MHealth

Modulul Web – Componenta de gestionare a urgențelor

LUCRARE DE LICENŢĂ

Absolvent: **Georgiana Bianca CORNEA**

Coordonator ştiinţific: **Senior Lector Ing. Cosmina IVAN**

2020

Cuprins

Capitolul 1. Introducere – Contextul proiectului

Acest capitol prezintă motivația care stă în spatele dezvoltării unui sistem precum *MHealth* și o scrută prezentare a capitolelor care alcătuiesc lucrarea.

1.1. Motivația

Odată cu progresul tehnologic din perioada modernă, interesul oamenilor asupra îngrijirii propriei persoane a crescut considerabil. Se promovează tot mai mult abordarea unui stil de viață sănătos, conștientizarea factorilor dăunători din viața unei persoane și informarea din cât mai multe surse.

Totuși, odată cu creșterea timpului petrecut în mediul online și sursele infinite de informare, de multe ori este greu de selectat ceea ce este cu adevărat important sau chiar corect. Exista o tendință majoră de a căuta pe internet informații legate de orice, fie pentru întărirea unor convingeri proprii, fie pentru găsirea unei soluții. Situațiile cu caracter medical nu fac excepție de la cele menționate anterior. Deși este vorba despre probleme ce pot avea caracter unic, adesea oamenii preferă sa se autodiagnosticheze în detrimentul unei vizite la cabinetul medical. Astfel, zilnic, Google ajunge să trateze de la banale răceli până la afecțiuni complexe, fie ele existente cu adevărat sau nu.

Afinitatea oamenilor către utilizarea dispozitivelor mobile precum și setea lor pentru informație, trebuie exploatate cât mai benefic, prin îndrumarea acestora către aplicații de calitate, care le pot oferi informații valide și care le pot fi de real ajutor atât în situații limită, cât și în distingerea realităților de neadevăruri.

O urgență medicală reprezintă, în general, o situație limită care necesită intervenția unor persoane de specialitate și / sau punerea unui diagnostic într-un timp cât mai scurt. În secolul vitezei, numărul acestor solicitări la apelul unic de urgență 112 s-a majorat, iar răbdarea solicitanților a scăzut, ceea ce face munca echipajelor de specialitate tot mai dificilă. Totodată, timpul de răspuns joacă un rol esențial în finalizarea cu bine a unei urgențe. Din cauza numărului mare de solicitări, al distanței până la punctul de interes, al condițiilor meteo sau al altor factori externi, acest timp poate crește, astfel că utilizarea perioadei dintre apel și intervenția serviciilor de specialitate, prin aplicarea unor măsuri potrivite, poate face diferența.

Însă nu doar situațiile limită necesită atenție din partea medicilor, ci și accidentele domestice, consultațiile de rutină sau evenimente neprevăzute fără caracter prioritar. În vremuri precum contextul actual, cel al pandemiei de COVID-19, prezentarea la spital din motive neîntemeiate poate împiedica activități prioritare și totodată poate cauza mai multe efecte nedorite decât beneficii. Cum oamenii sunt sfătuiți să nu se prezinte la spital decât in cazuri excepționale, aceștia au nevoie de alternative care sa le ofere soluții cu aceeași încredere ca și o discuție față în față cu un cadru medical.

Așadar, introducerea unui sistem care să permită gestionarea situațiilor de urgență într-un mod rapid și automat poate fi doar benefică, iar înglobarea în acest sistem a unei componente care să faciliteze interacțiunea cu doctorii, pune bazele unui sistem medical electronic complex, prin care utilizatorul să fie complet conectat la ajutor de specialitate indiferent de motiv sau moment.

1.2. Conținutul lucrării

În continuare, se vor prezenta capitolele care alcătuiesc această lucrare împreună cu o scurtă descriere a conținutului acestora:

- **Capitolul 1**: reprezintă capitolul introductiv al lucrării, care pune pilonii ideii ce urmează a fi dezvoltată.

- **Capitolul 2**: prezintă o listă a obiectivelor propuse, secționate în două categorii principale: obiective generale și obiective specifice.

- **Capitolul 3**: sumarizează conceptele asimilate pe durata perioadei de studiu bibliografic. Se realizează o prezentare generală a domeniului din care face parte sistemul, o comparație cu sisteme asemănătoare existente și descrierea în detaliu a unor concepte de actualitate care vor fi folosite mai departe în dezvoltarea sa.

- **Capitolul 4**: cuprinde arhitectura conceptuală și prezentarea tuturor cerințelor sistemului împreună cu cazurile de utilizare. De asemenea, se descrie setul de tehnologii ales pentru implementare.

- **Capitolul 5**: prezinta diagramele reprezentative ale sistemului și detaliile de implementare.

- **Capitolul 6**: sintetizează întreg procesul de testare și validare al sistemului, prezentând testele și modurile de testare în ordine: testarea unităților, testarea integrării modulelor, testarea de sistem, testarea de acceptanță.

- **Capitolul 7**: descrie modul în care sistemul trebuie folosit în mod corect de către un utilizator sub forma unui manual de instalare și utilizare.

- **Capitolul 8**: prezintă concluziile din urma dezvoltării sistemului și setul de contribuții personale adus. Totodată se propun niște metode de dezvoltare ulterioară a sistemului.

Capitolul 2. Obiectivele Proiectului

În acest capitol se descriu obiectivele proiectului. Mai întâi se descrie obiectivul general al sistemului *MHealth*, iar apoi obiectivele specifice ale modulului implementat.

2.1. Obiectivul general

Obiectivul principal al sistemului *MHealth* este de a pune la un loc o componentă de administrare a urgențelor medicale publice și private cu una de gestiune virtuală, a discuțiilor față în față cu un medic prin interacțiunea dintre o aplicație mobilă și una web. Scopul sistemului este de a permite maximizarea ajutorului oferit pe durata timpului de răspuns al autorităților și de a înlesni procesul de consultații medicale.

Acest obiectiv a fost atins prin împărțirea sistemului pentru managementul urgentelor medicale în trei părți majore prezentate și în tabelul 2.1:

- **MHealth -** *Modul Android pentru gestionarea urgențelor*: dezvoltat de către Dascălu Cosmin-Ștefan.
- **MHealth -** *Modul Web pentru gestionarea urgențelor:* dezvoltat și prezentat în lucrarea curentă.
- **MHealth –** *Modul Web și Android: Componenta Personal:* dezvoltat de către Ifrim Andreea.

Tabel 2.1 – Componentele sistemului MHealth

	Modulul mobil	**Modulul Web**
Utilizatori	Obișnuiți și cu pregătire medicală	Doctori
Componenta gestionării urgențelor	Utilizatorul obișnuit poate raporta o urgență adăugând dintr-o interfață prietenoasă o descriere text, o poză, un video, o înregistrare audio și un nivel de severitate al situației estimat de către el. Locația este transmisă în mod automat. Se trimite o notificare cu adresa urgenței din apropiere și un email cu informațiile necesare.	Detaliile despre acest modul se regăsesc în următoarele secțiuni
Componenta personal	Se ocupă de punerea în legătură a pacienților cu medicii de familie sau cu alți doctori de specialitate pentru sfaturi medicale, schimb de fișiere sau realizarea de programări. De asemenea, pune la dispoziție un chat pentru a facilita comunicarea.	

2.2. Obiective specifice

În următoarea secțiune se vor prezenta obiectivele specifice ale modulului Web de gestionare al urgențelor:

- Posibilitatea preluării urgențelor transmise prin intermediul aplicației mobile din baza de date comună și afișarea lor pe o interfață intuitivă.
- Posibilitatea distingerii între urgențe diferite și aceeași urgență raportată de mai mulți utilizatori.
- Posibilitatea prelucrării automate a datelor transmise de la fața locului (imagini, video, audio, text) folosind inteligență artificială și afișarea detaliilor relevante din rezultate.
- Posibilitatea validării rezultatelor obținute în mod automat și introducerea de detalii suplimentare.
- Posibilitatea transmiterii unui mail cu informații legate de caz tuturor utilizatorilor aplicației mobile dornici să participe la caz.
- Posibilitatea vizualizării unor date statistice legate de cazuri.
- Posibilitatea vizualizării cazurilor pe o hartă.
- Posibilitatea păstrării informațiilor despre urgențe într-un mod sigur și accesarea lor sub forma unui istoric.

Aceste obiective au fost remodelate sub forma cazurilor de utilizare și a cerințelor funcționale și au fost implementate în totalitate. Totodată, modul de implementare și deciziile de proiectare sunt descrise amănunțit în capitolele ce urmează.

Capitolul 3. Studiu Bibliografic

În acest capitol se vor prezenta niște detalii reprezentative din perioada de studiu bibliografic. Pentru început, se descrie legătura dintre domeniul de care aparține tema aleasă și tehnologie. Ulterior, se descriu conceptele de *Cloud Computing* si *Blockchain*, studiul finalizându-se cu realizarea unei analize comparative a sistemului cu alte sisteme existente deja pe piață si descrierea sumară a acestora.

3.1. Tehnologia în sistemul medical

De-a lungul timpului, relația dintre medicină și tehnologie a avansat în mod considerabil, precum și interesul oamenilor pentru monitorizarea sănătății. Astfel, tot mai multe companii[1] adaugă dispozitivelor de pe piață funcții menite să ajute la urmărirea și menținerea unui stil de viață cât mai sănătos. Fie că este vorba de senzori încorporați în ceasuri inteligente care urmăresc pulsul, numără pașii sau măsoară nivelul de stres, fie aplicații Web sau mobile care ne ajută cu schimbul de documente, programarea unei consultații sau stabilirea unui tratament, cu toții folosim într-o oarecare măsură aceste soluții medicale din mediul online.

Datorită interesului ridicat pentru aplicațiile mobile, tot mai multe organizații medicale aleg să folosească mediul virtual pentru comunicare cu pacienții[2].

O alta dovadă a translatării consultațiilor medicale în mediul online sunt și cele 70000 de întrebări adresate în fiecare minut lui „Doctor Google”[3], conform statisticilor oferite de motorul de căutare. Datorită gamei largi de întrebări adresate, aferente multor afecțiuni și de gravitate variabilă, este necesară oferirea unor răspunsuri de specialitate și personalizate. Această dorință, împreună cu lipsa timpului, impulsionează oamenii spre alternative online și rapide ale metodelor convenționale precum mersul la cabinet.

Cu toate că acum lumea are alternative online pentru sfaturi medicale, numărul apelurilor la numerele de urgență nu a scăzut, iar media timpului în care oamenii ajung în legătura cu dispeceratul variază între 1 si 15 minute[4]. Deși timpul este crucial când vine vorba de tratarea unei urgențe, statisticile arată că timpul mediu de răspuns al unui echipaj de intervenție este între 7 și 180 de minute, în funcție de gravitatea estimată a cazului. În medie, pentru un caz de urgență de categorie 1 (cazuri care necesită intervenție imediată sau resuscitare precum: infarct, respirație anormală etc.) timpul de răspuns este de 14 minute[5]. Pentru a micșora acești timpi, serviciile își propun abordarea diferitor soluții care vizează atât victimele (prin campanii de informare care să îi învețe când și de ce ar trebui apelat serviciul de urgențe) cât și o mai bună gestiune a resurselor și a timpului petrecut la telefon. De exemplu, în [1] se prezintă o aplicație care minimizează durata apelului folosind o aplicație mobilă, care este folosită pentru

[1] https://www.medicaldevice-network.com/comment/smartwatches-healthcare-leading-companies/
[2] https://www.biopharmatrend.com/post/113-top-healthcare-mobile-apps-you-should-know-in-2020/
[3] https://www.telegraph.co.uk/technology/2019/03/10/google-sifting-one-billion-health-questions-day/

[4] https://www.statista.com/statistics/794483/average-answer-time-of-calls-made-to-emergency-services/
[5] https://www.nuffieldtrust.org.uk/resource/ambulance-response-times

preluarea datelor personale (datele fiind introduse la înregistrarea în aplicație și preluate când se face apelul propriu-zis).

Tehnologiile moderne precum blockchain-ul și inteligența artificială contribuie semnificativ la îmbunătățirea securității și acurateței răspunsurilor oferite. Acum, utilizarea soluțiilor tehnologice în sistemul medical a evoluat de la simple sisteme de gestiune a fișierelor și a programărilor la sisteme complexe și eficiente care oferă servicii variate și îmbunătățesc timpul și calitatea răspunsurilor oferite într-un mod sigur.

3.2. Tehnologia Cloud – acum si în viitor

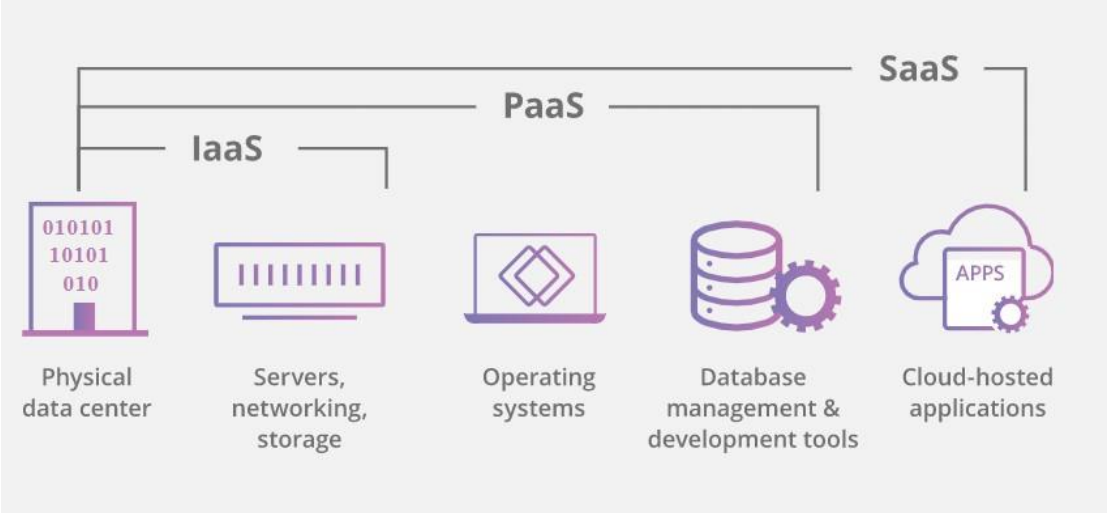
Cu siguranță, în zilele noastre, ideea de *Cloud* nu mai este străină pentru majoritatea oamenilor. Ceea ce este însă probabil necunoscut este diversitatea și resursele aparent nemărginite pe care acesta le pune la dispoziție. Majoritatea utilizatorilor interacționează cu acesta doar pentru o extensie a spațiului de stocare (soluții oferite de Google, Apple sau Amazon) și astfel văd cloud-ul ca pe o bază de date imensă, dar sigură[6].

