TICKETING HELPER – SISTEM DE MANAGEMENT AL TICHETELOR

LUCRARE DE LICENŢĂ

Absolvent: **Paul Șandor-Lari**

Coordonator ştiinţific: **Asist. Ing. Cosmina IVAN**

2020

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Paul Șandor-Lari**

TICKETING HELPER – APLICAȚIE DE MANAGEMENT AL TICHETELOR

1. **Enunţul temei:** *Acest proiect abordează problema managementului sesizărilor clienților. Clientul transmițând problema lui companiei, aceasta este transformată într-un tichet pentru a fi gestionată în funcție de aria în care se încadrează. De asemenea, acest sistem ajută la urmărirea activității angajaților.*

2. **Conţinutul lucrării:** *Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie, Anexe*

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

4. **Consultanţi**:

5. **Data emiterii temei:** 1 februarie 2020

6. **Data predării:** septembrie 2020

Absolvent: Șandor

Coordonator ştiinţific: _____

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul **Paul Șandor-Lari** legitimat(ă) cu **CI** seria **XB** nr. **618937** CNP **1970905060605** autorul lucrării **Ticketing Helper – Sistem de management al tichetelor** elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea **Calculatoare** din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea **Septembrie** a anului universitar **2019-2020** declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informațiilor 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
08.09.2020	**Șandor-Lari Paul**
	Semnătura
	Șandor

Cuprins

1. Introducere

Acest proiect a fost creat cu scopul implementării unui sistem software care să ajute o companie să dezvolte o relație cât mai bună cu clienții acesteia. Acest sistem reprezintă o unealtă pe care cei de la departamentul de suport al companiei o folosesc cu scopul de a gestiona cât mai bine mesajele clienților.

1.1. Contextul proiectului

În orice companie relațiile cu clientul reprezintă un factor important în dezvoltarea acesteia. Cu toate că pentru a avea o relație bună cu acesta, oferirea sentimentului că acesta este important și respectat, presupune costuri suplimentare dar realizează o reputație pozitivă pentru aducerea de noi clienți și a-i păstra pe cei din prezent.

De aceea un departament dedicat relației cu clienții este important în cadrul oricărei companii, aplicațiile și metodologiile utilizate în cadrul acestui departament jucând un rol important pentru a îmbunătății sau pastra o relație cât mai bună cu clienții. Acest departament dedicat relației cu clienții este compus dintr-o echipă care are rolul de a interacționa cu clienții, de a le oferi îndrumare în rezolvarea unor probleme sau de a le răspunde întrebărilor ce țin de companie.

În cazul unui număr mare de clienți, utilizarea unor sisteme software care să ajute la organizarea mesajelor de la client devine o necesitate. Sistemele de mangement al tichetelor au fost create special pentru acest lucru. Acestea au ca scop gestionarea mesajelor primite de la clienți și soluționarea acestora.

Un sistem de management al tichetelor reprezintă unealtă cu ajutorul căreia o instituție sau organizație urmărește problemele clienților și cine se ocupă de ele. Aceste sisteme au rolul de a centraliza modul de comunicare al clientului cu o instituție sau o companie.

1.2. Structura lucrării

În cele ce urmează, va fi prezentată structura lucrării împreuna cu o scurta descriere a fiecărui capitol din aceasta.

- **Introducere**: Acest capitol prezintă importanța departamentului de suport într-o companie și a sistemelor de management a tichetelor pentru acesta.
- **Obiectivele Proiectului:** În acest capitol este descris obiectivul principal al acestui proiect, dar și obiectivele specifice care doresc sa fie implementate în cadrul acestuia.
- **Studiu Bibliografic**: Acest capitol are ca scop prezentarea contextului de utilizare a unui sistem asemănator celui propus în acest proiect și prezentarea sistemelor asemănătoare prezente în prezent, precum și realizarea unei comparații a funcționalităților acestora și a sistemului propus de acest proiect.

- **Analiză și Fundamentare Teoretică**: În acest capitol vor fi prezentate cerințele sistemului implementat, cazurile de utilizare care au fost luate în considerare la momentul implementării și actorii sistemului. De asemenea, se vor descrie și tehnologiile care au fost utilizate pentru realizarea acestui sistem.
- **Proiectare de Detaliu și Implementare**: Acest capitol este dedicat descrierii în detaliu a structurii sistemului cu ajutorul diagramelor UML. Sunt prezentate secvențe de cod relevante și decizile care au dus la implementarea acestora.
- **Testare și Validare**: În acest capitol sunt prezentate metodele, testele realizate pentru verificarea funcționalității sistemului.
- **Manual de Instalare și Utilizare**: Acest capitol ajută utilizatorul în instalarea acestui sistem prezentând pașii pe care un nou utilizator trebuie să îi realizeze pentru a avea un sistem funcțional. De asemenea, conține și cerințele software și hardware minime de care un astfel de sistem are nevoie.
- **Concluzii**: Ultimul capitol al acestei lucrări conține concluziile deduse pe parcursul dezvoltării acestui sistem și vor fi prezentate posibilele îmbunatățiri ce pot fi aduse acestui sistem.

2. Obiectivele Proiectului

În acest capitol vor fi prezentate obiectivul principal al acestei lucrări, poziționarea produsului și obiectivele secundare.

Această lucrare urmărește implementarea unui sistem de management al tichetelor care să ofere posibilitatea urmăririi activității angajaților destinat companiilor cu un departament suport care pun accentul pe părerea clientului și doresc să ofere spijin acestora cât mai rapid cu putință.

2.1. Obiectivul principal

Obiectivul aplicației este de a eficientiza centralizarea mesajelor de la client. Acest aspect îmbunătățește modul prin care angajatul își desfașoară activitatea, găsind într- un singur loc toate întrebările trimise de clienți.

În tabelele de mai jos va fi prezentat obiectivul principal al acestei lucrări împreună cu problema pe care și-a propus să o rezolve și poziționarea produsului, precum și responsabilitățile utilizatorilor.

Tabel 2.1 Specificația problemei

Problema	Gestionarea comunicării cu clienții
afectează	Departamentul Relații Clienți/ Suport
impactul este	Dificultatea gestionării sesizărilor clienților
o soluție	Ținerea evidenței tichetelor O ușoară împărțire a tichetelor între angajați.

Tabel 2.2 Poziționarea produsului

Pentru	Companii cu un număr ridicat de clienți
Care	Doresc o gestionare cât mai bună a mesajelor primite de la clienți.
Ticketing Helper	Este un sistem software.
Care	Oferă posibilitatea urmăririi progresului tichetelor și a activității angajaților.
Spre deosebire de	Sistemele electronice de mesaje (ex: Yahoo Mail)
Acest produs	Oferă o imagine de ansamblu asupra sesizărilor clienților și gestionarea lor.

Tabel 2.3 Descrierea utilizatorilor și a responabilităților acestora

Actor	Descriere	Responsabilitatea principală
Utilizator anonim	Utilizator neautentificat care poate doar sa scrie mesaje	Poate sa scrie mesaje companiei fara a necesita autentificarea
Client autentificat	Utilizator autentificat, are acces la istoricul mesajelor și poate scrie unele noi	Scrie mesaje companiei, poate accesa istoricul de mesaje și poate vedea statusul acestora.
Agent	Utilizator autentitficat, reprezinta compania	Rezolvarea sesizărilor clientilor
Administrator	Utilizator autentitficat, reprezinta compania	Dirijarea și supervizarea angajaților

2.2. AObiective specifice

Pe lânga obiectivul principal al sistemului, centralizarea mesajelor de la client, sistemul mai oferă suport la alocarea eficientă a resurselor și tratarea tichetelor în funcție de importanța lor, la monitorizarea activității angajaților și a echipelor.

Alocarea Resurselor Umane

Aplicația va avea o lista de angajați și tichete. Angajații vor fi alocați per tichet, având posibilitatea de a-și actualiza stadiul în care se află.

Monitorizarea Activității Angajaților

Angajații își pot actualiza stagiul în care se află cu rezolvarea tichetului. Se poate observa toată activitatea acestuia. Acest sistem este de folos departamentului de suport clienți, fiecare client își va crea un tichet online, urmând ca acesta sa fie salvat intr- o bază de date și asignat unuia dintre angajații companiei.

Monitorizarea Activității Echipelor

Activitatea echipelor reprezintă suma activităților tuturor angajațiilor care fac parte din echipa respectivă.

3. Studiu Bibliografic

La începutul acestui capitol va fi prezentat contextul de utilizare al unui astfel de sistem, beneficiile pe care le aduce departamentului de suport și importanța lui într-o companie.

Vor fi discutate aspectele generale ale unui sistem informatic pentru tichete și aplicațiile de acest tip disponibile pe piață. Vor fi enumerate funcționalitățile acestora și comparate cu sistemul propus.

3.1. Context de utilizare și Departamentul de suport

Un sistem de management al tichetelor are ca scop, precum spune si articolul [1], oferirea de suport și informații clienților unei companii/instituții care oferă anumite produse sau servicii. Un asemenea sistem este folositor în cadrul departamentului de suport, ajutând la centralizarea mesajelor clienților venite de pe mai multe canale de comunicare.

Departamentul de suport reprezintă o echipă care interactionează cu clienții prin diferite metode de comunicare (ex: email, telefon, aplicații, platforme de socializare).

Acesta poate fi împărțit în două categorii [2]:

- **Suport client**
  Are ca scop să raspundă la întrebările clienților.
- **Suport tehnic**
  Scopul acestuia este de a rezolva problemele tehnice. De exemplu, în cadrul companiilor IT, acesta ar avea rolul de a asista/ajuta la instalarea, mentenanță, actualizarea sau ștergerea produsului oferit de companie.

Odată cu creșterea numarului de clienți, pentru a putea gestiona cât mai bine toate cererile clienților, automatizarea serviciului oferit de acest departament devine o necesitate.

Crearea unei baze de cunoștințe și automatizarea acestui serviciu face ca acesta sa fie disponibil tot timpul. Acest lucru este posibil prin utilizarea unui sistem de management al tichetelor.

3.2. Sisteme de management al tichetelor

Un sistem de management al tichetelor reprezinta unealta cu ajutorul careia o institutie sau organizatie urmareste problemele clientilor si cine se ocupa de ele. Aceste sisteme au rolul de a centraliza modul de comunicare al clientului cu o institutie sau companie.

Un sistem de management al ticketelor poate fi impartit in 5 parti, conform [2]:

1. Crearea unui tichet

   Un client poate crea un tichet prin completarea unui formular pus la dispoziție de companie, cu ajutorul căruia își poate transmite mesajul lui

departamentului de suport, și trimiterea lui, urmând ca acesta să fie salvat în baza de date a companiei.

De fiecare data când un formular este completat și trimis, sistemul de management al tichetelor va crea un tichet, indiferent de proveniența lui sau dacă cineva se ocupa de el.

2. Vizualizarea tichetelor

Membrii departamentului de suport sau un administrator pot vedea toate tichetele existente. Aceste tichete pot fi filtrate dupa statusul lor, care poate fi: neasignat, în așteptare, nerezolvat sau terminat/rezolvat. Aceste posibilități de vizualizare sunt esentiale deoarece acestea pot reprezenta modele de grupare a tichetelor.

Un tichet nou creat va avea mereu statusul de neasignat.

3. Raspunsul la tichete

Când statusul unui tichet este modificat clientul trebuie notificat.
Clientul trebuie sa primeasca urmatoarele tipuri de mesaje

- Email de confirmare ca mesajul a fost primit și un nou tichet a fost creat
- Email de notificare a schimbarii statusului unui ticket

Aceste mesaje de notificare trebuie trimise automat de sistem, fiind declanșate la momente cheie din ciclul de viață al unui tichet.

4. Rezolvarea tichetului

După ce un angajat a fost asignat să rezolve un tichet acesta trebuie să vizualizeze tichetul respectiv și sa încerce sa rezolve problema clientului.

După ce a terminat, acesta trebuie să actualizeze statusul tichetului ca fiind terminat/rezolvat.

5. Analizarea tichetelor

Tichetele din cadrul unui astfel de sistem pot fi asignate automat de catre acesta. Prin urmare, este nevoie ca tichetele să poată să fie analizate de un administrator. Analiza are ca rol observarea obiectivă a unor statistici cu scopul de a eficientiza asignarea de resurse în cadrul companiei.

3.3. Sisteme similare

În acest subcapitol vor fi analizate cele mai importante sisteme de management a tichetelor disponibile pe piață. O selecție a sistemelor importante a fost realizată[1], urmând ca după aceea să se compara funcționalitățile oferite de aceste sisteme, dar și de aplicația propusă în această lucrare.

[1] https://www.softwaretestinghelp.com/best-help-desk-software/

FreshDesk[2]

Este un sistem de management al tichetelor, dezvoltat de compania *Freshwwork*, potrivit atat pentru companiile mari cat și cele mai mici. Ofera mai multe facilitati, cum ar fi:

- Convertirea automată a email-urilor in tichete și un timp de raspuns rapid.
- Un sistem de rapoarte integrat cu ajutorul caruia se poate urmari activitatea echipei, nivelul de satisfactie al clientilor.
- Un sistem inteligent care automatizeaza operatia de asignare a tichetelor.
- O baza de cunostinte integrata cu ajutorul careia clientii pot sa gaseasca raspuns la intrebarile lor in cel mai scurt timp.

LiveAgent [3]

Aceasta aplicație ofera posibilitatea de unificare a tuturor canalelor de comunicare intr-un singur loc, astfel, comunicarea intre client și companie este simplificata. De asemenea, aplicația este intuitiva, fiind usor de utilizat, oferind peste 175 de functionalitati. Printre acestea se numara:

- Un sistem de tichete bine implementat care ofera posibilitatea transformarii mesajelor in tichete
- Chat live
- Automatizarea mesajelor, făcând posibilă oferirea de răspunsuri automat clienților.
- Un portal de suport bun care este disponibil 24 de ore din 24 pentru clienti lor.

Zendesk[4]

Aceast sistem este recomandat in cazul in care interactiunea cu clientul se realizeaza telefonic, pe un chat, prin email sau retele de socializare. Aplicația are urmatoarele avantaje:

- Flexibilitate în contextul de managementul tichetelor asigurand un suport pe mai multe canale de comunicatie.
- O bază mare de cunostinte și un forum în care clienții pot scrie pentru a-și rezolva problemele.
- Posibilitatea de creare de forumuri, atat publice cat și private, care sunt suportate de Android, iPhone și iPad.

[2] https://freshdesk.com/
[3] https://www.liveagent.com/
[4] https://www.zendesk.com/

Kayako [5]

Kayako este sistem de management al tichetelor care poate fi instalat pe un computer fizic sau poate fi folosit online. Simplifică interacțiunea cu clientii și ajuta la organizarea cererilor acestora. Functionalitatile acestui sistem sunt:

- Integrare buna a tichetelor cu emailul și slack.
- Gama larga de profile pentru clienti și organizatii care incurajeaza e-comertul.
- Posibilitatea de creare și personalizare de tichete, chat-uri și clienti pentru a eficientiza colectarea de informații.

Tabel 3.1 Tabel funcționalități sistem propriu și sisteme similare

Funcționalitate/ Aplicație	Ticketing Helper	FreshDesk	LiveAgent	Zendesk	Kayako
Secțiune de FAQs personalizată	Da	Da	Nu	Da	Da
Oferire de sugestii regăsite în FAQs	Da	Da	Nu	Nu	Nu
Spațiu dedicat conceperii formularului de sesizări	Da	Da	Da	Da	Da
Centralizarea mesajelor	Da	Da	Da	Da	Da
Urmărirea stadiului de rezolvare a tichetului	Da	Da	Nu	Nu	Nu
Urmărirea eficienței angajaților	Da	Nu	Nu	Nu	Nu
Asignarea automata a ticketelor	Da	Da	Da	Da	Da
Automatizarea mesajelor	Nu	Nu	Da	Nu	Nu
Crearea de tichete personalizate	Nu	Nu	Nu	Nu	Da
Crearea de forumuri dedicate	Nu	Nu	Nu	Da	Nu
Chat comun al angajaților	Nu	Da	Da	Da	Da

În tabelul de mai sus a fost realizată o comparatie a sistemelor prezentate. S-au luat în cosiderare funcționalitățile acestora și cele pe care doresc să le implementez în sistemul

[5] https://www.kayako.com/

propus. Pe prima coloana vor fi prezentate funcționalitățile, urmatoarele coloane vor corespunde sistemelor, (primul sistem fiind cel propus în această lucrare). Notații: **Da**(conține funcționalitatea), **Nu**(nu conține funcționalitatea).

Din analiza funcționalităților sistemelor din acest tabel se poate observa că prezența unui spațiu dedicat creări unui mesaj, centralizarea mesajelor și asignarea automată a tichetelor reprezintă funcționalități esențiale. Prin urmare, aceste trei funcționalități stau la baza unui astfel de sistem.

4. Analiză şi Fundamentare Teoretică

În acest capitol vor fi prezentate cerintele sistemului, cele funcționale cât și cele non-funcționale, cazurile de utilizare și actorii sistemului precum și tehnologiile și conceptele utilizate.

4.1. Cerințele sistemului

4.1.1. Cerințele funcționale

Cerințele funcționale reprezinta funcționalitățile sistemului care definesc un anumit comportament al acestuia în cazul unei intrări de date cu un scop specific.

Tabel 4.1 Cerințele funcționale ale sistemului

Identificator	Descrierea cerinței	Utilizatori
1	Autentificare	Client, Administrator, Angajat, Lider de echipă
2	Administrare mesaje	Client, Lider de echipă, Angajat
2.1	Adaugare mesaje	Client, Lider de echipă, Angajat
2.2	Vizualizare mesaje	Client, Lider de echipă, Angajat
3	Administrare tichete	Lider de echipă, Angajat
3.1	Adaugare tichete	Client
3.3	Actualizare informații tichete	Lider de echipă, Angajat
3.4	Căutare tichete după Id	Client, Lider de echipă, Angajat
3.5	Căutare tichete dupa email	Client, Lider de echipă, Angajat
4	Vizualizare activitate angajat	Administrator, Lider de echipă, Angajat
5	Vizualizare activitate echipa	Administrator, Lider de echipă, Angajat
6	Administrare conturi angajați	Administrator
6.1	Adaugare angajați	Administrator
6.2	Ștergere angajați	Administrator
6.3	Actualizare informații angajați	Administrator, Angajat
6.4	Căutare angajați	Administrator
7	Administrare echipe	Administrator
7.1	Adaugare echipe	Administrator
7.2	Stergere echipe	Administrator
7.3	Actualizare informații echipe	Administrator
7.4	Căutare echipe	Administrator

4.1.2. Cerințele non-funcționale

Cerințele non-functionale reprezintă caracterisitici ale calității sistemelor software. Oferă informații despre sistem din punctul de vedere al performanței, ușurinței utilizatorului de a învăța să folosească aplicația, securității, portabilității, timpului de răspuns, al scalabilității, fiabilitatea mentenanței și altele.

