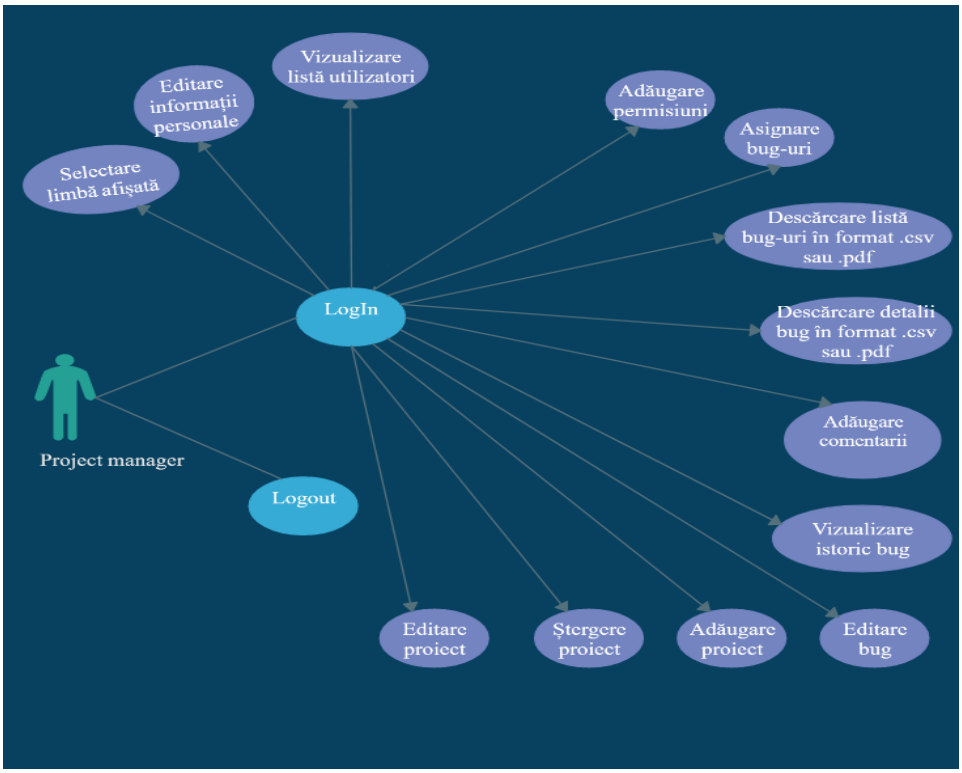


Figura 4.3 Cazuri de utilizare Project Manager

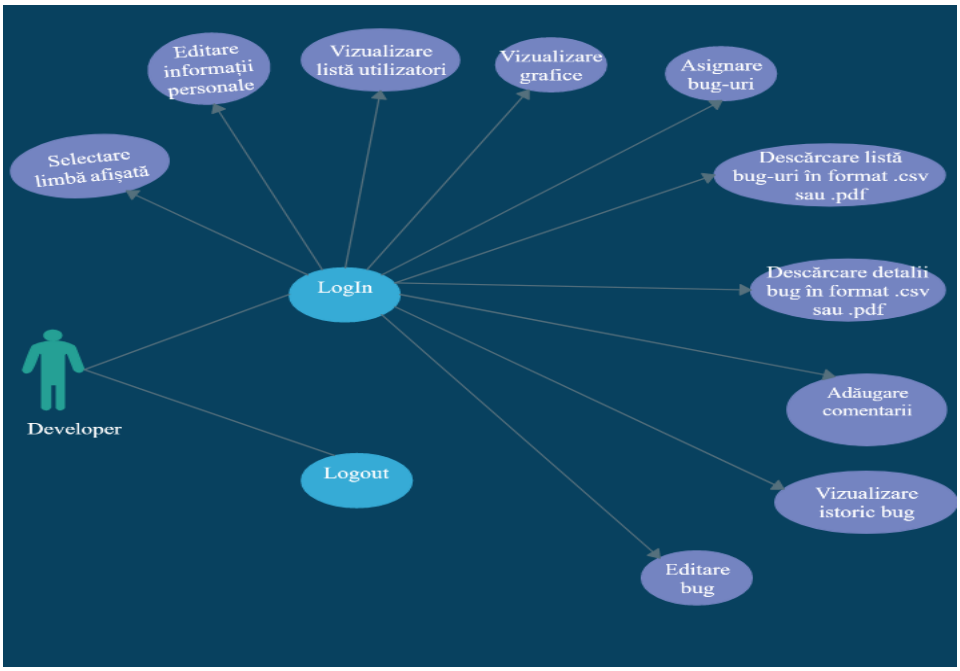


Figura 4.4 Cazuri de utilizare Developer

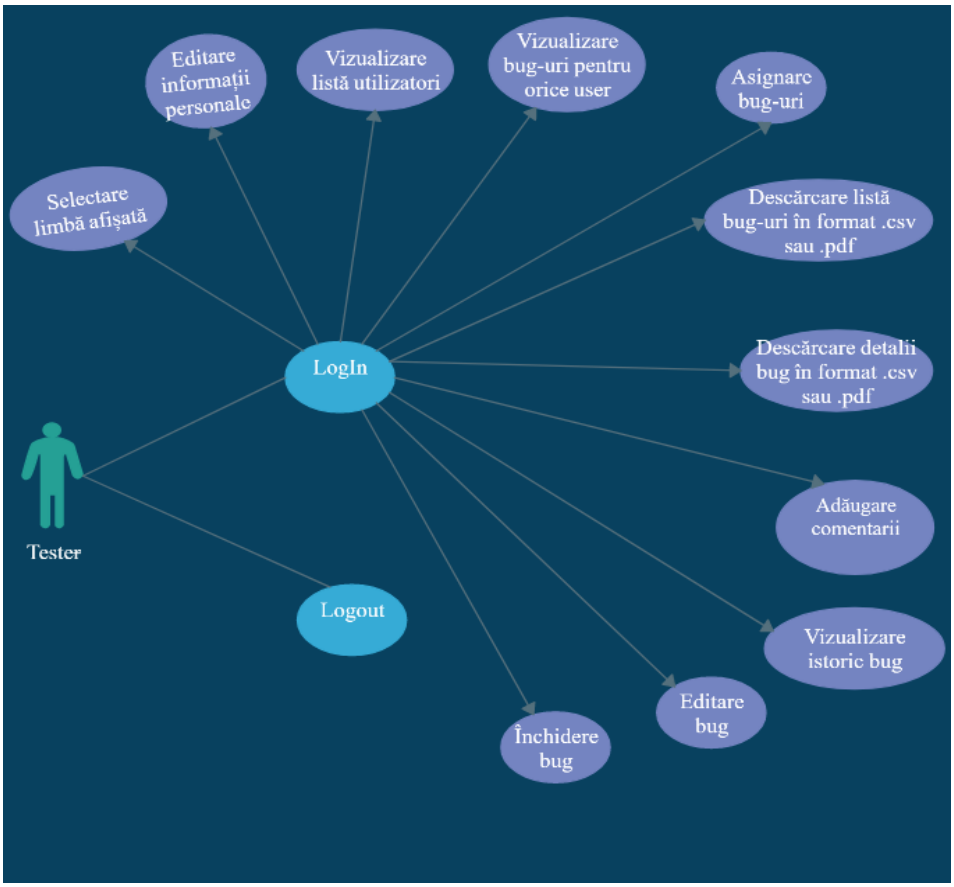


Figura 4.5 Cazuri de utilizare Tester

4.2.2. Descrierea scenariilor de utilizare

În cele ce urmează sunt descrise principalele cazuri de utilizare ale aplicației.

4.2.2.1. Cazul de utilizare login

Actor: toate tipurile de utilizatori sunt considerați ca și actori principali pentru acest caz

Precondiții:

- utilizatorul să acceseze pagina destinată logării sale
- utilizatorul care intenționează să se logheze să nu fie deja logat în browser-ul din care solicit o nouă operație de logare

Postcondiții

- logarea cu succes în cazul în care au fost introduse credențiale valide și redirecționarea către pagina corespunzătoare drepturilor deținute de persoana logată
- afișarea unui mesaj de eroare dacă informațiile necesare logării nu au fost regăsite în baza de date

Flow-ul execuției operațiunii de login:

1) Se accesează pagina de login
2) Sunt vizibile câmpurile asociate câmpurilor care trebuiesc obligatoriu completate
3) Utilizatorul completează spațiile libere cu datele sale și selectează limba dorită pentru afișaj dintr-un dropdown
4) Dacă toate câmpurile conțin informații, butonul de submit devine enable
5) Utilizatorul apasă butonul de submit
6) Sistemul verifică dacă credențialele introduse sunt asociate unui user din baza de date.
    a. Dacă credențialele sunt asociate unui cont de utilizator se creează un token și se salvează sesiunea, și se face redirecționare la pagina coresponzătoare rolului deținut de cel care accesează platforma
        i. Dacă este administrator va fi redirecționat la AdministratorPage
        ii. Dacă este project manager va fi redirecționat la ProjectManagerPage
        iii. Dacă este developer va fi redirecționat la DeveloperPage
        iv. Dacă este tester va fi redirecționat la TesterPage

    b. Dacă credențialele nu sunt valide, utilizatorul va fi atenționat printr-un mesaj de eroare și va rămâne la pagina curentă: LoginPage

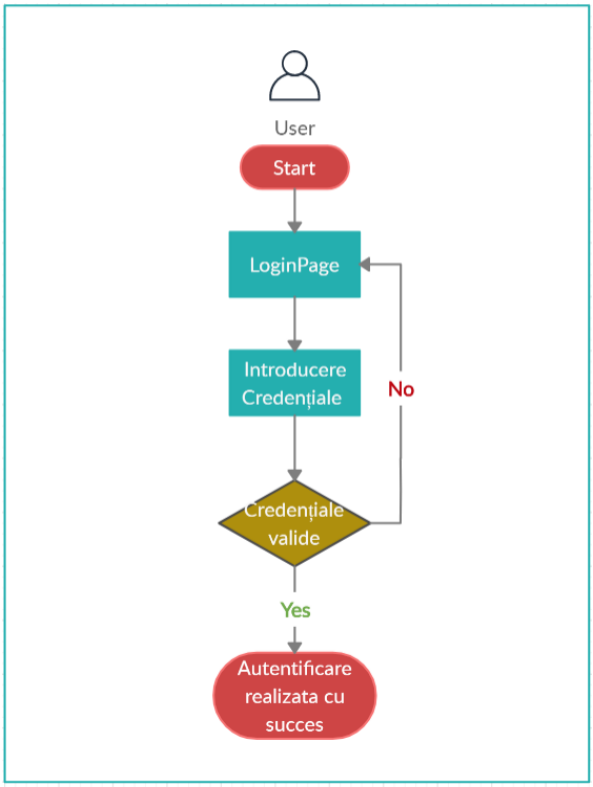


Figura 4.6 Cazul de utilizare pentru login

4.2.2.2. Cazul de utilizare adăugare utilizator nou

Actor: singurul actor care poate realiza această operație este administratorul
Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de administrator
- User-ul să fie situat în pagina principală dedicată administratorului în care se regăsesc informațiile legate de toți utilizatorii aplicației

Postcondiții:

- Utilizatorul nou introdus este salvat în baza de date
- Informațiile sale pot fi vizulizate în tabelul cu utilizatori

Flow-ul operației de adăugare user:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroduce date
2) Se verifică dacă utilizatorul logat este administrator
    a. Dacă are rolul menționat anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra utilizatorilor
    b. Dacă utilizatorul are alt rol, pagina de users nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare user
3) Administratorul apasă butonul de CreateUser
4) User-ul este redirecționat la un form unde câmpurile trebuie completate cu datele persoanei de introdus în platformă

5) Administratorul completează câmpurile solicitate
    a. Dacă introduce date valide
        i. Butonul de save va fi activat și apăsat
            1. Datele introduse sunt trimise la server
                a. Dacă informațiile introduse aparțin unui utilizator deja existent
                    i. se va trimite în frontend un mesaj de eroare și se revine la pasul 4)
                b. Dacă informațiile din pagină nu există deja inserate
                    i. Se adaugă un nou utilizator în baza de date
                    ii. Se trimite un mesaj de success
                    iii. Utilizatorul nou introdus este vizibil în tabela de users din pagina principală a administratorului
        ii. Butonul de cancel este apăsat
            1. Toate modificările sunt contramandate
    b. Dacă nu introduce date valide
        i. Acest lucru va fi marcat vizual
        ii. Butonul de save va rămâne dezactivat
        iii. Utilizatorul este nevoit să reintroducă datele, astfel procesul întorcându-se la pasul 5)

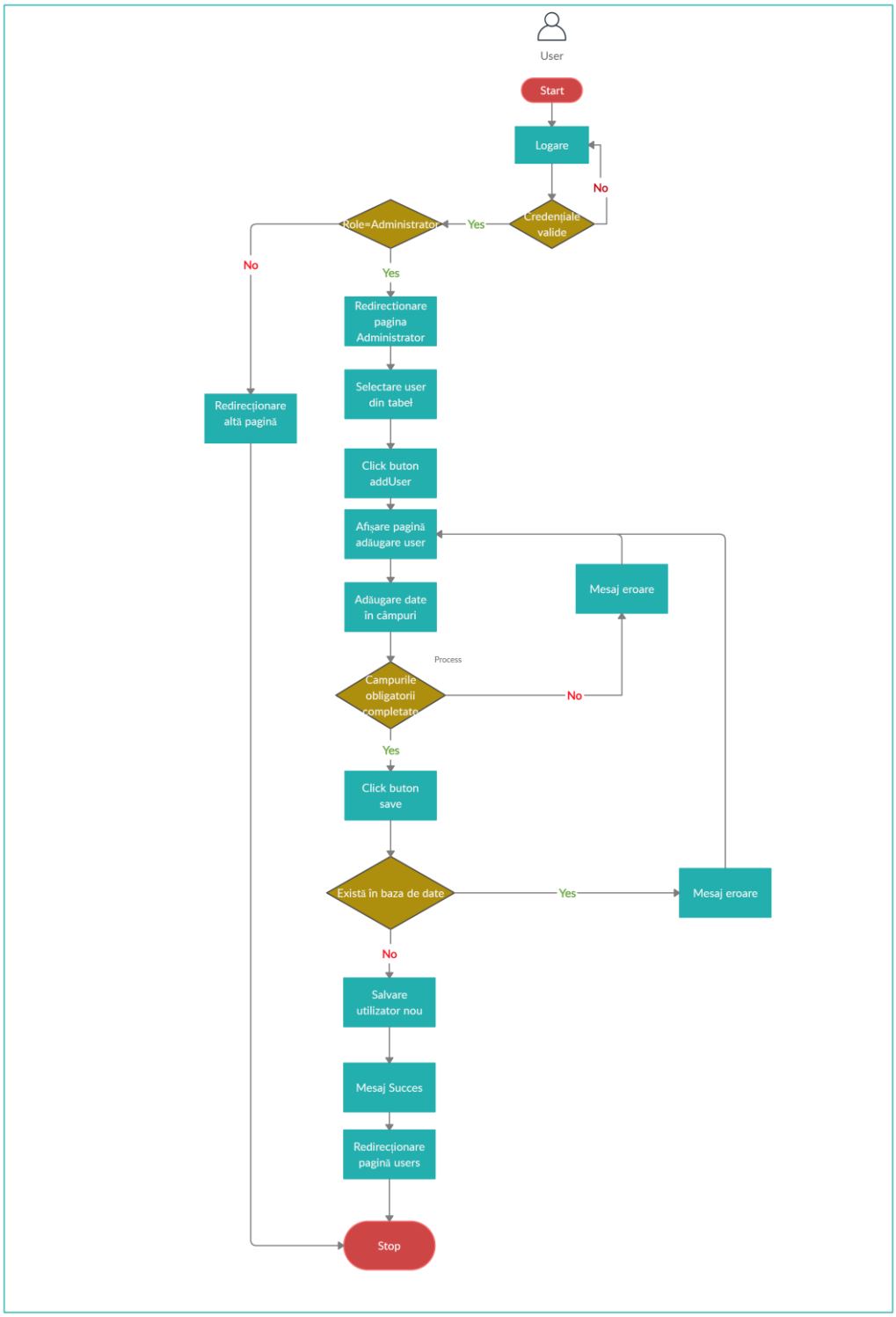


Figura 4.7 Cazul de utilizare pentru adăugarea unui utilizator nou

4.2.2.3. Cazul de utilizare adăugare permisiuni utilizator

Actori: Administratorul
Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de administrator
- User-ul să fie situat în pagina dedicată management-ului permisiunilor se regăsesc informațiile legate de toți utilizatorii aplicației și permisiunile pe care aceștia le dețin

Postcondiții:

- Utilizatorul are adăugate noi permisiuni
- Informațiile sale pot fi vizulizate în tabelul cu utilizatori

Flow-ul operației de adăugare user:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroduce date
2) Se verifică dacă utilizatorul logat este administrator
    a. Dacă are rolul menționat anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra permisiunilor unui utilizator
    b. Dacă utilizatorul are alt rol, pagina de Permisiuni nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare permisiuni
3) Administratorul selectează din tabel user-ul căruia dorește să îi adauge mai multe permisiuni
4) Administratorul apasă butonul de addPermission
5) User-ul este redirecționat la un form unde există un drop down cu permisiunile existente și care nu au fost încă asignate
6) Se selectează permisiunile dorite
7) User-ul logat va apăsa butonul
    a. AssignPermission
        i. Informațiile ajung în baza de date unde o să fie salvate
        ii. Tabelul cu permisiuni asignate user-ului asupra căruia s-a exercitat această operație va fi actualizat
    b. Cancel
        i. Modificările efectuate în pașii anteriori sunt contramandate