În realitate, noțiunea de *Cloud* se referă la niște servere care pot fi accesate prin intermediul Internetului și la bazele de date și programele software care rulează pe acele servere[7]. Acestea se găsesc în centre de date, răspândite pe tot globul. Principalul lor avantaj îl constituie că utilizatorii nu trebuie să se ocupe de părțile fizice ale serverelor, iar aplicațiile nu trebuie să ruleze pe aparatura proprie. Astfel, un utilizator poate rula orice oricând, fără să se îngrijoreze de competența serverului propriu sau performanța aparaturii sale. Aceste lucruri sunt posibile prin intermediul conceptului de virtualizare. Aceasta permite crearea unor mașini virtuale care pot folosi resursele hardware într-un mod mult mai eficient.

Există trei tipuri principale de servicii integrate de *Cloud,* definite după ceea ce oferă:

- **Software-as-a-Service**: după cum arată și figura 3.1 este cel mai „cuprinzător” tip; utilizatorii nu trebuie nici măcar să instaleze aplicația pe propriul dispozitiv deoarece acestea sunt găzduite pe platforma de *Cloud*.
- **Platform-as-a-Service**: aplicațiile nu sunt găzduite pe *Cloud,* dar utilizatorul are acces la resursele de care are nevoie pentru a-și dezvolta aplicația, inclusiv sisteme de operare și infrastructură.
- **Infrastrucuture-as-a-Service**: utilizatorul folosește doar serverele și / sau spațiul de stocare oferit de *Cloud*.

[6] https://authorscanvas.blog/2019/03/08/what-do-people-really-think-the-cloud-is/
[7] https://www.cloudflare.com/learning/cloud/what-is-the-cloud/



Figură 3.1 - Tipuri de Cloud

De asemenea, tipurile de *Cloud* se pot deosebi și prin cum se face deployment-ul:

- **Cloud privat**: serverul este complet dedicat unui utilizator.
- **Cloud public**: serviciul este deținut de o altă persoană, iar la același server au acces mai mulți utilizatori (fiecare pe o anumită componentă).
- **Cloud hibrid**: o combinație între tipul privat și cel *public*, utilizatorul folosind un Cloud *privat* pentru anumite servicii și altul / altele publice pentru alte servicii.

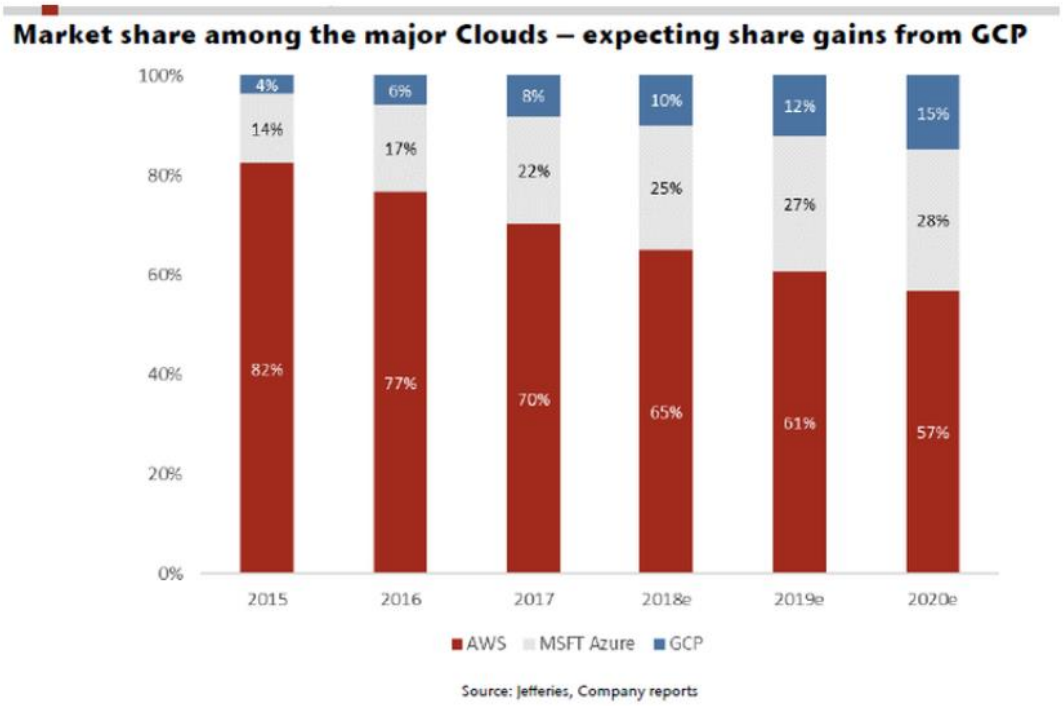
În prezent, tot mai multe companii aleg să se îndrepte spre *Cloud*, fie prin migrarea serviciilor existente, fie prin dezvoltarea noilor aplicații folosind doar aceste servicii pentru obținerea unor beneficii precum:

- Ușurința utilizabilității
- Scalabilitate
- Securitate
- Disponibilitate
- Mentenanța ușoară
- Costuri accesibile și de tip „Pay as you go”

Totodată, viitorul pare unul strălucit pentru *Cloud*, preconizându-se că până în anul 2024, peste 90% dintre companii vor folosi sisteme de acest tip[8]. Odată cu investițiile suplimentare și atenția masivă primită în ultima perioadă, această tehnologie va continua să surprindă prin ceea ce face, atrăgând tot mai mulți utilizatori.

În zilele noastre există mai multe companii care dețin astfel de sisteme *Cloud*. Printre cele mai importante se numără: Amazon, Azure, Google, IBM, Oracle sau Huawei[9].

[8] https://learn.g2.com/future-of-cloud-computing
[9] https://medium.com/aelfblockchain/top-cloud-computing-platform-comparison-d12708bcc12c



Figură 3.2 - Împărțirea pieței în domeniul Cloud

Conform articolelor și studiilor făcute, în prezent cea mai utilizată platformă Cloud este cea oferită de Amazon: Amazon Web Services, precum arată și figura 3.2. Deși această platformă este cea mai matură, alte companii pot fi deja luate în considerare drept competiție serioasă. Astfel, cele mai importante trei variante de luat în considerare sunt: Amazon Web Services, Google Cloud și Azure[10].

În tabelul 3.1 se regăsește o analiză comparativă a celor trei, sumarizând[11] elementele esențiale dezvoltării sistemului *MHealth*.

Tabel 3.1 – Analiză comparativă a sistemelor Cloud

Funcționalitate	**Amazon Web Services**	**Azure**	**Google Cloud**
Siglă	aws	Azure	Google Cloud
Companie	Amazon	Microsoft	Google
Server virtual	Amazon EC2	Azure Virtual Machine	Compute Engine
Autoscalare	Da	Da	Da

[10] https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/
[11] http://comparecloud.in

Deployment	Da, prin AWS Elastic Beanstalk	Da, prin Azure Web Apps sau Azure Cloud Services	Da, prin Google App Engine
Stocare	Amazon Simple Storage Service	Azure Blob Storage	Cloud Storage
Load Balancer	Da	Da	Da
Chatbot	Lex Chatbots	Azure Bot Service	Dialogflow
Cloud Software Development Kit	AWS SDK	Azure SDK Visual Studio	Cloud SDK Cloud Tools for IntelliJ Cloud Tools for Android Studio Cloud Tools for Powershell Google Plugin for Eclipse
Management Tools	Amazon CloudWatch AWS CloudTrial	Log Analytics Azure portal	Google StackDriver Monitoring Logging Error Reporting Trace Debugger
Securitate	Un larg ecosistem de parteneri si soluții de securitate	Este multi-layerd și folosește inteligența artificială.	Oferă cea mai buna securitate și în cazul în care gestionează ei setul de chei sau se alege gestiunea personală (cheile pot fi schimbate cu ușurință)
Cost	Pay-as-you go, cel mai accesibil	Costuri după timpul de utilizare si per gigabit de stocare	Costuri după timpul de utilizare, dar oferă la înscriere un buget de 300$ care poate acoperi cheltuielile inițiale.
Dezavantaje principale	Prețul inițial nu acoperă toate serviciile necesare. Există o limită maximă de stocare, care poate fi mărită doar contra cost.	Nu oferă gestionarea datelor companiilor sau monitorizarea serverelor.	Fiind cel mai nou, momentan are mai puține caracteristici decât principalii competitori.
Popularitate	În scădere	În creștere	În creștere
Notificari	Amazon SNS	Azure Notification Services	Firebase Cloud Messaging
Procesare de text	Amazon Lex	LUIS Azure Bot Service	Natural Language API Cloud Text-to-Speech DialogFlow Enterprise Edition
Recunoastere vocala	Amazon Polly Amazon Transcribe Amazon Translate	Bing Speech API	Translation API Speech API
Procesare de imagini	Amazon Rekognition	Emotion API Face API	Vision API

3.3. Blockchain – cheia digitalizării sistemului medical

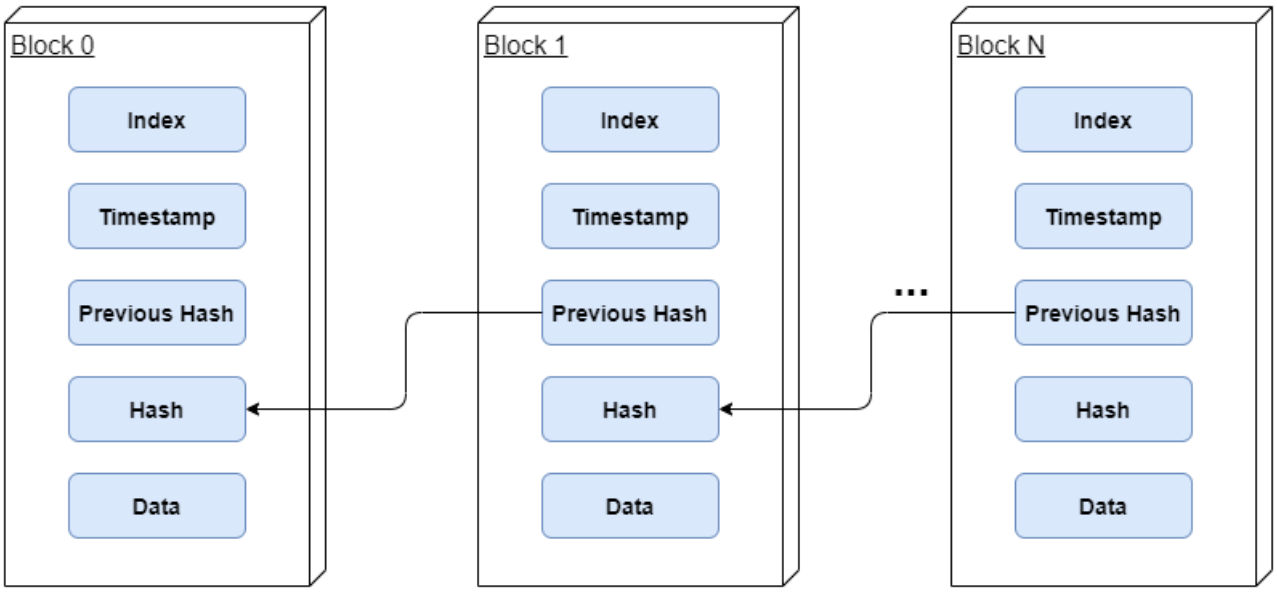
Un *blockchain* nu este altceva decât un grup de intrări de date, *decentralizat* care stochează datele *independent* de o entitate. Ceea ce îl face însă special este marcarea blocurilor de date cu un *timestamp* și modul în care își păstrează caracterul *transparent* deși se aplică principii *criptografice* asupra informațiilor.

Dezvoltat inițial pentru criptomonede, acesta a fost primit cu reticență de către publicul larg în 2009, când a fost introdus de Satoshi Nakamto prin intermediul Bitcoin. Ulterior, lumea a început să asocieze într-un mod eronat această tehnologie cu criptomonedele, considerându-le același lucru[12]. De atunci și până acum, lucrurile s-au mai schimbat, oamenii găsind metode tot mai variante de a integra blockchain și domenii tot mai vaste în care ar putea aduce diferențe remarcabile.

3.3.1. Structura blocurilor

Elementul structural al unui blockchain este blocul. Aceasta structură de date conține datele care trebuie stocate si alte elemente menite să sporească siguranța și confidențialitatea datelor[13].

Figura 3.3 prezintă o înșiruire de astfel de blocuri și modul în care acestea sunt interconectate.



Figură 3.3 - Structura generală de blocuri

[12] https://medium.com/the-mission/the-popularity-of-blockchain-technology-in-the-world-today-33ca8f36c0f6

[13] https://www.spheregen.com/blockchain-technology-basics/

Elementele unui bloc:

- **Index**: similar unui identificator.
- **Timestamp**: *data* și *ora* la care blocul a fost creat
- **Hash**: rezultatul *criptării* datelor; se folosește o funcție de *hashing* asupra datelor care indiferent de lungimea datelor de intrare returnează un șir de dimensiune *fixă*, și cea mai mică modificare a datelor cauzează efecte semnificative în șirul rezultat
- **Previous Hash**: *hash*-ul blocului *anterior*; permite conectarea blocului curent la restul blockchain-ului și semnalează *integritatea* datelor. Dacă în datele blocului anterior apare o modificare, hash-ul blocului respectiv se va schimba și nu va mai corespunde cu *previous hash-ul* blocului curent.
- **Data**: datele propriu-zise care se doresc stocate în blockchain.