- **Performanța**

  Performanta reprezinta cât de repede sistemul software sau o anumita componenta a acestuia răspunde la un anumit numar de cereri. Aceasta cerinta non-functionala se referă la timpul pe care un utilizator trebuie să- l aștepte pentru a obține un răspuns solicitării sale din partea sistemului.

  Sistemul propus în această lucrare urmărește ca limita superioară a timpului de răspuns de la server sa fie de 3 secunde. Pentru obținerea unui timp cât mai scurt s-a urmărit pe partea de fronte-end filtrarea datelor neimportante care vor fi trimise serverului pentru un timp de serializare cât mai scurt, iar pe partea de back-end s-a realizat paralelizarea operațiilor.

- **Scalabilitate**

  Reprezintă capacitatea sistemului de a fi atât disponibil cât și funcțional mai multor utilizatori care interacționeaza cu sistemul în același timp.

  Scalabilitatea poate fi de doua tipuri: scalare pe orizontală și scalare pe verticală. Scalarea pe orizontală are loc în momentul în care se adaugă noi resurse hardware sistemului pentru a facilitata paralelizarea, iar scalarea verticală, prin îmbunatățirea resurselor existente sau prin îmbunătățirea algoritmilor utilizați în sisitem.

  Limita sistemului este reprezentată de numărul maxim de conexiuni concurente permise cu baza de date. Astfel, sistemul va permite un numar maxim de 500 de utilizatori. Acest număr poate fi crescut prin tehnici de scalare pe orizontală sau verticală.

- **Securitate**

  Securitatea unui sistem este o caracteristica esențială in orice sistem. În cadrul dezvoltarii unei aplicații trebuie luate in considerare ce permisiuni trebuie acordate unui utilizator in cazul unei accesări a aplicației fără autentificare. De asemenea, include și protejarea datelor cu conținut sensibil prezente într- un sistem.

  Sistemul folosește tokeni la nivelul de transfer de date între server și client. Acest lucru asigură împiedicarea acțiunilor neautorizate în sistem limitând acțiunile pe care un utilizator neautentificat sau cu permisiuni insuficiente le poate realiza.

- **Extindere**

  Aplicația trebuie să suporte adaugări noi de funcționalități. Această caracteristică a unui sistem ține cont de arhitectura sistemului, de împărțirea acestuia în componente și a elementelor în pachete.

4.2. Cazuri de utilizare

În acest subcapitol vor fi prezentați actorii principali ai sistemului, responsabilitățile lor și reprezentarea lor în diagrame de tip use case (caz de utilizare). Se

va descrie pentru fiecare actor câte un use case împreuna cu precondițiile, postcondițiile, flow-ul normal, și flow-urile alternative.

4.2.1. Actorii sistemului

Sistemul poate avea următorii actori: utilizator anonim, client autentificat, angajat și administrator. În tabelul de mai jos vor fi prezentați actorii împreună cu responsabilitățile lor principale.

Tabel 4.2 Actorii sistemului

Actor	Descriere	Responsabilitatea principală
Utilizator anonim	Utilizator neautentificat care poate doar sa scrie mesaje	Poate sa scrie mesaje companiei fara a necesita autentificarea
Client autentificat	Utilizator autentificat, are acces la istoricul mesajelor și poate scrie unele noi	Scrie mesaje companiei, poate accesa istoricul de mesaje și poate vedea statusul acestora.
Agent	Utilizator autentitficat, reprezinta compania	Rezolvarea sesizărilor clienților
Lider de Echipă	Utilizator autentificat, reprezintă compania	Rezolvarea sesizărilor clienților și organizarea unei echipe
Administrator	Utilizator autentitficat, reprezinta compania	Dirijarea și supervizarea angajaților

4.2.2. Model caz de utilizare

Într-un model al cazurilor de utilizare (use case) se prezintă diferiți actori/utilizatori care pot interacționa cu sistemul în vederea rezolvării unei probleme. Prin urmare, acesta descrie obiectivele utilizatorilor și interacțiunile sistem-utilizator posibile.

4.2.2.1. Diagramele cazurilor de utilizare

Cum este prezentat și in [3] rolul diagramelor de cazuri de utilizare este de a expune o reprezentare cât mai simplistă a interacțiunilor utilizatorului cu sistemul. O diagrama a cazurilor de utilizare prezintă tipurile de utilizatori prezenți în sistem. Fiecare caz de utilizare fiind reprezentat de un cerc. Rolul acestor diagrame este de a oferi o viziune de ansamblu asupra sistemului, la abstractizarea sistemului.

În cele ce urmează vor fi prezentate diagramele de cazuri ale actorilor principali ai sistemului:client/utilizator anonim, client autentificat, angajat și administrator.

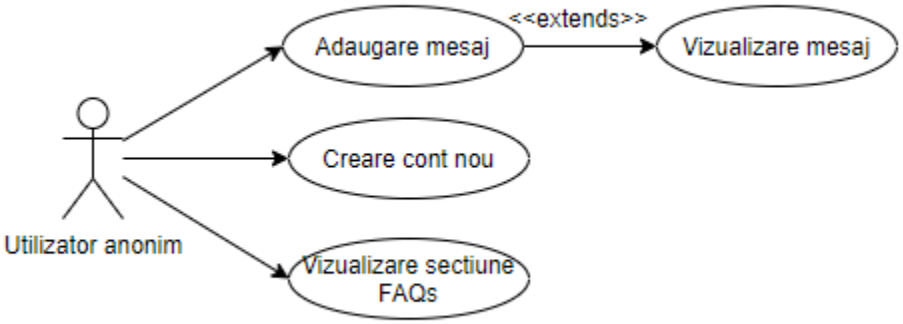

Figura 4.1 Diagramă cazuri de utilizare utilizator anonim

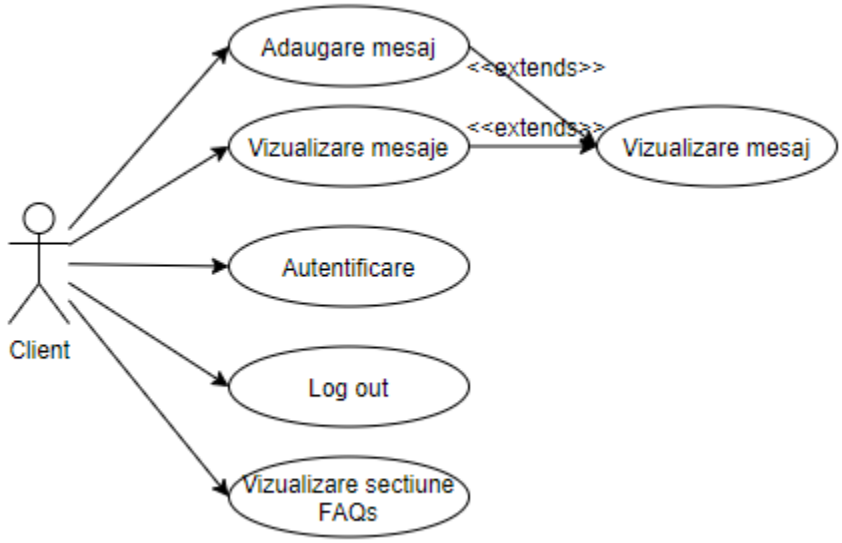

Figura 4.2 Diagramă cazuri de utilizare client

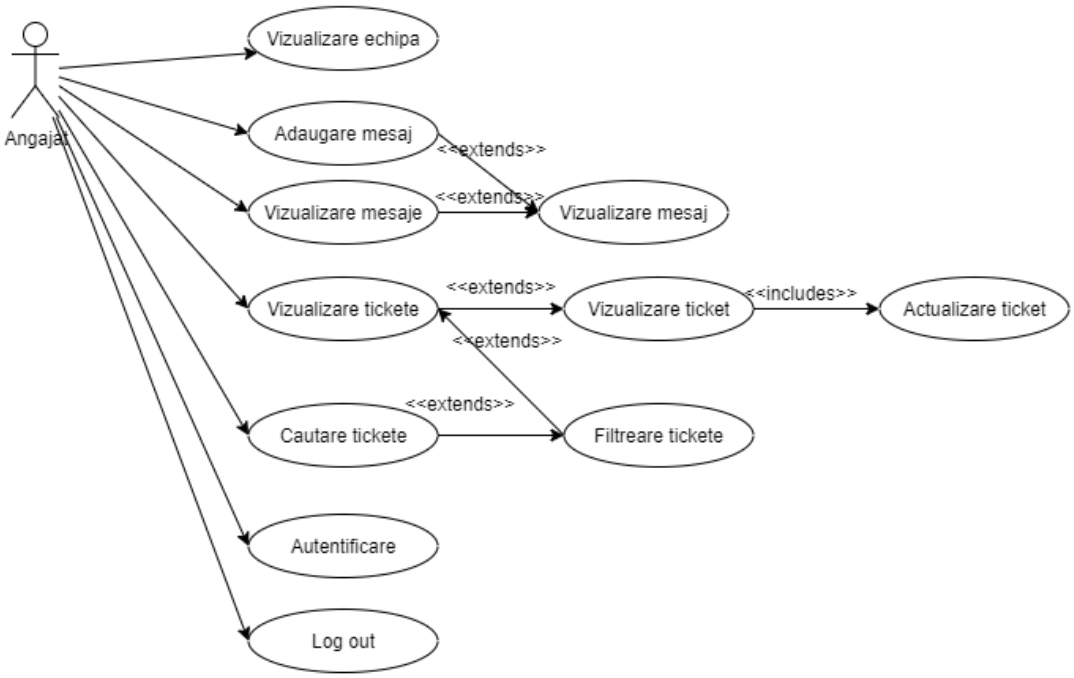

Figura 4.3 Diagramă cazuri de utilizare utilizator angajat

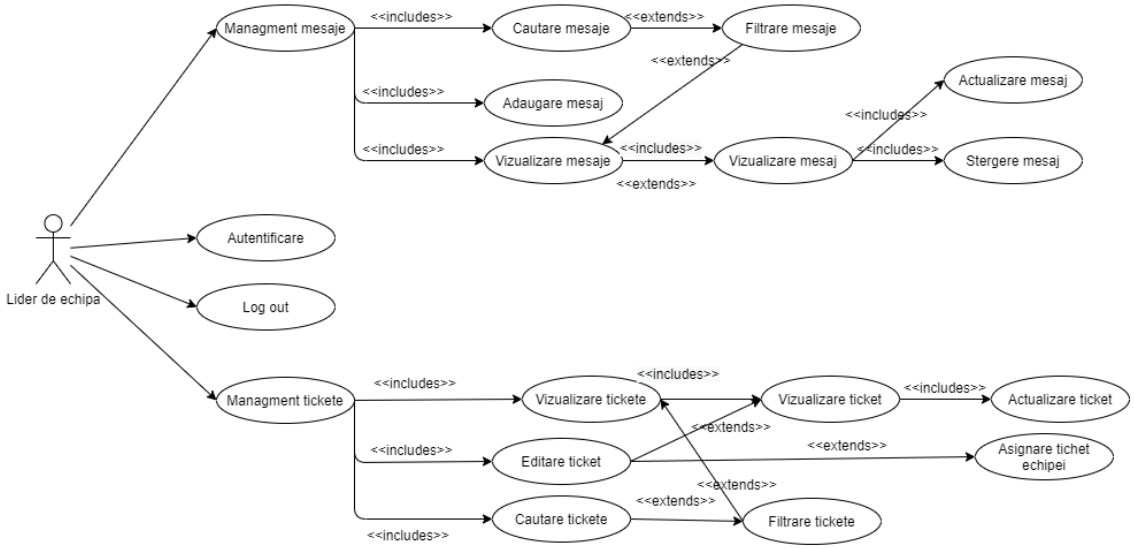

Figura 4.4 Diagramă cazuri de utilizare utilizator lider de echipă

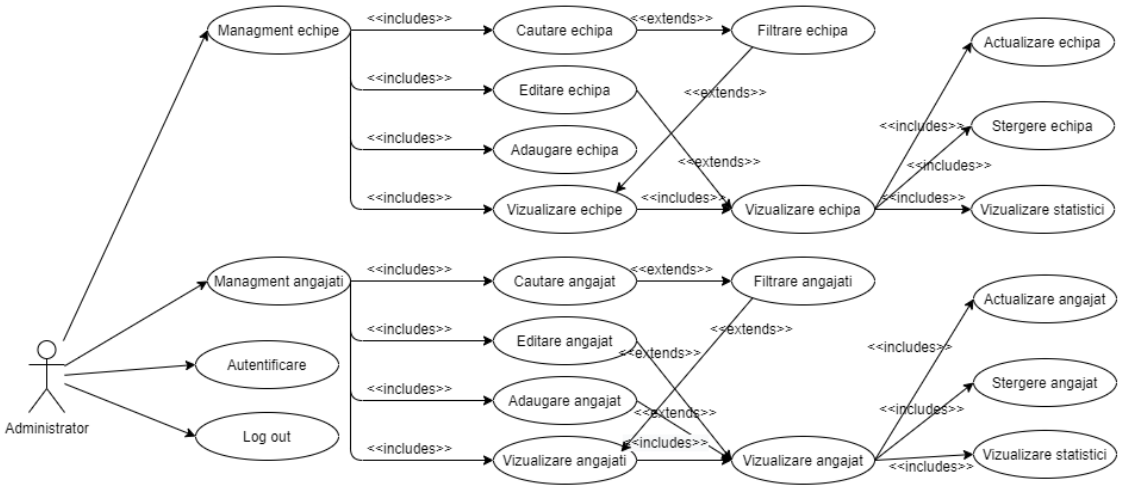

Figura 4.5 Diagramă cazuri de utilizare utilizator administrator

4.2.2.2. Descrierea cazurilor de utilizare

În cele ce urmează vor fi prezentate cele mai importante cazuri de utilizare întalnite în sistem.

Caz utilizare 1

Nume: Creare cont nou

Actor principal: Utilizator anonim

Descriere: Un utilizator, care nu are cont, doreste ca pe langa posibilitatea trimiterii de mesaje sa poate vedea și istoricul mesajelor și statusul acestora. Pentru aceasta trebuie sa își creeze un nou cont.

Precondiții:

1. Actorul sa nu fie autentificat.

Flow normal:

1. Actorul apasa pe butonul de creare de utilizator nou.
2. Se vor afisa campuri de completat necesare crearii unui nou cont.
3. Actorul completeaza campurile cu informațiile necesare.
4. Actorul apasa pe butonul de finalizare creare cont.
5. Se va afisa mesajul de confirmare.

Flow alternativ:

4. Actorul completeaza incorect sau nu completează informațiile necesare.
   a. Se va afisa un mesaj de eroare care il informeaza pe actor ca nu se poate crea un cont nou. Evidentierea campurilor invalide.

b. Actorul va realiza pasul 3 sau va renunta la crearea unui cont nou.

Postcondiții:

1. Un nou cont este creat cu informațiile despre noul utilizator.

Caz utilizare 2

Nume: Autentificare

Actor principal: Client, Angajat, Administrator

Descriere: Actorul dorește să se autentifice în sistem pentru a putea beneficia de funcțiile oferite de acesta.

Precondiții:

1. Un cont deja creat pentru actorul respectiv.

Flow normal:

1. Accesarea paginii de autentificare.
2. Sistemul afiseaza campurile care trebuie completate in vederea realizarii autentificarii.
3. Actorul apasa pe butonul de autentificare.
4. Accesarea paginii de prestabilite in cazul unei autentificari reusite.

Flow alternativ:

2. Actorul a gresit credentialele sau nu a completat campurile necesare.
    a. Se va afisa un mesaj de eroare care va informa actorul de eroare (credentiale incorecte sau campuri necompletate).
    b. Va mai introduce din nou datele de autentificare sau renunta la autentificare.

Postcondiții:

1. Actorul va putea folosi beneficile oferite de aplicație in functie de rolul acestuia.

Caz utilizare 3

Nume: Adăugare mesaj

Actor principal: Client

Descriere: Clientul poate trimite mesaje companiei, dorind sa își exprime aprecierea, nemultumirea sau orice alta observatie referitoare la servicile primite.

Precondiții:

1. Actorul sa fie autentificat.

Flow normal:

1. Actorul va intra in fereastra de creare de mesaj.
2. Se vor afisa campurile ce trebuie completate pentru trimiterea mesajului.
3. Actorul completeaza câmpurile necesare împreună cu mesajul dorit.
4. Apasă pe butonul de trimitere a mesajului.
5. Afișare mesaj de confirmare trimitere.

Flow alternativ:

4. Actorul nu completeaza anumite campuri obligatorii sau le completeaza incorect.
    a. Se va afisa un mesaj de eroare care il informeaza pe actor ca nu se poate trimite mesajul. Evidentierea campurilor invalide.
    b. Actorul va realiza pasul 3 sau va renunta la trimiterea mesajului.

Postcondiții:

1. Mesajul este trimis și salvat in baza de date.
2. Mesajul va aparea in istoricul clientului.

Caz utilizare 4

Nume: Asignarea tichetelor noi echipei

Actor principal: Lider de echipă

Descriere: Liderul de echipă decide ce tichete din cele noi să le rezolve echipa acestuia.

Precondiții:

1. Liderul de echipă să fie logat.
2. Liderul de echipă se află pe pagina cu tickete noi.

Flow normal:

1. Selectarea tichetelor dorite.
2. Apăsarea pe butonul de asignare a tichetelor echipei.
3. Actorul realizeaza modificarile dorite.

Flow alternativ:

1. Liderul de echipă selecteaza un tichet pe care nu dorește să îl asigneze echipei lui.
    a. Actorul va apăsa din nou pe butonul de selectare, acest lucru având ca efect deselectarea unui tichet deja selectat.

Postcondiții:

1. Tichetele sunt asignate echipei.
2. Tichetele vor apărea în lista de tichete a echipei respective.

Caz utilizare 5

Nume: Actualizare informații ticket

Actor principal: Angajat

Descriere: Angajatul doreste sa actualizeze statusul unui ticket sau alta informație referitoare la acesta.