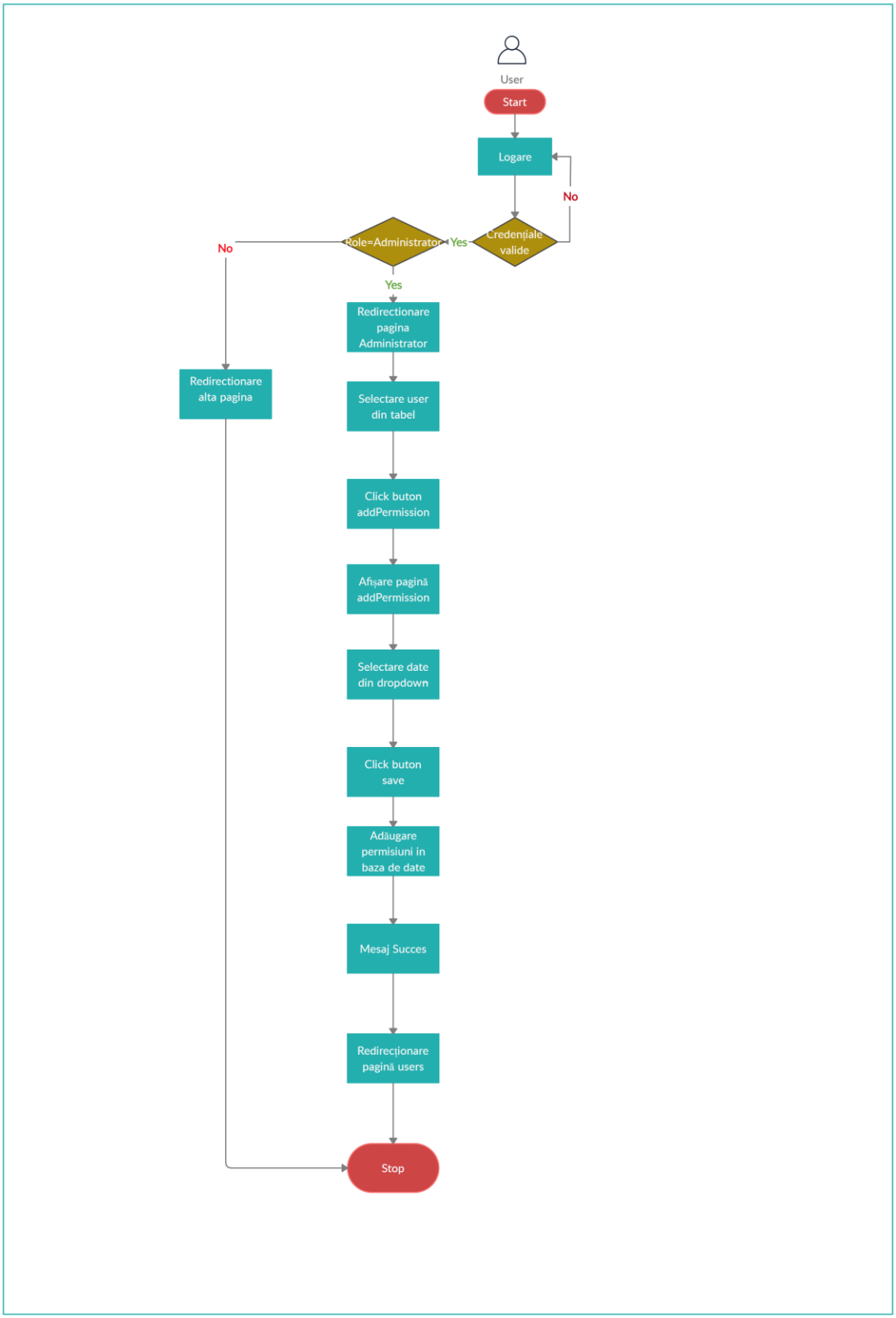


Figura 4.8 Cazul de utilizare pentru adăugare permisiuni

4.2.2.4. Cazul de utilizare adăugare proiect

Actor: singurul actor care poate realiza această operație este project manager-ul
Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de Project Manager
- User-ul să fie situat în pagina dedicată Project Manager-ilor în care se regăsesc informațiile legate de proiectele existente în platformă

Postcondiții:

- Un nou proiect va fi adăugat în baza de date
- Informațiile noului element adăugat pot fi vizibile din pagina utilizatorului care a creat acel element

Flow-ul operației de adăugare proiect:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroducă date
2) Se verifică dacă există un utilizatorul logat să fie Project Manager
    a. Dacă are rolul menționat anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra proiectelor
    b. Dacă utilizatorul are alt rol, pagina menționată anterior nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare proiect
3) Project Manager-ul apasă butonul de AddProject
4) User-ul este redirecționat la un form unde câmpurile trebuie completate cu datele ce țin de noul proiect
5) Project Manager-ul completează câmpurile solicitate
    a. Dacă introduce date valide
        i. Butonul de save va fi activat
            1. Project Manager-ul apasă acest buton
            2. Datele introduse sunt trimise la server
                a. Dacă informațiile sunt deja existente în baza de date
                    i. Se va trimite in frontend un mesaj de eroare
                    ii. Se revine la pasul 4)
                b. Dacă informațiile din pagină nu există deja inserate
                    i. Se adaugă un nou proiect în baza de date
                    ii. Se trimite un mesaj de success
                    iii. Proiectul nou introdus este vizibil în pagina de vizualizare proiecte a utilizatorului logat
        ii. Butonul de cancel este apăsat
            1. Toate modificările sunt contramandate

b. Dacă nu introduce date valide
    i. Acest lucru va fi marcat visual
    ii. Butonul de save va rămâne dezactivat
    iii. Utilizatorul este nevoit să reintroducă datele, astfel procesul întorcându-se la pasul 5)

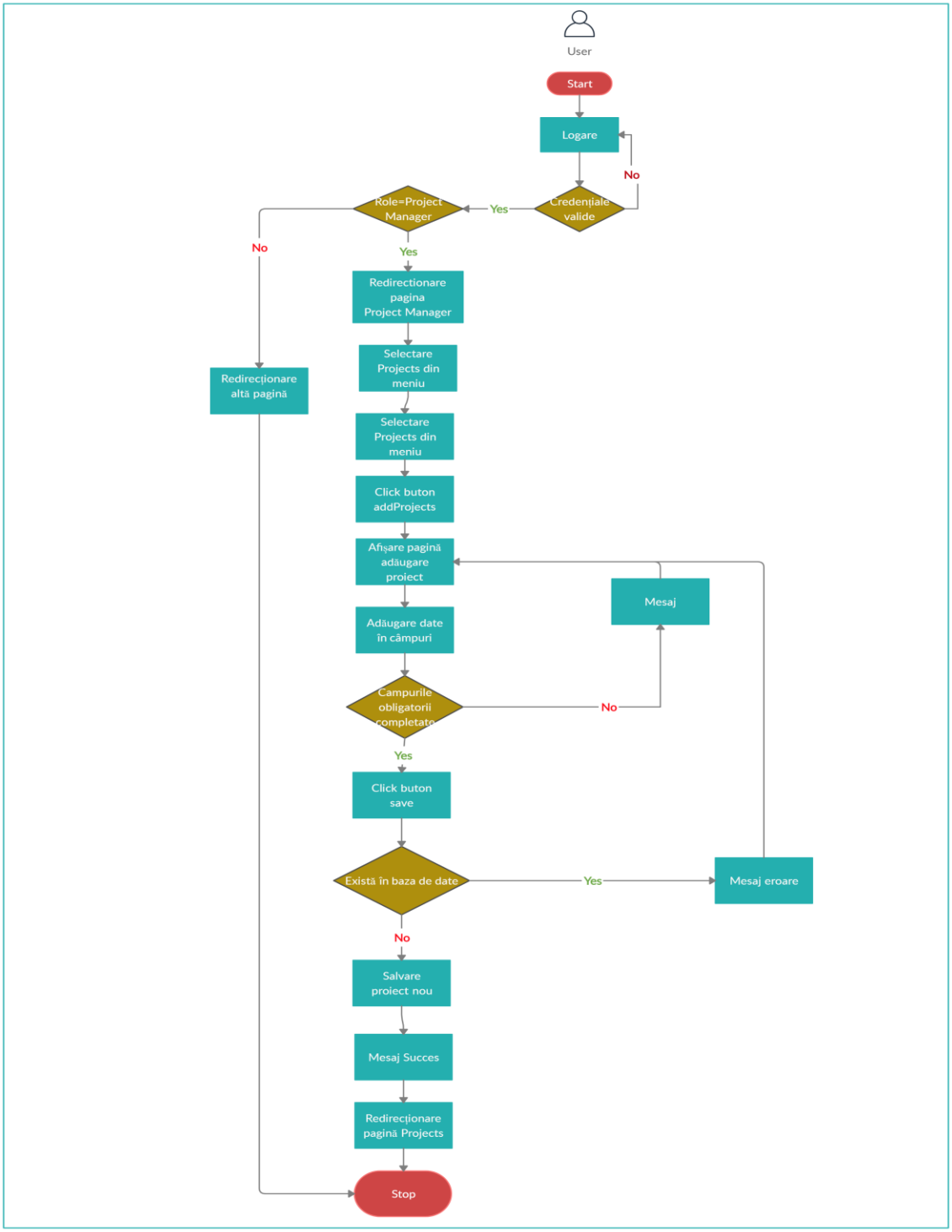


Figura 4.9 Cazul de utilizare pentru adăugare proiect

4.2.2.5. Cazul de utilizare asignare membru proiect

Actor: singurul actor care poate realiza această operație este project manager-ul
Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de Project Manager
- User-ul să fie situat în pagina dedicată Project Manager-ilor în care se regăsesc informațiile legate de proiectele existente în platformă

Postcondiții:

- Un nou user va fi assignat unui proiect
- Informațiile noului element adăugat pot fi vizibile din pagina utilizatorului care a creat acel element

Flow-ul operației de adăugare proiect:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroducă date
2) Se verifică dacă există un utilizatorul logat să fie Project Manager
    a. Dacă are rolul menționat anterior va fi redirecționat la pagina principală unde se regăsește o listă cu toate proiectele create de acel Project Manager
    b. Dacă utilizatorul are alt rol, pagina menționată anterior nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare proiect
3) Utilizatorul logat selectează proiectul la care dorește să asigneze membru
4) Project Manager-ul apasă butonul de AddMember
5) Platforma oferă o listă cu utilizatorii existenți și care nu au rol de administrator sau Project Manager și care nu au fost încă asignați pe proiectul ales la pasul anterior
6) User-ul este redirecționat la un form unde câmpurile trebuie completate cu datele ce țin de noul proiect
7) User-ul logat va apăsa butonul
    a. AssignMember
        i. Informațiile ajung în baza de date unde o să fie salvate
        ii. Apare denumirea membrului nou assignat în pagina de vizualizare a proiectului la care a fost adăugat ca și membru
    b. Cancel
        i. Modificările efectuate în pașii anteriori sunt contramandate

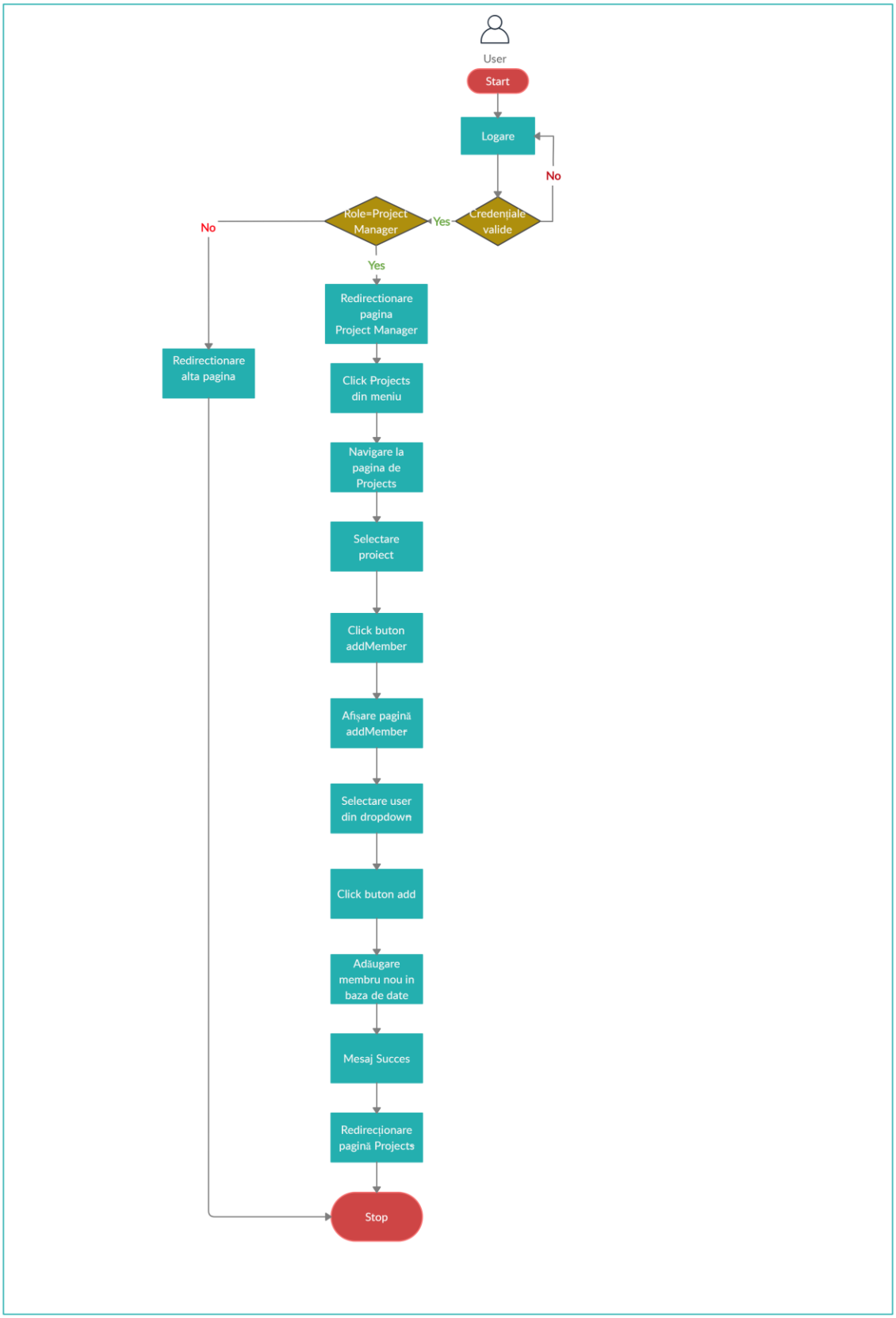


Figura 4.10 Caz de utilizare pentru asignare membru proiect

4.2.2.6. Cazul de utilizare Adăugare Bug

Actori: Project Manager, Developer, Tester

Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de Project Manager/Developer/Tester
- User-ul să fie situat în pagina de Bugs

Postcondiții:

- Un nou bug va fi adăugat în baza de date
- Informațiile noului element adăugat pot fi vizibile din pagina de Bugs a utilizatorului care a creat acel element

Flow-ul operației de adăugare proiect:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroducă date
2) Se verifică dacă există un utilizatorul logat să fie Project Manager/Developer/Tester
    a. Dacă are unul din rolurile menționate anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra bug-urilor
    b. Dacă utilizatorul are alt rol, pagina menționată anterior nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare proiect
3) Utilizatorul selectează proiectul corespunzător unde dorește să adauge un bug nou
4) Utilizatorul apasă butonul de CreateBug
5) User-ul este redirecționat la un form unde câmpurile trebuie completate cu datele ce țin de noul proiect, această pagină conține o listă cu utilizatorii cărora li se pot asigna bug-uri, precum și o listă cu posibilele valori ale bugStatus-ului
6) Utilizatorul completează câmpurile solicitate
    a. Dacă introduce date valide
        i. Butonul de save va fi activat
            1. User-ul apasă acest buton
            2. Datele introduse sunt trimise la server
                a. Dacă informațiile sunt deja existente în baza de date
                    i. Se va trimite in frontend un mesaj de eroare
                    ii. Se revine la pasul 5)
                b. Dacă informațiile din pagină nu sunt deja inserate
                    i. Se adaugă un nou bug în baza de date
                    ii. Se trimite un mesaj de success
        ii. Butonul de cancel este apăsat
            1. Toate modificările sunt contramandate

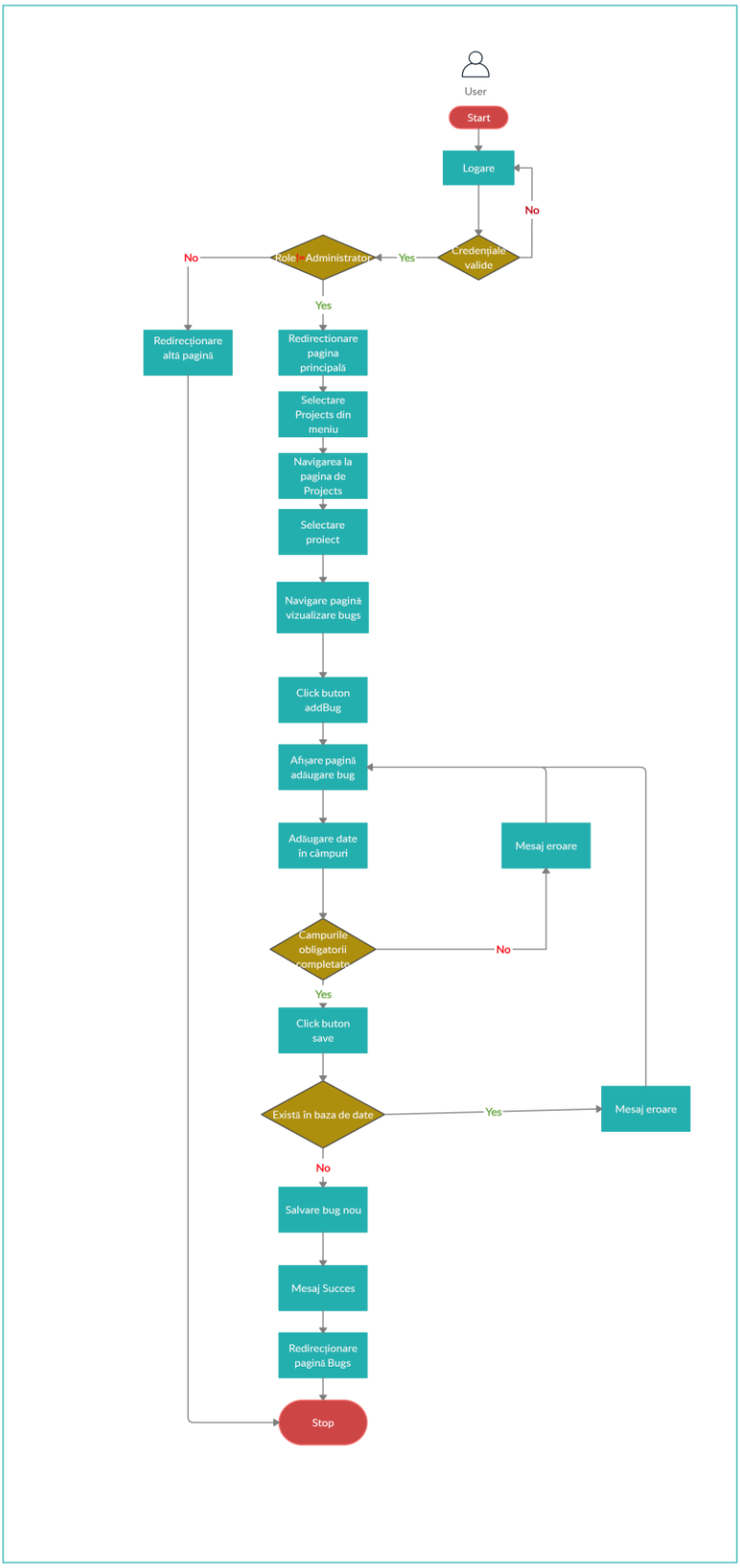


Figura 4.11 Caz de utilizare pentru adăugare bug

4.2.2.7. Cazul de utilizare Editare Bug

Actori: Project Manager, Developer, Tester

Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de Project Manager/Developer/Tester
- User-ul să fie situat în pagina de Bugs

Postcondiții:

- Un bug va fi actualizat în baza de date
- Informațiile actualizate pot fi vizualizate in pagina de Bugs

Flow-ul operației de adăugare proiect:

1) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroducă date
2) Se verifică dacă există un utilizatorul logat să fie Project Manager/Developer/Tester
    a. Dacă are unul din rolurile menționate anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra bug-urilor
    b. Dacă utilizatorul are alt rol, pagina menționată anterior nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare proiect
3) Se selectează un proiect, dacă bug-ul țintă se regăsește în proiectul selectat se trece la pasul următor, altfel se repeat acest pas
4) Se selectează bug-ul de editat
5) Utilizatorul apasă butonul de EditBug
6) User-ul este redirecționat la un form care pune la dispoziție câmpurile cu informațiile actuale ale bug-ului, inlusiv un dropdown cu tranzițiile posibile a câmpului de bugStatus
7) Utilizatorul completează câmpurile solicitate
    a. Dacă introduce date valide
        i. User-ul apasă acest buton
            1. Datele introduse sunt trimise la server
            2. Se salvează modificările bug-ului selectat în baza de date
            3. Se trimite un mesaj de success
        ii. Butonul de cancel este apăsat
            1. Toate modificările sunt contramandate