3.3.2. Conținutul unui blockchain

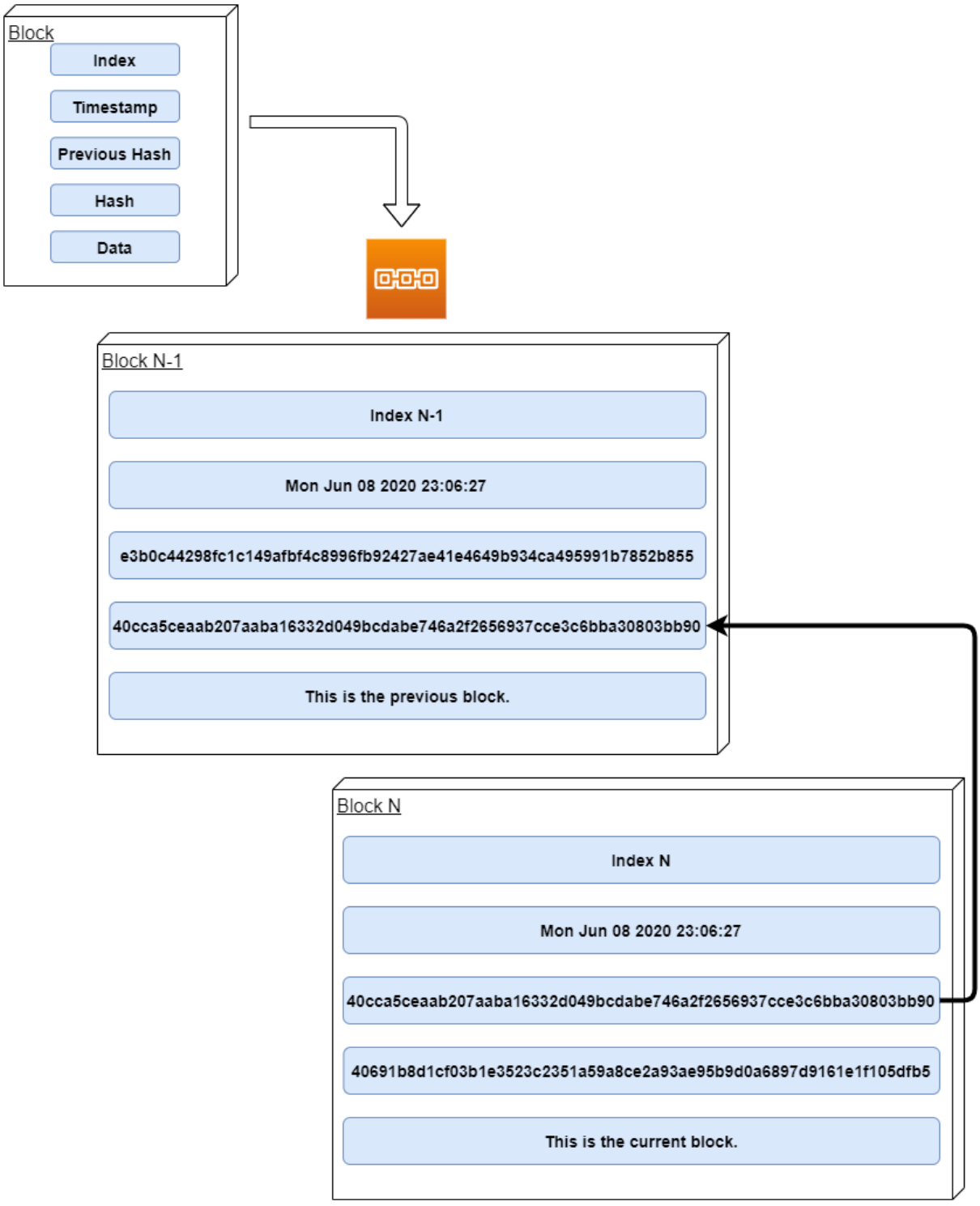
După cum indică și numele, un blockchain este o *înlănțuire* de blocuri care respectă următoarele caracteristici:

- **Scalabil**: blocurile pot fi adăugate *gradual*.
- **Permanent**: odată ce un bloc ajunge în blockchain rămâne stocată pentru o perioadă nedeterminată de timp.
- **Cronologic**: blocurile sunt însemnate cu un *timestamp* și adăugate în ordine cronologică; mereu ultimul bloc este cel mai recent adăugat ca oră și dată.

Primul bloc se numește bloc de *geneză* și nu conține date. Acest bloc nu va avea pointer către blocul anterior (fiind inexistent), nu va stoca date similare celor din restul blockchain-ului și va fi marcat cu timestamp-ul creării blockchain-ului.

Restul blocurilor vor încapsula datele reale respectând tiparul după cum se poate observa in figura 3.4. Pentru generarea valorilor de *hash* s-a utilizat un convertor disponibil online[14], convertor care folosește algoritmul de criptare SHA-256.

[14] https://xorbin.com/tools/sha256-hash-calculator



Figură 3.4 - Structura blockchain

3.3.3. Blockchain pentru sistemul medical

Odată cu creșterea popularității *blockchain-ului* a crescut și numărul de domenii în care acesta poate fi utilizat cu succes[15]. Printre domeniile care ar putea integra această tehnologie pentru a îmbunătăți semnificativ calitatea serviciilor si siguranța lor, se numără și domeniul *medical*.

Una dintre problemele majore înfruntate de sistemul medical în ceea ce privește gestiunea datelor, este *interoperabilitatea*[16]. Mai multe entități trebuie să modifice aceleași date, iar pacienții ar trebui să furnizeze aceleași date tuturor entităților solicitante. Astfel că, apare problema conexiunii corecte între pacient și seturile de *EHR* corespunzătoare.

EHR - *Electronic Health Record* a fost introdus cu scopul de a *digitaliza* informațiile cu caracter medical ale persoanelor. Există numeroase *standarde* care își propun să reglementeze fie ce date se stochează, fie modul în care acestea sunt stocate precum este prezentat in [8]. Ceea ce au în comun toate aceste standarde sunt suportul pentru fișiere *multimedia*, stocarea *persistentă* a datelor și folosirea unor *șabloane* pentru structurarea datelor, dar ceea ce le lipsește cu desăvârsire este implementarea unor mecanisme de *siguranță* care să asigure *integritatea* datelor și *nerepudierea* și o modalitate de stocare, ce poate oferi accesul tuturor la aceleași date.

Stocarea datelor într-un sistem *descentralizat* și *transparent* ar permite accesul persoanelor autorizate oricând și ar asigura *integritatea* și *actualitatea* datelor. Astfel, pacienții ar putea transmite datele de la un medic la altul cu ușurință, iar actualizarea acestora s-ar realiza într-un mod *transparent*. Fiind identificați *unic* prin intermediul unui *hash*, nu ar mai exista problema pierderii datelor sau a potrivirii greșite între pacienți și *EHR*.

Pe de altă parte, și sistemul farmaceutic ar putea beneficia din integrarea unui *blockchain*. De multe ori, medicamentele „se pierd” pe parcursul lungului lanț de custodie dintre furnizor si destinatar. Astfel că, medicamentele ajung să fie vândute pe piața neagră fără prescripție, la un preț neadecvat și cauzând probleme atât pentru persoanele care le consumă fără recomandare, dar și pentru persoanele care au nevoie dar nu le pot procura. În prezent, IBM a integrat un astfel de sistem, *Rapid Supplier Connect*[17], pentru a putea ajuta în lupta cu noul *Coronavirus*. Sistemul ajută la alocarea resurselor acolo unde sunt necesare, permițând introducerea de cereri pentru materiale sau medicamente. Datorita transparenței acestei tehnologii, cererile pot fi văzute și gestionate cu ușurință, furnizorii având posibilitatea să cunoască exact cantitatea de materiale necesare.

Totodată, din anul 2012, Estonia folosește *blockchain* pentru a stoca toate datele medicale sub formă *criptată*. Există și alte exemple de companii care au abordat deja această tehnologie pentru diferite motive[18]. Patentory[19], Robomed[20], BurstIQ[21] sau

[15] https://blockgeeks.com/guides/what-is-blockchain-technology/
[16] https://blockgeeks.com/guides/blockchain-in-healthcare/
[17] https://www.ibm.com/blogs/blockchain/2020/04/ibm-rapid-supplier-connect-getting-covid-19-responders-the-equipment-they-need/
[18] https://builtin.com/blockchain/blockchain-healthcare-applications-companies
[19] https://patientory.com
[20] https://www.robomed.io
[21] https://www.burstiq.com

Medical Chain[22] sunt doar câteva dintre companiile care folosesc *blockchain* pentru a stoca datele pacienților, rezultatele analizelor și rețetele emise. Clinicile, laboratoarele sau pacienții pot accesa datele si le pot modifica pentru a fi în conformitate cu actualitatea si diagnosticele, iar ordonarea lor cronologică face mult mai ușor de urmărit parcursul unui pacient.

Așadar, pentru sistemul propus s-a ales stocarea urgențelor într-un *blockchian*, în conformitate cu standardele *EHR*, pentru ca datele sa poată fi păstrate în siguranță și accesate doar de persoane autorizate. Totodată, dacă este nevoie de acestea pentru analiza lor ulterioară, statistici sau studii clinice se pot pune la dispoziție într-o manieră ordonată și transparentă având certitudinea integrității datelor.

3.4. Aplicații similare

Telefonul mobil a devenit practic o „extensie” a omului deci nu este de mirare ca exista o aplicație pentru orice. Astfel că, există o gamă largă de aplicații menite să ușureze luatul deciziilor în situații de criză. Fie că e vorba de servicii integrate în aplicații consacrate precum Facebook (serviciul care permite anunțarea prietenilor ca ești bine într-o situație de urgență), fie aplicații de sine stătătoare care au ca unic scop raportarea unei urgențe în diferite moduri, oferirea de informații ajutătoare în situații limită sau consultații video.

În continuare se vor descrie trei aplicații similare sistemului propus.

3.4.1. Descrierea unor aplicații asemănătoare

My Flare Alert[23] este o aplicație care permite raportarea situațiilor de urgență prin intermediul unei descrieri text.

Odată raportată, se trimit *mailuri, mesaje* sau *notificări* către persoanele de contact prestabilite cu locația *GPS* și cu descrierea. În plus, odată la 20 de secunde trimite înregistrările *video* realizate cu scopul oferirii mai multor detalii pentru rezolvarea problemei. Aplicația este conectată și cu serviciul de urgență 911 și poate fi apelată din aplicație la raportarea urgenței. Aplicația mai pune la dispoziția utilizatorilor și o *alarmă* sonoră puternică care să semnaleze incidentul.

[22] https://medicalchain.com/en/
[23] http://myflare911.com

Figură 3.5 - Interfața aplicației MyFlare Alert

Life360[24] este o altă aplicație disponibilă în magazinul Google Play[25]. Acesta oferă o *hartă* interactivă pe care pot fi localizate toate persoanele conectate, trimite *notificări* cu locația și facilitează comunicarea prin intermediul unui *chat*. Versiunea Premium pune la dispoziție un serviciu de „Crash Detection" care trimite notificări persoanelor de contact și *alertează* autoritățile.

Figură 3.6 - Aplicația Life360

[24] https://www.life360.com
[25] https://play.google.com/store/apps/details?id=com.life360.android.safetymapd&hl=ro

Tot din seria aplicațiilor care își propun să *digitalizeze* sistemul medical este și **Babylon**[26]. Această aplicație are atât o componentă web cât și una mobilă și pune în legătură persoane cu probleme medicale și doctori.

Utilizatorul introduce *simptomele* sale și răspunde la întrebări care sunt procesate prin intermediul inteligenței artificiale. După această analiză preliminară, se decide dacă trebuie să ia legătura cu un doctor sau problema a fost doar una superficială. Aplicația permite discuția prin apel video sau chat cu un doctor, schimbul de fișiere și vizualizarea unor date medicale și al propriului istoric.

Figură 3.7 - Aplicația Babylon

3.4.2. Analiză comparativă

Tabelul 3.2 prezintă o analiză comparativă între sistemul propus și cele 3 aplicații descrise anterior.

După cum se poate observa, nu există un sistem care sa înglobeze mai multe tipuri de urgențe, ci doar aplicații specializate pe raportarea incidentelor sau pe consultații online.

Sistemul propus dorește să îmbine gestionarea cu succes a unor situații limită și aducerea în mediul online a tuturor beneficiilor mersului la cabinetul medical. Prin colaborarea celor trei tipuri de utilizatori, sistemul poate fi de folos pentru varii situații micșorând numărul de apeluri la serviciul de urgență 112 și oferind sfaturi personalizate, validate de către persoane specializate.

[26] https://www.babylonhealth.com/product

Tabel 3.2 – Analiza comparativă a aplicațiilor

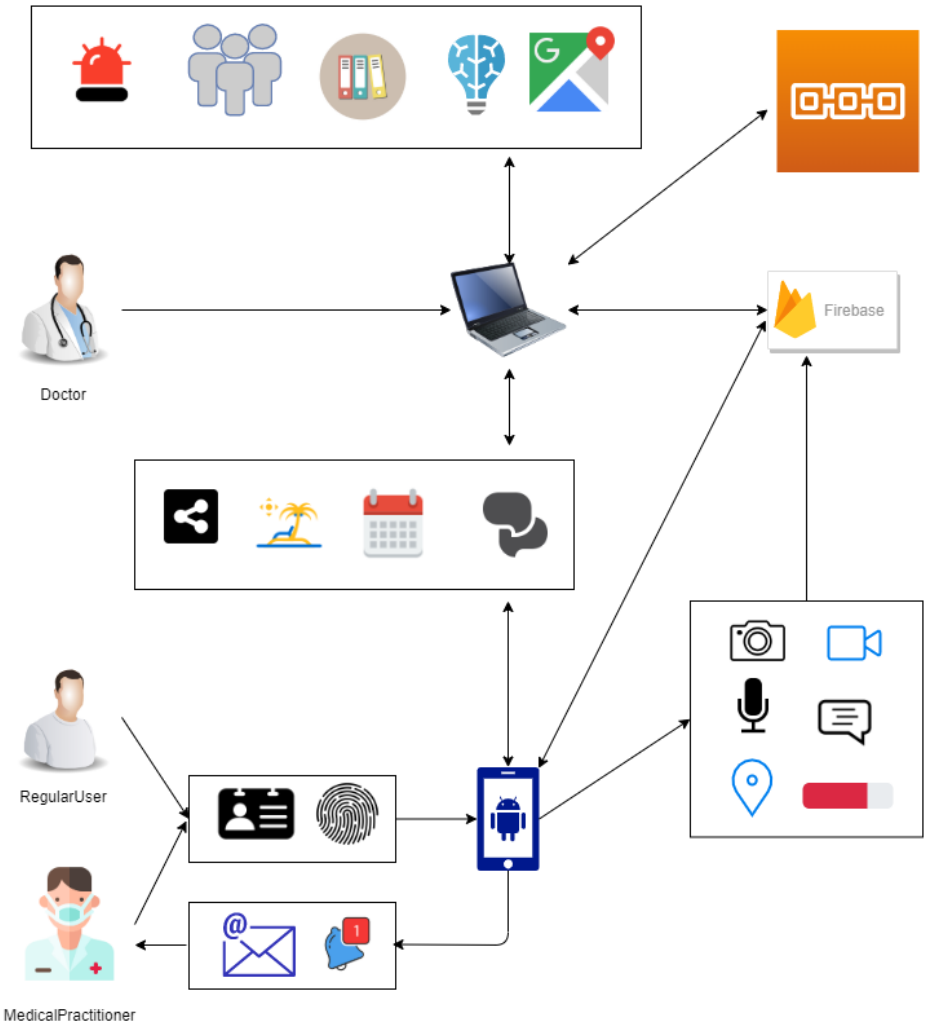
Funcționalitate	Include	My Flare Alert	Life360	Babylon	MHealth
Componeta mobila	-	✓	✓	✓	✓
Componeta web	-	×	×	✓	✓
Raportare urgență	Descriere text	✓	×	×	✓
	Imagine	✓	×	×	✓
	Video	✓	×	×	✓
	Audio	✓	×	×	✓
	Nivel de severitate	×	×	×	✓
Localizare GPS	-	✓	✓	×	✓
Traducere în timp real	-	×	×	×	✓
Autodetecție limbă	-	×	×	×	✓
Autentificare biometrică	-	×	×	×	✓
Creare cont cu automatizare date buletin	-	×	×	×	✓
Push notifications folosind geofencing	-	✓	✓	×	✓
Procesare urgențe	Cluster urgențe	×	×	×	✓
	Procesare imagini	×	×	×	✓
	Procesare text	×	×	✓	✓
	Procesare audio	×	×	×	✓
Vizualizare harta interactiva	-	×	✓	×	✓
Stocare date in blockchain	-	×	×	×	✓
Vizualizare istoric	-	×	✓	×	✓
Date statistice	-	×	×	✓	✓
Chat	-	×	✓	✓	✓
Programare consultație	Gestionare progamari pentru doctor	×	×	✓	✓
	Adaugare programari pentru pacient	×	×	✓	✓
Distribuire fișiere intre doctor si pacient	-	×	×	×	✓
Gestionare perioada concediu	Programare concediu	×	×	×	✓
	Alegere înlocuitor	×	×	×	✓
	Disponibilitate in caz de urgență	×	×	×	✓

Capitolul 4. Analiză și Fundamentare Teoretică

În acest capitol se prezintă arhitectura conceptuală a sistemului și cerințele acestuia. Totodată, se realizează și descrierea cazurilor de utilizare și se argumentează setul de tehnologii ales pentru implementare.