Precondiții:

1. Angajatul sa fie logat.
2. Angajatul se află pe pagina cu tickete, iar ticketul dorit sa fie vizibil in pagina.

Flow normal:

1. Apasare pe ticket.
4. Sistemul afiseaza informațiile despre acel ticket.
5. Actorul realizeaza modificarile dorite.
6. Salveaza modificarile.
7. Inchide fereastra cu informațiile despre ticket.

Flow alternativ:

In cazul in care nu s-a realizat pasul 4 și s-a sarit de la pasul 3 la pasul 5 sau la un refresh de pagina se va afisa un mesaj care informeaza actorul de existenta unor modificari și cere confirmarea realizarii actiunii.

Postcondiții:

3. Informațiile despre ticket vor fi actualizate.
4. Daca un alt utilizator vizualizeaza detalile despre acel ticket și s-au realizat intre timp schimbari acesta este nevoit sa efectueze un refresh de pagina pentru a vedea modificarile.

În figura de mai jos este afișat flow-ul celui de-al cincelea caz de utilizare (Actualizare informații tichet).

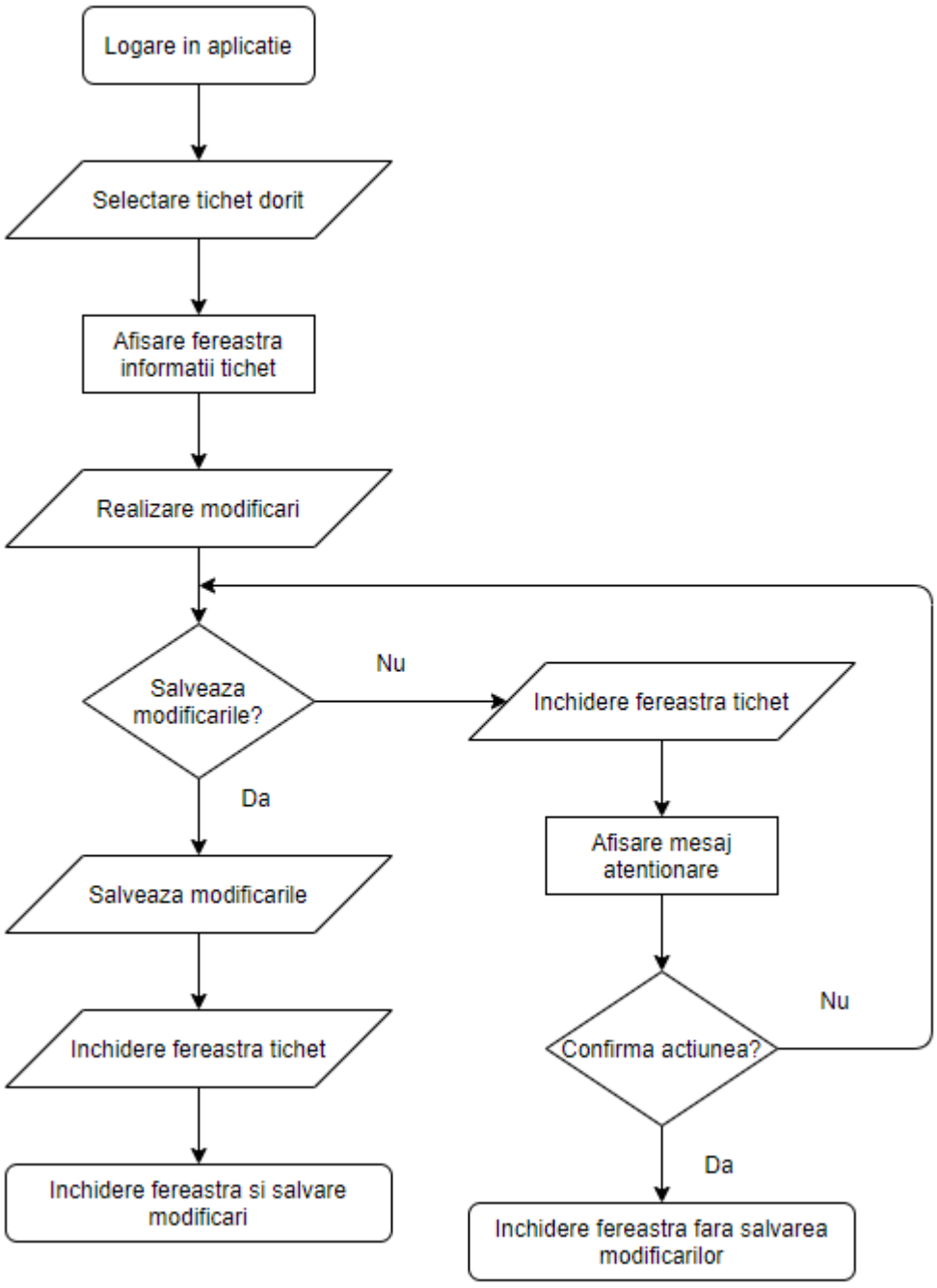

Figura 4.6 Flow actualizare informații tichet

4.3. Tehnologii și concepte utilizate

4.3.1. .NET Framework

.NET[6] este o platformă de dezvoltare open-source pentru crearea mai multor tipuri de aplicații. Aplicațiile pot fi scrise în C# (ales de mine pentru realizarea sistemului), F# sau Visual Basic. Motivele pentru care am ales această platformă sunt urmatoarele:

- un mediu de programare orientat pe obiect consistent indiferent daca acel cod este stocat și executat local, dar distribuit pe web sau executat complet remote
- un mediu de executie al codului care sa minimizeze deploymentul și versionarea
- un mediu de executie al codului care sa promoveze executarea sigura a codului, chiar și a codului creat de un intermediar necunoscut sau de incredere
- un mediu de executie al codului care elimină problemele de performanță ale altor medii.

4.3.2. JSON Web Token (JWT)

JSON Web Token[7] este un standard care definește modul în care se poate realiza securizat un transfer de date de tip JSON. Aceste informații transmise pot fi verificate și se poate avea încredere în ele deoarece conțin o semnătură digitală. Această "semnare digitală" este realizată folosind o cheie care poate fi generată folosind mai mulți algoritmi.

JWT este folositor când se dorește autorizarea unor utilizatori, restricționarea de acțiuni permise. Folosind JWT, când un utilizataor se loghează într-o aplicație primește un token. Cu token- ul va putea să își valideze accesul la anumite rute, servicii și resurse. De asemenea, prin utilizarea unui JSON Web Token se pot realiza transmiteri de date intr-un mod sigur între două părți. Folosirea unor perechi de chei publice/private validează identitatea membrilor.

Am ales utilizarea folosirea JWT deoarece doresc realizarea de acțiuni autorizate în sistem, oferind un mecanism de autorizare a acțiunilor. De asemenea, NET oferă posibilitatea utilizării JWT prin intermediul unor pachete care asigura crearea de tokeni și de restrictionare a cererilor HTTP în funcție de rolul utilizatorului asociat fiecărui token.

4.3.3. Servicii Web RESTful și Web API

Representational State Transfer (REST), conform [4] este un stil arhitectural devenit foarte important în tehnologiile aplicațiilor web. Acest stil arhiectural ofera aplicațiilor și clienților care folosesc această aplicație un mod securizat prin care aceștia pot face anumite cereri serverului. Protocolul utilizat de o astfel de aplicație este HTTP.

Se spune despre un serviciu web ca este RESTful dacă implementează anumite elemente cheie[8]:

- Resurse- reprezintă elementul cheie, conținând adresa URL a serverului

[6] https://docs.microsoft.com/sr-cyrl-rs/dotnet/framework/get-started/overview
[7] https://jwt.io/
[8] https://www.guru99.com/restful-web-services.html

- Tipul cererii- pot fi mai multe tipuri de cereri, precum: GET (cerere de informații), POST (inserare), PUT (actualizare), PATCH (actualizare parțială) și DELETE
- Header-ul cererii- conține informații suplimentare despre cerere (autorizație, tipul de răspuns așteptat)
- Corpul cererii - reprezintă datele trimise de client serverului
- Corpul răspunsului- după procesarea cererii, serverul va returna un raspuns clientului în funcție de tipul cererii și tipul de răspuns așteptat
- Codul statusului răspunsului- sunt coduri care sunt trimise alături de corpul răspunsului (ex: 200 este codul care informează clientul că cererea a fost corectă și procesată).

.NET Framework poate asigura legătura între partea de frontend și backend prin intermediul Web API[9] [10]. Este un framework construit peste .NET Framework, folosind HTTP ca un protocol al aplicație, nu ca un protocol de transfer, pentru a oferi informații clientului. Web API poate returna date în mai multe formate, putând fi specificate de cerere (request). În mod implicit, este JSON, dar poate returna date și în formatul XML.

Cererile de la client pot fi interpretate într-o maniera RESTful folosind metode HTTP, cum ar fi GET (cerere de informații), POST (inserare), PUT (actualizare), PATCH (actualizare parțială) și DELETE.

Acesta mai oferă și alte posibilități referitor la ce poate sa conțină header- ul unei cereri, cum ar fi:

- Accept (obligatoriu)- acest câmp se referea la modelul de date acceptat (json, text javascript, xml)
- Accept-Charset (obligatoriu)
- Authorization (obligatoriu)

Când clientul primește răspunsul la cererea lui, acesta poate avea unul din urmatoarele coduri:

- 200 OK- un raspuns standard în cazul in care cererea a fost realizata cu succes
- 201 Created- Cererea a fost îndeplinită
- 400 BadRequest- Cererea nu a fost corectă (ex: a fost omis un parametru)
- 401 Unauthorized- Utilizatorul care a realizat cererea nu are permisiunea de a realiza acțiunea cerută
- 404 Not found- Sursa cererii nu a putut fi gasită
- 500 Internal Server Error- Este o eroare generică, dată de obicei când nu se știe sursa erorii

[9] https://docs.smartstore.com/display/SMNET32/Web+API
[10] https://docs.smartstore.com/display/SMNET32/Web+API+in+Detail

Am ales folosirea acestui tip de arhitectură deoarece oferă posibilitatea de separare a interfeței clientului de server și baza de date, scalabilitate și independență față de platformele și limbajele de programare folosite.

4.3.4. MongoDB

Conform [5], MongoDB este un sistem de management al bazelor de date organizat pe documente care oferă performanță crescută, o disponibilitate crescută și scalabilitate automată.

O salvare in baza de date este reprezentată de o structură de forma unei perechi câmp/cheie și valoare. Documentele MongoDB sunt similare obiectelor JSON. O valoare poate contine alte documente, șiruri sau șiruri de documente.

Figura 4.7 Exemplu de înregistrare in MongoDB

MongoDB fiind o bază de date non-relațională oferă avantajul posibilității creării și gestionării unui volum mare de date care poate avea un model bine structurat cât și a unuia nestructurat. Acest lucru oferă flexibilitate, modificările modelelor din server neavând un impact semnificativ asupra modelelor din baza de date precum în bazele de date de tip relațional. Datorită acestor beneficii oferite de MongoDB, am ales sa utilizez acest tip de bază de date, deoarece am dorit crearea rapidă a unui sistem a cărui model de date să se poată schimba ușor și eficient.

4.3.5. HTML, CSS

HTML (HyperText Markup Language), poate fi descris ca și un document scalabil care poate fi folosit pentru schimbul de informații pe orice platformă virtuală. Acest limbaj permite crearea cu ușurință a unor interfețe de natură vizuală sau audio, fiind folosit atât pentru aplicații-utilizator interactive cât și pentru publicații text dat fiind faptul că acesta oferă instrucțiuni specializate pentru crearea de liste, paragrafe, etc.

Pentru a avea o dezvoltare software cât mai curată și lizibilă, s-a apelat la interfața standard DOM (Document Object Model) care poate fi folosită în mai multe medii și aplicați, de pildă documente XML. În acest fel au structurat logica ce poate fi asemănată cu un arbore[11]. În momentul când pagina va fi încarcată, browser- ul va genera DOM- ul acesteia facilitând modul în care elementele unui HTML sunt identificate[12].

[11] https://www.w3.org/TR/WD-DOM/introduction.html
[12] https://www.w3schools.com/whatis/whatis_htmldom.asp

Acest limbaj de dezvoltare web poate îngloba programe de scripting precum JavaScript, care influențează comportamentul elementelor, sau de stilizare precum CSS, care aduce un aport estetic paginii respective.

Deoarece gama de stilziare a elementelor de care dispune limbajul HTML nu era de ajuns pentru dezvoltarea de efecte vizuale complexe, limbajul CSS (Cascading Style Sheets) are un rol important în designul unei pagini. Marele avantaj este ca fisierul CSS este independent și conține anumite stiluri personalizate de dezvoltare, precum culori, fonturi, animații și permite adaptarea la diferite dispozitive din punct de vedere al dispunerii pe ecran. De asemenea, poate fi aplicat mai multor elemente HTML și chiar mai multor documente HTML[13].

Am ales utilizarea paginilor statice HTML împreuna cu CSS deoarece am dorit implementarea unei interfețe utilizator independente de partea de server care sa comunice cu acesta doar prin intermediul protocolului HTTP, conexiune realizata cu ajutorul limbajului JavaScript.

4.3.6. JavaScript și TypeScript

În contextul manipulării obiectelor dintr-o pagină web, aceasta trebuie făcută într-o manieră specializată, așadar limbajul JavaScript[14] se ocupa întocmai de acest aspect.

JavaScript este limbajul ce ajuta la realizarea unui pagini web interactive, fiind flexibil și accesibil. Unul din avantajele acestui limbaj este faptul că poate rula in browser- ul utilizatorului fara sa mai fie nevoie de un server web.

TypeScript poate fi descris ca un succesor al limbajului JavaScript, dar mult mai specializat pe tipurile de date folosite. Dacă în JavaScript exista libertatea de a nu asigna un tip anume unei variabile, putând fi folosită fară constrângeri, acest lucru având atât avantaje cât și dezavantaje, dar TypeScript- ul oferă posibilitatea impunerii unor constrângeri care pot preveni viitoare erori.

Încă diferența între aceste două limbaje este că TypeScriptul este un limbaj compilat, acest lucru putând sa ajute la detectarea mai eficientă și rapidă a erorilor.

Crearea unei interfețe utilizator web interactivă se poate realiza cu ușurința prin utilizarea JavaScript/TypeScript, acesta fiind motivul principal pentru care am ales acest limbaj pentru implementarea sistemului meu.

4.3.7. React

React[15] este o librărie de interfață grafica JavaScript creata de Facebook folosita pentru crearea componentelor de interfață grafică. Cea mai notabilă functionalitate a React-ului este reprezentată de utilizarea componentelor, acestea reprezentând la nivel abstract

[13] https://www.w3.org/standards/webdesign/htmlcss.html
[14] https://www.guru99.com/typescript-vs-javascript.html
[15] https://www.w3schools.com/whatis/whatis_react.asp

un element din DOM. De asemenea, la crearea unor astfel de componente se pot trimite informații prin intermediul unor proprietăți sau “props”.

Aceste componente pot fi de două feluri, componente funcționale și componente de tip clasă. Componenta funcțională este o funcție care trebuie sa returneze un element de tipul JSX, iar componenta de tip clasă trebuie sa implementeze o metodă care să returneze tot un element de tipul JSX.

Elementele de tip JSX[16] (JavaScript XML) sunt o extensie a sintaxei limbajului JavaScript. Este asemanator cu HTML- ul, dar poate conține și alte componente, atribute sau expresii JavaScript.

Am ales sa utilizez React deoarece este o librărie de JavaScript care ofera funcționalități de JavaScript, un mare avantaj oferit de această librărie este posibilitatea reutilizării elementelor, de actualizare a componentelor. Această librărie este recomandată site- urilor în care clientul interacționează mult cu elementele din interfața vizuală a acestuia, cum este și cazul sistemului dezvoltat.

```
const RegisterConfirmation = ({ message, buttonMessage, onClick }) => {
    return (
        <>
            <div>{message}</div>
            <input type='button' onClick={() => onClick()} value={buttonMessage} />
        </>
    );
};
```

Figura 4.8 Componenta funcțională care returneaza element de tip JSX

4.4. Arhitectura conceptuală a sistemului

În acest subcapitol se va prezenta arhitectura conceptuală a sistemului. De asemenea, vor fi enumerate conceptele care au fost luate în considerare pentru o cât mai bună implementare care au ajutat la crearea acestui sistem.

Sistemul este împărțit în trei mari componente:

- Front-end (Client): în aceasta componentă sunt afișate informațiile utilizatorului, este componenta cu care acesta interacționează si comunică prin intermediul protocolului HTTP cu back-end -ul (serverul).
- Back-end (Server): această componentă se ocupă de interpretarea mesajelor primite din front-end și de oferirea unui răspuns acestor cereri. În aceasta componenta are loc toată logica din aplicație
- Baza de date: componentă care are rolul de persistare a datelor.

Componenta de front-end este componenta sistemului care conține fisierele statice (fisierele HTML, cele de JavaScript, TypeScript și React) cu ajutorul cărora interfața

[16] https://en.wikipedia.org/wiki/React_(web_framework)

utilizator este creată. Utilizatorul interacționează cu sistemul direct prin aceasta componentă. Pentru realizarea comunicării cu back-end -ul se utilizează protocolul HTTP prin intermediul căruia front-end- ul trimite o cerere HTTP serverului, asteptând un raspuns din partea acestuia.

Back-end-ul are ca rol principal interpretarea și procesarea cererilor clientului venite din front-end. În acesta are loc toată logica responsabilă modificarii, citirii, adaugarea sau ștergerea datelor sau a interpretării lor cu scopul de a răspunde cererii clienților.

Componenta de bază de date este reprezentată de MongoDB. MongoDB este o bază de date care folosește documente pentru organizarea datelor, fiecare document putând să conțină mai multe înregistrări. Aceasta comunică prin intermediul documentelor JSON binare (BSON) cu back-end- ul.

În figura de mai jos va fi prezentată arhitectura conceptuală a sistemului, componentele principale ale acestuia și structurarea acestora.