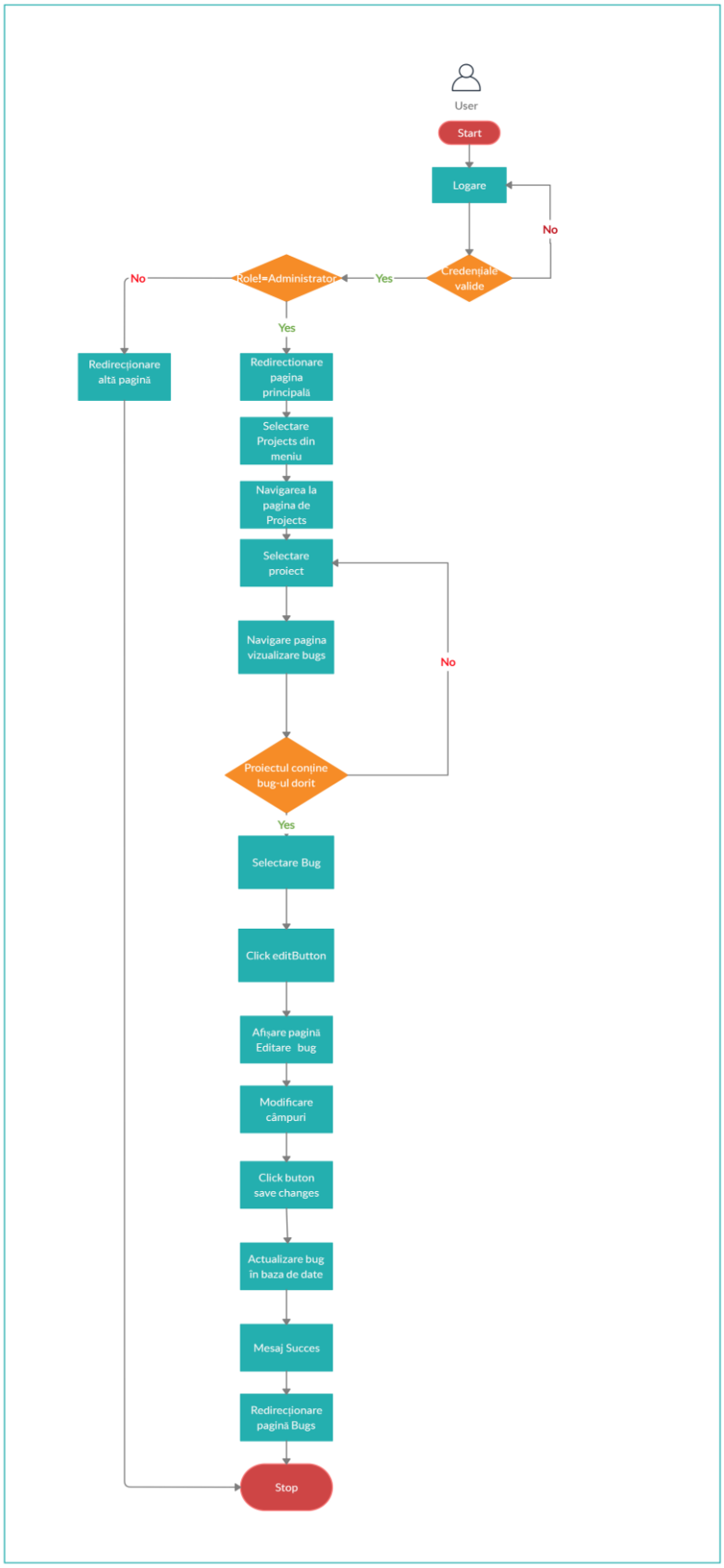


Figura 4.12 Caz de utilizare pentru editare bug

4.2.2.8. Cazul de utilizare setarea unui bug pe closed

Actor: Tester

Precondiții:

- Să existe un user logat în aplicație care să aibă rolul de Tester
- User-ul să fie situat în pagina de Bugs

Postcondiții:

- Un bug va fi actualizat în baza de date
- Informațiile actualizate pot fi vizualizate in pagina de Bugs

Flow-ul operației de adăugare proiect:

8) Se verifică dacă utilizatorul care dorește să se logheze are credențiale existente în baza de date, în caz contrar se va reexecuta acest pas unde utilizatorul va fi nevoit să reintroducă date
9) Se verifică dacă există un utilizatorul logat să fie Tester
    a. Dacă are rolul menționat anterior va fi redirecționat la pagina principală unde se regăsesc elementele grafice de execuție a operațiilor CRUD(Cread-Read-Update-Delete) asupra bug-urilor
    b. Dacă utilizatorul are alt rol, pagina menționată anterior nu îi va fi vizibilă și implicit nu va putea executa operația de adăugare proiect
10) Se selectează bug-ul de editat
11) Se verifică dacă bug-ul se află în una din următoarele stări: FIXED, REJECTED sau DUPLICATED
12) Utilizatorul apasă butonul de EditBug
13) User-ul este redirecționat la un form care pune la dispoziție câmpurile cu informațiile actuale ale bug-ului, inlusiv un dropdown cu tranziția de CLOSED
14) Utilizatorul selectează opțiunea de CLOSED
    a. Dacă introduce date valide
        i. Butonul de save este apăsat
            1. Datele introduse sunt trimise la server
            2. Se editează bug-ul selectat anterior
            3. Se trimite un mesaj de success
        ii. Butonul de cancel este apăsat
            1. Toate modificările sunt contramandate

4.3. Perspectiva tehnologică

4.3.1. Spring Boot

Este un framework pentru realizarea diverselor aplicațiii dezvoltat de Pivotal Software, care a apărut pe piață în 1 octombrie 2002. A fost lansat cu scopul de a elimina neajunsurile programării utilizand Spring: realizarea configurărilor destul de complicate, lipsa unui server încorporat și deja configurat, precum și reducerea dimensiunii codului pentru o implementare mai rapidă a funcționalităților dorite. Un lucru important de menționat este acela că Spring Boot oferă un suport excelent pentru utilizarea de servicii web REST.

Spring Boot oferă o multitudine de caracteristici:

- Bazat pe platforma Java
- Are încorporate serverele Tomcat, Jetty sau Undertow
- Utilizeaza POM.xml-uri ceea ce ușurează configurările maven
- Facilitează creearea rapidă și ușoară a diverselor aplicații web
- simplifică utilizarea altor framework-uri Java: Hibernate, JPA, Struts etc.
- permite scrierea de cod ușor testabil datorită principiului dependency injection. *Dependency injection*/*Inversion of Control*(Inversarea controlului) definește legăturile dintre obiecte evitând operatorul *new*(operatorul pentru crearea unui obiect nou Java), lucru care se poate realiza fie prin injectarea obiectului în constructor, fie prin injectarea lui utilizând un *setter*(o metodă *setter* este o metodă pentru setarea unui câmp dintr-un obiect Java).

4.3.2. Servicii REST

REST(Representational State Transfer) este un tip de arhitectură software care a fost introdusă în anul 2000 de către Roy Fielding, fiind o alternativă a unor servicii deja existente cum ar fi: SOAP, WSDL, RPC etc. Principiile pe care arhitectura REST le utilizează sunt:

- arhitectura client-server: separarea interfeței utilizator de partea de stocare și procesare, lucru care duce la creșterea portabiliății și a scalabilității
- stateless: fiecare request al unui client conține toate informațiile necesare pentru a trata acel request
- cacheable- marcarea datelor din body-ul unui request ca și cacheable sau noncacheable
- interfață uniformă
- layered system- există o ierarhie a componentelor, o componentă nu poate accesa decât layer-ul cu care interacționează direct

Rolul serviciilor REST din cadrul aplicației implementate este deosebit de importantant deoarce furnizează un mijloc de comunicare între aplicația server și aplicația client. Mai precis, legătura dintre cele două aplicații este menținută prin intermediul request-urilor *HTTPS* sau *HTTP* de tip GET, POST, PUT, DELETE etc.

Protocolul HTTPS(Secure Hyper Text Transfer Protocol) reprezintă încapsularea protocolului *HTTP* (Hyper Text Transfer Protocol) folosind protocoalele criptografice SSL(Secure Sockets Layer) sau TLS(Transport Layer Security) cu scopul de a cripta informațiile transmise între un browser și un server pentru a oferi o identificare sigură a server-ului.

4.3.3. Framework-ul Hibernate

În capitolul introductiv din [6] Hibernate ORM, sau pe scurt Hibernate este definit ca persistarea automată și transparentă a diverselor obiecte dintr-o aplicație Java în tabelele dintr-o bază de date relațională, utilizând metadate care descriu maparea dintre obiecte și baza de date.

Oferă numeroase avantaje, printre care:

- Productivitate – codul legat de persistența datelor este cel mai greoi și obositor dintr-o aplicație. Hibernate elimină o mare parte a muncii costisitoare legate de persistență, câștigând mai mult timp pentru implementarea altor elemente
- Mentenabilitate – utilizând Hibernate, codul va fi astfel mai scurt și mai ușor de refactorizat și menținut
- Performanță – ORM a fost construit astfel încât să optimizeze automat operațiile cu baza de date

4.3.4. Typescript

Este un limbaj open source dezvoltat de către compania Microsoft. La bază, Typescript-ul este o nouă formă a JavaScript. Este un limbaj compilat static, care poate fi rulat pe Node Js sau în orice browser care suportă cel puțin ECMAScript 3(ECMA = specificație de limbaj de script cu rolul de a standardiza JavaScript și de a sprijini multiple implementări ale JS) .

Typescript poate fi convertit cu ușurință în JavaScript, oferă o structurare mai bună a codului, precum și utilizarea bibliotecilor JS. S-a optat pentru utilizarea limbajului Typescript deoarece este un limbaj flexibil, există numeroase librării utile pentru dezvoltarea de aplicații software care oferă suport pentru acest limbaj(ex. *Redux*), oferă un suport global pentru refactorizarea elementelor din cod și ajută la identificarea timpurie a problemelor la compilare.

4.3.5. React Typecsript

React este este o librărie JavaScript dezvoltată de Facebook, lansată inițial în 29 mai 2013. În ultimii ani a devenit un framework foarte cunoscut datorită usurinței cu care se scrie cod. Identifică eventuale erori care pot apărea la compilare, lucru care scutește dezvoltatorii de timpul petrecut pentru ca aplicația să efectueze build-ul și să eșueze.

React utilizează abstractizarea componentelor, astfel încât dezvoltatorul trebuie doar să înțeleagă conceptele de “state” si “props” ale unei componente și să știe cum și cand să le folosească.

S-a optat pentru utilizarea React deoarece:

- Este declarativ, adică se pot crea cu ușurință interfețe utilizator interactive, updatează rapid starea unei pagini, actualizând doar componenta a cărei elemente s-au modificat
- reutilizează componente, ceea ce duce la scăderea timpului pentru a realiza o aplicație
- se pot crea aplicații mobile utilizand ReactNative, așadar cunoștințele dobândite învățând React vor putea fi utilizate și cu alt scop decât acela de a crea aplicații web

4.3.6. Redux

Redux este o componentă destinată menținerii și gestiunii stării unei aplicații JavaScript sau a derivatelor sale. Necesitatea utilizării Redux se datorează mai multor factori:

- Aplicațiile single-page au devenit mult mai complexe, ceea ce implică mai multe stări ale aplicației de gestionat
- Cu cât crește mai mult complexitatea unei aplicații, cu atât gestiunea acțiunilor din cadrul aplicației cum ar fi: routarea, selectarea tab-urilor, controlul paginilor ș.a.m.d trebuie realizată mai atent
- Aplicațiile pot scăpa de sub control dacă sunt greu de înțeles și urmărit

Cum este menționat și în [7], premisa de bază din Redux este ideea că toată starea aplicației dezvoltate să fie stocată într-un singur loc denumit **Store**. Logica din spatele Redux este următoarea: Store-ul trebuie să fie notificat când o acțiune a avut loc, ca mai apoi să modifice după logica aplicației starea influențată de evenimentul nou apărut.

4.3.7. Maven

Apache Maven este un instrument de gestiune și înțelegere a proiectelor software. Este un tool care oferă o definiție concisă a unui proiect, oferind o modalitate ușoară de a publica informații despre acesta.

Elementul central din cadrul acestui instrument de build se numește Pom.xml. Acesta este un fișier în care se regăsesc toate dependețele, configurările și plugin-urile proiectului construit. Avantajul semnificativ al utilizării tool-ului este descărcarea automată a librăriilor și gruparea jar-urilor rezultate.

4.3.8. Tomcat

Este un server creat pentru aplicațiile web care au la bază ca și limbaj Java, fiind dezvoltat de Apache Software Foundation. Deși este utilizat în deosebi ca server de aplicație, acesta poate fi configurat să funcționeze ca și un server obișnuit sau împreună cu serverul Apache HTTP. Este simplu și ușor de administrat, fiind deja integrat în aplicațiile Spring Boot și Java EE.

Deoarece aplicația dezvoltată este considerată ca fiind una de dimensiuni mici, Tomcat este varianta cea mai bună pentru dezvoltarea aplicațiilor de acestă mărime. Principalele avantaje sunt acelea că vine deja configurat în cadrul aplicațiilor Spring Boot și faptul că rulează pe Java Virtual Machine lucru care îl face independent de platforma pe care este instalat și utilizat.

4.3.9. Axios

Este un client HTTP bazat pe *promises*(promisiuni = finalizarea cu succes sau eșec al unei operațiuni și valoarea rezultată în urma efectuării operațiunii) ușor de utilizat, dezvoltat atât pentru browsere, cât și pentru Node.js. Un avantaj al utilizării Axios este acela de a scrie un cod mai lizibil și cu comportament asincron datorită faptului că se bazează pe *promises*. Mai mult oferă posibilitatea de a intercepta și anula solicitări, oferind și un nivel de protecție mai ridicat împotriva cererilor cross-origin falsificate.

În cadrul aplicației implementate, s-a utilizat Axios pentru a trimite request-uri din aplicația client spre server și pentru a procesa datele primite ca și răspuns. Pentru utilizarea acestui consumator de request-uri a fost necesară instalarea lui în cadrul proiectului React utilizând comanda ***npm axios***.

4.3.10. JavaMailSender

Aplicația de managemnt al bug-urilor oferă posibilitatea utilizatorilor de a fi notificați prin email atunci când apar anumite evenimente(ex. Un utilizator a fost asignat pe un nou proiect, un bug a fost actualizat de către altă persoană, un bug a fost marcat ca și *closed*(închis) de către tester ș.a.m.d.).Pentru acest lucru a fost necesară implementarea unui serviciu de mail în aplicația server. Deoarece componenta de server este scrisă utilizând framework-ul Spring Boot, a fost configurată o instanță a JavaMailSender pentru a satisface această cerință de emitere a email-urilor.

JavaMailSender este o interfață care extinde JavaMail și care permite comunicarea prin email între diverși utilizatori. Implementarea din producție al acestei interfețe este JavaMailSenderImpl care necesită configurarea următoarelor proprietăți: host-ul, portul corespunzător protocolului SMTP[13], username-ul și parola persoanei care emite mail-ul (în cazul nostru credențialele contului de gmail asociate aplicației prezentate în secțiunea 5.1.1.3) și tipul protocolului de transport.

Pentru a putea utiliza pachetul javax.mail în care se regăsește și JavaMailSender a fost necesară adăugarea dependenței de mail în fișierul pom.xml.[14]

4.3.11. JWT

JWT[15](JSON Web Token) este un element utilizat pentru autentificarea utilizatorilor în cadrul diverselor aplicații. Dacă un utilizator introduce credențiale valide, operația de logare se execută succes și un astfel de token în format JSON va fi generat.

[13] https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/
[14] https://mvnrepository.com/artifact/javax.mail/mail/1.4
[15] https://jwt.io/

Acest token va conține informația criptată a utilizatorului(adresă de email și permisiuni), informație care va însoți fiecare cerere executată de client după autentificare. Acestă *JWT* va fi utilizat în setarea header-ului de *Authorization* a request-urilor trimise spre componenta de server.

Server-ul pentru fiecare *request* primit testează dacă valoarea token-ului este validă, și în funcție de operațiile solicitate, server-ul realizează verificări suplimentare pentru a se asigura că utilizatorul logat are permisiunile necesare pentru execuția operațiilor respective.

Capitolul 5. Proiectare de Detaliu si Implementare

În cadrul acestui capitol sunt descrise elementele componenete ale sistemului implementat, rolul acestora și modul în care interacționează. De asemenea vor fi prezentate și o serie de diagrame (diagrama de deployment, diagrama bazei de date, structura componentelor diferitelor layere ș.a.m.d) pentru înțelegerea modului în care aplicația funcționează.

5.1. Arhitectura Sistemului

Platforma implementată a fost construită utilizând arhitectura de tip CLIENT-SERVER. Componenta de CLIENT este scrisă utilizând React typescript și Redux(pentru o gestiune mai eficientă a stării aplicației) și este structurată pe module. Componenta de SERVER utilizează Spring Boot și utilizează o arhitectură de tip layers. Rolul deținut de componenta de SERVER este acela de a procesa request-urile care vin din partea de CLIENT, a extrage datele necesare, a filtra conținutul obținut și de a livra răspunsul corespunzător request-ului efectuat înapoi la CLIENT.

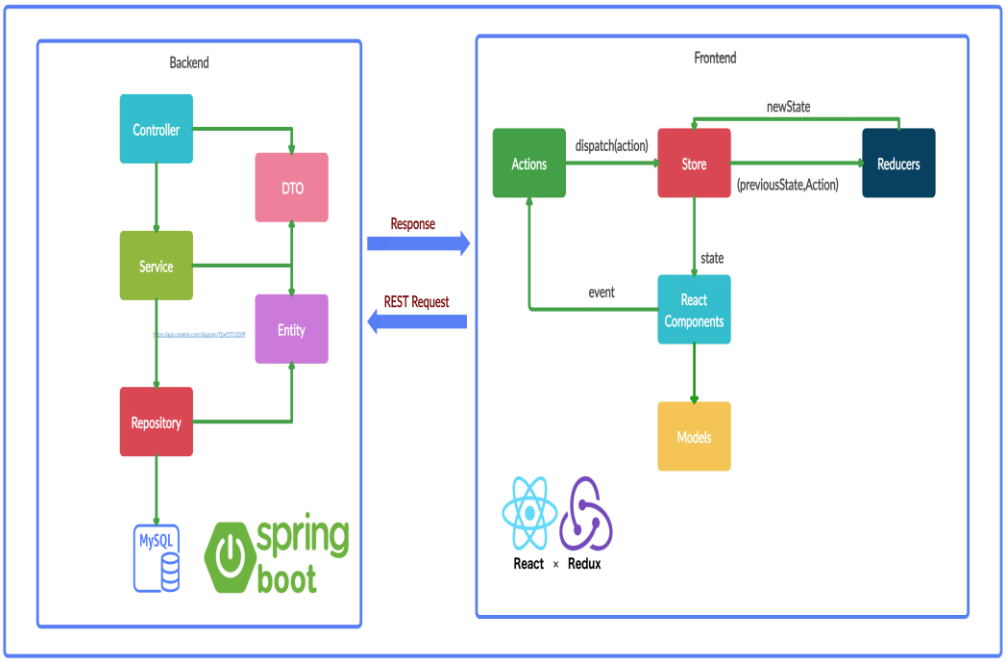


Figura 5.1 Arhitectura generală a aplicației

5.1.1. Server side

Așa cum a fost menționat în capitolul 5.1, această componentă are o importanță deosebită deoarece are rolul de a deservi cererile clientului, fiind și singura componentă care interacționează cu baza de date.