4.1. Arhitectura conceptuală a sistemului

În figura 4.1 este prezentată arhitectura conceptuală a întregului sistem. Aplicația se împarte în două module principale: unul mobil și unul Web care interacționează prin schimb de informații și prin acces la resurse comune, precum aceeași bază de date (modelată prin intermediul Firebase). Totodată, se pot remarca și cele trei tipuri de utilizatori: doctor, utilizator cu pregătire medicală și utilizator obișnuit, dar și modulele la care aceștia au acces.



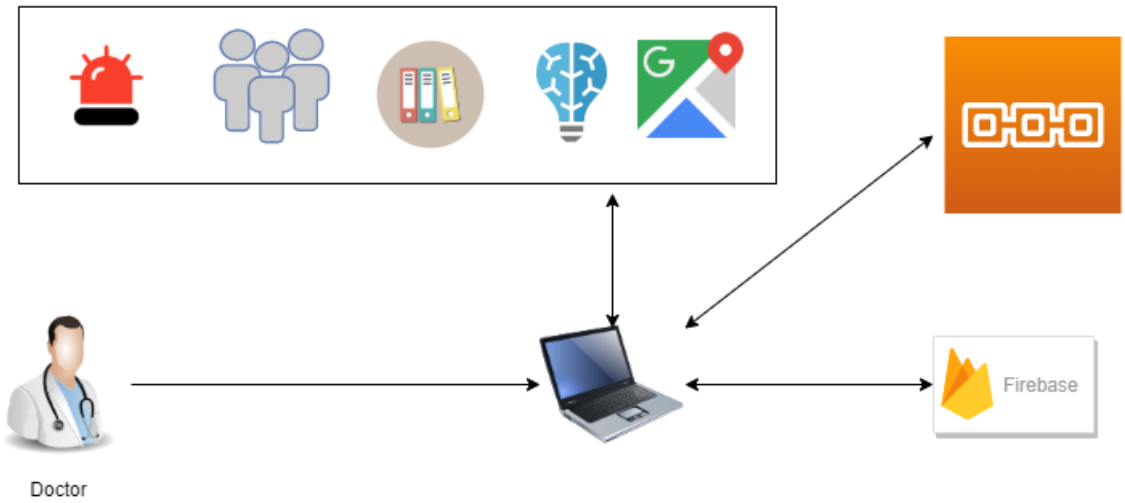
Figură 4.1 - Diagrama arhitecturii conceptuale

Componenta web este integrată pe Cloud pentru ca performanța acesteia să nu fie influențată de aparatura utilizatorului, iar aplicația să fie accesată cu ușurință de aceștia.

Atât aplicația mobilă, cât și cea web, sunt împărțite în 2 module: cel personal și cel dedicat gestionării urgențelor. Partea personală se ocupă de programări și gestionarea fișierelor dintre doctori și pacienți, având la dispoziție ca și canal de comunicare un chat. În ceea ce privește modulul de urgențe, datele sunt preluate prin intermediul aplicației mobile și prelucrate de aplicația web.

Pentru un strat adițional de securitate, urgențele care nu mai necesită intervenție vor fi mutate și păstrate într-un blockchain.

Figura 4.2 reprezintă arhitectura modulului Web care se ocupă cu gestionarea urgențelor și are ca unic tip de utilizator, doctorul.



Figură 4.2 - Diagrama modulului implementat

Modulul gestionării urgențelor este compus eminamente din:

- vizualizarea urgențelor curente
- vizualizarea persoanelor specializate disponibile să răspundă urgenței
- vizualizarea unui istoric al urgențelor
- Machine Learning pentru procesarea automată a datelor
- vizualizarea unui hărți interactive pe baza Google Maps

De îndată ce o urgență are statusul „closed”, adică nu mai este necesară intervenția asupra sa, aceasta este ștearsă din baza de date comună și mutată în blockchain. Astfel, datele necesare prezentării istoricului sunt preluate din blockchain, unde vor rămâne stocate pentru o perioadă nedeterminată de timp, într-un mod sigur, pentru protecția datelor medicale care au un caracter sensibil.

Restul datelor, se obțin prin citirea bazei de date Firebase, populate cu ajutorul aplicației mobile.

4.2. Cerințe de sistem

4.2.1. Cerințe funcționale

Cerințele funcționale sunt cele care validează sistemul, asigurând că s-a realizat ceea ce s-a dorit / a fost cerut. Acestea definesc într-un mod complet toate acțiunile pe care un utilizator poate să le execute și ce rezultate poate obține utilizând sistemul în modul în care a fost definit si proiectat.

Componenta de gestiune are ca unic tip de utilizator doctorul. Astfel, în continuare se prezintă ce poate face un doctor care deja s-a autentificat în sistem:

- Vizualizarea, în ansamblu, a urgențelor active.
    - Vizualizarea unei liste a urgențelor active, în ordine cronologică, de la cele mai recente la cele mai vechi raportate
    - Vizualizarea numărului total de urgențe active
    - Vizualizarea numărului de persoane cu pregătire medicală înregistrate
    - Sortarea după tip a urgențelor
    - Sortarea după status a urgențelor
    - Sortarea după nivelul de severitate a urgențelor
    - Căutarea după locație a urgențelor
    - Căutarea după nivelul de severitate a urgențelor

- Vizualizarea unei urgențe active
    - Vizualizarea detaliilor generale despre urgență
        - Vizualizarea tipului de urgență
        - Vizualizarea locației urgenței
        - Vizualizarea statusului urgenței
        - Vizualizarea unei liste de persoane care au preluat urgența
        - Vizualizarea nivelului de severitate al urgenței
    - Vizualizarea datelor transmise de către utilizatorii aplicației mobile
        - Vizualizarea imaginilor preluate de la fața locului
        - Vizualizarea secvențelor video preluate de la fața locului
        - Vizualizarea descrierilor text realizate la fața locului
        - Ascultarea înregistrărilor vocale realizate la fața locului
    - Vizualizarea rezultatelor prelucrării datelor transmise
        - Vizualizarea unui pie-chart / grafic cu detaliile extrase din imagini
        - Vizualizarea cuvintelor cheie și a categoriei din care face parte urgența, extrase din descrierile text și audio
    - Introducerea unei păreri proprii, a unor detalii care să ghideze persoanele de la fața locului

- Vizualizarea unei hărți interactive
    - Vizualizarea numărului de urgențe active
    - Vizualizarea numărului de persoane care intervin la urgențe
    - Vizualizarea punctelor în care se află urgențele active

- Vizualizarea unui istoric al urgențelor
    - Vizualizarea numărului de urgențe închise
    - Vizualizarea numărului de persoane care au intervenit la urgențe
    - Vizualizarea numărului de locații din care au fost raportate urgențe
    - Vizualizarea unei liste a urgențelor închise, în ordine cronologică, de la cele mai recente la cele mai vechi raportate

- Vizualizarea notificărilor primite

4.2.2. Cerințe non-funcționale

Cerințele non-funcționale reprezintă un factor extrem de important în dezvoltarea unui sistem de calitate, care nu doar să își deservească scopul, dar să o facă într-un mod optim. Dacă cerințele funcționale ne asigură validarea sistemului („building the right product"), cerințele non-funcționale contribuie la procesul de verificare al sistemului („building the product right"). Respectarea acestor cerințe conduce la obținerea unor rezultate nu doar corecte, dar și de încredere și la certitudinea că sistemul funcționează acum la un randament maxim și întreținerea / dezvoltarea lui ulterioară se pot realiza cu ușurință.

În dezvoltarea sistemului *MHealth* s-a ținut cont de îndeplinirea și respectarea unor astfel de cerințe pentru ca experiența utilizatorului să fie cât mai plăcută, modulele și aplicațiile să poată comunica cu ușurință între ele, mentenanța să poată fi realizată fără probleme și de către persoane care nu cunosc detaliile de implementare, iar sistemul să poată fi scalat oricând.

În continuare, se vor detalia niște cerințe non-funcționale definitorii pentru modulul de gestiune al urgențelor al aplicației Web:

- **Securitatea**: această cerință este una ce nu ar trebui să lipsească din nicio aplicație, cu atât mai mult una cu caracter medical, domeniu în care siguranța datelor și intimitatea / caracterul anonim al persoanelor sunt aspecte primordiale. Pe lângă *autentificarea* în sistem folosind logarea cu username și parolă, tokeni și sesiuni active, detaliile legate de cazuri sunt păstrate într-un *blockchain*. Astfel, imagini și secvențe video care prezintă victime sau martori, înregistrări sau descrieri detaliate ale unor situații critice sau sfaturi cu caracter confidențial nu vor fi la îndemâna oricărui utilizator, ci doar al doctorilor autentificați. Totodată, blockchain-ul păstrează o evidență completă a datelor și a evoluției lor (modificările care apar) prin intermediul unui *timestamp*.

- **Elasticitatea**: este definită de maniera în care un sistem reușește să se adapteze la schimbările de încărcătură prin alocarea automată a resurselor într-un mod echilibrat. Această *echilibrare* se referă la raportul cerere / răspuns și nu neapărat la alocarea resurselor în mod egal între utilizatori, servere sau aplicații. Sistemele bazate pe *Cloud* îndeplinesc această cerință printr-un așa numit „*Load Balancer*" care alocă resurse într-un mod optim între servere și între instanțele aplicației. Astfel, dacă se fac foarte multe cereri dintr-o instanță a aplicației din Cluj-Napoca, serverului din Germania (care este principalul responsabil de sistemul

MHealth) îi vor fi alocate mai multe resurse decât restul serverelor puse la dispoziție de Google, iar instanța respectivă va avea alocate mai multe resurse decât orice altă instanță.

- **Scalabilitatea**: este o altă caracteristică recunoscută în principal la aplicațiile bazate pe Cloud. Tocmai *elasticitatea* este cea care permite o scalare ușoară pe *verticală* a aplicației, nefiind necesară o alocare suplimentară manuală de resurse sau modificarea hardware-ului în vreun fel. De asemenea, aplicația permite adăugarea de noi module sau integrarea unor altor servicii Cloud cu ușurință făcând posibilă astfel scalarea pe *orizontală* . Baza de date poate fi extinsă, de asemenea, prin adăugarea de date sau de colecții, iar modul în care informația este structurată poate fi modificată nefiind vorba de o bază de date relațională.

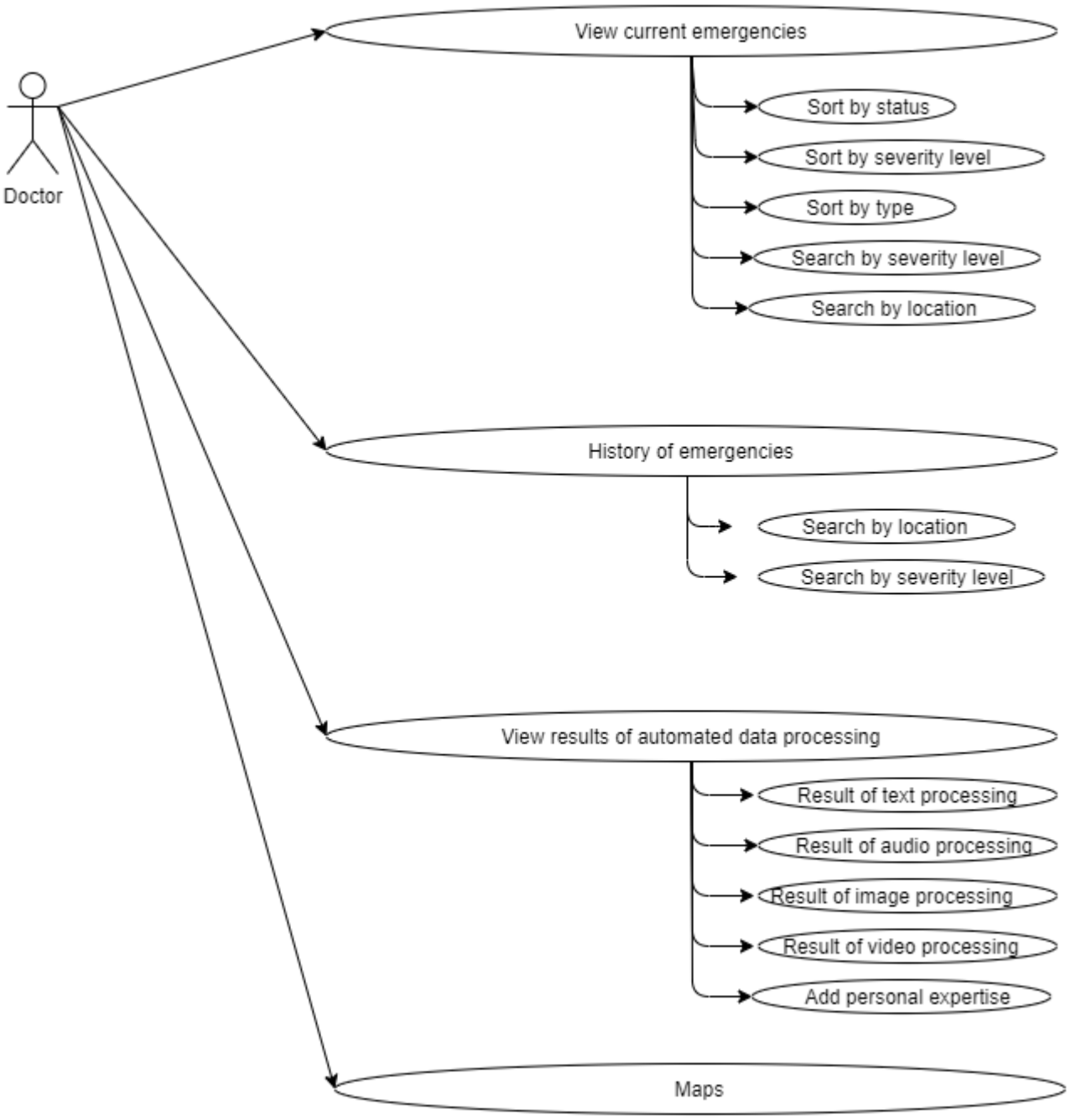
- **Performanța**: împreună cu *utilizabilitatea,* determină succesul sistemului din punctul de vedere al utilizatorului. Pe lângă punerea la dispoziție a unor servicii într-un mod organizat și ușor de înțeles, sistemul oferă răspunsuri într-un timp scurt indiferent de numărul de utilizatori sau performanța dispozitivului de pe care e utilizată aplicația. Sistemul este găzduit pe *Cloud,* deci resursele sunt aparent infinite, baza de date integrată notifică schimbările *în timp real*, serviciile care implică *Machine Learning* sunt fie bazate pe seturi de date alese în concordanță cu nevoile (pentru a obține răspunsuri detaliate, dar specifice), fie folosind agenți deja antrenați de platforma de *Cloud* (antrenați pe un set de date extins și care oferă răspunsuri de calitate și cu un factor de încredere mare).