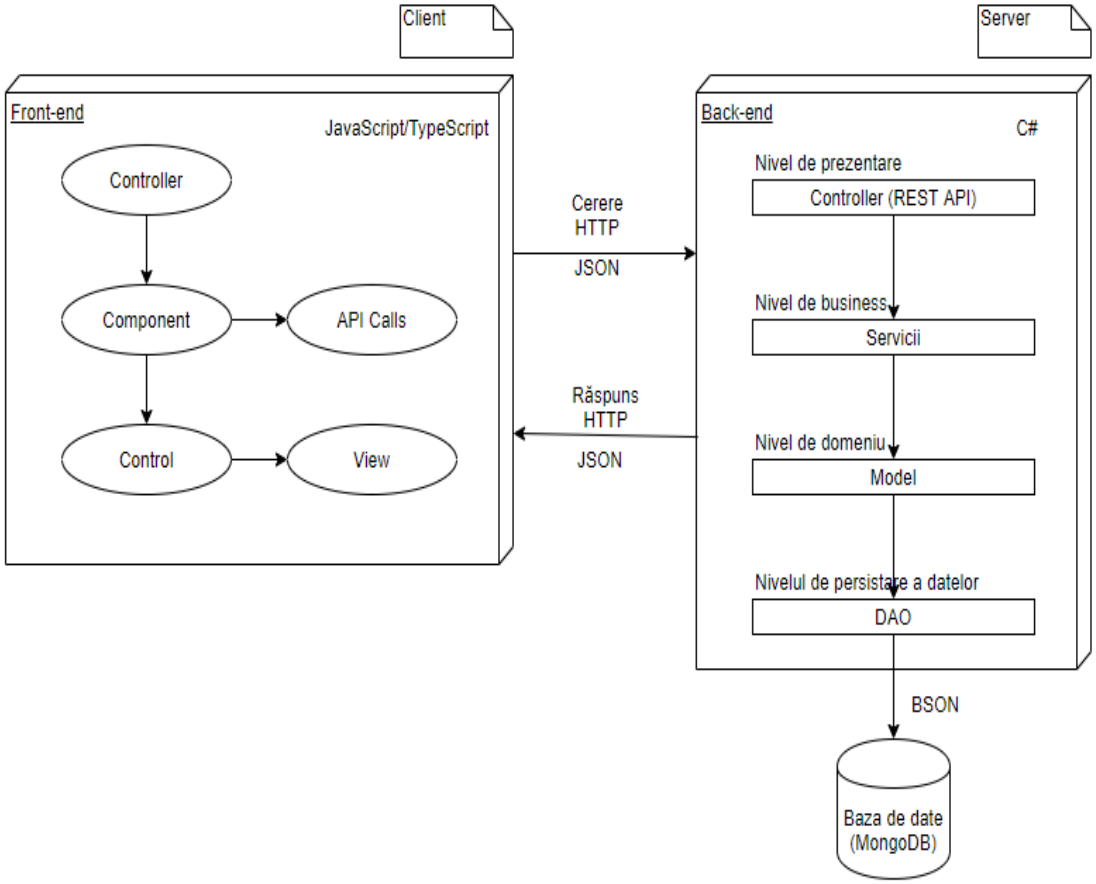

Figura 4.9 Arhitectura sistemului

5. Proictare de Detaliu și Implementare

În acest capitol vor fi prezentate în detaliu deciziile luate pe parcursul implementării sistemului, structura aplicației și secvențe de cod relevante.

7.1. Arhitectura sistemului

Arhitectura de baza a sistemului, ca orice aplicație de tip client-server, este împărțită în trei mari parți:

- nivelul de prezentare (front-end)
- nivelul de business (back-end)
- nivelul de acces la baza de date (Data Access Layer)

Am ales împărțirea sistemului în aceste componente pentru o cât mai buna separare a responsabilităților. Acest lucru ajutând la o dezvoltare a sistemului cât mai eficientă când vine vorba de adăugarea iterativă de noi funcționalități.

7.1.1. Nivelul de prezentare

Cu acest nivel intracționează clientul. Acesta are rolul de a interpreta datele de intrare provenite de la client, de a le decodifica și de a efectua cererile corespunzătoare către server. De asemenea, trebuie să afișeze răspunsul serverului, să afișeze datele primite de la acesta într-un mod cât mai ușor de înțeles de către client și cât mai intuitiv.

Cu toate că acest nivel are drept scop principal afișarea informațiilor clientului și metode de introducere sau modificare a datelor din sistem, exista și logică în acest nivel. Aceasta are drept scop prevenirea utilizatorului de a introduce date invalide în aplicație (exemple de validări: validare de email, validarea unei date introduse, validare a parolei în cazul în care se cere o confirmare a acesteia). Logica din nivelul de prezentare ajută la oferirea unui feedback rapid utilizatorului în cazul introducerii unor date invalide, fără să mai trimită acele date serverului, așteptând invalidarea acestora de către server.

Un exemplu de acest tip de validare din sistemul implementat este prezent la crearea unui nou mesaj de către un utilizator anonim, acesta neputând să trimită mesajul dacă nu a introdus o adresă de email validă.

În imaginea de jos va fi prezentat un exemplu pentru modul în care sistemul a fost împărțit nivelului de prezentare, componentele din care este alcătuit și responsabilitatea fiecăreia.

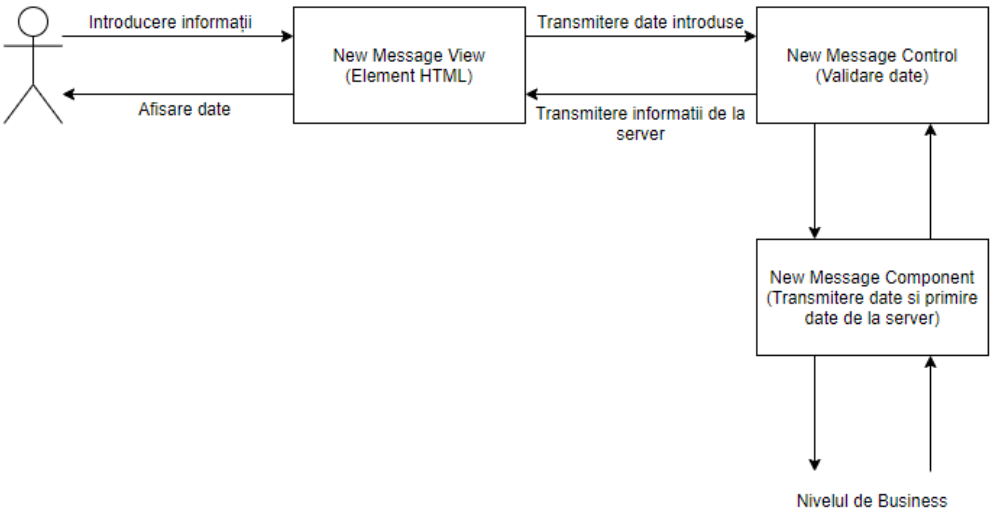

Figura 5.1 Validarea datelor în cazul introducerii unui nou mesaj

Folosind React, am ales împarțirea nivelului de prezentare în componente responsabile de afișare a informațiilor si componete responsabile de validarea informațiilor introduse sau afișate și de transmitere mai departe a acestora.

Arhitectura front-end -ului este împărțită în Controllere, Componente, Controale si View-uri (elementele care vor fi afișate).

Elemente de View reprezintă elementele de HTML care vor fi afișate utilizatorului. Scopul lui principal este de afisa informațiile și de a le trimite elementului de Control din care fac parte datele introduse de utilizator și acțiunile realizate de acesta.

Elementul de control este responsabil de validarea datelor introduse de utilizator în elementul de View, de transmitere a datelor și erorilor către elementul de View. După validarea datelor, daca a fost necesară, acestea sunt transmise Componentei părinte, acesta la rândul lui transmițând răspunsul înapoi elementului de Control.

O componentă are rolul de a realiza apelurile de HTTP sau alte acțiuni mai complexe și de a interpreta răspunsul de la server și transmiterea de date unui element de Control cu scopul de a afișa informația primită sau există cazuri în care această acțiune sau apel HTTP este doar un pas dintr-o acțiune mai complexă, caz în care componenta transmite confirmarea realizării acțiunii elementului părinte, care este un element de tipul Controller, acesta hotărând în funcție de rezultatul primit următorul pas care poate reprezenta utilizarea unei alte componente.

În imaginea de mai jos este prezentată arhitectura conceptuala a front-end -ului cu toate elementele ei (Controllere, Componente, Controale si View- uri).

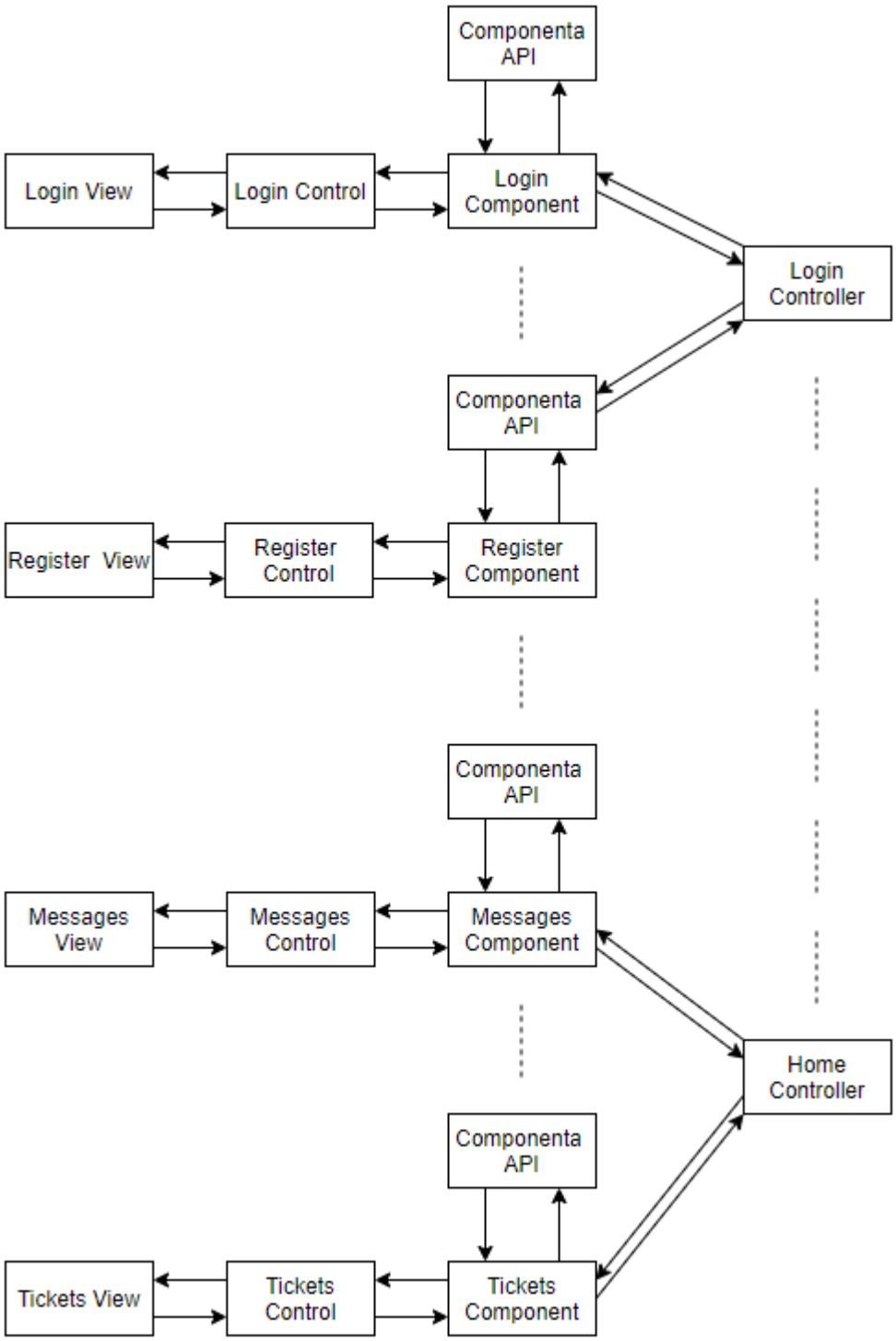

Figura 5.2Arhitectura front-end -ului

7.1.2. Nivelul de logică (Business Layer)

Acest nivel are drept scop realizarea de logică/business pentru soluționarea cererilor clientului și de a face legătura între front-end si back-end cu ajutorul unui controller al cărui rol este de a gestiona cererile HTTP din partea clientului. De asemenea, acest nivel are rolul de a realiza legătura între server și baza de date.

În figura de mai jos este prezentată structura acestui nivel si componentele principale ale acestuia:

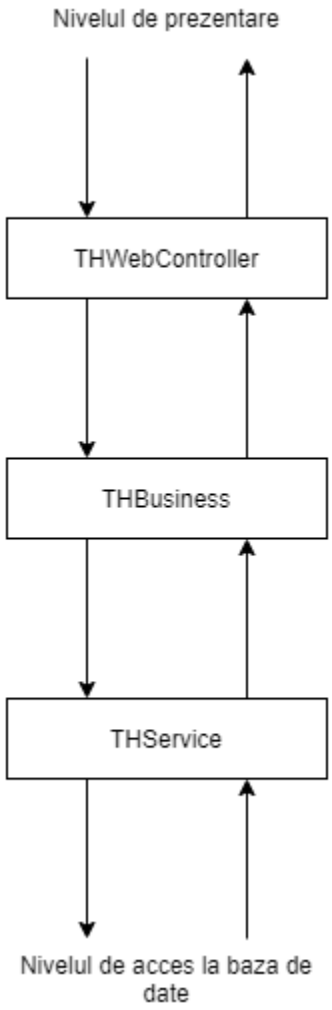

Figura 5.3 Nivelul de business al sistemului

Controllerul API are rolul de a gestiona cererile clientului din front-end, de transmitere a datelor serverului și a răspunsului serverului inapoi clientului.

Aceste răspunsuri din partea controllerului returnează clientului un anumit cod care reprezintă statusul operației realizate (acesta poate informa utilizatorul că acțiunea a fost realizată cu succes sau nu, că cererea este una invalidă sau ca utilizatorul nu are permisiunile necesare realizării unei asemenea acțiuni).

Componenta de business are drept scop validarea datelor primite, de a apela metodele din componenta de service cu scopul de realizare a unei cereri mai complexe și oferirea unui răspuns Controllerului.

Am ales utilizarea unei clase separate de service pentru a avea o cât mai buna separare a responsabilităților în acest sistem. Fiecare serviciu se va ocupa de logica unui singur tip de date, componenta de business având ca scop, după cum a fost spus mai sus, realizarea de acțiuni mai complexe, pe mai multe tipuri de date, prin intermediul componentelor de service.

7.1.3. Nivelul de acces la baza de date (Data Access Layer)

Nivelul de acces la baza de date are rolul de a realiza transferul de date de la server la baza de date si de la baza de date către server. Aceste date sunt transmise în format BSON (JSON Binar).

În acest nivel are loc serializarea și deserializarea datelor după un anumit model predefinit. Fiecărui model i s-a definit modul de mapare a datelor. La tipurile de date cum ar fi numerele sau șirurile de caractere și vectorii de aceste tipuri, datele sunt mapate fara modificări, dar pentru datele mai complexe a fost nevoie de o logică suplimentară pentru o folosire cât mai eficientă a memoriei disponibile. Un exemplu de asemenea mapare este la obiectele între care exista o relație de 1 la 1, 1 la N, sau N la N: am ales în implementarea mea ca aceste obiecte la care există referință să nu fie stocate în totalitate pentru fiecare obiect, acest lucru eliminând posibilitatea de duplicare a datelor, ci doar identificatorul acestora. Acest lucru ajută și la un transfer mai rapid al datelor deoarece nu trebuie serializate sau deserializate informații pot să nu fie relevante într-o anumită situatie (de exemplu: citirea credențialelor unui utilizator nu necesita și citirea tuturor mesajelor trimise de acesta).

De asemenea, în acest nivel au loc și operațiile efectuate pe baza de date,cum ar fi citirile, actualizările, ștergerile sau inserările.

În urmatoarea imagine va fi prezentată componenta nivelul de acces la baza de date și cum interacționează aceasta cu celelalte componente ale sistemului.

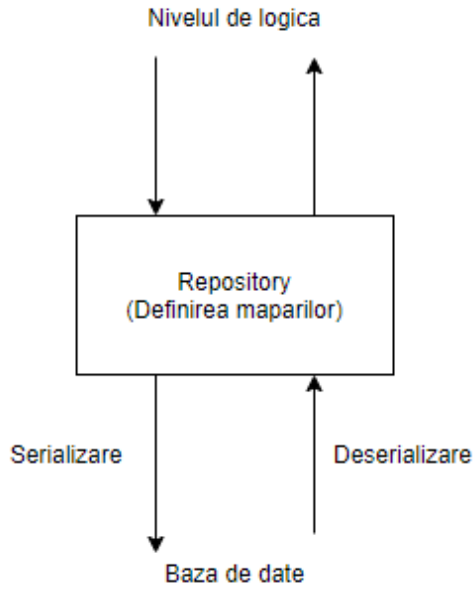

Figura 5.4 Nivelul de acces la baza de date (Data Access Layer)

7.1.4. Baza de date

MongoDB este baza de date utilizată în acest sistem. Rolul bazei de date este de persistare a datelor, dar și de manipulare a acestora. Legătura între baza de date și server este realizată de nivelul de acces la baza de date.

Mai jos va fi prezentată relația între baza de date și back-end.

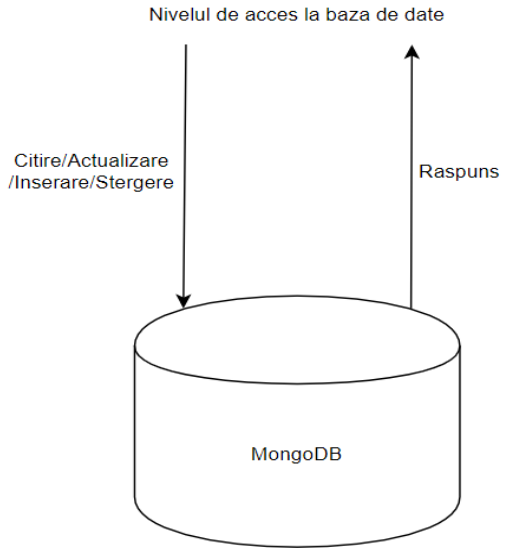

Figura 5.5 Legătura între baza de date și back-end a sistemului

7.2. Diagramele sistemului

7.2.1. Diagrama de deployment

Diagrama de deployment[17] are rolul de a prezenta componentele fizice ale unui sistem de componente software. Acestea au rolul de a se concentra asupra topologiei hardware a sistemului, fiind utilizate de inginerii sistemului.