5.1.1.1. Prezentare generală

Elementele componente ale arhitecturii Layer implementate în cadrul sistemului de management al bug-urilor sunt: API Layer, Business Logic, Persistance Layer și Database Layer. Partea de Presentation Layer din cadrul arhitecturii tradiționale este reprezentată în acest caz de aplicație client efectivă.

Toate aceste layere comunică între ele utilizând fie obiecte de tip Entity, fie obiecte de tip DTO a căror importanță și necesitate vor fi detaliate în capitolele următoare.

Aplicația având la bază arhitectura Layers, fiecare nivel superior va interacționa doar cu layer-ul imediat inferior. Layer-ul superior va avea acces doar la metodele expuse de interfața layer-ului inferior, fiind independent de implementarea efectivă a metodelor pe care dorește să le utilizeze. Aceste interfețe reprezintă un contract între layer-ele aplicației, astfel încât orice modificări de implementare de la un layer inferior nu va afecta apelul metodelor din layer-ul superior.

5.1.1.2. Nivelul de API(Controllerele)

În cazul aplicației implementate, aceste clase sunt adnotate cu @RestController și au rolul de a prelua request-urile HTTP emise de client, de a le procesa și de a transmite un răspuns clientului în formatul așteptat de acesta, tot prin intermediul protocolului HTTP.

Deoarece în cadrul aplicației se utilizează protocolul HTTP (Hypertext Transfer Protocol), Spring Boot utilizează adnotările @GetMapping, @PostMapping, @DeleteMapping, @PutMapping etc. pentru a semnala tipul de request care urmează să fie primit de metoda deasupra căreia se regăsește o astfel de adnotare.

Fiind o arhitectură Layered, acest nivel de API poate utiliza doar nivelul imediat inferior, și anume nivelul de servicii. Când un request primit de la client se suprapune peste semnătura unei metode dintr-un Controller Rest, acea metodă se apelează. Datele primite de Controller în cazul în care request-ul are atașat și un body(cum este în cazul request-urilor POST și PUT) sunt mapate în obiecte de tip DTO și transmise mai departe nivelului de servicii pentru a realiza procesarea lor. După ce partea de Business Logic a fost finalizată, Controller-ul Rest primește un răspuns pe care îl formatează pentru a respecta standardul HTTP și îl trimite mai departe clientului.

O notă deosebit de importantă când vine vorba de Controllere Rest în Spring este accea că este necesară utilizarea adnotării @CrossOrigin pentru metode sau pentru întreg Controller-ul. Acestă adnotare permite tratarea request-urilor provenite din alte origini, în cazul aplicației implementate request-urile o să provină de la client, prin intermediul applicației React. Lipsa acestei adnotări este o greșeală frecventă și nu permite aplicațiilor web să acceseze resursele regăsite în altă origine.

```
@CrossOrigin
@GetMapping(value = "/bugs")
public List<BugDTO> findAllBugs()
```

Figura 5.2 Exemplu de utilizare al adnotărilor în Controllere

În Figura 5.3 este prezentată o porțiune din aplicație în care se poate vizualiza interacțiunea dintre nivelul de *API(Controllere)* și nivelul imediat inferior: nivelul de *Service*.

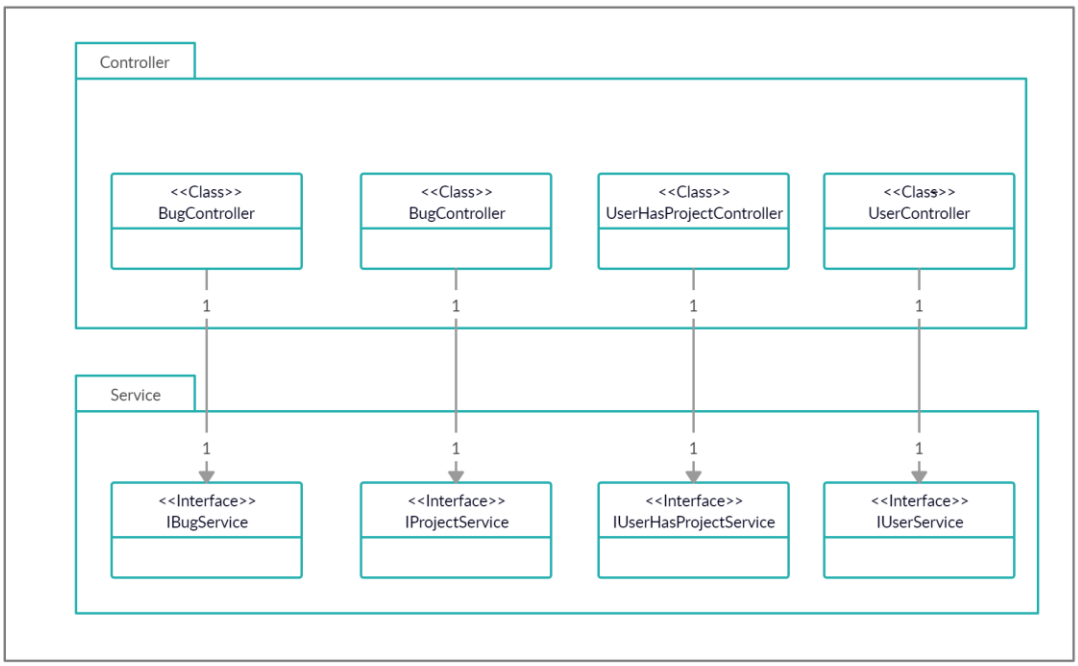


Figura 5.3 Fragment din structura nivelului de API

5.1.1.3. Nivelul de service

Acesta este nivelul în care este conținută logica aplicației, de la validare și convertirea datelor, până la apelul metodelor din repository. Clasele de implementare se regăsesc în pachetul *serviceImplementation*, iar interfețele sunt situate în pachetul *service*.

Și în cadrul nivelului de Bussiness Logic au fost utilizate interfețe pentru expunerea metodelor relevante nivelului superior: API.

Elementele necesare pentru ca Spring Boot să poată utiliza clasele de service au fost adnotările: @Service înaintea unei clase - pentru a indica faptul că deține logica de bussines și @Autowire – pentru a injecta obiectele din pachetul de *repository*.

Acest layer se ocupă și de trimitere de email-uri cu mesaje personalizate persoanelor care le-au fost asignate bug-uri noi sau cărora li s-au actualizat informațiile unor bug-uri deja existente cu scopul de a menține întreaga echipă la zi cu ce se întâmplă în aplicație.

Pentru a putea trimite email-uri din aplicație Spring a fost necesară adăugarea unor elemente de configurare în fișierul *application.properties*, cum poate fi vizualizat în Figura 5.4:

```
#mail configuration
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.properties.mail.transport.protocol=smtp
spring.mail.properties.mail.smtp.port=25
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.username="bugtrackingapplication@gmail.com"
spring.mail.password="bugTracker!2020"
spring.data.rest.return-body-on-update=true
```

Figura 5.4 Configurearea server-ului de mail în Spring Boot

Cum am menționat mai sus, în partea de bussiness logic se realizează și partea de validare a datelor și de convertire a acestora dintr-un model în altul.

Clasele de validare se regăsesc în pachetul de *validation* și în cadrul metodelor de validare se verifcă date toate câmpurile din DTO primit ca și parametru respectă cerințele: nu sunt câmpuri goale și au o anumită dimensiune sau un anumit format.

Clasele de mapare a obiectelor de tip entity în obiecte de tipul DTO și viceversa sau maparea dintre un obiect de tip projection în DTO se regăsesc în pachetul intitulat *mapper*.

Comunicarea la nivelul de Bussines Logic se face utilizând obiecte DTO și obiecte de tipul enity. Mai precis, metodele de service când sunt apelate din controller primesc ca și parametri DTO-uri pe care le iau și le procesează, iar dacă datele din aceastea sunt valide, le convertesc în obiecte entity pentru a le transmite mai departe nivelului de repository.

În Figura 5.5 se poate vizualiza legătura dintre interfețele de servicii și implementările lor, precum și legătura dintre clasele de servicii și interfețele de repository.

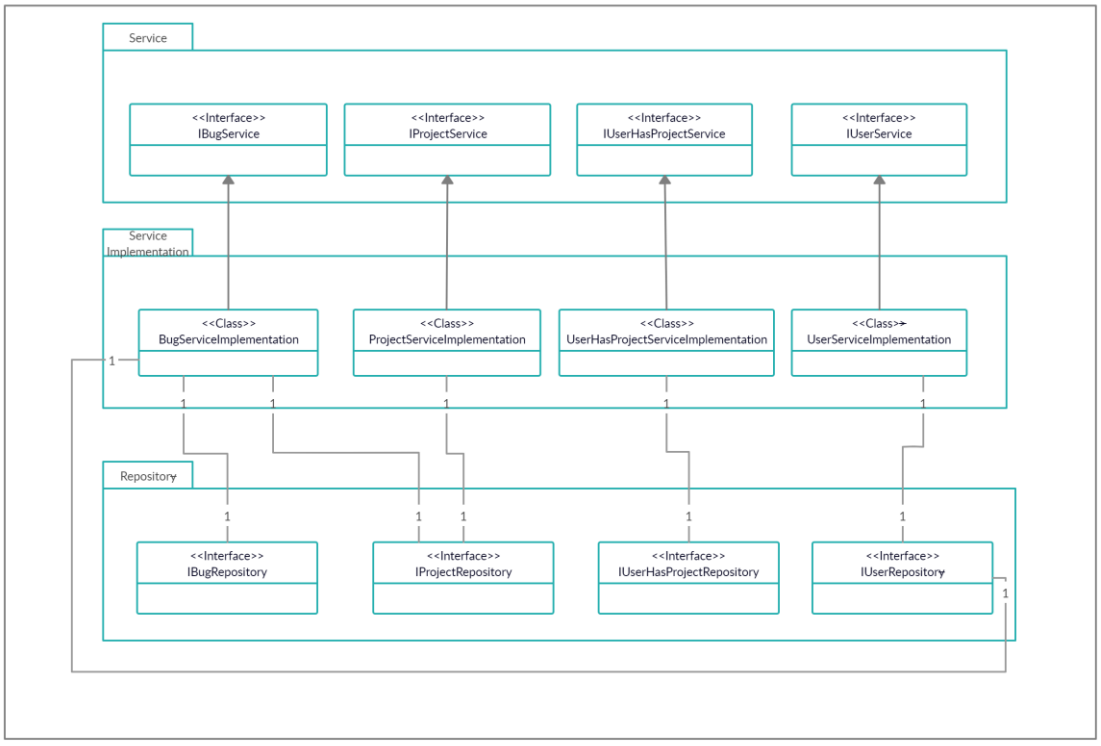


Figura 5.5 Fragment din structura nivelului de *Service*

5.1.1.4. Nivelul de repository

Acesta este singurul model care interacționează cu baza de date și este destinat realizării acțiunilor de save, update, delete, find, precum și executarea de interogări necesare obținerii diferitelor date sau de persistarea unor date noi.

Pachetul de clase corespunzător acestui layer se numește *repository*. Aici se regăsesc interfețele de repository asociate fiecărei clase corespondente unui tabel din baza de date.

Acesta este nivelul în care se utilizează JPA(Java Persistence Layer) prin implementarea interfeței JpaRepository. JPA vine în ajutorul dezvoltatorilor prin expunerea metodelor de bază deja implementate necesară persitării și obținerii datelor din baza de date.

Pe lângă metodele de bază(save, delete,find, ...) JPA permite scriere de interogări particulare care ușurează munca celor care utilizează Hibernate. În cadrul aplicației de bug tracking s-au utilizat atât Query Methods, cât și a *native queries*(interogări native).

Query Methods nu mai necesită scrierea efectivă a query-urilor de tip SELECT în cazul în care trebuie realizate căutări obișnuite în baza de date după anumite coloane ale unui în tabel, acest select fiind apelat implicit atunci când sunt utilizate în repository metodele care respectă patternul: *find*DenumireClasaEnitity*By*FieldClasa. Un exemplu pentru clasa User care are ca și câmpuri: username și password dacă dorim să obținem utilizatorii care au un anumit username și o anumită parolă se poate scrie metoda asociata: *findUserByUsernameByPassword(username,password).*

Query-urile native au fost utilizate atunci când au fost necesare filtrări mai complexe ale datelor. Pentru a scrie o astfel de interogare, este necesară plasarea adnotării @Query înaintea unei metode și amplasare interogării în corpul adnotării.

În cadrul pachetului de repository, interfețele conținute utilizează doar clase care conțin adnotarea de @Enity sau proiecții. Mai mult detalii despre proiecții în secțiunea 5.1.1.7.

Pentru a utiliza Hibernate, a fost necesară scriere unei configurații în fișierul de application.properties ca și în Figura 5.6:

```
#Hibernate configuration
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto = update
```

Figura 5.6 Configurarea Jpa Hibernate în Spring Boot

5.1.1.5. Interacțiunea dintre componente

Cum am menționat și în secțiunea 5.1.1, arhitectura aplicației server este una Layered. În acest caz, nivelele superioare ale aplicației apelează nivelele imediat inferioare: nivelul de API apelează nivelul de servicii,iar nivelul de servicii apelează nivelul de repository.

În Figura 5.7 este prezentat o porțiune din sistemul dezvoltat, corespunzătoare adăugării unui nou *bug* cu scopul de a evidenția modul în care a fost structurată aplicația.

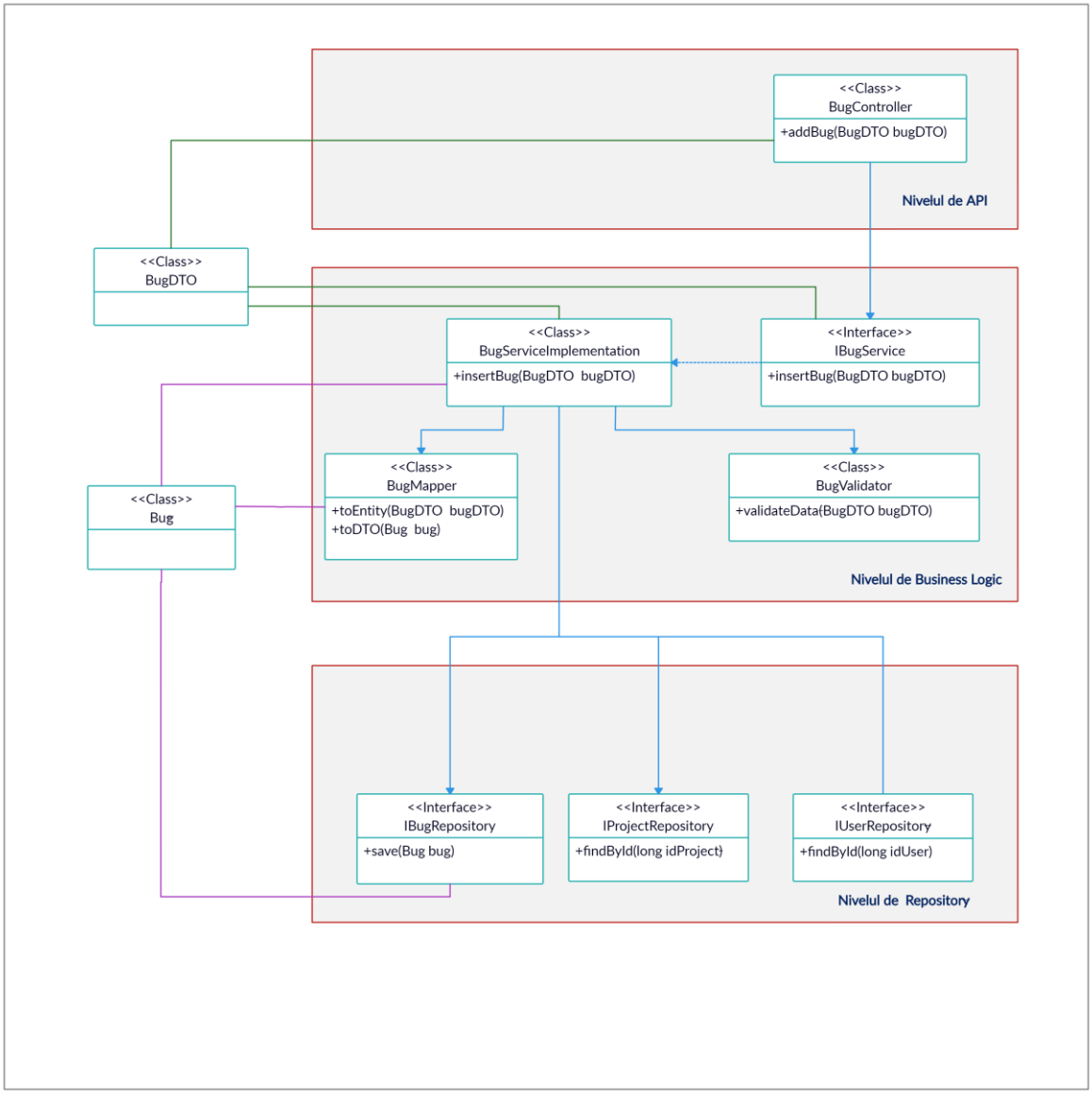


Figura 5.7 Interacțiunea între nivelurile aplicației pentru operația de adăugare bug

În Figura 5.8 este descrisă diagramă de secvență pentru operația de creare a unui *bug* nou, unde sunt prezentate pas cu pas, clasele utilizate și metodele apelate pentru finalizarea unei astfel de operații.