4.3. Cazuri de utilizare

Cazurile de utilizare ale unui sistem reprezintă o colecție a interacțiunilor posibile dintre utilizator (numit actor) și sistem. Acestea sunt adesea folosite pentru o analiză completă a cerințelor sistemului.

În ceea ce privește actorii, sistemul *MHealth* are 3 astfel de roluri: utilizator obișnuit, utilizator cu pregătire medicală și doctor.

În figura 4.3, sunt prezentate cazurile de utilizare pentru unicul actor al componentei de gestiune a urgențelor: doctorul.



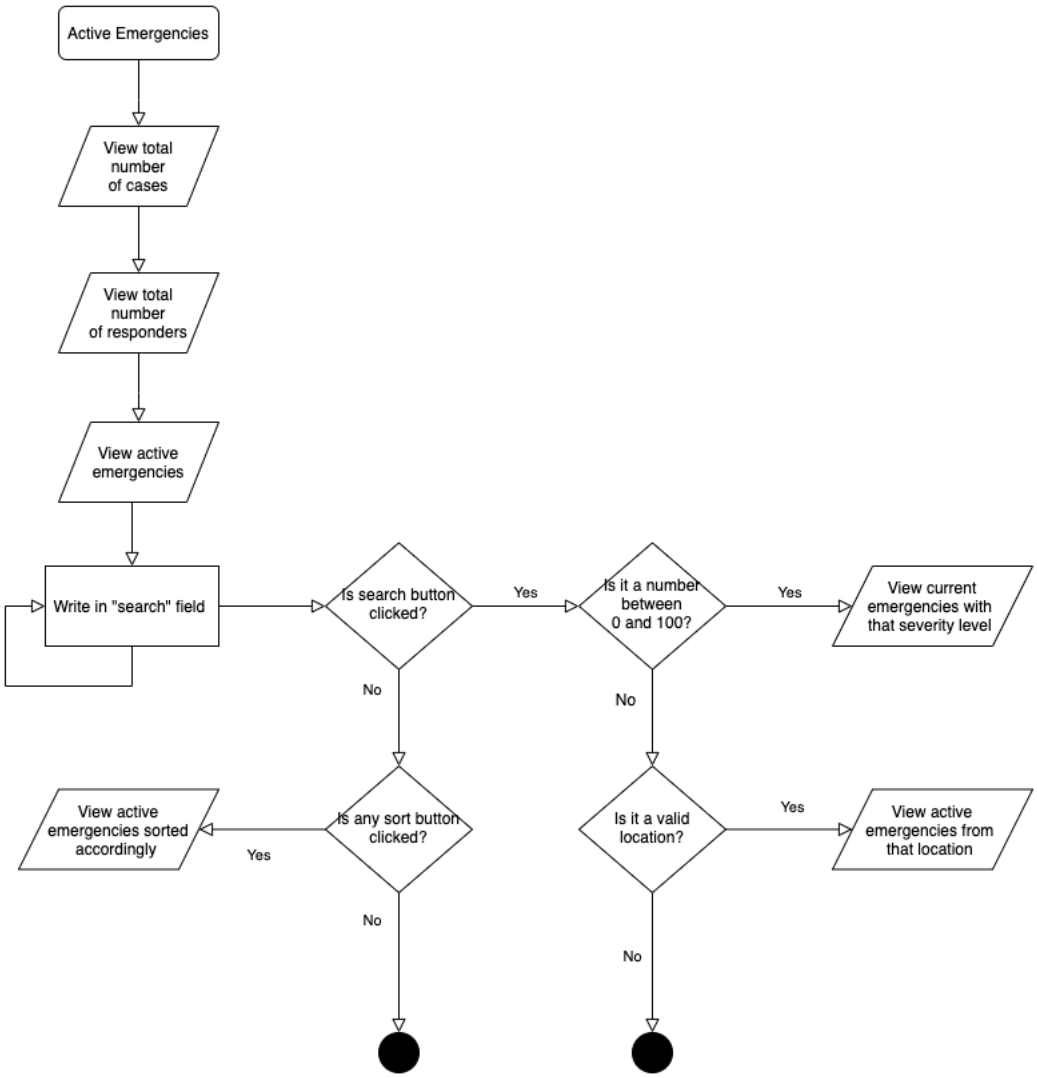
Figură 4.3 - Use-case pentru utilizatorul Web

În cele ce urmează, prin intermediul tabelelor 4.1, 4.2, 4.3 și 4.4 se vor prezenta cele patru cazuri de utilizare aferente utilizatorului de tip doctor:

Tabel 4.1 – Primul caz de utilizare

Caz de utilizare 1	
Nume caz de utilizare:	Vizualizarea urgențelor active
Include:	• Sortare după status, tip sau grad de severitate • Căutare după locație sau grad de severitate
Precondiții:	• Actorul are o conexiune la internet • Actorul este autentificat în sistem • Actorul se află pe pagina „Active Emergencies”
Postcondiții:	• Informațiile afișate rămân neschimbate pe durata întregului scenariu • Sortările si căutările returnează rezultate în conformitate cu alegerile actorului
Scenariu așteptat:	Acest caz de utilizare începe după ce utilizatorul a fost autentificat cu succes în sistem și redirecționat spre pagina de „Active Emergencies” sau când alege în mod explicit navigarea către aceasta pagină și se sfârșește când actorul alege să schimbe pagina. 1. Actorul vizualizează un tabel cu informații generale despre cazurile active la momentul respectiv. 2. Actorul vizualizează date statistice precum numărul total de cazuri sau de utilizatori cu pregătire medicală. 3. Utilizatorul caută urgențele după criterii precum locația, statusul sau nivelul de severitate. Căutarea se face prin introducerea criteriului în câmpul destinat căutării și acționarea butonului de căutare sau apăsarea tastei „enter”. 4. Vizualizarea rezultatelor.
Scenariu alternativ:	1. Actorul nu are conexiune / pierde conexiunea la internet (poate să apară la pasul 1, 2 sau 3). În acest caz, este redirecționat spre o pagină de eroare, fără ca acest lucru să afecteze datele în vreun fel. 2. Actorul introduce date eronate în câmpul destinat căutării (poate să apară la pasul 3). În acest caz, se va returna o listă goală însoțită de un mesaj de informare.

Figura 4.4 prezintă fluxul de date al cazului de utilizare descris anterior.



Figură 4.4 – Flux de date pentru primul caz de utilizare

Tabel 4.2 – Al doilea caz de utilizare

Caz de utilizare 2	
Nume caz de utilizare:	Gestionarea rezultatelor procesării automate
Include:	• Rezultatele procesării textului • Rezultatele procesării imaginilor • Rezultatele procesării înregistrărilor audio • Rezultatele procesării înregistrărilor video • Vizualizarea datelor transmise despre caz • Introducerea părerilor proprii despre caz
Precondiții:	• Actorul are o conexiune la internet • Actorul este autentificat în sistem • Actorul se află pe pagina „Current Emergency”
Postcondiții:	• Informațiile afișate rămân neschimbate pe durata întregului scenariu • Procesările returnează rezultate relevante • Datele introduse de utilizator sunt salvate în sistem • Emailul cu informații este transmis către utilizatorul solicitant al aplicației mobile
Scenariu așteptat:	Acest caz de utilizare începe după ce utilizatorul a fost autentificat cu succes în sistem și alege în mod explicit navigarea către pagina „Current Emergency” (prin acționarea butonului de „More Info” din dreptul unei urgențe de pe pagina „Active Emergencies”) și se sfârșește când actorul alege să schimbe pagina. 1. Actorul vizualizează un tabel cu informații generale despre cazurile active la momentul respectiv. 2. Actorul vizualizează datele transmise de la fața locului. 3. Actorul vizualizează rezultatele procesării automate a datelor transmise de la fața locului 4. Actorul introduce propriile păreri despre caz, în câmpul destinat acestui lucru. 5. Transmiterea emailului se face prin acționarea butonului „send”.
Scenariu alternativ:	1. Actorul nu are conexiune / pierde conexiunea la internet (poate să apară la pasul 1, 2, 3, 4, 5 sau 6). În acest caz, este redirecționat spre o pagină de eroare, fără ca acest lucru să afecteze datele în vreun fel. 2. Actorul nu apasă tasta de trimitere a emailului (poate să apară la pasul 5). În acest caz, nu se va trimite informația procesată sau informațiile de la medic către solicitant.

Tabel 4.3 – Al treilea caz de utilizare

Caz de utilizare 3	
Nume caz de utilizare:	Vizualizarea istoricului urgențelor
Include:	• Sortare după dată • Sortare după caz • Sortare după locație • Sortare după gradul de severitate
Precondiții:	• Actorul are o conexiune la internet • Actorul este autentificat în sistem • Actorul se află pe pagina „History”
Postcondiții:	• Informațiile afișate rămân neschimbate pe durata întregului scenariu • Sortările și căutările returnează rezultate în conformitate cu alegerile actorului
Scenariu așteptat:	Acest caz de utilizare începe după ce utilizatorul a fost autentificat cu succes în sistem și alege în mod explicit navigarea către pagina „History” (prin alegerea acestei opțiuni din bara laterală) și se sfârșește când actorul alege să schimbe pagina. 1. Actorul vizualizează un tabel cu informații generale despre cazurile închise la momentul respectiv. 2. Actorul vizualizează date statistice precum numărul total de cazuri închise sau de utilizatori cu pregătire medicală. 3. Actorul caută urgențele după criterii precum locația, data, cazul sau nivelul de severitate. Căutarea se face prin introducerea criteriului în câmpul destinat căutării și acționarea butonului de căutare sau apăsarea tastei „enter”. 4. Vizualizarea rezultatelor.
Scenariu alternativ:	1. Actorul nu are conexiune / pierde conexiunea la internet (poate să apară la pasul 1, 2 sau 3). În acest caz, este redirecționat spre o pagină de eroare, fără ca acest lucru să afecteze datele în vreun fel. 2. Actorul introduce date eronate în câmpul destinat căutării (poate să apară la pasul 3). În acest caz, se va returna o listă goală însoțită de un mesaj de informare.

Tabel 4.4 – Al patrulea caz de utilizare

Caz de utilizare 4	
Nume caz de utilizare:	Vizualizarea unei hărți interactive
Include:	-
Precondiții:	• Actorul are o conexiune la internet • Actorul este autentificat în sistem • Actorul se află pe pagina „Maps”
Postcondiții:	• Informațiile afișate rămân neschimbate pe durata întregului scenariu
Scenariu așteptat:	Acest caz de utilizare începe după ce utilizatorul a fost autentificat cu succes în sistem și alege în mod explicit navigarea către pagina „Maps” (prin alegerea acestei opțiuni din bara laterală) și se sfârșește când actorul alege să schimbe pagina. 1. Actorul vizualizează o hartă pe care sunt marcate urgențele active. 2. Actorul vizualizează date statistice precum numărul total de cazuri sau de utilizatori cu pregătire medicală.
Scenariu alternativ:	1. Actorul nu are conexiune / pierde conexiunea la internet (poate să apară la pasul 1 sau 2). În acest caz, este redirecționat spre o pagină de eroare, fără ca acest lucru să afecteze datele în vreun fel.

4.4. Setul de tehnologii utilizate

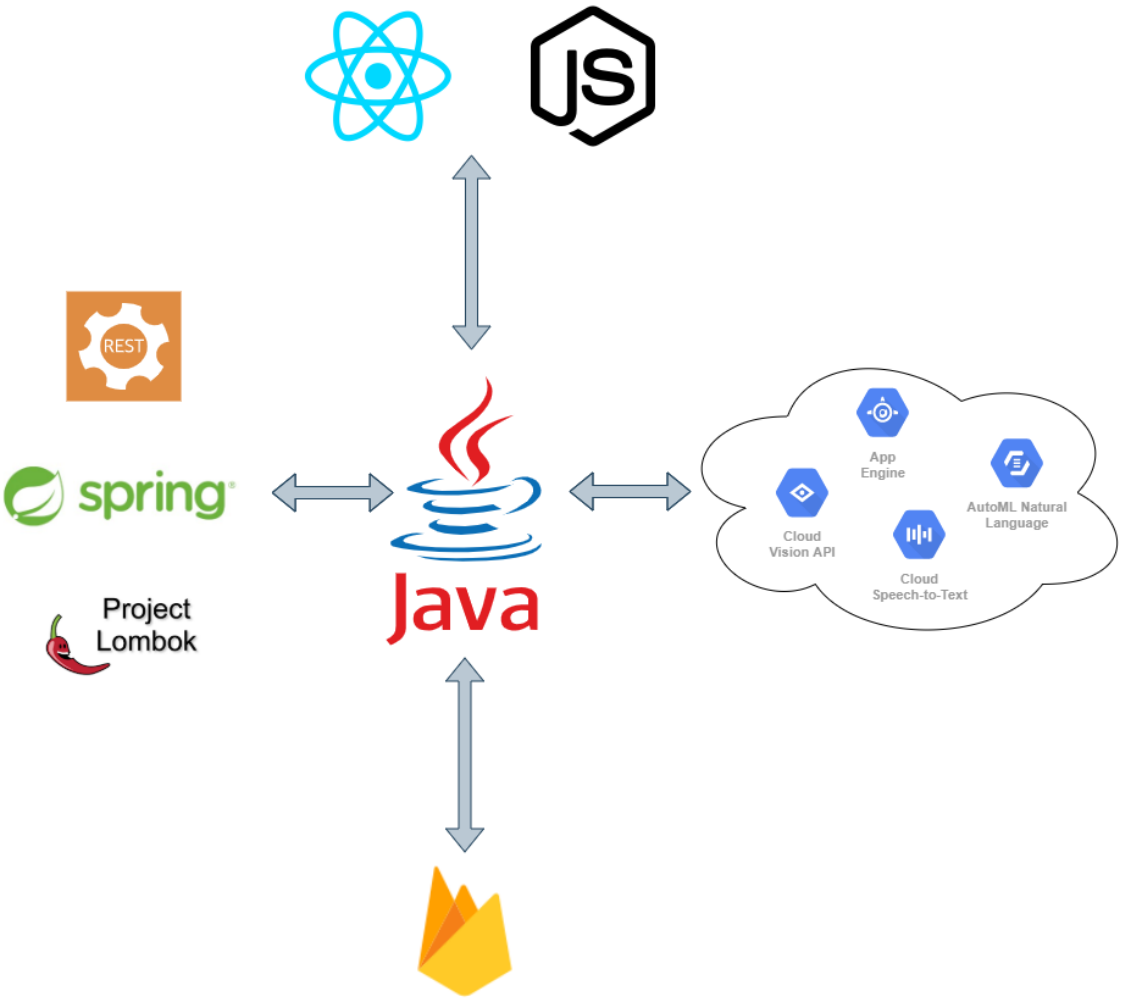
Pentru dezvoltarea acestui sistem s-au analizat câteva tehnologii candidat pentru realizarea sistemului. Astfel, în tabelul 4.5 sunt prezentate soluțiile alese pentru fiecare problematică și cum au evoluat acestea pe parcursul studiului în funcție de ce probleme s-au întâmpinat.