Diagrama de deployement a sistemului propus în aceasta lucrare este afișată mai jos în figura următoare. În aceasta se poate observa separarea clientului de server, clientul comunicând cu serverul prin intermediul protocolului HTTP.

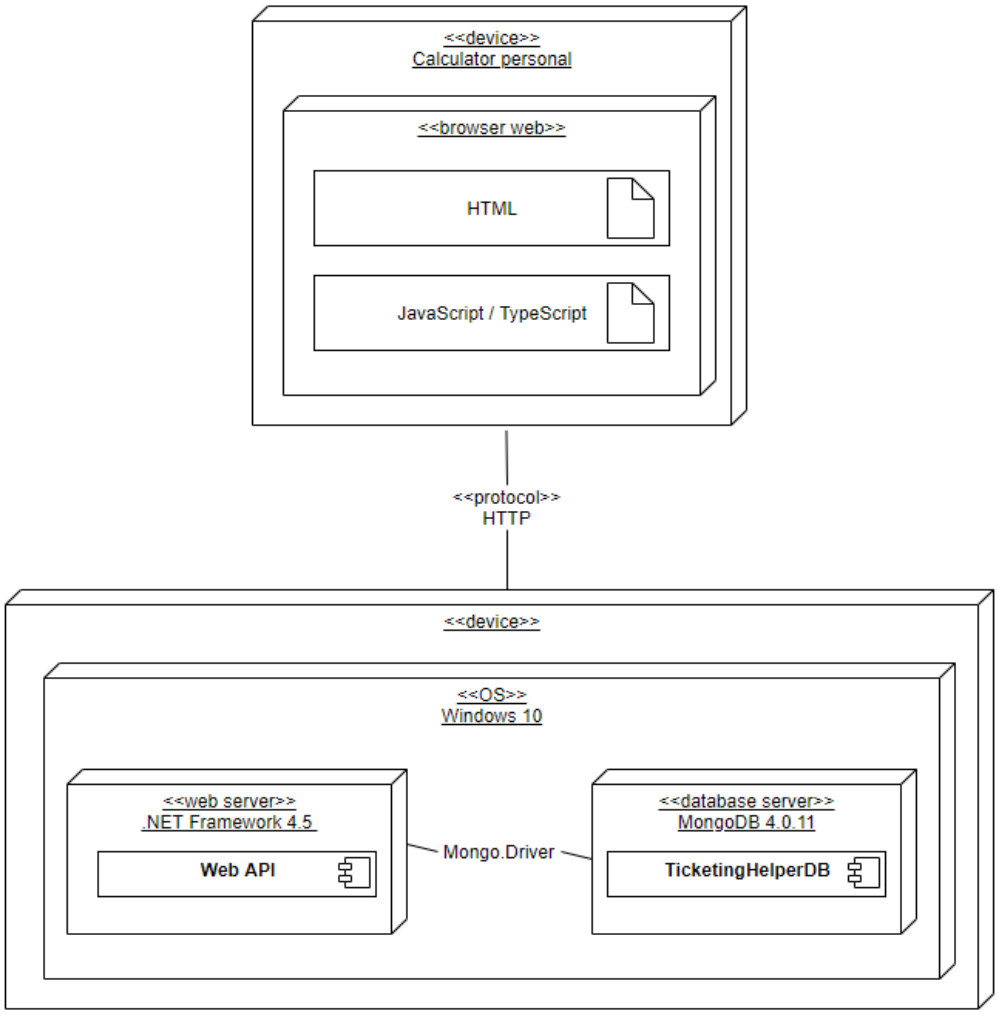

Figura 5.6 Diagrama de deployment a sistemului

[17] https://sites.google.com/site/uml4students/diagrama-de-desfurare-diagrama-de-componente-diagrama-de-pechete

7.2.2. Diagrama de pachete

În dezvoltarea sistemului s-a ținut cont de principiul separării responsabilităților. Prin urmare, în front-end exista urmatoarele pachete:

- Controller: responsabil de managementul componentelor
- Component: apelarea metodelor responsabile de cererile la server si crearea de controale
- Control: se ocupă de validarea datelor introduse de client
- View: responsabil de afișarea propriu-zisă a elementelor HTML
- Service: executa cererile la server
- Utils: conține elemente de funcționalitate care pot fi folosite în întreaga aplicație.

Diagrama de pachete a front-end -ului este prezentată în imaginea urmatoare.

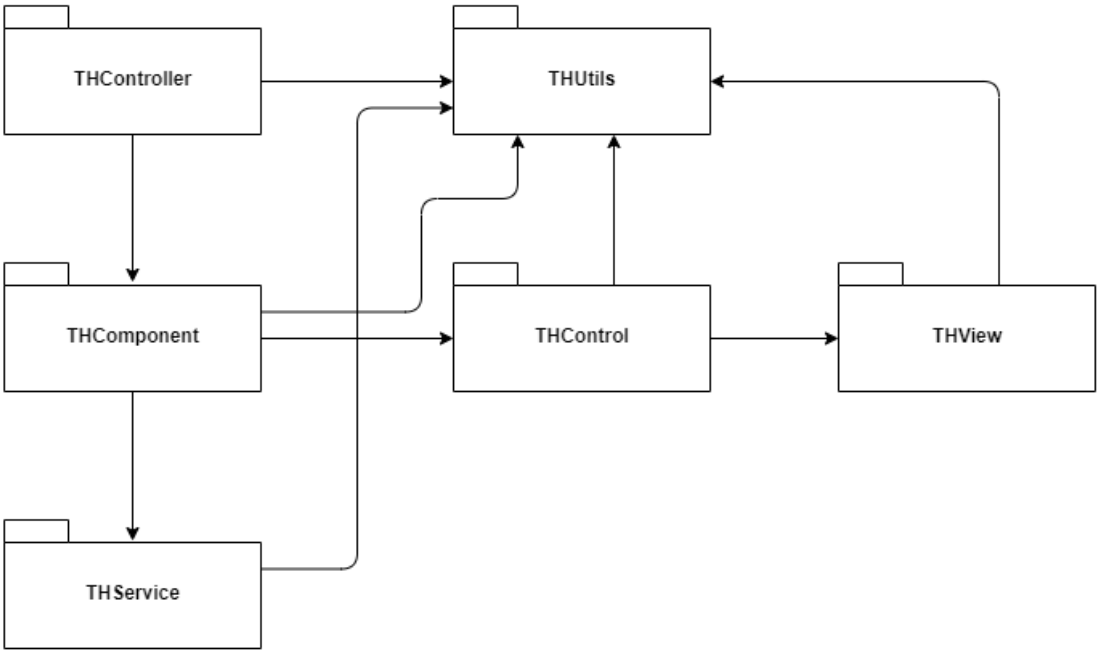

Figura 5.7 Diagrama pachetelor pentru front-end

Pentru backend împărțirea pe pachete a fost gândită astfel:

- Security: acest pachet este responsabil de verificarea și acordarea permisiunilor utilizatorilor care efectuează cereri prin intermediul protocolului HTTP.
- Controller: obiectivul elementelor din acest pachet este de a expune datele aplicației prin intermediul uni API de tip RESTful, de a răspunde cererilor de tip HTTP din partea de front-end.

- Business: responsabil de executarea de acțiuni complexe folosind elementele din pachetul de servicii.
- Service: in acest pachet împărțirea pe clase este una strictă, va exista un serviciu pentru fiecare model din baza de date.
- Model: în acest pachet se găsesc modelele utilizate în întreaga aplicație.
- Repository: pachet responsabil cu executarea de actiuni în baza de date și de definire a metodelor de mapare a modelelor în baza de date.
- Config: se realizează configurările back-end -ului, în special cele de configurare a Web API -ului, utilizat pentru implementarea servicilor RESTful.
- Utils: conține elemente de funcționalitate care pot fi folosite în întreaga aplicație.

Diagrama de pachete este prezentată în următoarea figură.

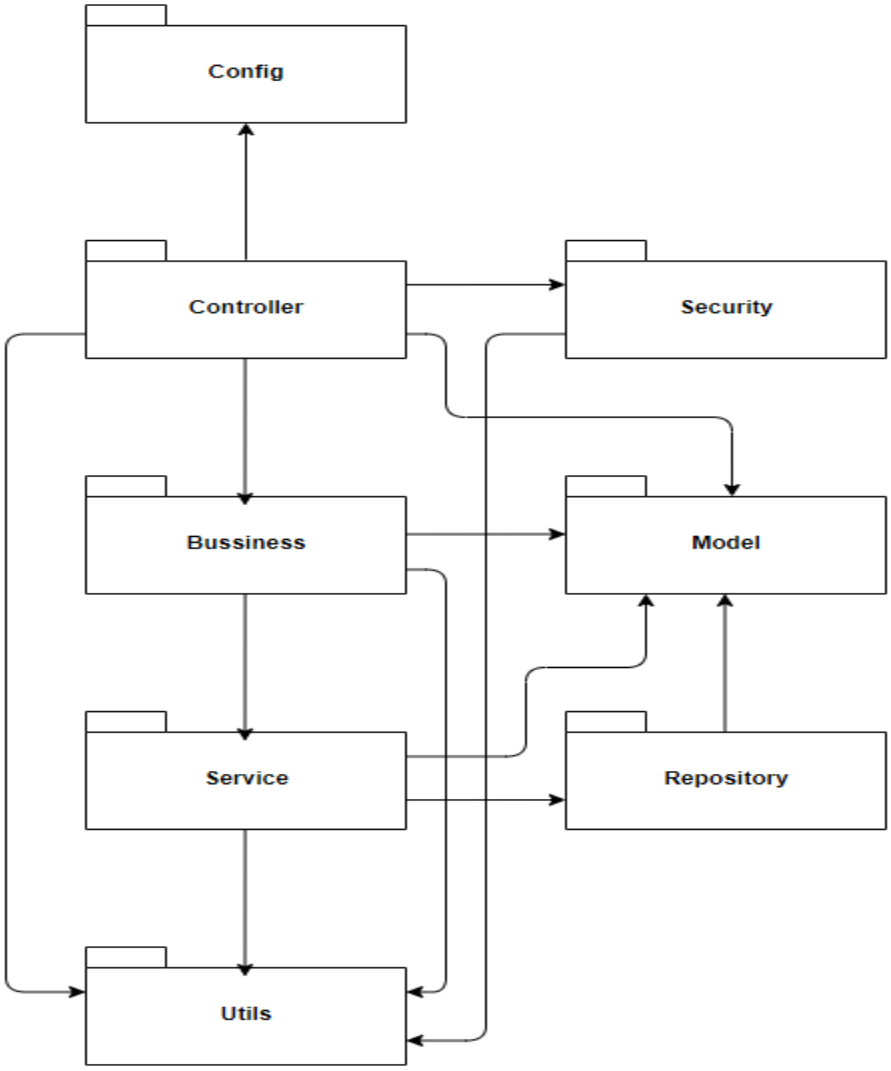

Figura 5.8 Diagrama pachetelor pentru back-end

7.3. Elemente de implementare

Acest subcapitol are ca scop prezentarea secvențelor de cod semnificative utilizate în sistemul implementat, avantajele aduse de această implementare, dar și posibilele compromisuri aduse de aceasta.

7.3.1. Implementare front-end

Restricționarea accesului la anumite rute persoanelor neautorizate

Pentru a trata cazul în care anumite persoane neautorizate încearca să acceseze rute la care nu au acces si pentru a evita cod duplicat am ales crearea unui componente funcționale care să verifice dacă există un utilizator logat și dacă este logat sa verifice dacă are permisiunile necesare accesării acelei rute. În cazul în care utilizatorul care încearcă sa acceseze o rută la care nu are suficiente permisiuni va fi redirecționat pe o pagină prestabilită cum ar fi pagina de login sau o pagină care sa informeze utilizatorul că nu are acces la resursele respective.

Aceasta componentă este prezentată în imaginea de mai jos.

```
const AuthorizedRoute = (WrappedComponent, allowedRoles) => {
    const Authorization = (props) => {
        const [user] = useContext(UserContext);
        return <>{allowedRoles.includes(user.Details.Role) ?
            <WrappedComponent {...props} /> :
            <unauthorizedAccessRedirectPage />}</>;
    };

    return <Authorization {...props} />;
};
```

Figura 5.9 Componenta funcțională pentru testarea permisiunilor utilizatorului

Executarea de cereri către server

Dupa trimiterea unei cereri prin intermediul protocolului HTTP serverului se așteaptă un răspun cu un anumit format. Astfel, pentru a nu scrie de fiecare dată testare validității răspunsului am ales crearea unei metode care să execute un astfel de cerere și sa trateze cazurile peste care poate da la un astfel de apel. Această metodă asteaptă ca parametrii funcției care trebuie apelată pentru realizarea cererii, metoda care va fi apelata daca statusul răspunsului de la server este unul favorabil și încă două metode care vor fi apelate în cazul în care se primește un răspuns de la server, dar nu s-a putut realiza cererea respectiva din anumite cauze, respectiv o metodă care va fi executată în cazul în care nu s-a putut realiza conexiunea cu serverul.

Codul metodei va fi prezentat în figura următoare:

```
export const ExecuteApiCall = (
    apiCall: Function,
    onSuccess: Function,
    onFail?: Function,
    onError?: Function
    ) => {
        apiCall()
            .then((rawResponse) => {
                if (!rawResponse.ok) {
                    if (!IsNullOrUndefined(onFail))
                        onFail(rawResponse);
                    else
                        console.error(rawResponse);

                    return;
                }
                rawResponse.json().then((response) => {
                    onSuccess(response);
                });
            })
            .catch((error) => {
                if (!IsNullOrUndefined(onError)) {
                    onError(error);
                } else {
                    console.error(error);
                }
            });
};
```

Figura 5.10 Metoda generică de tratare a apelurilor către server

7.3.2. Implementare back-end

Maparea obiectelor din back-end în MongoDB

În implementarea sistemului, am decis ca obiectele care conțin alte obiecte ca atribute, la momentul salvării lor în baza de date să li se salveze doar o referință către acel obiect. Prin urmare, am avut nevoie de un mod de a identifica fiecare obiect în mod unic. Această problemă a fost realizată prin crearea unei clase de bază pe care o moștenesc toate clasele din sistem care să conțina un atribut cu ajutorul căruia să se fie identificat obiectul.

De asemenea, a fost nevoie de o metoda de serializare si deserializare a acestor obiecte în momentul trimiterii acestora către baza de date sau primirii lor în urma interogării acesteia. Datorită faptului ca toate clasele moștenesc clasa cu ajutorul careia se realizeaza identificarea unica a fiecărui obiect, a fost nevoie doar de implementarea a două clase pentru serializarea și deserializarea obiectelor, una în cazul obiectul conține ca atribut doar un obiect și alta în momentul în care exista o referință la o listă de obiecte.

În urma creării acestei clase, s-a putut realizare serializarea și deserializarea obiectelor din backe-end în MongoDB și invers fără probleme, singurul dezavantaj al acestei implementări fiind că dupa ce s-a realizat o interogare a bazei de date și s-a returnat un astfel de obiect care are referință la alte obiecte, pentru a afla informațiile despre acel obiect către care face referință va mai fi nevoie de încă o interogare a bazei de date pentru acel obiect.

Implementarea clasei care realizeaza serializarea si deserializarea în cazul în care exista referință doar către un singur obiect este prezentată în imaginea de mai jos.

```
18 references
public class BaseModelByUidCustomBsonSerializer<T> : SerializerBase<T>
    where T : BaseModel, new()
{
    0 references
    public override T Deserialize(BsonDeserializationContext context,
                                  BsonDeserializationArgs args)
    {
        if (context.Reader.CurrentBsonType == BsonType.String)
        {
            return new T
            {
                Uid = new Guid(context.Reader.ReadString())
            };
        }

        context.Reader.ReadNull();
        return null;
    }

    0 references
    public override void Serialize(BsonSerializationContext context,
                                   BsonSerializationArgs args, T value)
    {
        var serializer = BsonSerializer.LookupSerializer(typeof(string));
        serializer.Serialize(context, value != null ? value.Uid.ToString() : null);
    }
}
```

Figura 5.11 Implementarea serializării si deserializării obiectelor

Crearea unei clase de bază pentru efectuarea operațiilor pe baza de date

Având în vedere că pe toate documentele din baza de date trebuie să se execute operații de citire, actualizare, inserare sau ștergere am considerat că o clasă de bază care să

execute toate operațiile de bază pe un document va ajuta la reducerea de cod duplicat. Prin urmare am creat o clasă generică. Acesta mai executa pe lângă operațiile de bază si posibilitatea de citire dupa anumite valori dintr-un câmp din valorile din baza de date. Astfel, după crearea acestei clase, pentru a putea face modificări sau citiri pe baza de date trebuie creată o clasă nouă care sa specifice doar numele documentului din baza de date și tipul obiectului așteptat în urma unei interogări.

Imaginea de mai jos prezintă implementarea acestei clase.

```
6 references
public class Base<T> : IBasicTableQueries<T>
    where T : BaseModel
{
    Properties

    Constructor

    Members

    #region DatabaseOperations

    1 reference
    public void DeleteByUid(string uid)...

    1 reference
    public void DeleteByUids(IEnumerable<string> uids)...

    5 references
    public List<T> ReadAll()...

    2 references
    public T ReadByUid(string uid)...

    2 references
    public List<T> ReadByUids(IEnumerable<string> uids)...

    8 references
    public List<T> ReadByColumnValue(string columenName, object value)...

    3 references
    public List<T> ReadByColumnValues(string columenName, IEnumerable<object> values)...

    5 references
    public bool Save(T dataSet)...

    3 references
    public bool Save(IEnumerable<T> dataSet)...

    2 references
    public bool Update(T dataSet)...

    1 reference
```

Figura 5.12 Clasa de bază pentru efectuarea operațiilor pe baza de date

Autorizarea cererilor HTTP cu ajutorul JSON Web Token

Pentru a împiedica sau reduce cât mai mult accesul persoanelor neautorizate am ales utilizarea în implementarea sistemului a unui sistem de autentificare de tip JSON Web Token. Un astfel de sistem are ca și componente principale: o componentă de validare a unui nou utilizator care se autentifică în sistem și una de configurare a Controllerului Web API.