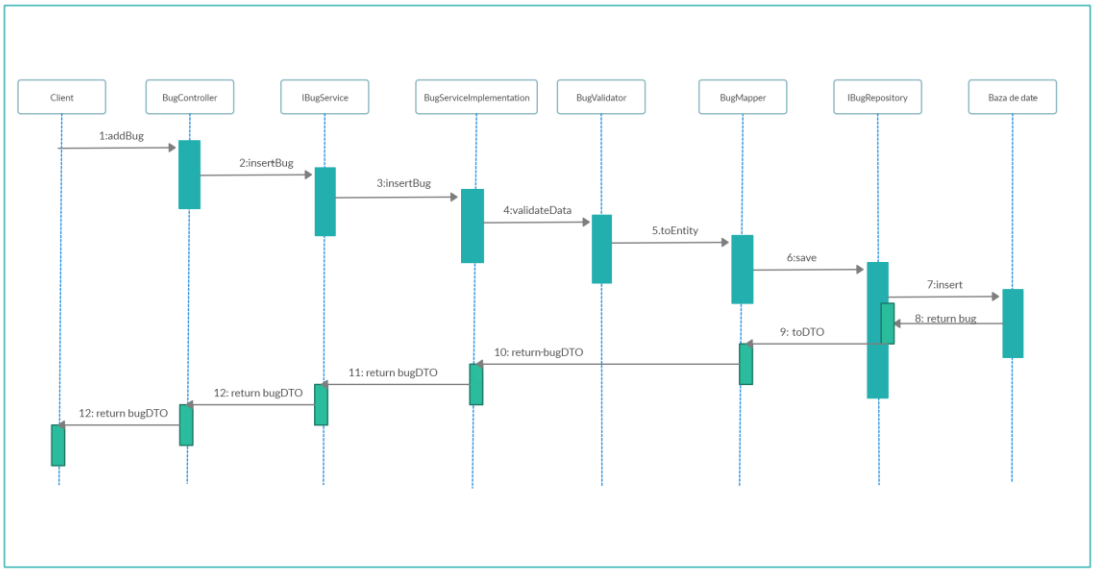


Figura 5.8 Diagrama de secvență pentru operațiunea de adăugare *bug*

5.1.1.6. Structura bazei de date

Pentru a persista datele asociate aplicației, s-a utilizat o bază de date relațională SQL. Generarea tabelelor a fost realizată utilizând Hibernate în Spring-Boot, însă pe lângă configurările pentru JPA prezentate în Figura 5.2 a mai fost necesară adăugarea configurației pentru SQL :

```
#Configure the connection to MySql WorkBench
database.ip = ${MYSQL_IP:localhost}
database.port = ${MYSQL_PORT:3306}
database.person = ${MYSQL_USER:root}
database.password = ${MYSQL_PASSWORD:root}
database.name = ${MYSQL_DBNAME:bug_tracker_schema}


#Spring configuration for database
spring.datasource.url = jdbc:mysql://${database.ip}:${database.port}/${database.name}?allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&useJDBCC
spring.datasource.username = ${database.person}
spring.datasource.password = ${database.password}
```

Figura 5.9 Configurarea bazei de date MySQL și a conectării cu MySQL Workbench

Pentru vizualizarea conținutului bazei de date, a fost necesară realizarea conexiunii aplicației Spring cu instrumentul de vizualizare a bazelor de date MySQL Workbench.

În design-ul bazei de da s-a urmărit evitarea creării de cicluri între tabele, țintindu-se spre creare unei structuri de arbore. Astfel reducându-se posibilitatea de pierderea datelor sau de stocare a informațiilor eronate. Pentru evitarea problemelor menționate anterior, în etapa de design al bazei de date s-a utilizat procesul de *normalizare* (reprezintă optimizarea bazei de date prin minimizarea redundanței datelor și

a anomaliilor de adăugare, modificare și ștergere ale acestora). În continuare vor fi prezentate condițiile formelor normale pe care baza de date dezvoltată de îndeplinește.

Baza de date realizată se regăsește în forma normală 1, deoarece fiecare atribut al unui tabel primește o singură valoare atomică(nu există grupuri de date în câmpuri), iar înregistrările din tabele sunt unic identificate prin cheia primară.

Forma normală 2 cere ca baza de date să fie în forma normală 1 și elementele din tabele să fie total dependente de cheia primară . Baza de date dezvoltată respectă aceste condiții, deoarece fiecare tabel are o cheie primară formată dintr-un singur atribut.

Forma normală 3 este îndeplinită dacă baza de date se află în forma normală 2, dacă toate atributele non-cheie ale unei relații sunt dependente doar de *cheia candidată*(orice cheie din relație) și dacă nu există dependințe tranzitive. Baza de date implementată respectă aceste cerințe.

Baza de date dezvoltată se regăsește în forma normală Boyce-Codd(versiune mai restrictivă a formei normale 3) deoarece se regăsește în forma normală 3, iar toate atributele tabelelor sunt dependente în totalitate de o singură cheie: cheia primară.

Elementele centrale ale bazei de date sunt:

- Tabelul bug – acest tabel conține toate informațiile asociate unui bug, de la denumire până la status, severitate și persoana care a creat bug
- Tabelul de proiecte – acest tabel conține informațiile de bază legate de un proiect
- Tabelul de utilizator – un tabel indispensabil în orice aplicație

Pe lânge elementele principale menționate mai sus, a fost nevoie de crearea unor tabele auxiliare necesare pentru soluționarea relațiilor Many-to-many dintre tabele înrudite. Aceste tabele suplimentare sunt:

- User_has_permission - face legătura între un utilizator și permisiunile sale
- User_has_project - acest tabel este utilizat pentru a descrie relația dintre un utilizator și proiectele în care acesta este membru

Există câteva tabele care conțin informații suplimentare asociate elemntelor principale ale bazei de date:

- Notification - conține informațiile despre evenimentele pentru care un utilizator trebuie să fie atenționat
- User_history - necesar pentru persistarea activității utilizatorului în cadrul aplicației
- Bug_history - pentru logarea modificărilor aduse unui bug
- Comment - acest taebel este necesar pentru stocarea informațiilor fișierelor atașate unui bug

În Figura 5.1 este prezentat design-ul bazei de date a sistemului dezvoltat:

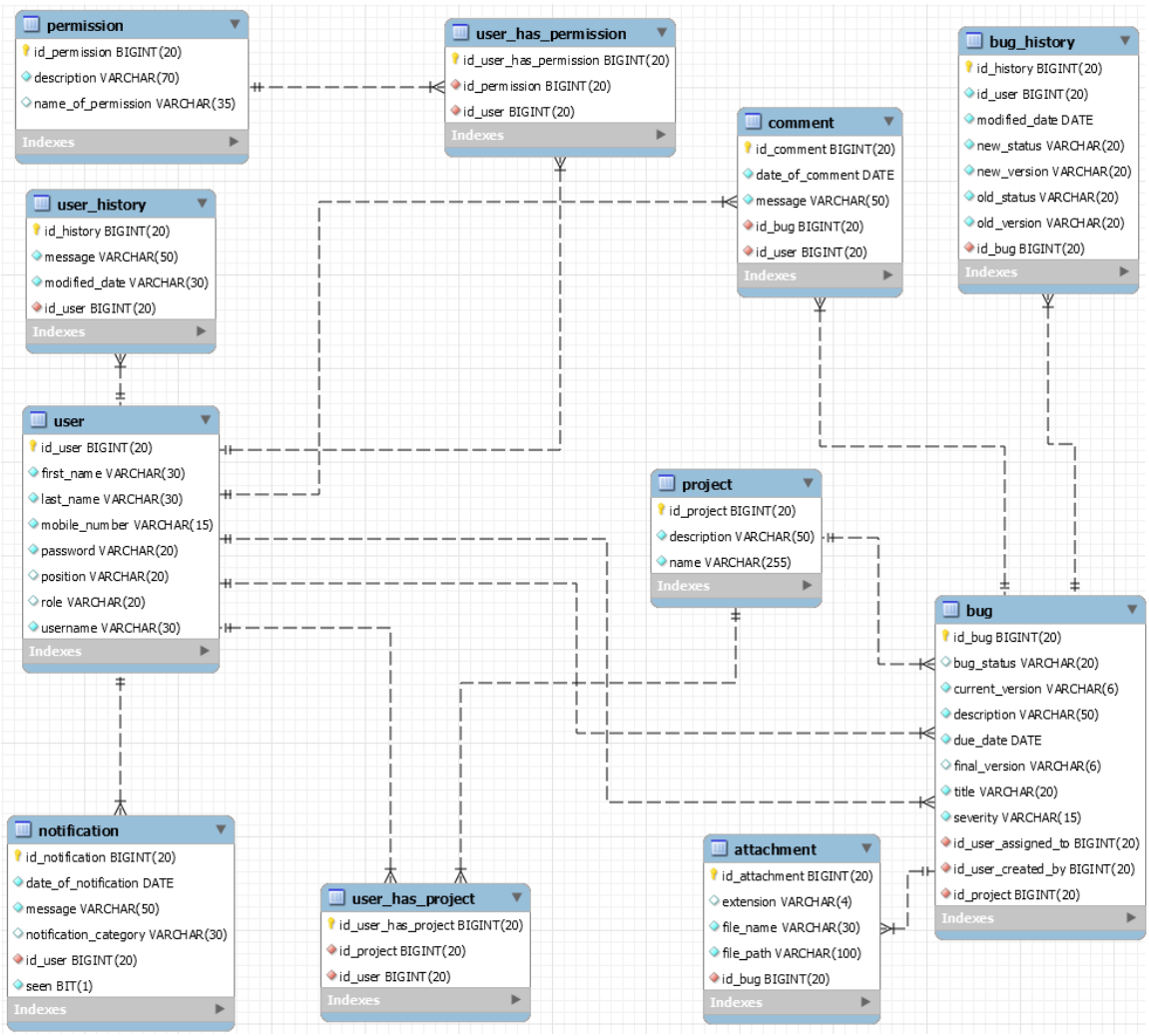


Figura 5.10 Structura bazei de date

5.1.1.7. DTO

Obiectele DTO(Data Transfer Object) sunt obiecte care au rolul de a încapsula date, aceste obiecte fiind utilizate pentru a transfera informații de la un sistem la altul.
În contextul aplicației de bug tracking, DTO-urile reprezintă mesajul de comunicare dintre CLIENT și SERVER.

Avantajul principal al utilizării unuor astfel de obiecte este faptul că ajută la reducrea cantității de informație care trebuie pasată între sisteme. Un alt avantaj semnificativ reprezintă faptul că într-un DTO se pot acumula informațiile semnificate din mai multe clase de tip Enity. Acest lucru reduce numărul de request-uri pe care un client ar trebui să le realizeze atunci când dorește informații care se regăsesc stocate în mai multe tabele din baza de date, dar utilizând DTO-uri acele informații sunt culese într-un astfel de obiect și un singur request va fi necesar pentru obținerea datelor.

5.1.1.8. Entity

Deoarece nivelul de repository utilizează Hibernate, pentru a putea manipula datele din baza de date a fost necesară crearea de enitități JPA care sunt *POJO*-uri(Plain Old Java Objcet) ce reprezintă date care pot fi persistate. Câmpurile declarate în aceste clase vor reprezenta și coloanele tabelelor din baza de date.

Adnotările utilizate pentru definirea enităților JPA sunt:

- @Entity – prin utilizarea ei se menționează că datele asociate POJO-ului marcat cu acestă adnotare doresc a fi socate în baza de date
- @Table – este utilizată pentru a stabili denumirea tabelului din baza de date în cazul în care denumirea enității și a tabelului nu corespund
- @Id – definește cheia primară a tabelului asocit POJO-ului
- @GeneratedValue – echivalentul auto-increment-ului din SQL
- @Column – specifică faptul că elemntul clasei asupra căruia se aplică corespunde unei coloane din baza de date
- @OneToMany și @ManyToOne – aceste adnotări sunt utilizate pentru a reprezenta relațiile de tip *One-to-many* dintre tabele unei baze de date
- @OneToOne – utilizată pentru a specifica relații *One-to-one* ale unei baze de date
- @Enumerated – pentru a persita valori de tip enum

Important de reținut în cazul enităților JPA sunt următoarele: trebuie neapărat să conțină o cheie primară(definită cu ajutorul @Id), să conțină un constructor fără parametri, iar declararea claselor marcate cu adnotarea @Entity să nu fie marcate *final*(previne moștenirea claselor în Java).

5.1.1.9. Spring Data JPA Projections

Aceste proiecții sunt interfețe care conțin doar semnături ale metodelor de get utilizate pentru culegerea informațiilor la execuția unei interogări ai cărui rezultat nu poate fi mapat pe o clasă deja existentă. O altă situație în care se dorește utilizarea lor este atunci când se dorește omiterea unor proprietăți considerate irelevante în anumite cazuri.

5.1.1.10. Clasa de excepție

În cadrul layer-ului de servicii, atunci când se efectuează diverse verificări ale datelor, în cazul în care datele nu respectă un anumit format sau au fost corupte este necesar ca utilizatorul să trimită layer-ului următor un mesaj sugestiv care să evidențieze problemele apărute la procesare. Pentru acest lucru, aplicația de management al bug-urilor utilizează o clasă care extinde clasa *Exception* din Java și a cărui constructor primește ca și parametru un mesaj care descrie problema întâlnită la procesarea datelor. Pentru o utilizare mai eficientă și mai clară, aceste mesaje reprezintă elementele unei enumerări.

5.1.1.11. JWT

În cadrul sistemului dezvoltat, pentru utilizarea a *JSON Web Tokens* a fost necesară implementarea unuei clase de configurare care să se specifice tipurile de request

acceptate și serviciile oferite, o clasă care să filtreze cererile venite din partea clientului, precum și o clasă care să genereze *token*-uri(utilizând o cheie secretă), care să conțină credențialele și permisiunile utilizatorului care dorește să se logheze.

Pentru utilizarea metodelor necesare utilizării acestei forme de autentificare a utilizatorilor prin token și parole criptate, a fost neccesară adăugarea dependențelor de *Spring Security* în cadrul proiectului.

5.1.1.12. HTPPS

Pentru utilizarea protocolului *HTTPS* între browser și aplicația Spring Boot a fost necesară creare unui certificat SSL, ca și în Figura 5.11

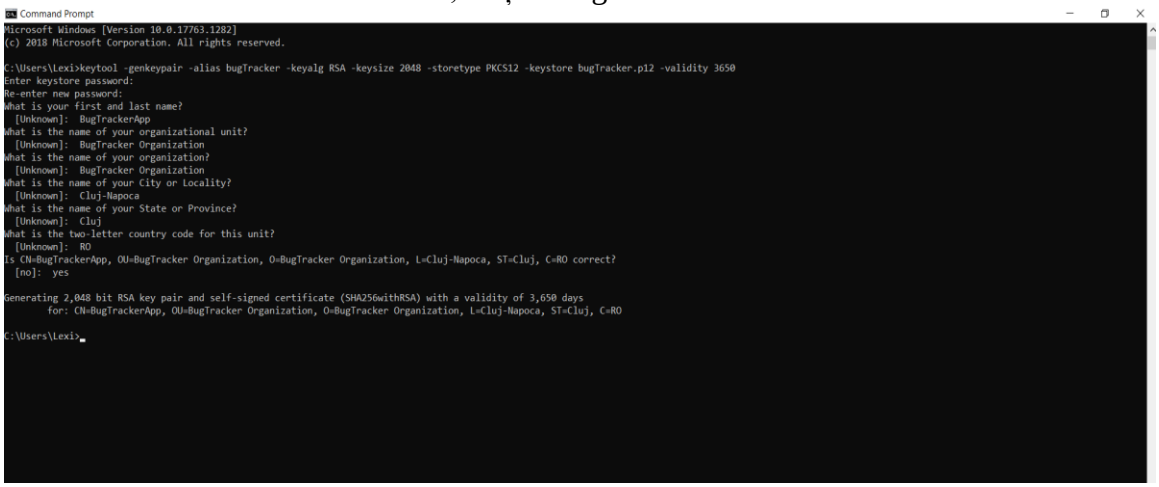


Figura 5.11 Genereare certificatului SSL bugTracker.p12

În continuare, în cadrul fișierului *application.properties* a fost necesară introducerea informațiilor de configurare, ca și în Figura 5.12:

```
server.port=8443
server.ssl.enabled=true
server.ssl.key-store: src/main/resources/bugTracker.p12
server.ssl.key-store-password: bugTracker
server.ssl.keyStoreType: PKCS12
server.ssl.keyAlias: bugTracker
```

Figura 5.12 Configurarea protocolului HTTPS în Spring Boot

După realizarea configurărilor, a fost implementată o metodă cu rolul de a redirecționa toate cererile *HTTP*(acest protocol utilizează portul 80) spre protocolul *HTTPS*(utilizează portul 443).

În final a fost necesară adăugarea certificatului SSL generat anterior în lista certificatelor salvate de browser-ul utilizat pentru rularea aplicației: *Chrome*.

5.1.2. Client side

Acesta este componenta cu care utilizatorii interacționează, având un rol deosebit de important. De asemenea, este componenta care comunică cu aplicația de Spring Boot descrisă în secțiunea 5.1.1.

5.1.2.1. Prezentare generală

Acestă parte a platformei platformei de bug tracking a fost realizată utilizând React și Redux(pentru menținerea și gestiunea stării apliației).

Componenta centrală al aplicației React este App.tsx. Aici se regăsec declarațiile paginilor care sunt afișate utilizatorilor, rutele asociate lor, precum și condițiile care trebuie îndeplinite pentru ca un anumit utilizator să acceseze o pagină oarecare.

Spre deosebire de aplicația Spring Boot care este organizată pe layere, în cadrul aplicației client, fiecare componentă reprezintă un modul. Toate elementele sunt grupate în funcție de zona ocupată din pagină. Mai multe detalii despre împărțirea unei pagini în secțiunea 5.1.2.2.

Pentru ca aplicația să fie funcțională, clientul trebuie să comunice cu server-ul. Acestă comunicare se realizează printr-un schimb de mesaje în format JSON între cele două enitități prin utilizarea protocolului HTTP.

5.1.2.2. Descrierea componentelor grafice

Elementele principale ale unei pagini sunt:

- app bar-ul aplicației
- side bar-ul aplicației
- componenta principală al fiecărei pagini (conținutul efectiv al paginii)
- elementele de tip pop-up – exemplu modal-ul pentru adăugare bug

Pentru fiecare utilizator s-a creat o pagină separată care să adune la un loc toate componentele mai mici necesare efectuării acțiunilor asociate persoanei logate. De asemenea, deoarece există numeroase elemente în fiecare pagină care necesită stilizare, aplicația menține toate aceste stiluri în componente separate(fiecare pagină are propriul stil asociat), grupate în modulul de styles.

Tot în cadrul componentelor se regăsesc funcții de JavaScript care au rolul de a efectua diferite procesări ale datelor prezente în interfață: sortări, filtre de căutare, upload de fișiere, download de fișiere și multe altele.

Orice pagină are asociată o stare. Prin stare se înțelege valorile, datele conținute de o componentă la un moment dat. De fiecare dată când un elemente din interfață își modifică valoarea, este necesără și modificarea stării asociate acelui obiect. Datorită faptului că o componentă are multe elemente a căror stare trebuie gestionată, în implementare s-a utilizat Redux. Utilizând Redux, întreaga stare a aplicației se află stocată într-un singur loc: *Store* și singura modalitate de actualizarea a stării este prin emiterea de acțiuni și specificarea modului în care aceste *acțiuni* produc modificări utilizând *Reducers*. Mai multe detalii despre componentele Redux în secțiunile 5.1.2.5, 5.1.2.6 și 5.1.2.7.

În Figura 5.13 este prezentată interacțiunea dintre componentele care sunt afișate dacă utilizatorul logat este manager de proiect. Componenta principală este ProjectManagerPage în care sunt conținute toate celelalte componente mici care sunt afișate în funcție de diversele cereri solicitate de utilizator. Figura de mai jos nu descrie însă și managemntul stării aplicație, acesta fiind descris în secțiunile următoare.