Ulterior, s-a ales un set de tehnologii optim, care să corespundă standardelor actuale și care să deservească scopurilor aplicației. S-au folosit tehnologii și framework-uri de actualitate care pot, de asemenea, să fie utilizate împreună cu ușurință precum este ilustrat în figura 4.5.

Modulul de gestionare al urgențelor este reprezentat sub forma unei aplicații Web, formată dintr-o componentă de frontend, una de backend și o bază de date.

Tabel 4.5 – Tehnologii candidat pentru sistem

Problematica	Sursa solutiei	Evoluția tehnologiei alese +killer usecase
Securizare	Java	**Criptarea obiectelor de tip Emergency** - s-a găsit o metodă mai ușor de implementat și mai sigură **Blockchain**
Deploy Cloud	Google Cloud	**App Engine**
Procesare Imagini	Google Cloud	**Vision API**
Procesare Text	Google Cloud	**Natural Language API** - rezultatele furnizate nu sunt suficient de precise pentru scopul aplicației si este necesară antrenarea unui agent specializat **AutoML Language API**
Procesare Audio	Google Cloud	**Speech-to-text Language API**
Chat - web+mobile	PubNub	**RabbitMQ** - s-a dorit utilizarea unei tehnologii de ultimă generație chiar dacă sistemul MHealth nu beneficiază de toate funcționalitățile la dispoziție de către PubNub **PubNub Chat**
Push notifications	-	**Google Nearby Messages API -** aria de acoperire oferită de acest API este prea mică, în jur de 20m maxim, în timp ce sistemul MHealth își propune oferirea notificărilor indiferent de locația utilizatorilor **Firebase Cloud Messaging -** deși această tehnologie oferă funcționalitatea dorită de a trimite notificări indiferent de locație, nu se pot filtra utilizatorii în funcție de distanța față de urgență **Geofencing API**
Traducere Text	Firebase	**ML Kit**
Transalatare din coordonate în adresă și invers	-	**Geocoding API**
Stocare Fisiere	Firebase	**Firebase Storage**
Stocare date personale	Firebase	**Firebase Database**
Preluare date buletin	Firebase	**ML Kit**



Figură 4.5 - Setul de tehnologii

4.4.1. Backend

Prin backend se înțelege nivelul aplicației care se ocupă cu *accesul la date* și partea de *logică* a aplicației. Acesta joacă rol de *server* pentru aplicația client, căreia ii expune un set de *endpoint-uri* ce pot fi apelate fără a se dezvălui modul de implementare.

4.4.1.1. Java

Java[27] este un limbaj de programare *orientat-obiect* care permite dezvoltarea aplicațiilor desktop, web și chiar mobile. Programele Java sunt executate în *JVM* (Java Virtual Machine) și odată scrise, pot fi rulate oricând (*write once, run anywhere*) [28].

[27] https://www.java.com/en/

S-a ales versiunea 11 a Java Development Kit-ului, fiind (la momentul emiterii temei) cea mai nouă versiune de tip „Long Term Support”, tip adecvat dezvoltării aplicațiilor ce urmează să fie *deployed*.

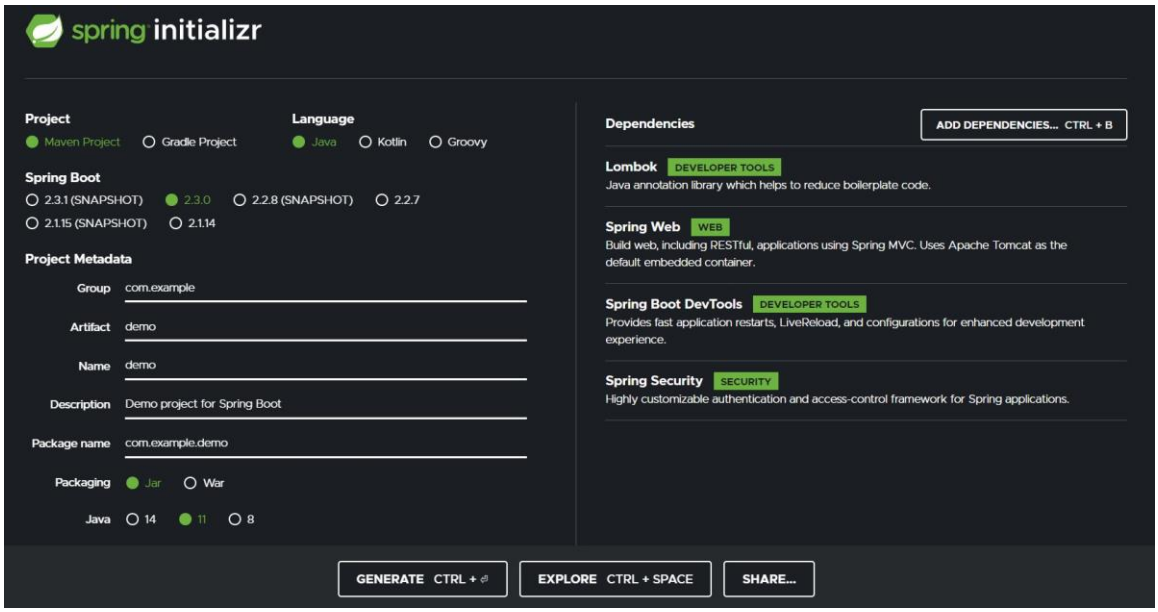
IntelliJ IDEA[29] este unul dintre mediile de dezvoltare integrate care permit dezvoltarea de aplicații folosind Java, considerat *superior* altor medii datorită *feedback*-ului oferit programatorului - prin propuneri de denumiri, avertizări și sugestii privind erorile și datorită modului de funcționare *mai rapid* și fără întreruperi.

4.4.1.2. Spring

Spring[30] Framework este o platformă open-source, care permite și simplifică scrierea aplicațiilor Web, în Java.

Aplicația va rula astfel într-un *container*, în care pot sau nu să fie incluse diferite elemente ale aplicației. Librăriile permit utilizarea diferitor *adnotări* care marchează elemente ce trebuie căutate la rulare. Astfel, la compilare, se realizează niște *bean*-uri, care sunt apoi *injectate* în restul codului, unde este necesar.

Un instrument util la începerea unui nou proiect este **Spring Initializr**[31], care permite crearea proiectului inițial, alegerea versiunilor de Java și Spring, dar și introducerea de dependințe într-un mod intuitiv și prietenos, după cum prezintă figura 4.6.



Figură 4.6 – SpringInitializr

[28] https://www.geeksforgeeks.org/why-is-java-write-once-and-run-anywhere/
[29] https://www.jetbrains.com/idea/
[30] https://spring.io
[31] https://start.spring.io

4.4.1.3. REST Services

Representional **S**tate **T**ransfer este un stil arhitectural care permite crearea unei aplicații ca un set de micro-servicii, prin introducerea de protocoale de comunicare între sistemele Web și adăugarea de constrângeri și proprietăți după cum se prezintă și în capitolul 6 din [3].

S-a ales acest tip de servicii în detrimentul serviciilor SOAP datorită performației superioare (prin prisma mecanisemelor de caching), și a posibilității integrării cu JSON pentru acceptarea mai multor tipuri de date.

Aplicațiile care folosesc servicii REST sunt considerate ușor și rapid de folosit, deoarece resursele sunt identificate prin intermediul unor URI.

Cele mai folosite metode HTTP folosite prin serviciile REST sunt:

- **GET**: folosită pentru returnarea unor rezultate de la server; este singurul tip de metodă care nu necesită prezența unui body.
- **POST**: este folosită pentru a crea o noua resursă; cererea conține un header cu permisiunile și un body cu informația care trebuie adăugată pe server.
- **DELETE**: folosită pentru eliminarea resurselor.
- **PUT**: folosită pentru actualizarea sau modificarea resurselor.

4.4.1.4. Lombok

Inclus într-un proiect Java, acest proiect open-source contribuie semnificativ la reducerea codului *boilerplate*. Astfel, secțiuni de cod redundante sau cu foarte puține modificări nu vor mai trebui scrise, fiind înlocuite cu succes de *adnotări*. Mai multe detalii se regăsesc pe pagina principală a librăriei [32].

Principalul avantaj îl constituie eliminarea *getter*-elor și a *setter*-elor prin simpla adnotare ***@Data***, deasupra definirii clasei. Alte adnotări utile sunt:

- **@AllArgsConstructor**: înlocuiește constructorul cu toți parametrii.
- **@NoArgsConstructor**: înlocuiește constructorul fără parametrii.
- **@NotNull**: introduce o constrângere și previne apariția erorilor de tip *Null Pointer*.

4.4.2. Frontend

Această componentă joaca rolul de *client* în aplicațiile client-server, fiind partea expusă utilizatorului. Aplicațiile de acest tip, nu au acces direct la date, fiindu-le expuse un *API* pe care îl apelează atât pentru a extrage date, cât și pentru a le adăuga sau modifica. Acest lucru contribuie la securitatea aplicațiilor, cunoscându-se doar *ce* se poate face, nu și *cum*.

[32] https://projectlombok.org

4.4.2.1. React și NodeJs

Deși Angular este framework-ul oferit de Google cand vine vorba de JavaScript, s-a ales React[33] datorita unei mai bune performanțe în dezvoltarea aplicațiilor multi-page. React este o librărie open-source de JavaScript care este utilizată pentru crearea *interfețelor* grafice, dezvoltată de *Facebook* în 2013.

Deoarece această librărie se ocupă în principal de randare, este necesară introducerea unor alte librării care să îi completeze funcționalitățile. Aceste pachete pot fi adăugate cu ușurință utilizând un manager de pachete pentru Javascript precum *npm*.

Niște pachete necesare dezvoltării unei aplicații Web complete sunt:

- **Router**: permite navigarea prin aplicație, mapând paginile la URL-ul corespunzător și oferind posibilitatea adăugării unei rute prestabilite.
- **Axios**: permite realizarea apelurilor către server
- **Material UI**: conține componente stilizate, gata de utilizat în componentele nou create.

Visual Studio Code este un editor de cod oferit gratuit de către *Microsoft*, care permite pe lângă scrierea de cod Javascript, instrumente de *debugging*, autocompletare a codului și sugestii de *refactorizare*. Totodată, oferă o bună integrare cu *git*-ul, iar terminalul pus la dispoziție permite instalarea pachetelor prin *npm*.

Odată cu introducerea mediului de rulare open-source NodeJs[34], în 2009, s-a făcut posibilă rularea codului Javascript, în afara unui browser Web.
Printre principalele caracteristici sunt *rapiditatea* cu care se execută codul, faptul că toate librăriile sunt asincrone (astfel nu apar blocaje) și ușor *scalabil*[35].

4.4.3. Google Cloud Platform

După analiza comparativă a celor trei furnizori (Google, Amazon și Microsoft) prezentată în capitolul anterior, cu accent pe serviciile și caracteristicile care corespund nevoilor aplicației noastre, s-a ales **Google Cloud**. Principalele motive fiind Software Development Kit-ul si Management Tools-urile foarte variate și în conformitate cu tehnologiile alese pentru dezvoltarea aplicației, multitudinea de API-uri ușor de folosit și integrat pentru procesarea datelor de care se folosește aplicația, dar și creditul inițial[36] de $300 care acoperă cel puțin nevoile aplicației din perioada de dezvoltare și testare.

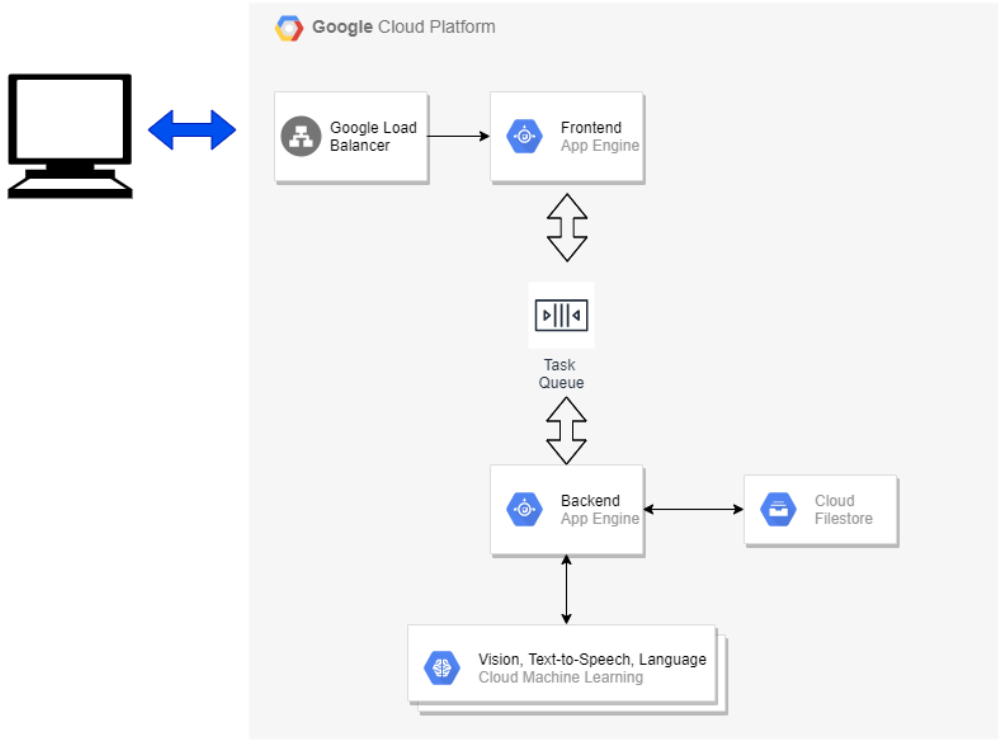
În cele ce urmează, se va prezenta *platforma* de cloud-computing și niște API-uri relevante în dezvoltarea sistemului *MHealth*.

[33] https://reactjs.org
[34] https://nodejs.org/en/
[35] https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm
[36] https://metrixdata360.com/cloud-comparison-amazon-web-services-vs-microsoft-azure-vs-google-cloud/

4.4.3.1. App Engine

Google permite dezvoltarea si găzduirea aplicațiilor în centrele sale de date prin intermediul serviciului de tip *Platform-as-a-Service,* App Engine[37]. Aplicațiile sunt rulate pe mai multe servere și se oferă *scalare* automată, raportată la numărul de cereri pentru fiecare server.

În figura 4.7, este prezentat fluxul unei interacțiuni cu platforma Google Cloud, prin App Engine. Aplicatiile de frontend și backend sunt găzduite în proiecte de tip App Engine (fie în același proiect sub forma a două servicii diferite, fie în doua proiecte distincte). Când utilizatorul face o cerere, Load Balancer-ul se ocupă de redirecționarea sa spre serverul portivit sau de alocarea mai multor resurse serverului în cauză. Cererile sunt trimise către backend în ordinea în care se primesc (similar unei cozi), iar aplicația de backend comunică atât cu datele stocate, cât și cu alte servicii Cloud.