Componenta de validare este reprezentată de o clasă care trebuie sa moștenească de la clasa *OAuthAuthorizationServerProvider* și să îi suprascrie metodele *GrantResourceOwnerCredentials* și *ValidateClientAuthentication*. Prima metodă are rolul de a valida credențialele introduse de utilizator la autentificare, iar în cazul în care acestea au fost corecte să creeze un token pe care îl va asocia acelui utilizator. A doua metodă are rolul de a valida cererile provenite prin protocolul HTTP, de a testa token-ul si de realiza maparea între token și client.

Componenta de configurare este prezentată în imaginea următoare.

```
1 reference
public class Startup
{
    0 references
    public void Configuration(IAppBuilder app)
    {
        OAuthAuthorizationServerOptions options = new OAuthAuthorizationServerOptions
        {
            AllowInsecureHttp = true,
            //The Path For generating the Token
            TokenEndpointPath = new PathString("/api/authentication/token"),
            AuthorizeEndpointPath = new PathString("/api"),
            //Setting the Token Expired Time (24 hours)
            AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
            //MyAuthorizationServerProvider class will validate the user credentials
            Provider = new AuthorizationServerProvider()
        };
        //Token Generations
        app.UseOAuthAuthorizationServer(options);
        app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());

        HttpConfiguration config = new HttpConfiguration();
        WebApiConfig.Register(config);
        app.UseWebApi(config);
        app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
    }
}
```

Figura 5.13 Clasa de configurare a JWT

În această componentă se configurează Controllerul Web API utilizat pentru gestionarea cererilor de la client provenite prin intermediul protocolului HTTP. In variabila *option* se poate observa că s-au realizat anumite setări cum ar fi permiterea cereri anonime (*AllowInsecureHttp = true*), s-a setat calea pentru autentificare si calea în care este nevoie

de autentificare. Linia de cod *Provider = new AuthorizationServerProvider()*, reprezintă definirea clasei care realizează validarea clienților și în care se asociază tokeni utilizatorilor, fiind prima componentă principala din sistemul de autorizare a cererilor de HTTP de care am precizat mai sus.

Prin intermediul unui token- ului se pot realiza identificarea utilizatorului printr-un identificator unic salvat în momentul autentificării și a rolului acestuia. De exemplu, în cazul unei cereri a mesajelor unui utilizator singurul identificator care va fi trimis serverului este token- ul utilizatorului. Astfel, serverul va identifica utilizatorul pe baza token- ului fără a fi nevoie de trimiterea de date suplimentare din browser (din front-end).

În figura următoare se poate observa cum serverul filtrează cererile în funcție de rolul utilizatorilor și că utilizează datele utilizatorului asociate token-ului trimis de acesta.

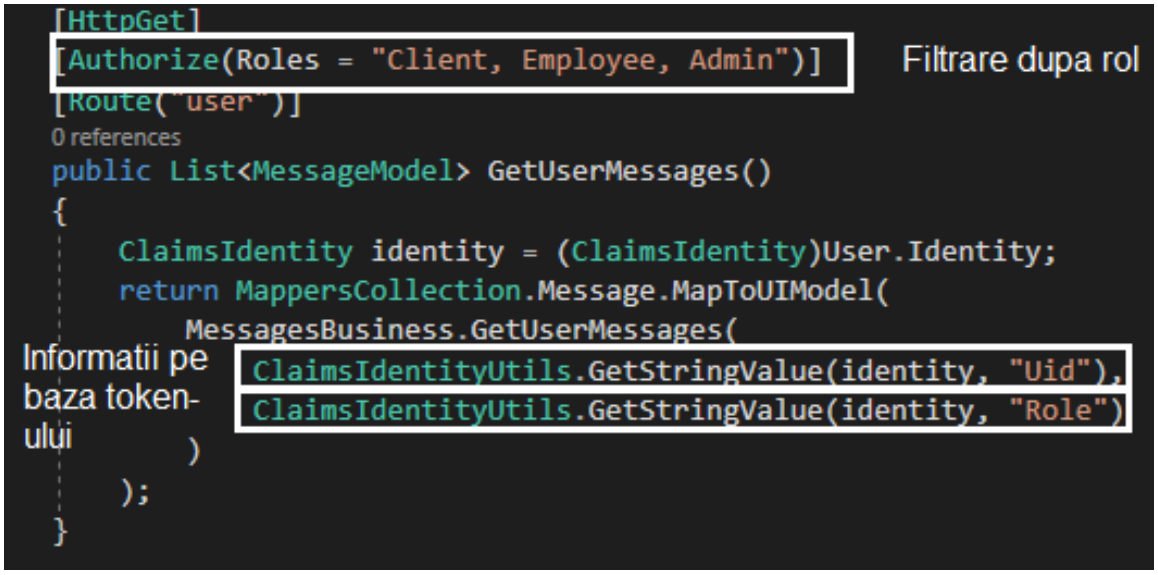

Figura 5.14 Citire mesaje utilizator

Se poate observa cum cererea din partea clientului nu trimite date suplimentare (metoda *GetUserMessages* nu are parametri). Serverul obține datele necesare prin intermediul token- ului. Variabila *identity* conține informațiile clientului și cu ajutorul acesteia se obțin datele necesare căutării și returnării datelor cerute de client.

Separarea logicii între componenta de Business și cea de Service

Am ales crearea unor clase de Servicii care să realizeze operații doar pe un anumit model pentru a respecta cât mai bine principiul responsabilității unice a unei clase. Prin urmare clasele de service realizează comunicarea cu nivelul de acces la baza de date, fiind responabile doar de o singură colecție din baza de date. Clasele de tip business au fost create pentru realizarea unor operațiuni mai complexe, acțiuni care necesită accesarea mai multor tipuri de date (colecții din baza de date) sau modificarea lor. De asemenea, acestea au fost create pentru realizarea de logică suplimentară care nu poate fi încapsulată într-o singură clasă de service.

Un astfel de exemplu este prezent în clasa de business pentru mesaje din imagina de mai jos.

```
2 references
public TicketModel AddNewMessageWithNewTicket(MessageModel newMessage)
{
    MessageModel message = MessagesService.AddNewMessage(newMessage);
    return TicketService.CreateTicketFromMessage(message);
}
```

Figura 5.15 Adăugare de mesaj nou și crearea de tichet

Secvența de cod de mai sus reprezintă o metodă din clasa de business pentru clasa de mesaje. Metoda realizează crearea unui mesaj prin apelarea metodei de creare de mesaj din clasa de servicii corespunzătoare mesajelor, după care a apelat din clasa de servicii pentru tichete metoda de creare a unui nou tichet care conține un mesaj, mesajul creat anterior.

7.4. Proiectarea bazei de date

Pentru persistarea datelor, sistemul folosește ca bază de date MongoDB, o baza de date non-relațională organizată pe documente. Baza de date este organizată conform figurii prezenate pe următoarea pagină.

Se va putea observa că există 6 documente în această baza de date, fiecare având cate o cheie primară (_id). Această cheie primară reprezintă un atribut obligatoriu oricărei înregistrări din baza de date, iar dacă aceasta nu este specificată, se va crea o cheie unică pentru aceasta. Nu există relații declarate propriu-zis în această bază de date între documente.

De asemenea, dacă există relații între două obiecte (pe partea de back-end), cum ar fi în cazul unui tichet care are în alcatuirea lui si un mesaj, am ales ca în înregistrarea respectivă din document să salvez doar identificatorul unic (_id) al înregistrării respective din celălalt document (valoarea atributului _id a înregistrării din documentul “Messages”). Prin salvarea doar a identificatorului unic și nu a tuturor informațiilor despre altă înregistrare s-a evitat salvarea datelor duplicate și nevoia de actualizare a unui set de date în mai multe locuri.

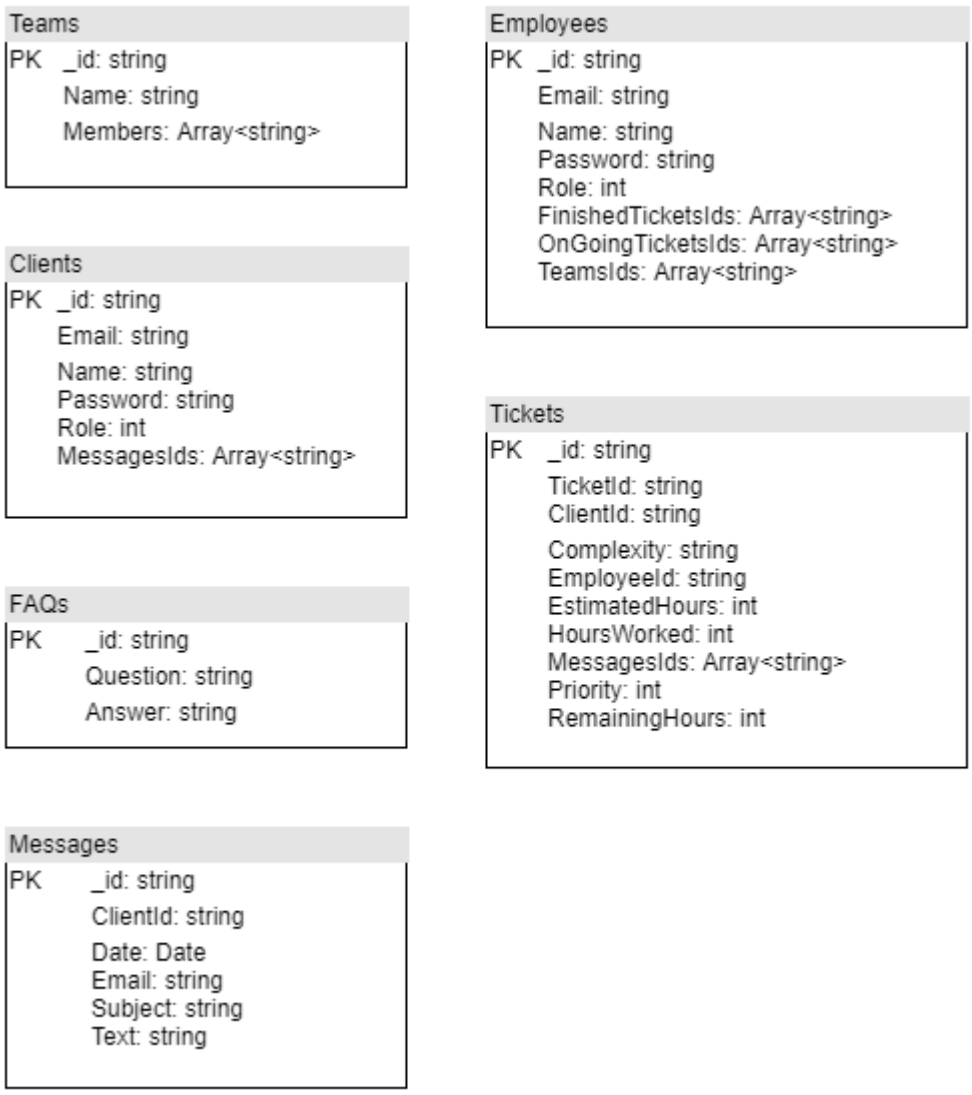

Figura 5.16 Diagrama bazei de date

Documentele din această bază de date sunt urmatoarele:

Tickets

Înregistrările din acest document conțin informațiile necesare identificării unui tichet, cum ar fi mesajele pe care le conține, stadiul în care se află tichetul, angajatul care se ocupă de acesta si informații referitoare la timpul necesar realizării acelei sesizări în urma căreia tichetul a fost creat.

Messages

Acest document conține toate mesajele sistemului.

Clients

În documentul destinat înregistrării datelor despre clienți se află informațiile despre fiecare client, precum datele necesare autentificării, o listă cu identificatorii fiecărui mesaj trimis si a tichetelor create de pe urma acestor mesaje.

Employee

Acest document este destinat salvării datelor despre angajații companiei care utilizează acest sistem. Printre aceste informații se numară credențialele, tichetele rezolvate și cele de care se ocupă în momentul de față și echipele/echipa din care face parte.

Teams

În acest document sunt salvate datele despre echipe, cum ar fi numele si o listă cu identificatorii unici ai fiecărui membru al echipei. Membrii echipei sunt angajați (înregistrări din documentul *Employees*).

FAQs

Înregistrările din acest document reprezintă întrebarile cel mai des întâlnite în cadrul companiei din partea clienților împreună cu rezolvarea acestor probleme, fără a exista nevoia asistenței din partea unui angajat al companiei.

6. Testare și Validare

În acest capitol vor fi prezentate noțiunile generale ale testării manuale și automate și vor fi prezentate și testele realizate pentru a testa sistemul prezentat în această lucrare.

În cadrul acestui capitol trebuie menționate următoarele tipuri de testări:

- Unit testing
- Integration testing
- Full-System testing
- Acceptance testing

Unit testing

Testarea în timpul implementării a fost realizată la crearea fiecărei clase cu ajutorul acestui tip de testare. Acest lucru ajutând la găsirea și rezolvarea problemelor.

În cadrul acestui sistem merită menționate testele realizate pentru a valida integritatea transmiterii și primirii datelor de la baza de date, adică serializarea și deserializarea datelor.

În imaginile de mai jos poate fi observat cum s-a realizat maparea modelului folosit în back-end și reprezentarea lui în baza de date.

```
BsonClassMap.RegisterClassMap<FrequentlyAskedQuestionModel>(cm =>
{
    cm.AutoMap();
    cm.SetIgnoreExtraElements(true);
    cm.MapProperty(faq => faq.Question)
      .SetElementName(Constants.FAQS_QUESTION);
    cm.MapProperty(faq => faq.Answer)
      .SetElementName(Constants.FAQS_ANSWER);
});
```

Figura 6.1 Maparea clasei de întrebari frecvente și răspunsuri în back-end

```
_id: "ebe26c4e-fee5-4683-9936-bc531f084cb6"
Question: "O intrebare des intalnita"
Answer: "Raspuns la intrebare"
```

Figura 6.2 O înregistrare a clasei de FAQ în baza de date

Se poate observa că în backend se specifică denumirea proprietăților clasei model (Constantele *FAQ_QUESTION* reprezintă șirul de caractere *Question* și *FAQ_ANSWER* reprezintă șirul de caractere *Answer*), iar cheile proprietăților coincid cu acestea.

Integration testing

Prin realizarea acestor teste se validează comunicarea între componentele individuale, testate înainte, ale sistemului.

În cadrul acestui tip de testare am ales testarea și validarea serviciilor REST ale sistemului și a componentei de validare a token-ului.

Pentru testarea cererilot HTTP din front-end am utilizat Postman. Au fost luate în considerare urmatoarele cazuri:

- cererea de autentificare
- cererea de date fără a avea permisiunile necesare
- cererea de date folosind un token care a fost asignat utilizatorului în urma unei autentificări reușite

În cazul testării unei cereri de autentificare se asteaptă un răspuns de la server. În urma unui astfel de apel, dacă datele de autentificare au fost introduse corect, serverul va trimite ca răspuns un token cu ajutorul căruia utilizatorul va putea să execute alte apeluri către server. Un astfel de răspuns este prezentat în imaginea de mai jos.

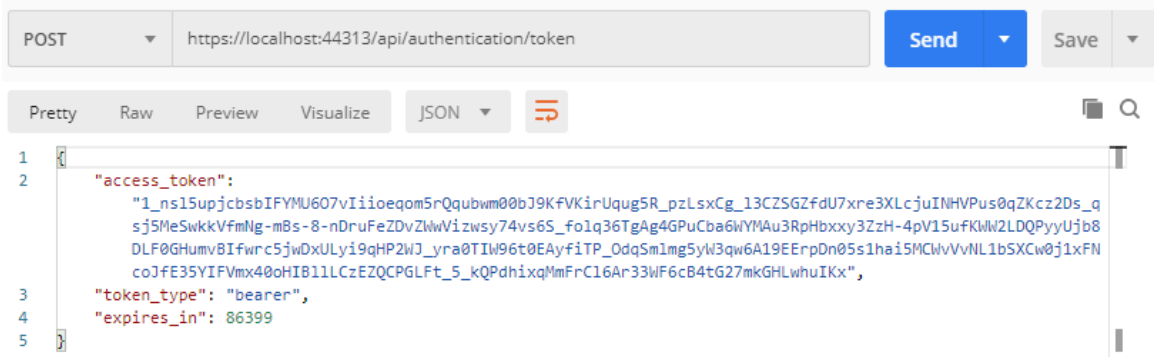

Figura 6.3 Răspuns din partea serverului cu token-ul asignat utilizatorului

În cazul în care nu s-a gasit un utilizator ale cărui credențiale sa coincida cu cele introduse de utilziator, serverul va trimite un mesaj sugestiv ca răspuns exact ca în figura de ma jos.

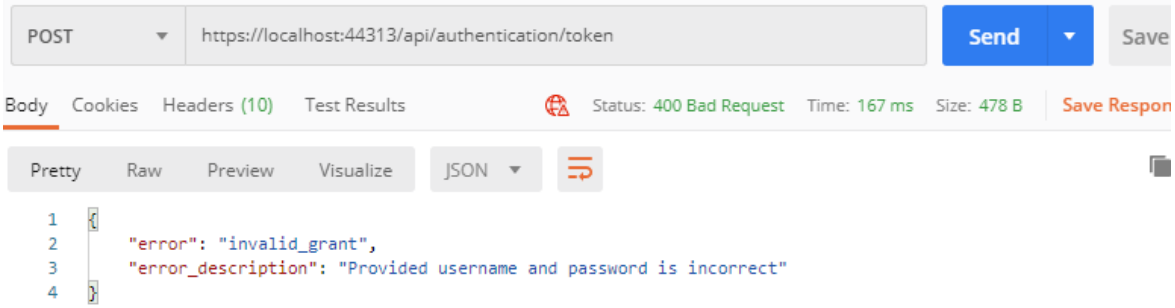

Figura 6.4 Răspuns din partea serverului la o autentificare invalidă

Pentru testarea securității serverului s-au facut apeluri HTTP care necesită autorizație din Postman fără a transmite un token în header-ul cererii. În urma acestui apel serverul în răspunsul acestuia nu au fost trimise datele cerute. În răspuns a fost trimis un mesaj care menționa că a fost respins accesul.

Pentru testarea cererilor la server cu token au fost urmați pașii:

- Autentificare cu un utilizator valid
- Realizare de apel HTTP către server, introducând în secțiunea de Header a cererii și token-ul primit în urma autentificării

S-a testat cererea de la server de mesaje ale unui utilzator și tichete asignate unui angajat.