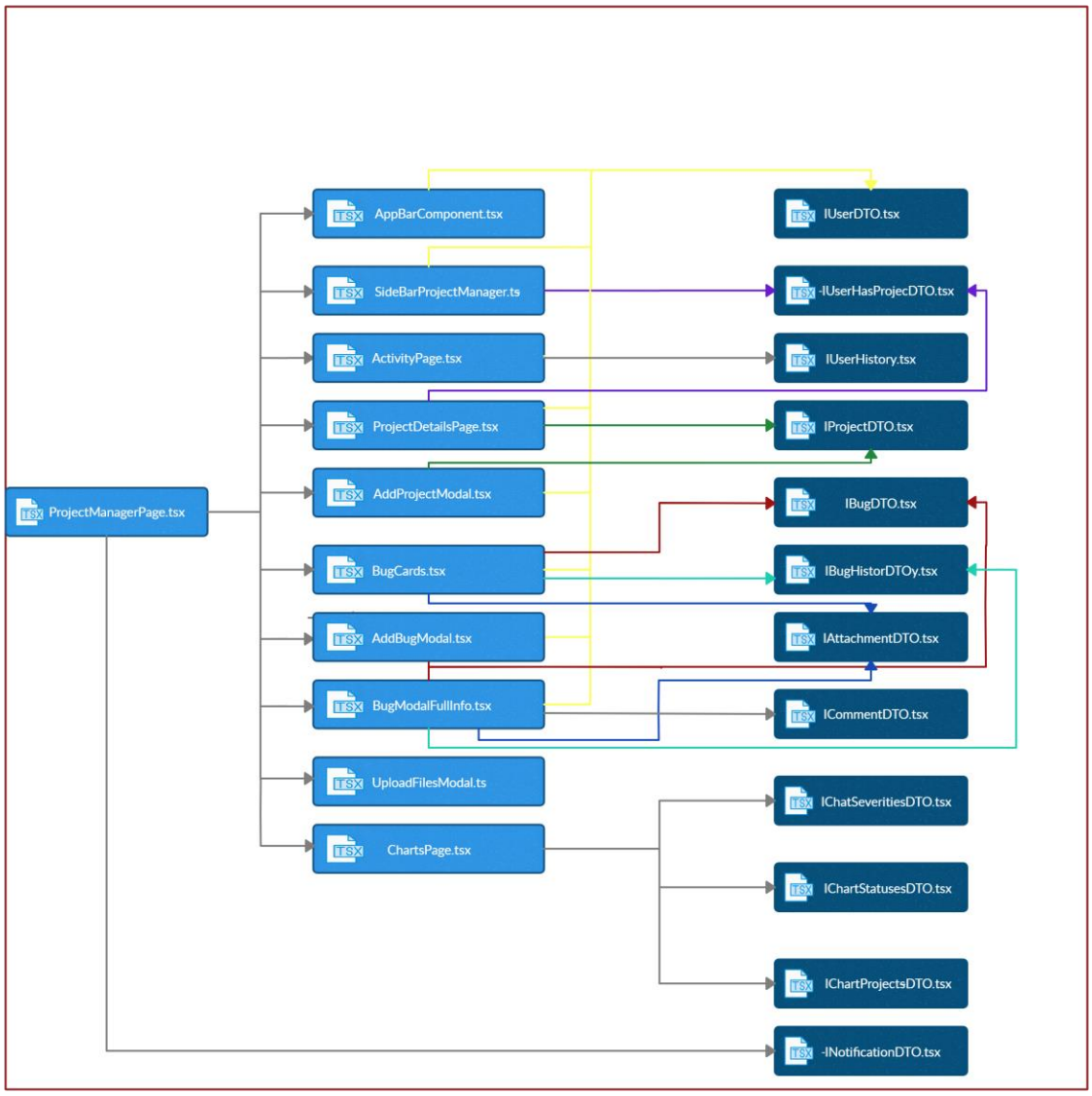


Figura 5.13 Interacținea componentelor pentru pagina de Project Manager

5.1.2.3. DTO

Deoarece aplicația React comunică cu aplicația Spring și în unele cazuri, mesajul primit de la server conține obiecte(în cadrul aplicației Spring Boot aceste obiecte sunt DTO-uri), pentru a le mapa a fost necesară declararea unor DTO-uri și pe partea de client. Aceste DTO-uri sunt simple interfețe care conțin declarațiile câmpurilor asociate obiectelor primite ca și răspuns din backend.

5.1.2.4. Comunicarea cu server-ul

Atunci când un utilizator interacționează cu interfața grafică, există un schimb de informații între aplicația client și aplicația server. Deoarece comunicarea între cele doua enități se realizează folosind protocolul HHTP, clientul trebuie să fie capabil să trimită cereri la server și să aștepte răspuns de la acesta.

Pentru a trimite un request spre server, aplicația client utilizează *Axios*. Axios fiind un client HTTP bazat pe *promise*, reușește să trimită cereri de tip GET, POST, PUT ș.a.m.d.

Elementele necesare utilizării unui request folosind Axios sunt:

- URL-ul asociat metodei din server care se dorește a fi accesată
- Parametrii – optional, în funcție de request
- .then – este ramura apelată dacă request-ul a fost efectuat cu succes, aici se poate realiza prelucrarea răspunsului primit de la server
- .catch – ramura apelată în caz de eroare
- .finally – ramura apelată de fiecare dată după trimitrea request-ului

Un exemplu de utilizare a modului de interacțiune dintre aplicația React și aplicația Spring Boot se regăsește în Figura 5.14:

```
let request = "http://localhost:8080/project/add";
axios.post(request, selectedProject)
    .then((response: { data: string }) => {
        Store.dispatch(mofifyRerenderProjectsCards(true));
        Store.dispatch(modifyOpenAddProjectModal(false));
    })
    .catch(() => { alert("Failed to insert the bug"); });
```

Figura 5.14 Exemplu de request tip POST utilizând Axios

5.1.2.5. Store

Este elementul central care deservește la stocarea stării întregii aplicații. Pentru a putea manipula datele menținute de acesta, au fost utilizate următoarele metode:

- *getState()* – care permite obținerea stării unui obiect
- *dispatch(acțiune)* – utilizată pentru actualizarea stării, acțiunea trimisă ca și parametru specifică ce obiect și ce câmp al obiectului să fie actualizat
- *subscribe(listener)* – înregistrarea de ascultători care să fie notificați când apare o schimbare a stării aplicației

Există un singur Store per aplicație, dar acest lucru nu împiedică separarea logicii de gestiune a datelor, deoarece se pot utiliza mai mulți Reducers separați care să implementeze logica necesară.

Comanda utilizată pentru crearea unui Store este: *createStore(reducers)*, comandă regăsită în Figura 5.15:

```
import { createStore } from "redux";
import allReducers from "../store/reducers/RootReducer";
const Store = createStore(allReducers)
export default Store
```

Figura 5.15 Crearea unui Store-ului aplicației

5.1.2.6. Reducers

Rolul aceștior *Reducers* în cadrul aplicației este accea de a specifica modul în care se modifică starea aplicației ca și răspuns la acțiunile trimise de către *Store*.

În esență un *Reducer* nu este altceva decât o *funcție pură*[16] care primește ca și parametrii: starea anterioară a unui obiect și o acțiune și returnează o nouă stare.

```
const openAddModalBugReducer = (state: boolean = initAddModal, action: Action) => {
  let newState = state;
  switch (action.type) {
    case actionConstants.OPEN_ADD_BUG_MODAL:
      {
        newState = action.open;
        break;
      }
  }
  return newState;
};
```

Figura 5.16 Exemplu de reducer

În Figura 5.16 este prezentat un *reducer* care modifică starea variabilei initAddModal în funcție de ceea ce specifică acțiune OPEN_ADD_BUG_MODAL.

Pentru o separare a logicii de actualizare a datelor au fost utilizați mai mulți Reducers, care utilizând comanda *combineReducers()* sunt adunați într-un singur oviect pentru o utilizare mai ușoară.

5.1.2.7. Actions

Acțiunile sunt sarcini utile cu informații care preiau informații din aplicație și pe care le trimit mai departe la Store. Aceste acțiuni sunt trimise spre Store utilizând metoda Store.dispatch(), ele reprezântând dealtfel singura sursă de informație primită de Store.

O astfel de actiune este alcătuită din:

- Un type – reprezintă tipul acțiunii care urmează să fie realizată; tipul este definit ca și constantă
- Informația utilă

```
export const OPEN_ADD_BUG_MODAL = "OPEN_ADD_BUG_MODAL";
```

Figura 5.17 Definirea unui type ca și constantă

```
export interface MofifySelectedBug {
  type: typeof actionConstants.SELECTED_BUG_MODIFY;
  bug: IBugDTO;
}
```

Figura 5.18 Definirea formatului unei acțiuni

```
export function modifyOpenAddBugModal(open: boolean = false): ModifyOpenAddBugModal {
    return { type: actionConstants.OPEN_ADD_BUG_MODAL, open: open };
}
```

Figura 5.19 Definirea acțiunii de actualizarea a stării modal-ului de adăugare bug

[16] https://en.wikipedia.org/wiki/Pure_function

5.2. Diagrama de deployment

Diagramele de *deployment*(dezvoltare) sunt alcătuite din trei elemente principale: noduri(reprezintă componenetele hardware ale sistemului),artifacte(reprezintă componentele software ale sistemului care rulează în noduri) și legăturile dintre noduri. Scopul unei astfel de diagrame este acela de a oferi o privire de ansamblu asupra arhitecturii fizice a sistemului, precum și vizualizarea componetelor software principale.

Nodurile sistemului implementat sunt:

- ClientPC- componenta hardware(PC sau laptop) de unde se accesează aplicația prin intermediul unui browser
- Client – componenta hardware unde rulează aplicația React
- Server– componenta hardware unde rulează aplicația Spring Boot
- Baza de date – componenta hardware unde serverul bazei de date MySQL

Artifactele sistemului implementat sunt:

- Aplicația React cu server-ul de Node.js
- Aplicația Spring Boot cu server-ul Tomcat
- Baza de date MySQL

Comunicarea între componenta *Client* și componenta *Server* se realizează utilizând protocolul HTTP. Server-ul primește cererea, o procesează, accesează baza de date pentru diverse informații ca mai apoi să trimită ca și răspuns clientului informațiile solicitate. Comunicarea între componenta *Server* și *Baza de date* se realizeză utilizînd *JDBC*(Java Database Connectivity).

În figura 5.20 sunt prezentate nodurile și artifactele sistemului dezvoltat, precum și interacțiunea dintre acestea:

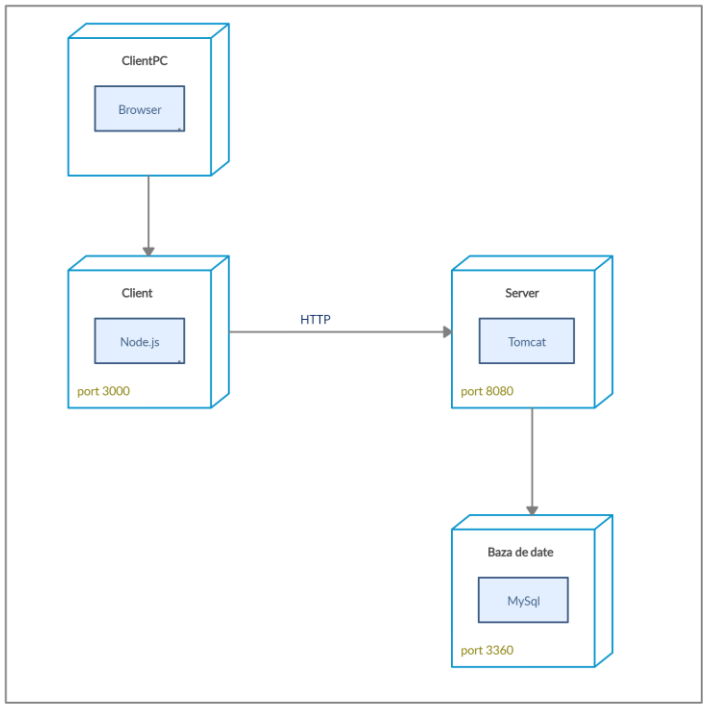


Figura 5.20 Diagarama de deployment a sistemului implementat

Capitolul 6. Testare şi Validare

Ca și în orice sistem software este nevoie de un proces prin care dezvoltatorii să se asigure că produsul final funcționează conform cerințelor de proiectare. Pentru aceasta, este necesară realizarea unor teste. Aceste teste sunt metode care primesc diverse date de intrare(valide sau invalide), apelează metode din aplicația țintă și compară la final dacă rezultatul obținut în urma executării este același cu rezultatul așteptat.

Pentru testarea aplicației de gestiune a bug-urilor au fost efectuate pe partea de server atât teste unitare, cât și teste de integrare. În cadrul implementării acestor nivele de testare au fost utilizate și testarea pozitivă cu scopul de a verifica dacă sistemul implementat se comportă conform așteptărilor, precum și testarea negativă pentru a acoperi și acele cazuri în care s-a dorit verificarea comportamentului sistemului în situațiile în care i-au fost furnizate date incorecte.

Tehnica de testare aplicată în cazul testelor unitare a fost tehnica “white box”, acesta fiind tehnica de creare a cazurilor de test asupra metodelor din cadrul aplicației pentru a detecta orice scenariu în care ar putea fi găsite eventuale probleme la implementare.

Tehnica de testare aplicată în cazul testelor de integrare a fost tehnica “black box”, acesta fiind tehnica care se bazează în întregime pe cerințele aplicației date, precum și ale specificațiilor solicitate de client. În cazul acestui tip de testare: testare de integrare se dorește a se evalua interacțiunea dintre componentele sistemului, aceste elemente fiind în prealabil testate independent utilizând testarea unitară.

De asemenea, este necesar de menționat faptul că s-a utilizat Impunerea contractelor prin aserțiuni în cazul testării cu JUnit.

Testele unitare aplicate metodelor de validare și mapare au fost contopite cu data-driven-test. Au fost utilizate adnotările @ParameterizedTest și @ValueSource pentru a specifica valorile de intrare utilizate în cadrul acelor teste.

Testele de integrare au fost realizate cu scopul de verifica dacă componentele aplicației Spring Boot funcționează corect împreună. În acest scop s-au testat metodele și interogările din nivelul de repository cu ajutorul adnotării @DataJpaTest. Aceste teste nu alterează starea bazei de date reale, deoarece utilizează o bază de date creată și ștearsă la fiecare rulare.

Tot teste de integrare au fost realizate utilizând @Mock, MockMvc și Mockito. Aceste teste au ca și scop verificarea întregului flux al aplicației de backend, de la nivelul de Controller și până la cel al bazei de date.

Pe partea de client verificarea a fost una clasică, realizată manual. S-a pus accentul pe validările câmpurilor, a restricțiilor impuse de rol și afișarea mesajelor primite de la server în consolă în faza de dezvoltare.

În Tabel 6.1 este prezentat cazul de test pentru logarea unui utilizator în aplicație:

Acțiune	Rezultatul așteptat	Rezultatul obținut
Utilizatorul accesează pagina de login	Pagina de login se deschide	Pagina de login se deschide
Utilizatorul introduce credențiale valide în câmpurile afișate, selectează limba engleză și apasă butonul de logare	Se așteaptă validarea credențialelor de la server	Se așteaptă validarea credențialelor de la server
Primirea răspunsului de la server și redirecționarea la pagina asociată rolului utilizatorului logat	Redirecționarea la pagina corespunzătoare	Redirecționarea la pagina corespunzătoare

Tabel 6.1 Cazul de test pentru logarea cu succes a unui utilizator

În tabelul următor este descris scenariul de test pentru adăugarea unui nou bug cu toate informațiile necesare valide:

Acțiune	**Rezultatul așteptat**	**Rezultatul obținut**
Utilizatorul logat cu success și cu rol de Project Manager se poziționează pe pagina cu bug-uri	Pagina cu bug-uri se deschide	Pagina cu bug-uri se deschide
Apasă butonul pentru adăugare bug	Fereastra de adăugare bug se deschide	Fereastra de adăugare bug se deschide
Utilizatorul completează toate datele și apasă SAVE	Se închide fereastra de adăugare Bug și se trimite request spre Server și utilizatorul este redirecționat la pagina cu lista de bug-uri	Se închide fereastra de adăugare Bug și se trimite request spre Server și utilizatorul este redirecționat la pagina cu lista de bug-uri
Utilizatorul poate să vizualizeze în lista cu bug-uri OPEN bug-ul adăugat anterior	Noul bug introdus este afișat în lista OPEN de bug-uri	Noul bug introdus este afișat în lista OPEN de bug-uri

Tabel 6.2 Adăugarea unui bug nou

Conform informațiilor din Tabel 6.2, rezultatul operației de adăugare bug(Figura 6.1) trebuie să fie Figura 6.2, ceea ce denotă faptul că rezultatele așteptate coincid cu rezultatele obținute.

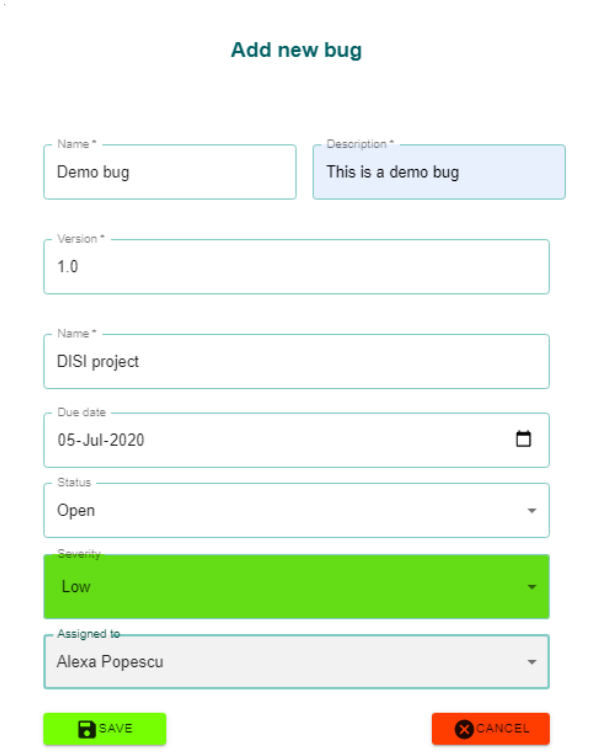
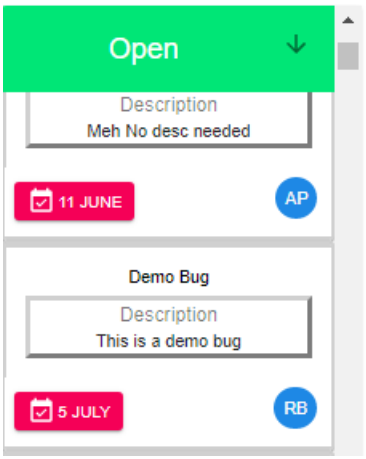


Figura 6.1 Adăugarea unui bou bug Figura



6.2 Lista bug-urilor cu statusul OPEN după operația de adăugare

Capitolul 7. Manual de Instalare si Utilizare

În această secțiune for fi detaliate resursele software și hardware necesare pentru instalarea şi rularea aplicaţiei, precum şi o descriere pas cu pas a procesului de instalare.

De asemenea se va prezenta și un manual de utilizare al sistemului implementat.

7.1. Instalare si rulare

În acestă secțiune se vor prezenta pașii necesari pentru rularea cu succes a aplicației de gestiune a *bug*-urilor.

7.1.1. Instalarea elementelor necesare

Resursele hardware necesare sunt:

- Arhitectura sistemului să fie pe 64 de biți(Arhitectură x64)
- Memorie RAM cu o capacitate de minim 8 GB
- Procesor cu frecvență minimă de 2GHz și minim și minim 4 nuclee

Aplicația a fost dezvoltată pe un sistem cu sistem de operare Windows. Se recomandă utilizarea aceluiași sistem de operare și pe sistemele unde urmează a fi instalată.