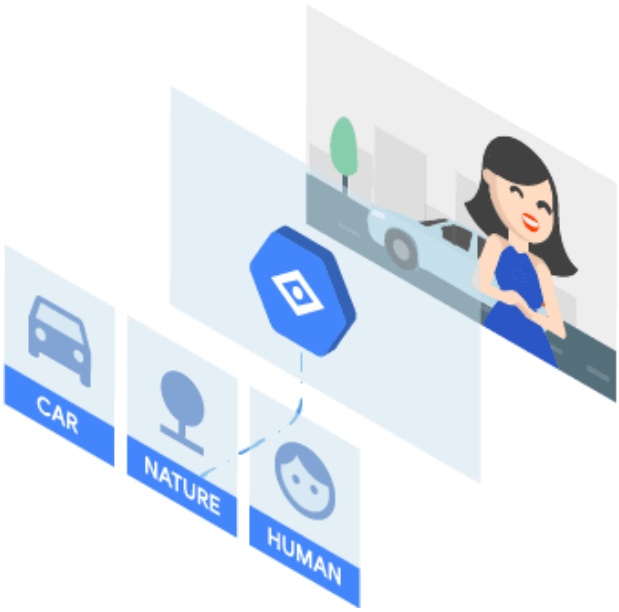
Figură 4.7 - App Egine Flow

4.4.3.2. Vision API

Google oferă un serviciu de *analiză* al imaginilor folosind *Machine Learning*, prin intermediul API-ului Vision[38]. Acest API permite realizarea apelurilor către modele pre-antrenate, pe seturi foarte mari de date și returnează răspunsuri diverse, dar specifice.

Vision API asignează etichete imaginilor și le clasifică în categorii predefinite, precum este exemplificat în figura 4.8.

[37] https://cloud.google.com/appengine
[38] https://cloud.google.com/vision



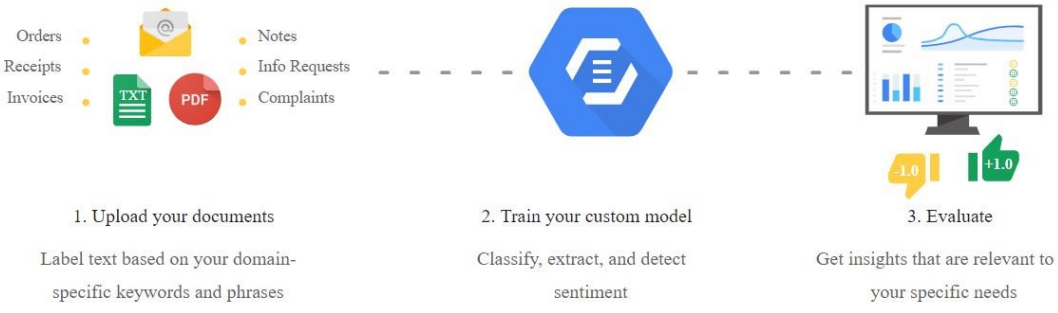
Figură 4.8 - Vision API

Pentru a realiza această clasificare, Google folosește propriul framework de Machine Learning, *TensorFlow*[39]. Prin intermediul *rețelelor neuronale*, imaginea introdusa este încadrată într-un *cluster*. Ulterior, se găsesc toate clusterele apropiate aceluia si se verifică potrivirea cu imaginea pentru o *etichetare* cât mai amănunțită și cu un grad de încredere cât mai ridicat.

4.4.3.3. Natural Language API

Este serviciul de analiză al textului oferit de platforma Google Cloud. Asemenea Vision API, acesta poate clasifica textul în categorii predefinite prin agenți antrenați.

Pe de altă parte, componenta sa de AutoML[40], permite antrenarea propriului model. Setul de date ales este încărcat și prelucrat conform fluxului din figura 4.9.



Figură 4.9 - Modul de funcționare al Natural Language[41]

[39] https://www.tensorflow.org
[40] https://cloud.google.com/natural-language/automl/docs

Astfel, datele sunt încărcate și *clasificate* după criterii specifice, alese de utilizator. Apoi, modelul este *antrenat* și *deployed*. Ulterior, se pot face *apeluri* către modelul personalizat la fel cum s-ar face către modelul pre-antrenat, dar rezultatele vor fi unele mult mai specifice, în deplină conformitate cu nevoile aplicației în care se folosește.

4.4.3.4. Speech-to-Text API

Acest API[42] permite dezvoltatorilor să transforme secvențe audio în text, aplicând modele de rețele neuronale. Sunt recunoscute în mod automat 120 de limbi și permite alegerea de cazuri de utilizare specifice pentru a putea obține rezultate cât mai specifice.

În plus, poate translata atât secvențe în timp real, cât și secvențe înregistrate anterior și încărcate.

4.4.4. Git & GitHub Desktop

Github[43] este un sistem de *control* al versiunilor care rulează pe majoritatea platformelor și care a fost proiectat pentru ca munca în echipă să poată fi coordonată cu ușurință, iar schimbările să poată fi urmărite cronologic.

Github oferă un mediu sigur de *păstrare* al codului, fiind de asemenea rapid și *scalabil*, astfel că dimensiunea proiectelor nu constituie o problemă.

Pentru a integra git într-un proiect, trebuie creat un *repository*. Prima oară când se încarcă ceva în repository, se creează automat un *branch* de *master*. Este indicat ca acolo să se păstreze mereu o versiune funcțională a proiectului. Ulterior, se adaugă branch-uri pe care vor rula versiuni noi, poate instabile, ale aplicației. După ce versiunea nouă este testată, poate să fie *merged* în master. Schimbările sunt detectate în mod automat și adăugate într-un mod rapid și sigur. Dacă mai multe persoane modifică același cod, se semnalează un *conflict* care trebuie *rezolvat* de către developeri înainte de a putea fi merged.

GitHub Desktop[44] permite o vizualizare mai prietenoasă a tuturor repository-urilor si branch-urilor. Totodată, înlocuiește realizarea acțiunilor de *push* și *commit* din terminal și semnalează numărul și statusul conflictelor.

[41] https://cloud.google.com/natural-language
[42] https://cloud.google.com/speech-to-text
[43] https://github.com
[44] https://desktop.github.com

Capitolul 5. Proiectare de Detaliu si Implementare

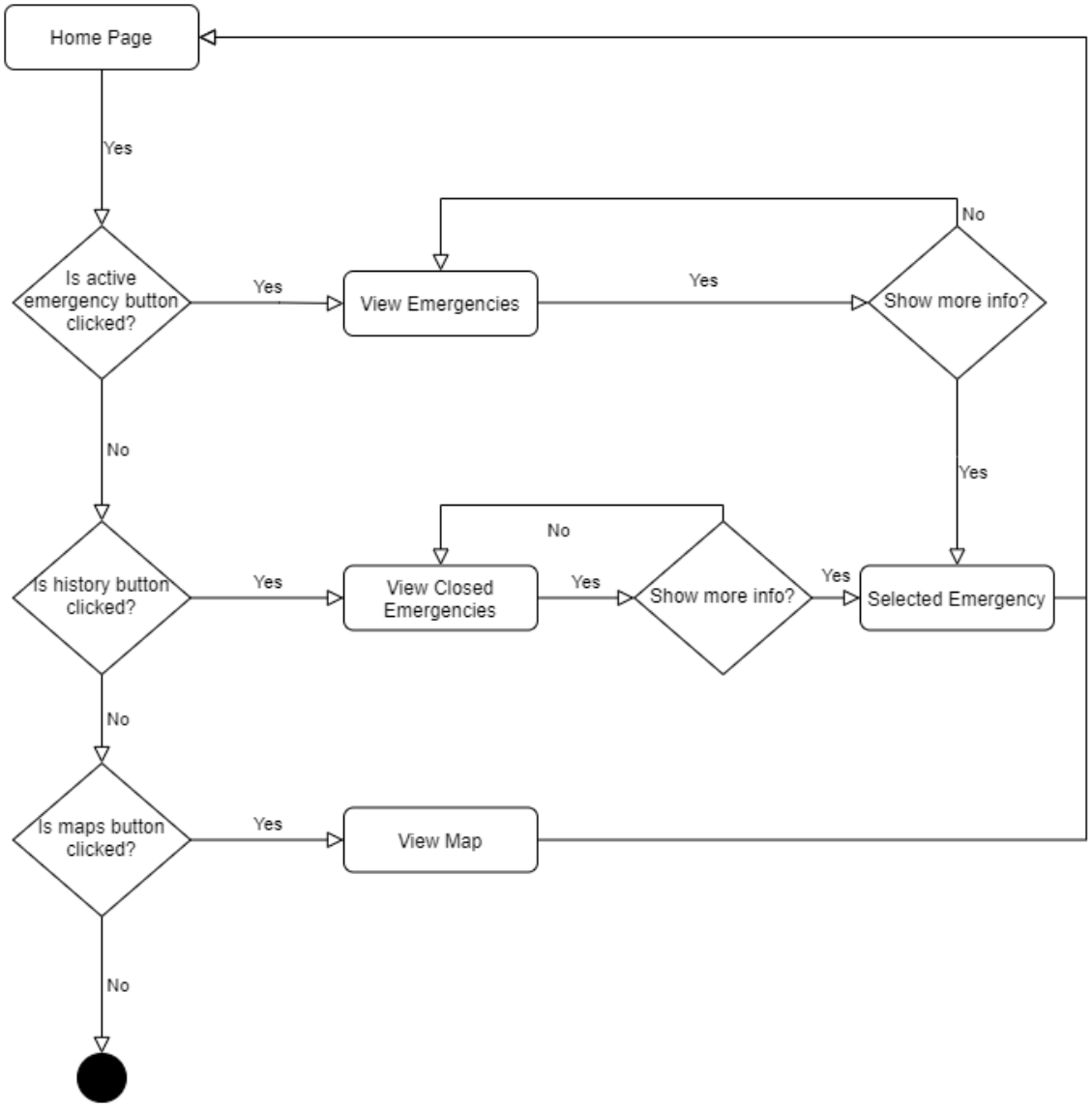
În acest capitol se prezintă diagramele reprezentative ale sistemului și se realizează descrierea implementării modulelor.

5.1. Diagrame reprezentative ale sistemului

5.1.1. Diagrama fluxului de control a aplicației Web

Funcționalitățile aplicației sunt foarte intuitive, iar trecerea de la una la alta se face într-o ordine logică și naturală.

Figura 5.1 prezintă modul în care utilizatorul de tip doctor poate naviga prin aplicație sub forma unei diagrame de flux de control care respectă criteriul exclusivității pe nu.



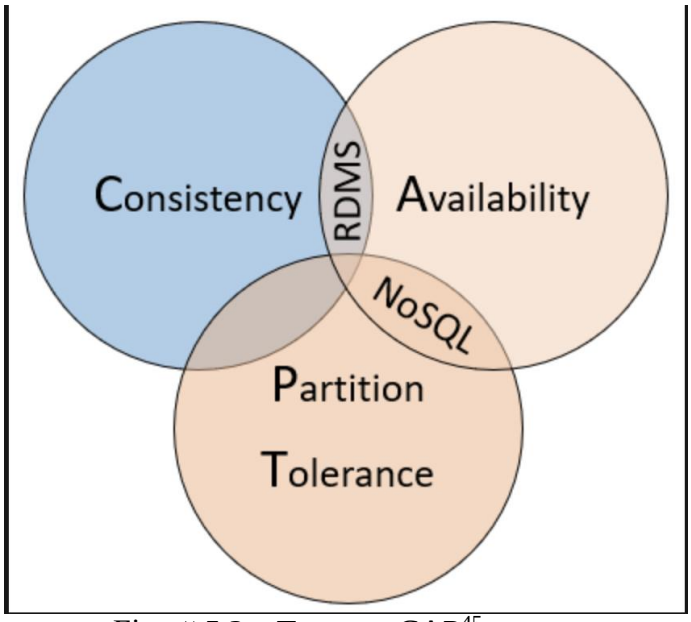
Figură 5.1 - Flowchart pentru utilizatorul Web

După logare, doctorul este redirecționat către pagina sa principala: Home Page. Dintr-un meniu lateral poate alege către ce altă pagină să navigheze. Dacă acționează butonul de Active Emergencies poate vedea urgențele active la momentul respectiv și poate vizualiza mai multe detalii despre o anumită urgență acționând butonul de More Info. Aceeași acțiune poate fi realizată și în cazul detaliilor legate de urgențele stocate în istoric, urgențe care au statusul Closed. Dacă doctorul acționează butonul de Maps, este redirecționat către pagina respectivă pe care îi este prezentată harta interactivă. În cazul în care nu este acționat niciun buton, utilizatorul rămâne pe pagina principală.

5.1.2. Diagrama bazei de date

Pentru stocarea datelor în contextul dezvoltării unui sistem distribuit, s-a ales un model non-relațional, Firestore, care este ușor integrabil în proiectul de Cloud și care primește suport prin intermediul platformei Google.

Când vine vorba de baze de date distribuite, un factor important este teorema CAP (consistență, disponibilitate și partiționare). După cum este prezentat și în figura 5.2, bazele de date de tip NO-SQL tind să sacrifice consistența datelor în favoarea vitezei și a disponibilității.



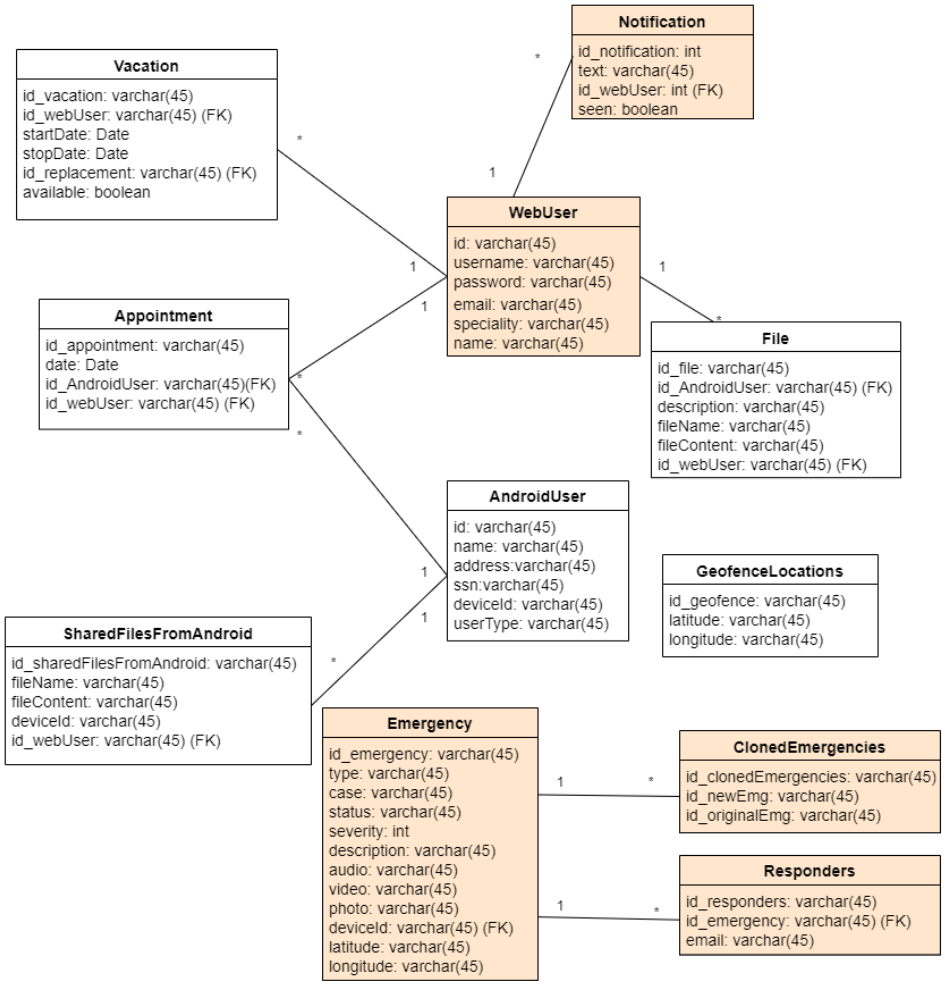
Figură 5.2 – Teorema CAP[45]

Pentru sistemul dezvoltat, viteza, disponibilitatea datelor și partiționarea sunt factori esențiali și constituie fundamentul alegerii unui astfel de model. Consistența datelor sensibile este asigurată prin adăugarea unui blockchain.