Atât în cazul testării unui apel neautorizat cât și a unuia cu token răspunsurile de la server au coincis cu cele așteptate.

Răspunsul de la server în cazul unui apel neautorizat și cel în care se cer mesajele unui client sunt prezentate în imaginile de mai jos.

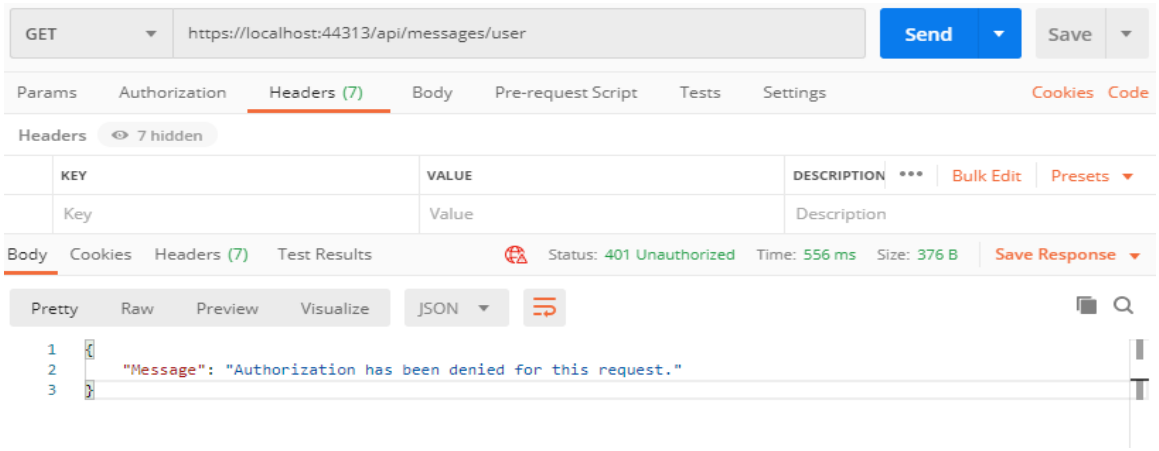

Figura 6.5 Răspuns de la server unei cereri neautorizate

Figura 6.6 Răspunsul de la server pentru cererea mesajelor uni client

System testing

Acest tip de testare are ca scop testarea întregului sistem. Acest lucru implică testarea sistemului de la interfața grafică (autentificare, completare de formulare, sau salvare de informații), până la testarea conexiunii cu baza de date.

Am ales testarea celor mai importante funcționalități ale sistemului. Pentru aceste teste sa va prezenta cazul de utilizare, precondițiile, pașii care trebuie urmați, postcondițiile, rezultatul așteptat în urma fiecarei acțiuni și rezultatul propriu-zis rezultat în urma realizării acțiunii respective.

În cele ce urmează va fi prezentat pașii ce trebuie urmați pentru a trimite un nou mesaj ca și client autentificat.

Precondiții: Clientul să fie autentificat

Tabel 6.1 Cazul de testare a trimiterii unui nou mesaj ca și client autentificat

Acțiune	Rezultat așteptat	Rezultat obținut
2. Clientul va accesa pagina dedicată creării unui nou mesaj	Utilizatorul va fi redirecționat către pagina dedicată creării de noi mesaje.	Utilizatorul este redirecționat către pagina dedicată creării de noi mesaje.
3. Clientul va completa mesajul cu informațiile cerute (Subiect și Mesaj)	Modificările vor fi afișate.	Modificările sunt afișate.
4. Clientul apasă pe butonul de trimitere mesaj	Se va afisa mesaj de informare de trimitere a mesajului și confirmare trimitere mesaj.	Se afisează mesaj de informare de trimitere a mesajului și confirmare trimitere mesaj.

Postcondiții: Mesajul este salvat în baza de date, un nou tichet este creat care să conțină acest mesaj si mesajul este adăugat listei de mesaje a clientului.

Ca urmare a aplicării acestui test, rezultatele obținute au coincis cu cele așteptate. De asemenea, și postcondițiile au fost îndeplinite după cum se poate observa din imaginile de mai jos care reprezintă colecțiile din baza de date.

Figura 6.7 Mesajul salvat în baza de date

Figura 6.8 Tichetul salvat în baza de date

Figura 6.9 Clientul cu noul mesaj salvat în baza de date

Se observa că identificatorii unici ai fiecărui element au fost adaugate la referințele fiecare înregistrări.

S-a mai testat răspunderea la un tichet nou de la un client. Pașii pentru a testa acest lucru sunt prezentați mai jos.

Precondiții: Angajatul să fie autentificat

Tabel 6.2 Cazul de testare pentru răspunderea la un tichet nou

Acțiune	Rezultat așteptat	Rezultat obținut
1. Angajatul va accesa pagina cu tichete noi / neasignate	Angajatul va fi redirecționat la pagina cu tichetele noi / neasignate	Angajatul este redirecționat la pagina cu tichetele noi / neasignate
2. Angajatul expandează tichetul pentru a vedea mesajul de la client	Tichetul va fi expandat și vor fi afișate mesajul / mesajul din tichet împreuna cu zona dedicată scrierii unui nou mesaj tichetului.	Tichetul este expandat și vor fi afișate mesajul / mesajul din tichet împreuna cu zona dedicată scrierii unui nou mesaj tichetului.
3. Angajatul completează mesajul și îl trimite	Se va afișa un mesaj de confirmare a trimiterii mesajului.	Se afișează un mesaj de confirmare a trimiterii mesajului.

Postcondiții: Mesajul este salvat în baza de date, tichetul asignat angajatului și are statusul de *Activ*.

În urma acestor teste, rezultatele obținute au coincis cu cele așteptate. De asemenea și postcondițiile au fost îndeplinite, după cum se poate observa din imaginile urmate fiecărui caz de testare.

Acceptance testing

Acest tip de testare a avut ca scop principal testarea respectării nevoilor utilizatorilor reali ai sistemului, folosindu- se metoda *black box*. Această metodă are ca scop verificarea cerințelor unui sistem (funcționale sau non-funcționale) fără a verifica structura internă a acestuia. S-au testat funcționalitățile sistemului și cu ajutorul aplicației *Postman* s-a testat timpul de răspuns de la server, iar acest timp nu a depășit 3 secunde.

7. Manual de Instalare și Utilizare

În acest capitol vor fi prezentați pașii care trebuie urmați pentru a utiliza aplicația, aplicațiile care trebuie instalate și cerințele software și hardware pentru ca aplicația să funcționeze în parametri optimi.

7.1. Cerințele sistemului

Sistemele software au anumite cerințe din punctul de vedere al resurselor hardware și software pe care trebuie să le îndeplinească orice calculator pentru a asigura o rulare optimă. Prin urmare pentru sistemul prezentat în această lucrare are următoarele cerințe:

- Sistem de operare: Windows 7, Windows 10
- Un sistem cu arhitectura pe 64 biți
- Memorie RAM mai cel puțin 8 GB
- Procesor cu frecvența minimă de 2.40 GHz
- Sistem de operare Windows
- Instalarea următoarelor programe: Visual Studio 2019, MongoDB, NodeJS.
- Un browser de internet cum ar fi Chrome sau Microsoft Edge.

7.2. Instalare

Pentru a utiliza sistemul de management al tichetelor propus de această lucrare este necesară instalarea urmatoarelor programe: Visual Studio 2019(pentru rularea părții de server), MongoDB (pentru persistarea datelor) și NodeJS (partea de client/browser).

Visual Studio 2019

1. Descărcarea și instalarea Visual Studio 2019, trebuie urmați pașii care pot fi găsiți aici[18]. În momentul instalării se vor selecta următoarele componente care vor fi introduse în Visual Studio din categoria *Workload*: ASP.NET and development.
2. După instalarea Visual Studio 2019 se va deschide aplicația se va selecta opțiune de deschidere proiect existent și se va selecta fișierul în care se află fișierul sursă pentru partea de server.

MongoDB

1. Pentru instalarea MongoDB vor fi urmați pașii de aici[19], selectându-se opțiunea *On-premise* de MongoDB.
2. Pentru a putea vizualiza datele din baza de date se poate instala și Compass, aplicație destinată afișării datelor din MongoDB. Aici[20] se găsește link-ul pentru descărcarea fișierului de instalare și a

[18] https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019
[19] https://www.mongodb.com/try/download/community
[20] https://www.mongodb.com/try/download/compass

instrucțiunilor de instalare a aplicației Compass. Pentru a vizualiza datele se va selecta baza de date cu numele “TicketingHelperDB”.

NodeJS

1. Pentru instalarea NodeJS se pot urma pașii de la linkul următor[21].
2. Pentru pornirea părții de front-end este necesară accesarea fișierului care conține fișierele destinate front-end -ului și rularea din linia de comanda a comenzii “npm instal”, iar după executarea comenzii mai trebuie rulată urmatoarea comandă “npm start”, aceasta având rolul de a rula componenta de front-end.

7.3. Utilizare

Acest subcapitol are rolul de a oferi utilizatorului sfaturi și de a-l îndruma în utilizarea cât mai eficientă a sistemului. În cele ce urmează vor fi prezentate cele mai importante acțiuni pe care un utilizator poate să le îndeplinească. Aplicația poate fi accesată din browser la URL- ul “localhost:3000” .

După accesarea acestui URL, utilizatorul va fi redirecționat la pagina de autentificare. Pe această pagină utilizatorul are opțiunea de a se autentifica, de a crea un mesaj nou, vizualiza secțiune de întrebări frecvente și răspunsuri și posibilitatea de a fi redirecționat către o pagină noua în care își poate crea un cont nou.

Trimitere mesaj fără autentificare

În cazul în care utilizatorul dorește să trimită un mesaj fără a se autentifica acesta trebuie sa completeze la crearea mesajului subiectul acestuia, mesajul pe care vrea să îl transmită și adresa de email la care să fie contactat în cazul în care compania va trebui să îl contacteze.

Autentificare

Dacă un client are un cont, după autentificare acesta va fi redirecționat pe pagina contului acestuia. În aceasta va avea posibilitatea de a vedea toate tichetele lui, mesajele trimise și va avea posibilitatea de a crea mesaje noi, dar nu va fi nevoit să completeze și email- ul ca și în cazul trimiterii unui mesaj din pagina de autentificare deoarece noul mesaj îi va fi direct asignat clientului autentificat.

[21] https://blog.teamtreehouse.com/install-node-js-npm-windows

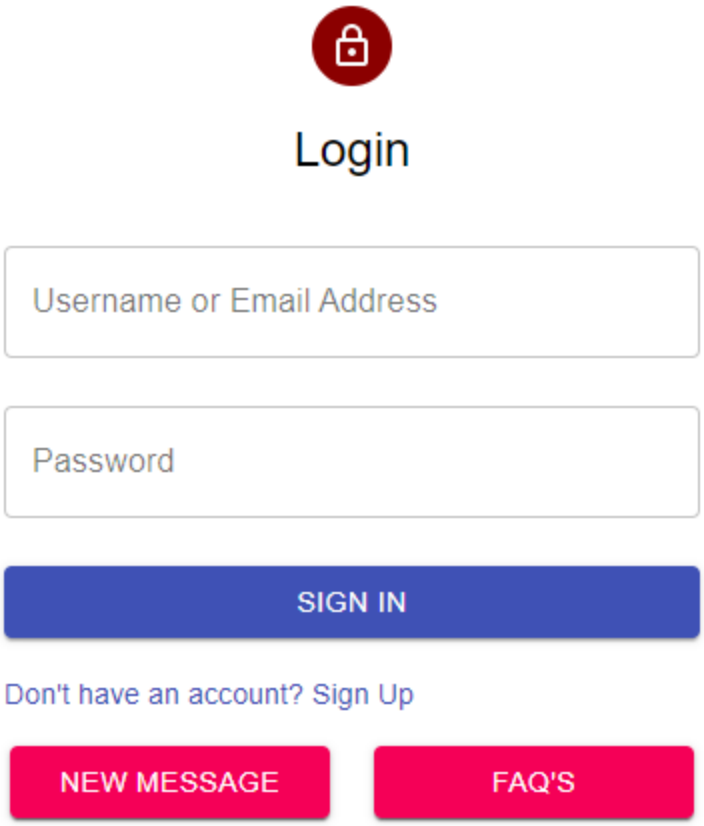

Figura 7.1 Pagina de autentificare

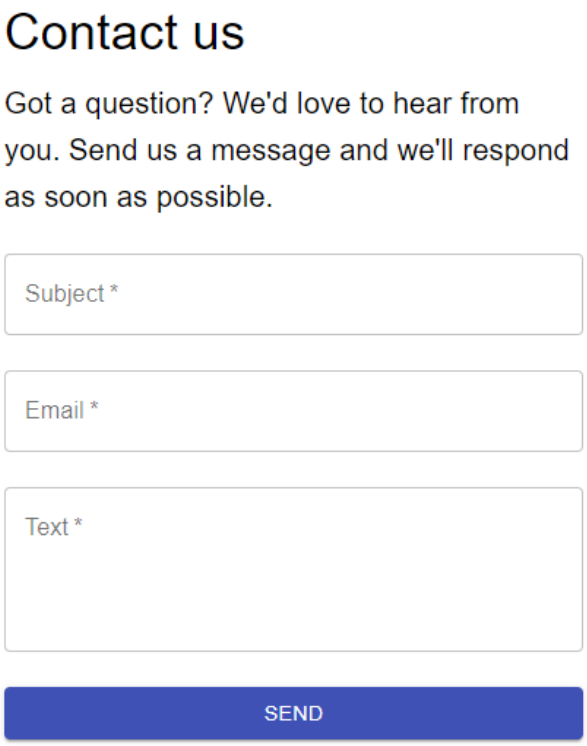

Figura 7.2 Creare mesaj nou ca utilizator neautentificat

Schimbare parolă

Orice utilizator poate să își schimbe parola din secțiunea *Profile*, selectând opțiunea de schimbare a parolei (butonul *Chage password*). Pentru schimbarea parolei va fi necesară confirmarea parolei curente și introducerea unei noi parola și confirmarea acestia.

În figura de mai jos se poate observa formularul care trebuie completat pentru a schimba parola.

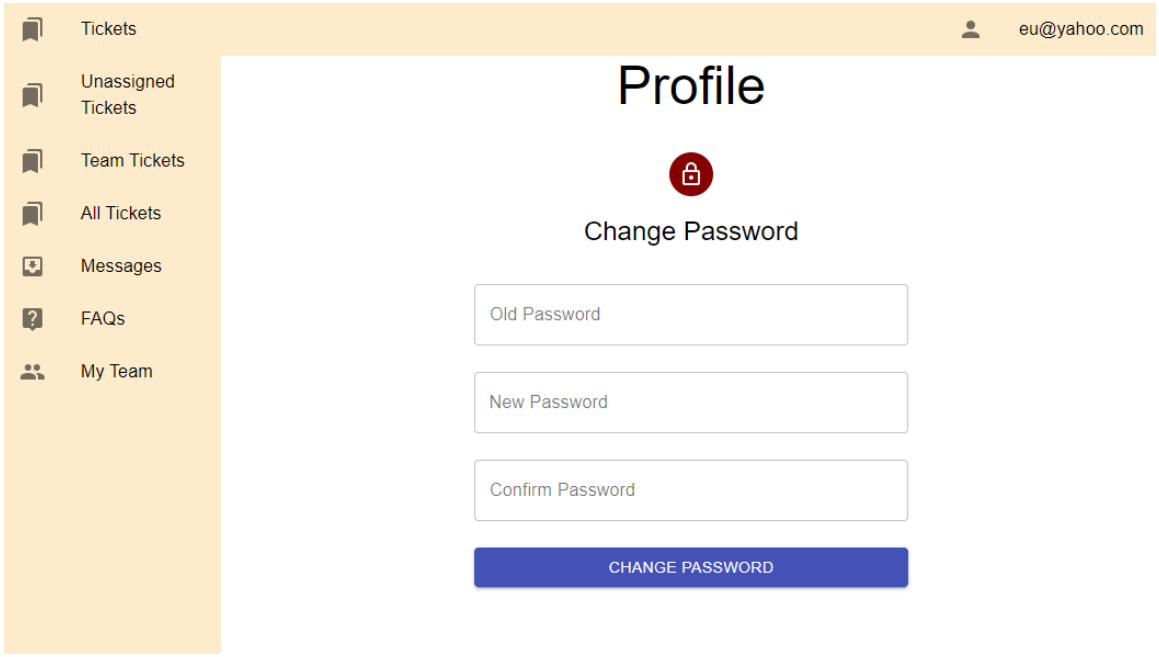

Figura 7.3 Schimbarea parolei unui utilizator

Vizualizare tichete

De asemenea, clientul poate vizualiza mesajele asociate unui tichet și să adauge noi mesaje unui tichet. Această funcționalitate ajută la comunicarea între angajat și client deoarece și angajații pot să scrie, adauge mesaje noi unui tichet existent cu scopul rezolvării sesizărilor clienților.

Imaginea de mai jos reprezintă lista de tichete ale unui client împreună cu mesajele asignate acestui mesaj și un formular care permite crearea de mesaj nou care să fie asociat acetui tichet.