Pentru a fi posibilă rularea aplicației, mai întâi va fi necesară instalarea următoarelor elemente:

1) **Java**

Java JDK(Java Development Kit) și Java JRE(Java Runtime Environment) pot fi descărcate de pe site-ul Oracle: https://www.oracle.com/technetwork/java/ la secțiunea *Java SE*. Este necesară descărcarea celei mai recente versiuni, sau cel puțin versiunea 1.8. După descărcare, se execută fișierul executabil, iar mai apoi se vor seta variabilele de mediu: JAVA_HOME și JRE_HOME cu *path*-ul(calea absolută) a locației unde au fost instalate JAVA JDK și JAVA JRE.

2) **MySQL Workbench**

Pentru a instala MySql se accesează: https://dev.mysql.com/downloads/installer/ și se execută pașii din intefața programului de instalare decărcat.

3) **NodeJs**

Instalarea se efectuează accesând: https://nodejs.org/en/download/. După ce se rulează executabilul, se va rula comanda *node -version* pentru a verifica dacă instalarea a fost realizată cu succes. De menționat faptul că *npm*(Node Package Manager – util pentru instalarea diverselor pachete în React și nu numai).

Ca și medii de dezvoltare(*IDE- Integrated Development Environment*) au fost utilizate:

- **Eclipse**[17] – pentru aplicația Spring Boot
- **Visual Studio Code**[18] – pentru aplicația React
- **MySql Workbench**[19] – pentru gestiunea bazei de date

[17] https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers
[18] https://code.visualstudio.com/download
[19] https://dev.mysql.com/downloads/installer/

7.1.2. Rularea aplicației

Primul pas pentru rularea server-ului de Spring Boot este acela de a crea o nouă conexiune la baza de date și o nouă schemă a bazei de date. Pentru configurarea unei noi conexiuni se vor utiliza următoarele credențiale: username: root și password:root, iar numele bazei de date utilizate va fi: bug_tracker_schema. Aceste configurări se regăsesc și în Figura 7.1:

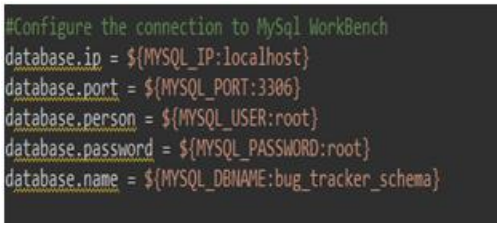


Figura 7.1Configurarea bazei de date

După ce baza de date a fost configurată, pentru a porni aplicația din backend va fi necesară rularea clasei LicențăApplication.java din *IDE*-ul Eclipse.

Pentru a rula aplicația client, se va deschide IDE-ul Visual Studio Code, se va rula mai întâi din linia de comandă ***npm -i***(instalează toate dependențele lipsă ale aplicației) și apoi se va rula ***npm start***(pornește aplicația client). Aplicația va putea fi accesată din browser, utilizând URL-ul: https://localhost:3000.

7.2. Manual de utilizare

În cadrul acestui capitol vor prezentate principalele funcționalități ale sistemului dezvoltat.

Prima pagină care este afișată este cea de login, în care urilizatorul își introduce credențialele și selectează limba dorită pentru afișare. În cazul în care sunt oferite credențiale invalide, se va afișa un mesaj de eroare. Pagina de login se regăsește în Figura 7.2 după cum urmează:

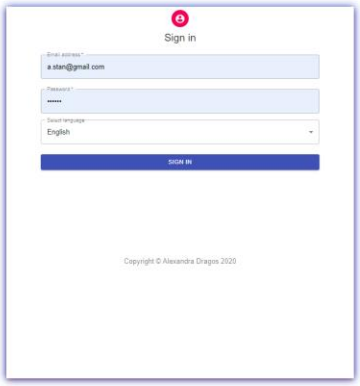


Figura 7.2Pagina de login

Dacă utilizatorul s-a logat cu succes și deține rolul de administrator va fi redirecționat la pagina din Figura 7.3:

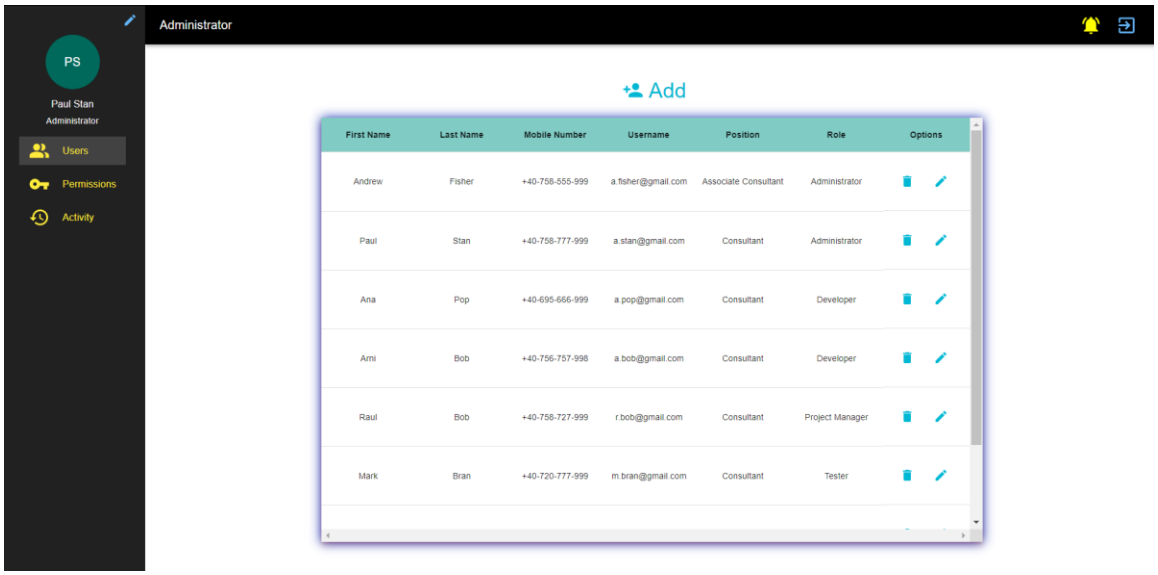


Figura 7.3Pagina de administrator

Din cadrul acestei pagini, administratorul poate să adauge un nou utilizator (Figura 7.4), să editeze detalii unui utilizator(Figura 7.5) prin selectarea utilizatorului dorit din tabel și apăsarea butonului sau să șteargă un utilizator prin apăsarea butonului .

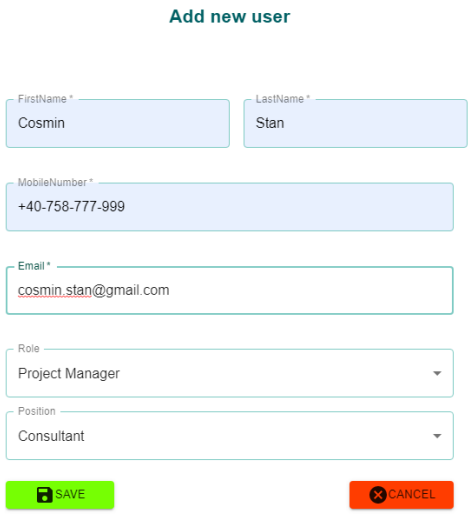


Figura 7.4Adăugare utilizator

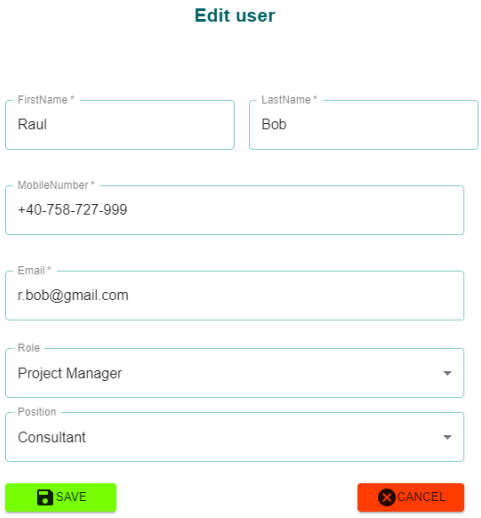


Figura 7.5 Editare utilizator

În secțiunea *Permission*(permisiuni), administratorul poate să adauge permisiuni unui utilizator sau să elimine din permisiunile sale prin bifarea/debifarea căsuțelor corespunzătoare permisiunilor dorite. În Figura 7.6 este prezentată pagina de permisiuni:

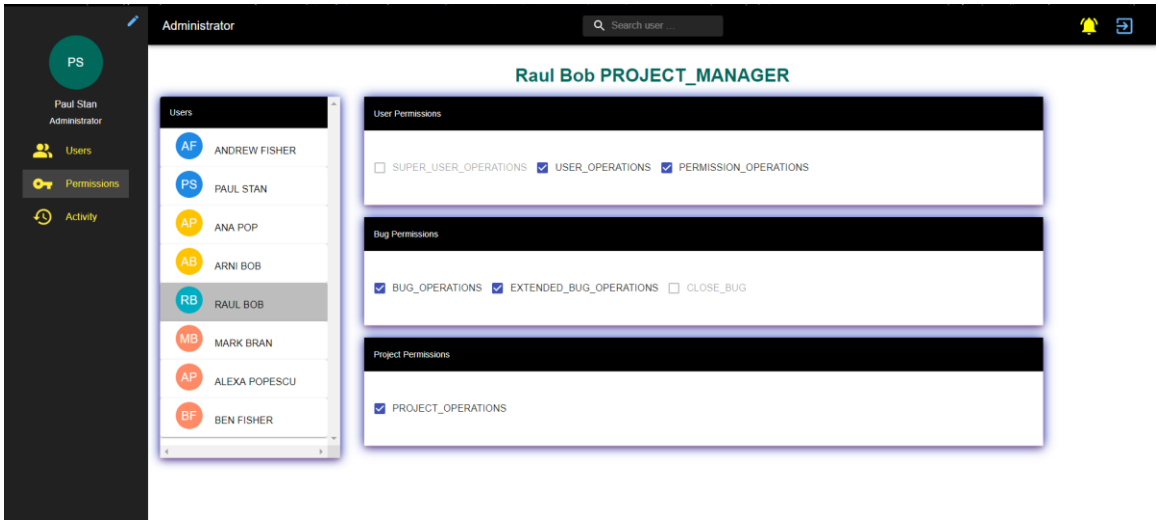


Figura 7.6 Pagina de permisiuni

În secțiunea *Activity*(Activity) utilizatorul logat(indiferent de rolul său) poate să vizualizeze un istoric al acțiunilor sale. În Figura 7.7 este prezentată acestă pagină:

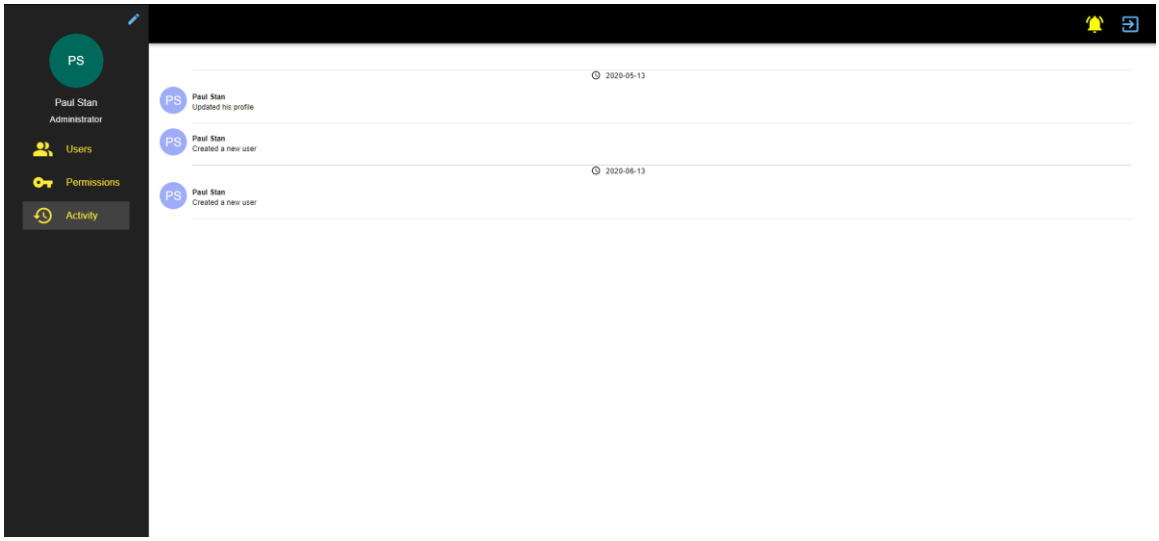


Figura 7.7Pagina de *Activity*(Istoric) a unui utilizator

Dacă utilizatorul logat are rolul de Project Manager, acesta are posibilitatea de a crea un nou proiect(Figura 7.8.a și Figura 7.8.b), de a modifica proiecte existente(Figura 7.9) și de a le șterge(Figura 7.9):

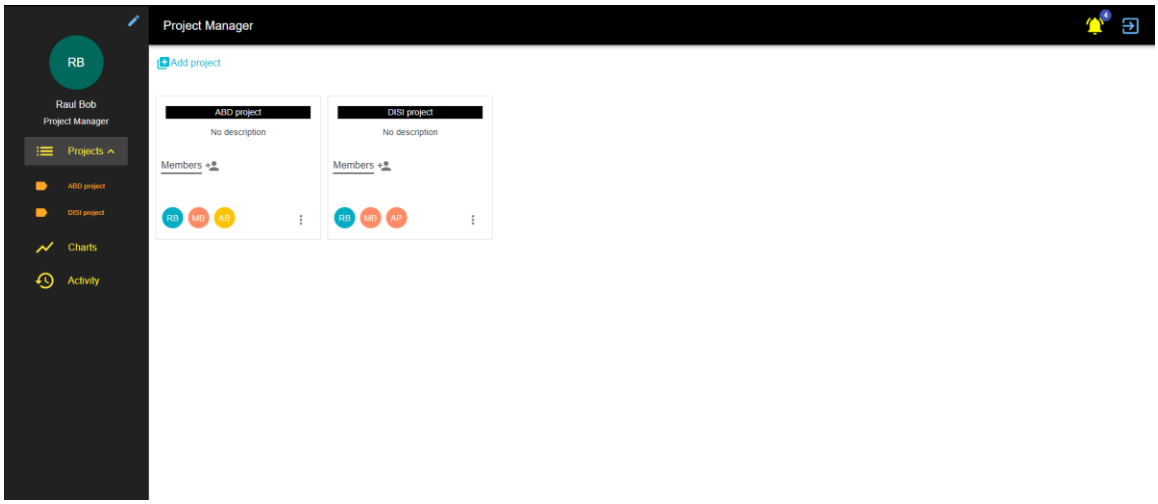


Figura 7.8.a Pagina de Project Manager

Figura 7.8.b Adăugare proiect

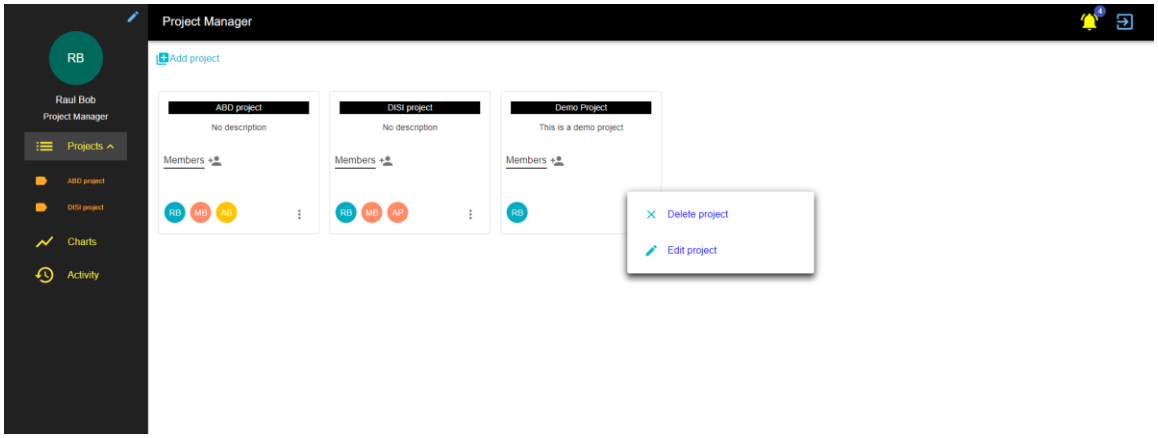
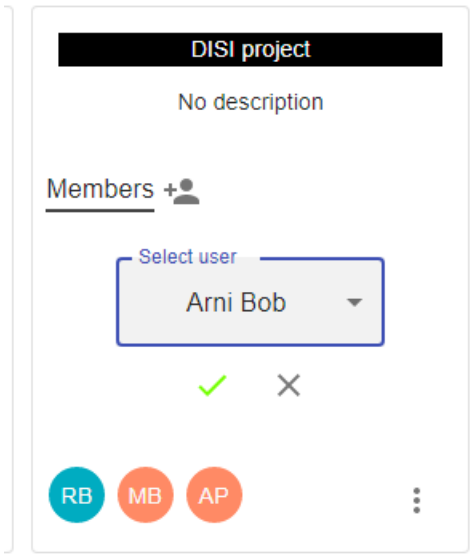


Figura 7.9 Modificarea sau ștergerea unui proiect

De asemenea, un Project Manager poate adăuga(Figura 7.10) sau elimina din membrii unui proiect(Figura 7.11):



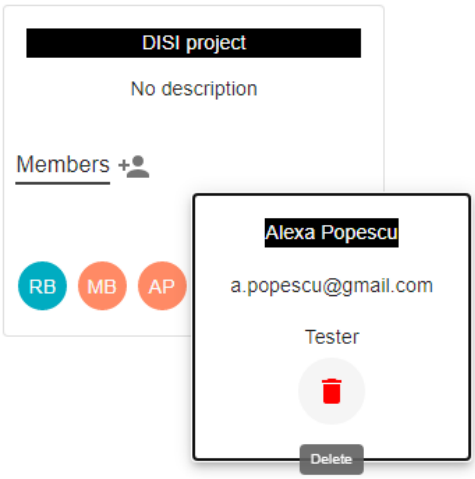


Figura 7.10 Adăugare membru proiect Figura 7.11 Ștergere membru proiect

O altă operație pe care un Project Manager o poate efectua este aceea de a vizualiza diverse grafice legate de statusul bug-urilor, severitatea lor și a numărului acestora din cadrul unui proiect(Figura 7.12):

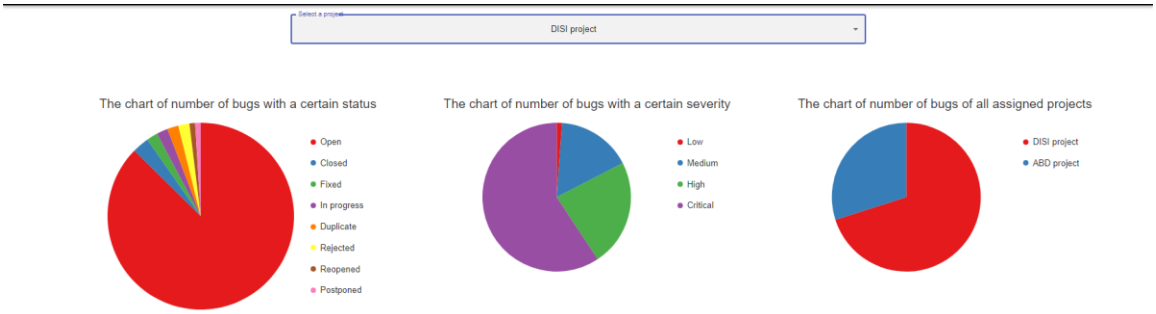


Figura 7.12 Pagina de statistici

Atât un Developer, cât și un Project Manager pot să vizualizeze lista de bug-uri asociate unui proiect ca și în Figura 7.13 , pot adăuga informații ca și în Figura 7.14 și pot modifica informațiile acestuia ca și în figura Figura 7.15:

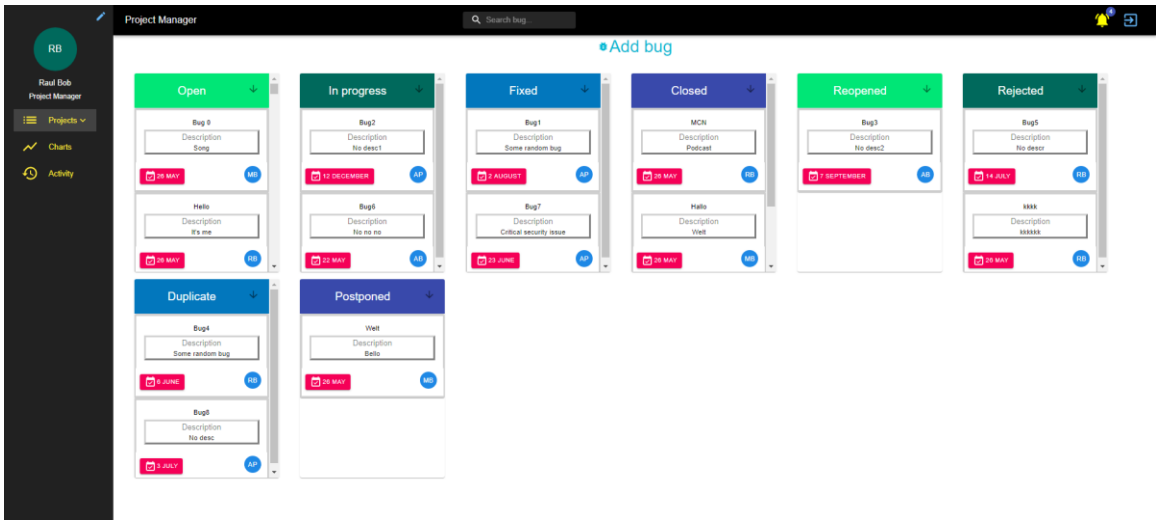


Figura 7.13 Vizualizare listă bug-uri Developer și Project Manager

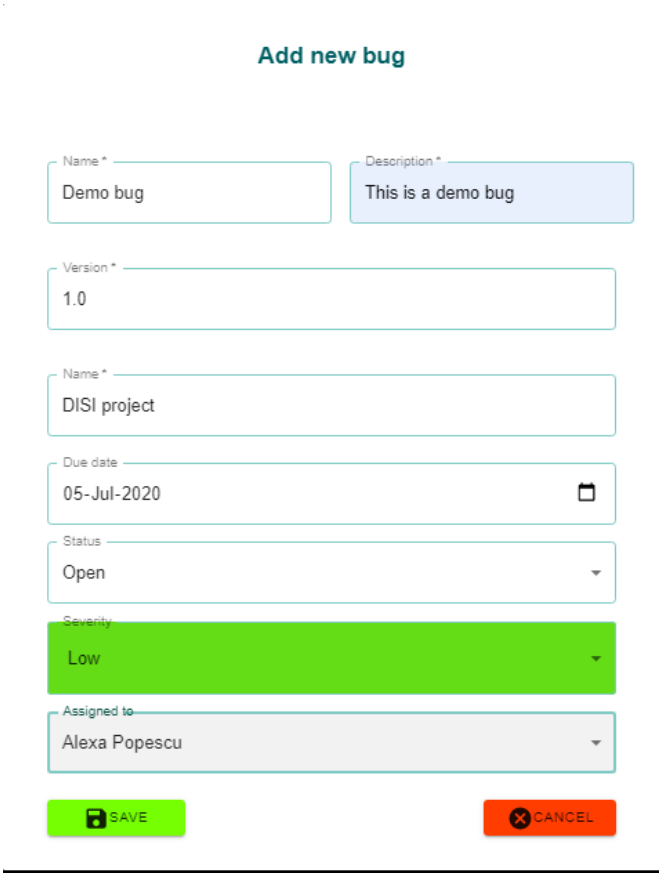


Figura 7.14 Fereastra de adăugare bug

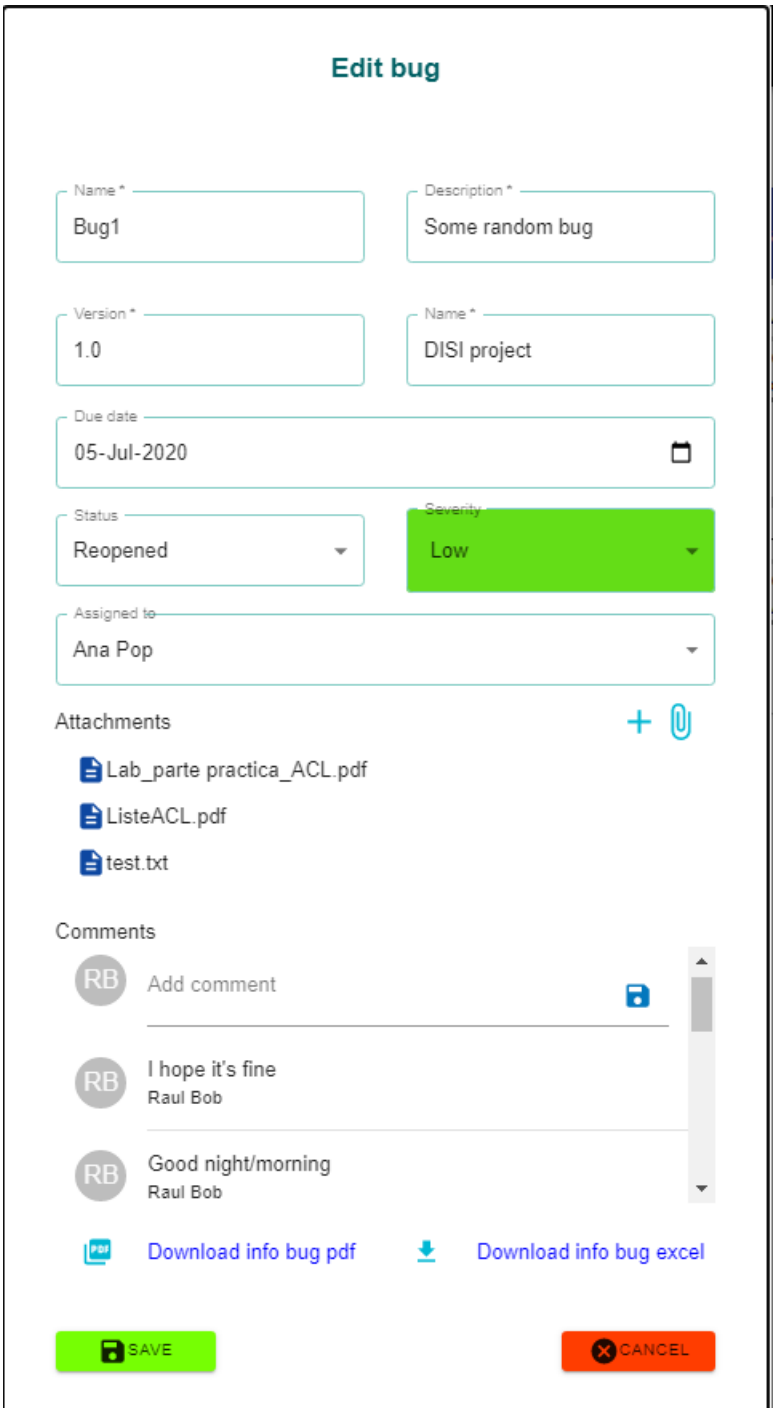


Figura 7.15 Fereastra de editare bug și vizualizare informații suplimentare

În Figura 7.16 este prezentată fereastra care se deschide atunci când un utilizator din echipa de dezvoltare dorește să adauge un nou atașament unui bug:

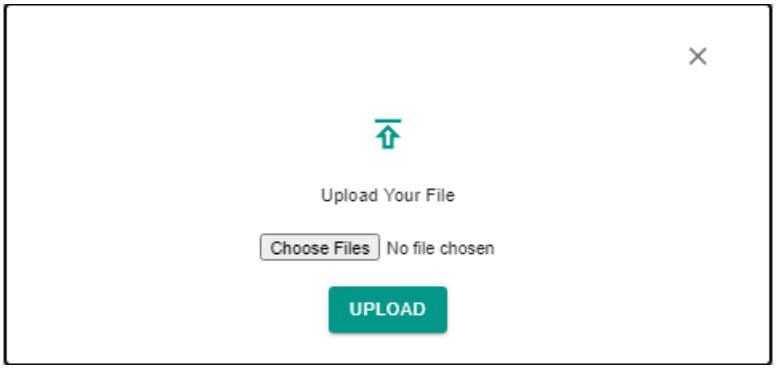


Figura 7.16 Fereastra pentru adăugare atașamente

În Figura 7.17 este prezentată pagina principală a unui Tester:

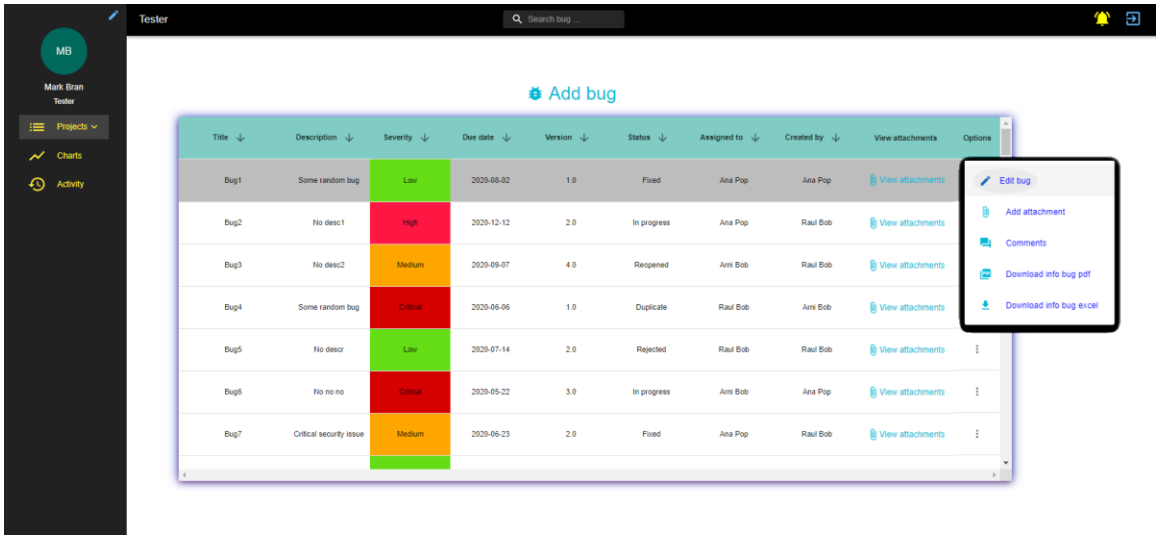


Figura 7.17 Pagina de Tester

Capitolul 8. Concluzii

În cadrul acestui capitol vor fi prezentate funcționalitățile aplicației dezvoltate, precum și o serie de funcționalități noi și utile care se doresc a fi implementate ulterior.

8.1. Analiza rezultatelor obținute

Sistemul dezvoltat reușește să își atingă toate obiectivele setate, oferind o aplicație care să vină în ajutorul companiilor mici și mijlocii în procesul de rezolvare a bug-urilor diverselor aplicații.

Aplicația a fost concepută într-o manieră care să permită vizualizarea rapidă a informațiilor depre *bug*-uri și proiecte, având o interfață modernă și ușor de utilizat. Funcționalitățile puse la dispoziția utilizatorilor sunt numeroase, de la adăugare bug-uri, până la încarcarea de atașamente unui bug și vizualizarea diverselor statistici.

Așadar, sistemul dezvoltat conferă utilizatorilor săi posibilitatea de a gestiona mai ușor procesul de fixare a *bug*-urilor, datorită structurării informațiilor despre bug-uri într-un singur loc, a notificărilor în timp real și prin email care au rolul de a menține membrii echipelor de dezvoltare la curent cu starea proiectelor la care lucrează și a modului de interacțiune dintre coechipieri prin scrierea de comentarii și încărcarea de atașamente bug-urilor

8.2. Dezvoltări ulterioare

Domeniul IT este un domeniu în continuă schimbare, ceea ce determină ca aplicațiile deja existente să necesite îmbunătățiri pentru a statisface cele mai noi cerințe de pe piață. Aplicația realizată a fost concepută astfel încât modificări ulterioare să poată fi încorporate cu ușurință, fără să afecteze funcționalitățile deja existente.

O primă îmbunătățire care poate fi adusă sistemului implementat este aceea de îmbunătăți interfața grafică, prin introducerea unei funcționalități de personalizare a acesteia:ex. modificarea schemei de culori, a modului în care sunt afișate listele de proiecte sau bug-uri, posibilitatea adăugării unei imagini ca și avatar sau implementarea unei funcționalități de tipul *drag&drop* pentru mutarea unui card de bug-uri dintr-o listă în alta.

De asemenea, o funcționalitate utilă ar fi și importarea bug-urilot dintr-un fișier excel de exemplu. Acest lucru ar rapidiza procesul de adăugare de bug-uri noi și ar facilita reutilizarea informațiilor deja existente. Este o funcționalitate importantă atunci când se dorește migrarea informațiilor bug-urilor dintr-o platformă în alta.

O altă îmbunătățire ar fi adăugarea unui chat online prin care utilizatorii să poată cumunica mai rapid. Acestă îmbunătățire este benefică în situațiile în care membrii unui proiect nu se regăsesc în același spațiu de lucru, lucru care împiedică purtarea unor discuții față în față.

O funcționalitate utilă ar fi implementarea unei logici care să determine automat dacă un bug este duplicat. Acest lucru ar scuti utilizatorii cu rol de tester să efectueze acțiuni de identificare ale bug-urilor duplicate, prin verificarea manuală a informațiilor fiecărui *bug*. Cum este menționat și în [13] o metodă posibilă pentru identificarea elementelor duplicate ar fi detectarea acestora prin analiza textului. Pentru aceasta au fost

utilizate 3 tipuri de *idf*-uri(*inverse document frequency* – frecvența inversă a documentului- aceasta redă importanța unui cuvânt într-un document): idfSum(idf-ul rezumatelor bug-urilor), idfDesc(idf-ul corespunzător descrierilor bug-urilor) și idfBoth(colecție de date hibride, atât rezumate, cât și descrieri). Aceste frecvențe inverse a documentelor au fost analizate în funcție de asemănările și deosebirile lor de către un algoritm de *Machine Learning,* cu scopul de a identifica defectele software duplicate.

O dezvoltare a sistemului se poate efectua și în direcția introducerii unui sistem de recompensare a utilizatorilor. Pentru fiecare dezvoltator și tester, managerii de proiect să ofere recenzii în funcție de numărul de bug-uri fixate de aceștia și în funcție de procentul de bug-uri introduse de ei în cadrul proiectului.

O altă caracteristică utilă ar fi introducerea unui sistem de estimare a timpului de rezolvare. Acest lucru ar oferi o privire de ansamblu asupra timpului necesar pentru rezolvarea bug-urilor și o modalitate de alocare a orelor destinate fixării mai avantajoasă atât pentru client,cât și pentru echipa de dezvoltare.

Bibliografie

[1] Bin Lin, “Bug Fixing in Action: Activities, Process, and Knowledge”, Master’s Thesis, Faculty of Informatics, Masaryk University, Brno, 2016 (https://pure.tue.nl/ws/files/46945200/855437-1.pdf)

[2] A.S.Syed Fiaz, N.Devi, S.Aarthi, “Bug Tracking and Reporting System”, International Journal of Soft Computing and Engineering Volume-3, 2013 (https://arxiv.org/ftp/arxiv/papers/1309/1309.1232.pdf)

[3] Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, London: Pearson Education, 2008.

[4] Karim, Md & Ihara, Akinori & Yang, Xin & Choi, Eunjong & Iida, Hajimu & Matsumoto, Kenichi, “Improving the High-Impact Bug Reports: A Case Study of Apache Projects”, 2019. (https://www.researchgate.net/publication/332151757_IPSJ_SIG_Technical_Report_Improving_the_HighImpact_Bug_Reports_A_Case_Study_of_Apache_Projects)

[5] Jiri Janak, “Issue Tracking Systems”, Diploma’s Thesis, Faculty of Informatics, Masaryk University, Brno, 2009 (https://is.muni.cz/th/60778/fi_m/thesis.pdf)

[6] Bauer, Christian, Gavin King, and Gary Gregory. *Java Persistence with Hibernate*, Shelter Iron, New York: Manning Publications Co., 2015.

[7] Eve Porcello, Alex Banks, *Learning React: Functional Web Development with React and Redux*, Sebastopol, California: O'Reilly Media, 2017

[8] Gemma Catolino, “Not all bugs are the same: Understanding, characterizing, and classifying bug types”, Journal of Systems and Software Volume 152, pp. 165-181, June 2019 (DOI: 10.1016/j.jss.2019.03.002)

[9] Naimul Islam Naim, “ReactJS: An Open Source JavaScript Library for Front-end Development”, Diploma’s Thesis, Metropolia University of Applied Sciences, Heksinki, 2017 (https://www.theseus.fi/handle/10024/130495)

[10] Luiz Alberto Ferreira Gomes, “Report severity level prediction in open source software: A survey and research opportunities“,Information and software technology, Volume 115, pp 58-78, November 2019 (DOI: 10.1016/j.infsof.2019.07.009)

[11] Xue Han,“Reproducing performance bug reports in server applications: The researchers’ experiences”, Journal of Systems and Software, Volume 156, pp. 268-282, October 2019
(DOI: 10.1016/j.jss.2019.06.100)

[12] F. Brooks, *The mythical man-month*, Boston, Addison-Wesley, 1975.

[13] Tao Zhang, He Jiang, Xiapu Lup, Alvin Chan, “A literature Review Of Research in Bug Resolution: Tasks, Challenges and Future Directions”, The Computer Journal, Volume 59, May 2016
(DOI: 10.1093/comjnl/bxv114)

Anexa 1

Lista Figurilor

Lista Tabelelor

Anexa 2 Glosar de termeni

API	Application Programming Interface
JSON	JavaScript Object Notation
SQL	Structured Query Language
URL	Uniform Resource Locator
JWT	JSON Web Token
REST	Representational State Transfer
ORM	Object-Relational Mapping
CMMI	Capability Maturity Model Integration
ISO	International Organization for Standardization
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hyper Text Transfer Protocol
SMTP	Simple Mail Transfer Protocol
DTO	Data Transfer Object
CRUD	Create, Read, Update and Delete
SSL	Secure Sockets Layer
TLS	Transport Layer Security
idf	inverse document frequency
idfSum	idfSummarise
idfDesc	idfDescription