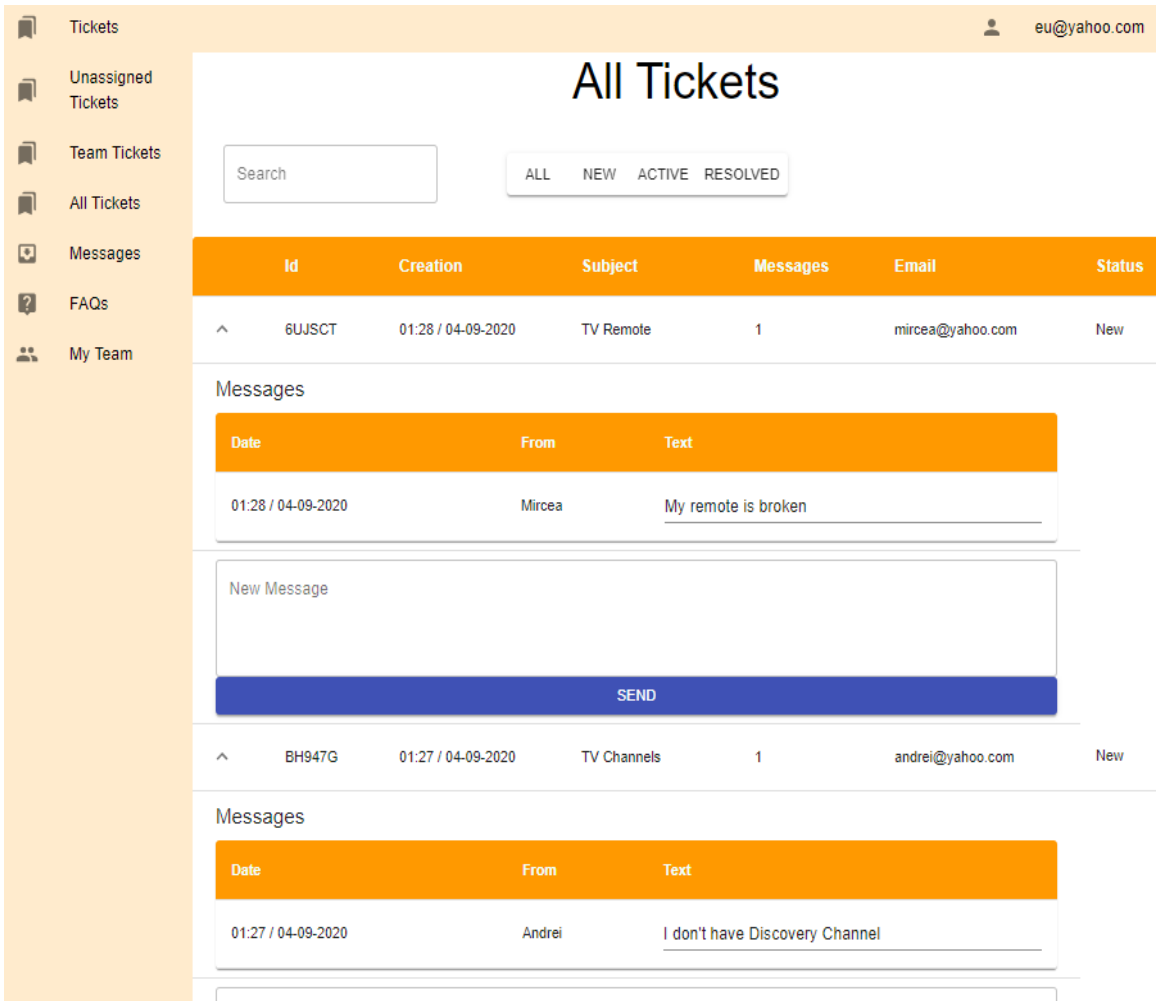

Figura 7.4 Afișare mesaje tichet

Vizualizare tichete neasignate

În cazul angajaților, pentru a-și asigna un nou tichet, trebuie să acceseze secțiunea de tichete noi (*Unassigned Tickets*) unde poate vizualiza toate tichetele noi la care nu lucrează nici un angajat.

Scrierea de mesaje în contextul unui tichet existent

Dacă dorește să răspundă unui mesaj din partea unui client, poate să scrie un mesaj nou prin expandarea tichetului. Prin expandarea tichetului se pot observa mesajele tichetului și secțiunea special dedicată trimiterii unui nou mesaj asociat tichetului.

În figura următoare se poate observa cum se poate răspunde unui tichet nou.

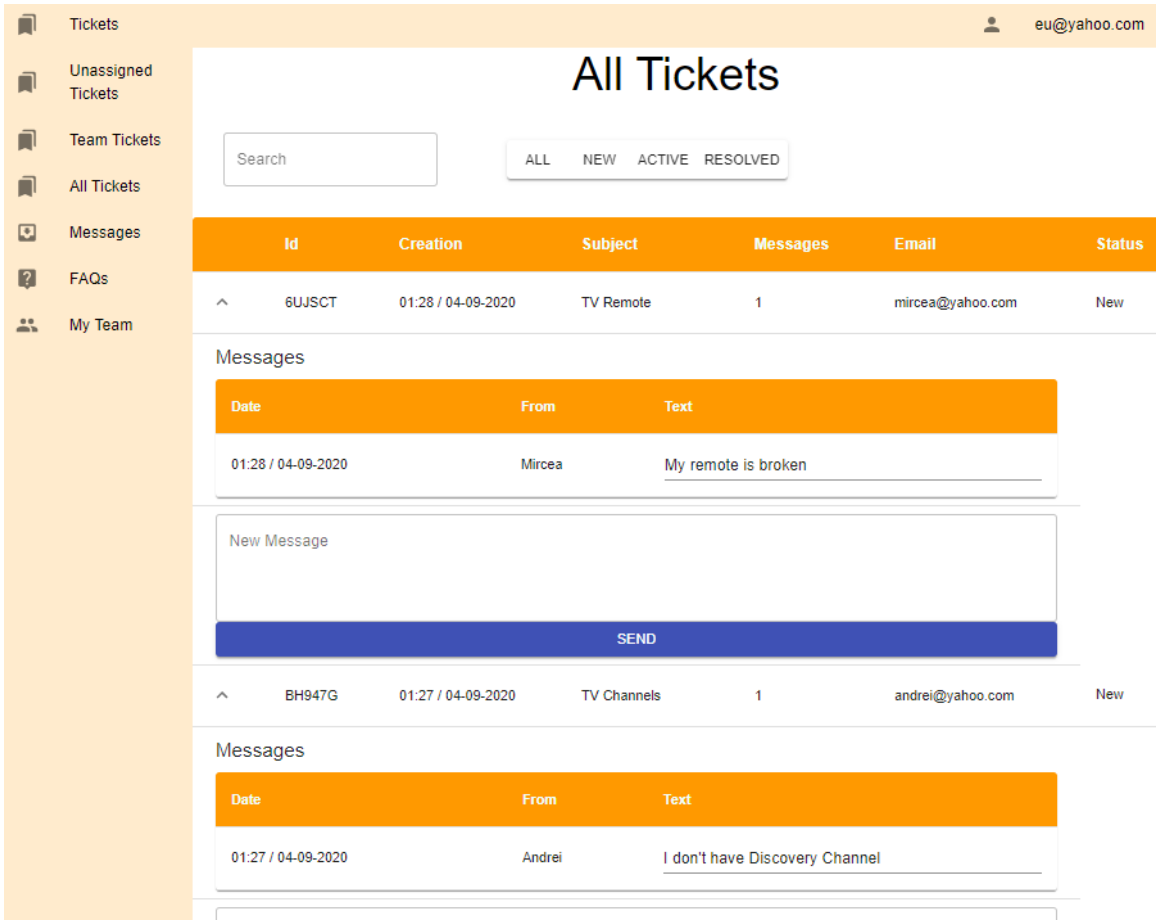

Figura 7.5 Scrierea de mesaj la un tichet nou (neasignat)

Dacă un angajat va scrie un mesaj unui asociat unui tichet nou, acest tichet va fi automat asignat angajatului respectiv.

Informațiile despre un tichet pot fi actualizate de către angajatul asignat tichetului din pagina de tichete asignate lui (*Tickets*) apăsând pe butonul de editare urmat de confirmarea modificărilor prin salvarea acestora.

Administratorii sistemului, pe lângă posibilitatea de a răspunde tichelelor de la clienți, pot vizualiza toți angajații si toate echipele împreună cu membrii acestora. De asemenea, pot adăuga noi echipe, șterge echipe, sau să modifice numele sau membrii acestora și să adauge noi angajați, să asocieze echipe acestora și să șteargă angajatul.

Managementul echipelor și angajaților

Pentru adăugarea de echipă nouă sau angajat nou trebuie accesată pagina în care se pot vizualiza toate echipele, respectiv toți angajații. În aceste pagini există în colțul din dreapta jos un buton de adaugare de echipă, respectiv client.

În cazul în care se dorește adăugarea unei noi echipe se va introduce doar numele echipei noi, iar în cazul unui nou angajat se mai poate selecta, pe lângă numele acestuia, se

vor introduce și adresa de email, o parolă temporară și poziția pe care o va avea (Angajat sau Administrator).

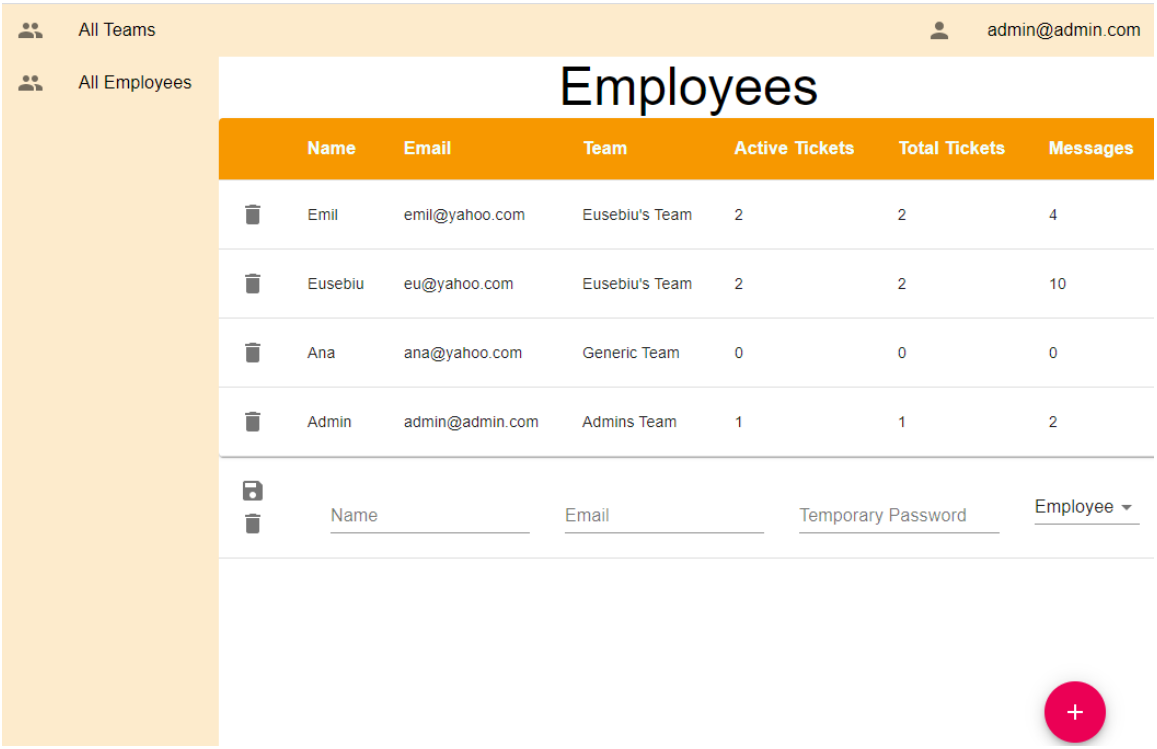

Figura 7.6 Vizualizare/ștergere/adăugare angazați

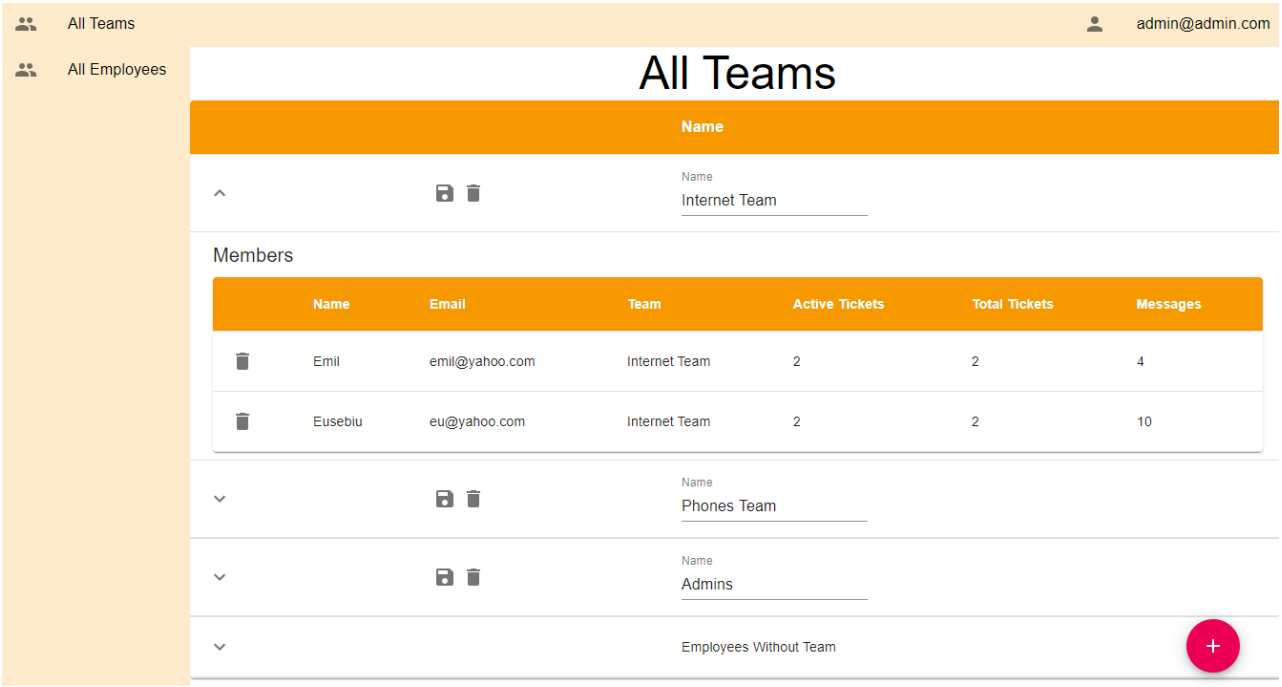

Figura 7.7 Vizualizare/ștergere/adăugare echipe

8. Concluzii

În acest capitol vor fi prezentate concluziile la care am ajuns în urma implementării acestui sistem de management al tichetelor, viziunea de ansamblu a sistemului și ce îmbunătățiri pot fi aduse sistemului.

8.1. Realizarea obiectivelor propuse

Am dorit crearea unui sistem de management al tichetelor care să asigure o metodă simplă și ușor de utilizat pentru un client de a transmite anumite sesizări unei companii. De asemenea, am urmărit să introduc în aplicație posibilitatea urmăririi activității angajaților companiei. Prin urmare am reușit implemntarea unui sistem de tip client-server care sa ofere următoarele funcționalități:

- Crearea de cont nou pentru client
- Trimiterea de mesaje companiei
- Urmărirea de către client a statusului mesajelor trimise de acesta
- Crearea de secțiune FAQs pentru a veni în sprijinul clienților
- Crearea automată a tichetelor la primirea unui mesaj nou de la client
- Managementul tichetelor
- Urmărirea activității angajaților în cadrul firmei

8.2. Dezvoltări ulterioare

Sistemul implementat poate fi îmbunătățit, conține funcționalitățile de bază ale unui sistem de management al tichetelor, pe lânga care mai pot fi adăugate funcționalități de urmărire a activității angajațiilor. Printre funcționalitățile noi care pot fi aduse sistemului se numără:

- Adăugarea de chat comun pentru angajați pentru a facilita comunicarea în interiorul companiei
- Citirea de mesaje de pe alte surse cum ar fi poșta electronică sau rețelele de socializare
- Posibilitatea de salvarea a unor mesaje predefinite de către angajați cu scopul de a răspunde mai repede unor mesaje des întâlnite care au aceeași rezolvare
- Trimiterea de email- uri automată clientului la anumite momente importante din ciclul de viață al unui tichet
- Automatizarea asignării tichetelor pe baza unor criterii stabilite de utilizator

În concluzie, sistemul implementat oferă pe lângă funcționalitățile de bază ale unui sistem de management al tichetelor posibilitatea de urmărire a activității angajaților. Astfel, acesta poate fi utilizat și pentru a îmbunătății relația cu clienții în cadrul unei companii, dar și pentru a urmării activitatea angajaților.

9. Bibliografie

[1] T. Nosek, "Kentico Help Desk for Internal Purpose", Master Thesis, Masaryk University, 2016.
[Online]: https://is.muni.cz/th/i740y/TomasNosek-MastersThesis.pdf

[2] G. Naveen, "Electronic Support Ticket Management System", International Journal of Engineering Research & Technology (IJERT), vol. 2, nr. 9, 2013.
[Online]: https://www.ijert.org/research/electronic-support-ticket-management-system-IJERTV2IS90559.pdf

[3] M. Seidl, M. Scholz, C. Huemer, G. Kappel, "UML@Classroom", Springer, 2015.

[4] L. Richardson, M. Amundsen, S. Ruby, "RESTful Web APIs, O'Reilly Media", 2013.

[5] "MongoDB Documentation," 29 Junie 2015
[Online]: http://web.cs.wpi.edu/~cs585/s17/Books/Books-PDF/MongoDB-manual.pdf.

[6] O. Pop, "Sisteme Informatice", Notițe curs, UTCN, Facultatea de Automatică și Calculatoare.

[7] K. WIeger, J. Beath, "Software Requirements", Microsoft Press, 2013.

[8] B. Jephson, R. Larsen, L. Coulson, "The HTML and CSS Workshop: A New, Interactive Approach to Learning HTML and CSS", Packt, 2019.

[9] A. Boehm, J. Murach, "Murach's C# 2015", Murach, 2016.

[10] M. Oprea, "Autentificarea JWT pentru Microsoft .NET", 2018.
[Online]: https://www.red-gate.com/simple-talk/dotnet/net-development/jwt-authentication-microservices-net/.

Anexa 1: Glosar de termeni

Termen	Descriere
Agent	Membru al departamentului de suport responsabil cu rezolvarea unui tichet.
API	Application Programming Interface – Aplicație care are rolul de a realiza o comunicare între două sisteme.
CRUD	Create Read Update Delete (Creare, Citire, Actualizare, Ștergere).
CSS	Cascading Style Sheets.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol - Protocol folosit pentru stabilirea regulilor de transmitere a mesajelor în cadrul World Wide Web
HTTPS	Hypertext Transfer Protocol Secure – Reprezintă varianta securizată prin SSL a protocolului HTTP.
JSON	Javascript Object Notation - Format folosit pentru schimbul de informaţii între diferite sisteme
REST	Representational State Transfer – Modalitate architecturală folosită pentru schimbul de informaţii între diferite sisteme prin intermediul protocolului HTTP.
Tichet	Elementul de legatură între client și departamentul de suport al unei firme. Mapează sesizările sau cererile clienților, acestea putând fi trimise prin email sau de pe site- ul companiei.
UML	Unified Modeling Language
URL	Uniform Resource Locator

Anexa 2: Lista imaginilor

Anexa 3: Lista tabelelor