Deși s-a ales utilizarea unei baze de date NO-SQL, orientată pe documente, în detrimentul unei baze de date clasice relaționale, figura 5.3 prezintă baza de date într-o formă convențională pentru o mai bună înțelegere a legăturilor dintre elemente și pentru prezentarea într-un mod logic, a elementelor definitorii fiecărei colecții. În anexa 5, se

[45] https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data

regăsește o reprezentare a bazei de date folosind un instrument specializat pentru bazele de date de tip NO-SQL. Totodată, în tabele se prezintă numărul maxim de câmpuri posibile în fiecare document. În funcție de caz, unele câmpuri pot să aibă caracter obligatoriu iar altele caracter opțional pentru modelarea și dezvoltarea sistemului *MHealth*.



Figură 5.3 - Diagrama completă a bazei de date

Colecțiile reprezentative modulului Web de gestionare a urgențelor sunt colorate. Colecția principală folosită de modul este cea de „Emergecy” care este structurată după cum urmează:

- **Id_emergency**: câmp unic, autogenerat de către Firebase, similar unei chei primare a tabelei, care identifică în mod unic fiecare document stocat.
- **Type**: câmp care semnalează dacă urgența este cu caracter „*public*” (de exemplu, un accident rutier sau un incendiu) sau cu caracter „*personal*” (de exemplu, accidente casnice precum tăieturi sau arsuri).
- **Case**: câmp care semnalează categoria din care face parte urgența, după analiza textului.

- **Status**: câmp ce poate lua valorile „*pending*” (cazul este adăugat, dar nicio persoană nu a semnalat participarea la acesta; valoarea „defaut” a acestui câmp), „*active*” (valoarea pe care o are câmpul când cel puțin o persoană acceptă cazul) sau „*closed*” (valoare atribuită când urgența s-a încheiat și semnalează că documentul poate fi mutat în blockchain).
- **Description, Audio, Video, Photo, Severity**: câmpuri cu caracter opțional pentru utilizatorul aplicației mobile, dar care stochează date ce vor fi ulterior procesate prin metode care utilizează Machine Learning.
- **Latitude, Longitude**: date preluate în mod automat de către aplicația mobilă, care marchează locația de unde este raportată o urgență.

Colecțiile „Cloned Emergencies” și „Responders” sunt în strânsă legatură cu colecția „Emergency”.

Deoarece o urgență cu caracter public poate fi remarcată și semnalată de mai mulți utilizatori, este necesară o prelucrare a acestora astfel încât să nu existe urgențe duplicate, iar datele aferente unui singur caz să poată fi prelucrate împreună pentru un rezultat mai complex și mai precis. Colecția are la rândul ei un identificator unic, autogenerat, un câmp cu identificatorul unei urgențe unice și un alt câmp care reprezintă identificatorul aceleiași urgențe, dar adăugată ulterior. Așadar, când se adaugă un document în colecția de „Emergency”, se face o verificare după locație (dacă mai există o urgență raportată în raza imediată) și după timp (dacă urgența nou raportată în aceeași arie este una din prezent), iar dacă se returnează un rezultat valid, în această colecție, se adaugă un nou document care să marcheze duplicatul.

Colecția „Responders” păstrează adresele de email asociate fiecărei persoane dornice să ajute la rezolvarea unei urgențe. Totodată, are și ea un identificator unic autogenerat.

Colecția „WebUser” conține date despre utilizatorii aplicației Web. Aceștia sunt unic identificați printr-un id autogenerat, email și nume de utilizator. Totodată, la crearea contului, aceștia își setează și o parolă și își introduc numele complet.

Utilizatorii vor primi notificări prin intermediul colecției cu același nume. Documentele au câmpuri cu caracter obligatoriu precum identificatorul unic autogenerat, text (textul notificării), seen (câmpul care indică dacă notificarea a fost deja văzută de utilizator și previne afișarea ei de mai multe ori) și identificatorul utilizatorului Web căruia îi este destinată notificarea.

5.1.3. Diagrama de pachete a aplicației Web

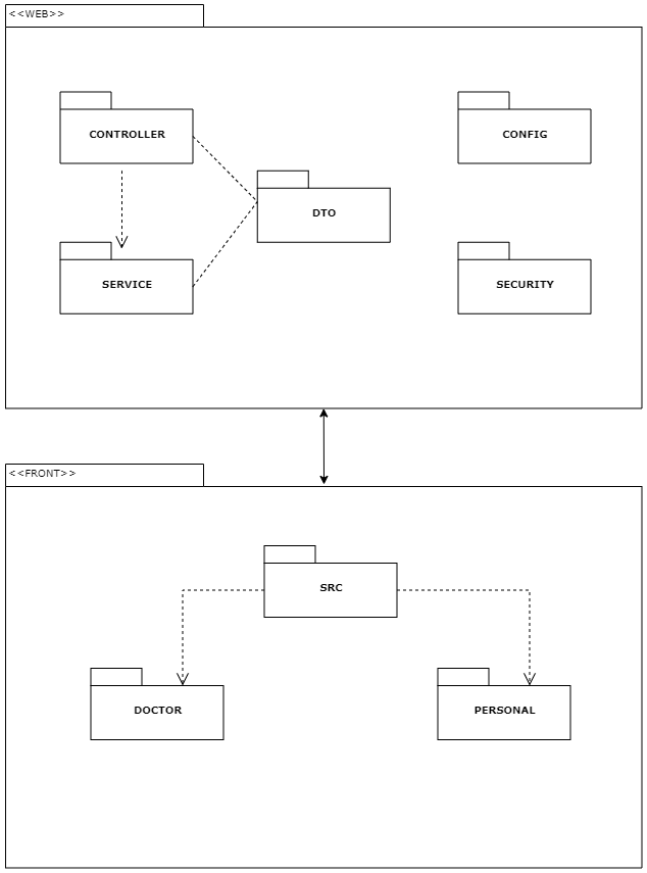
În figura 5.4 este prezentată diagrama de pachete a modulului de gestiune a urgențelor.

În partea superioară se regăsesc pachetele care țin de partea de backend a aplicației:

- Config: este pachetul care conține configurările proiectului legate de baza de date și de politica CROS; de asemenea, include clasele necesare configurării și inițializării blockchain-ului.
- Security: conține clasele care gestionează sesiunile și gestionarea codului de resetare a parolei.

- Dto: înglobează toate clasele de tip model, care modelează obiectele prin intermediul cărora se vor transmite date de la aplicația de frontend către backend, între starturile aplicației de backend și invers.
- Controller: conține toate controller-ele aplicației
- Service: conține toate serviciile cu implementările metodelor, fiind utilizate de controller pentru realizarea operațiilor.

Partea de frontend este structurată mai simplu, având un pachet pentru gestiunea urgențelor, unul pentru modulul personal și unul cu clasele principale, care folosește celelalte două pachete, după caz.

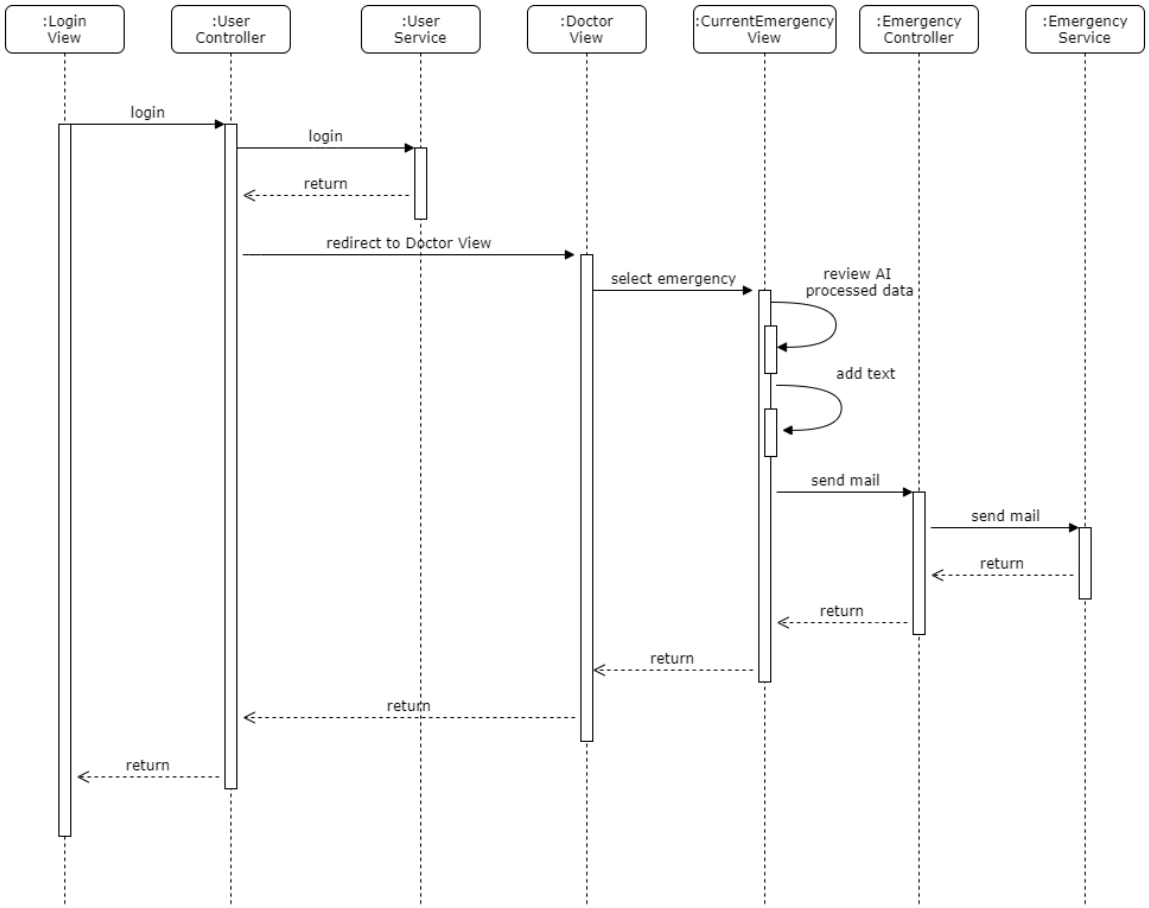


Figură 5.4 - Diagrama de pachete

5.1.4. Diagrama de secvențiere

Figura 5.5 prezintă diagrama de secvențiere a unei funcționalități realizate de doctor. Ca și precondiție la cazul descris, un utilizator al aplicației mobile trebuie să fi raportat o urgență. Se poate observa cum realizarea unei acțiuni de trimitere mail trece prin fiecare clasă implicată.

Caz: completarea detaliilor unui caz și trimiterea mailului către utilizatorul Android.



Figură 5.5 - Diagrama de secvențiere

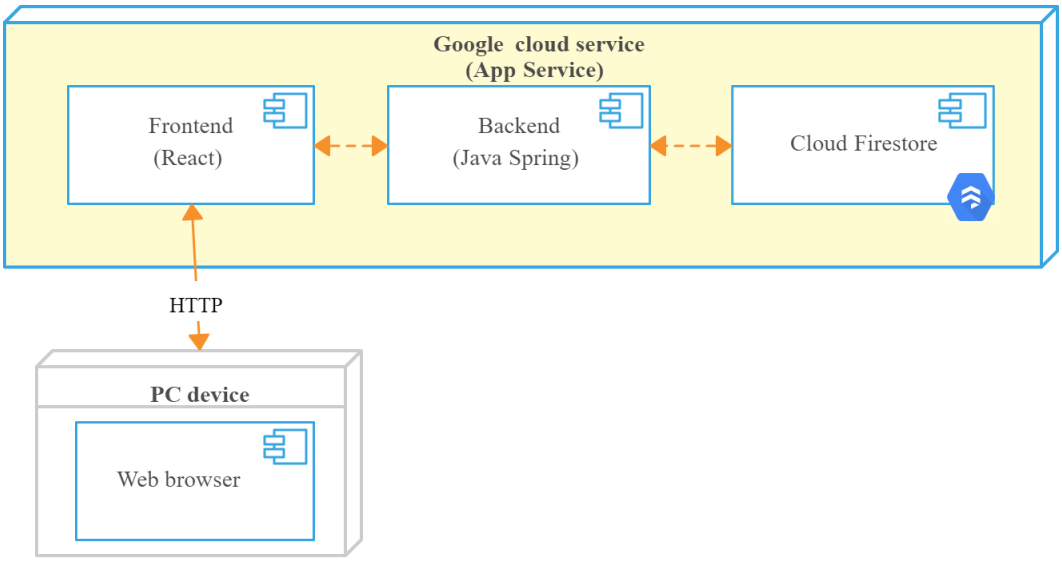
Mai întâi, se realizează legarea în aplicație din vederea respectivă. Acțiunea ajunge ulterior în controller care apelează serviciul corespunzător. Datele sunt verificate și dacă sunt corecte se face redirecționarea către pagina principală a doctorului; altfel, utilizatorul rămâne pe pagina de logare. După cum se poate observa, timpul de viață al logării este cel mai mare deoarece utilizatorul trebuie să fie și să rămână logat pe durata întregului caz.

Odată redirecționat spre pagina de urgențe active, doctorul alege o urgență și este redirecționat către pagina cu informații. Acolo, poate să verifice și confirme datele rezultate în urma procesării automate, poate să adauge propriile păreri și sfaturi și apoi trebuie să acționeze butonul de trimitere mail. Se apelează metoda corespunzătoare din serviciu, mailul este trimis către utilizatorii respectivi, iar în final, se returnează mesaje de succes.

5.1.5. Diagrama de deploy a aplicației Web

Întreaga aplicație este găzduită pe Cloud, iar interacțiunea cu aceasta se realizează prin intermediul internetului. În figura 5.6 se prezintă modul de interacțiune dintre

utilizator (de la un dispozitiv conectat la internet, prin intermediul protocolului HTTP) și modulul web de gestiune a urgențelor.

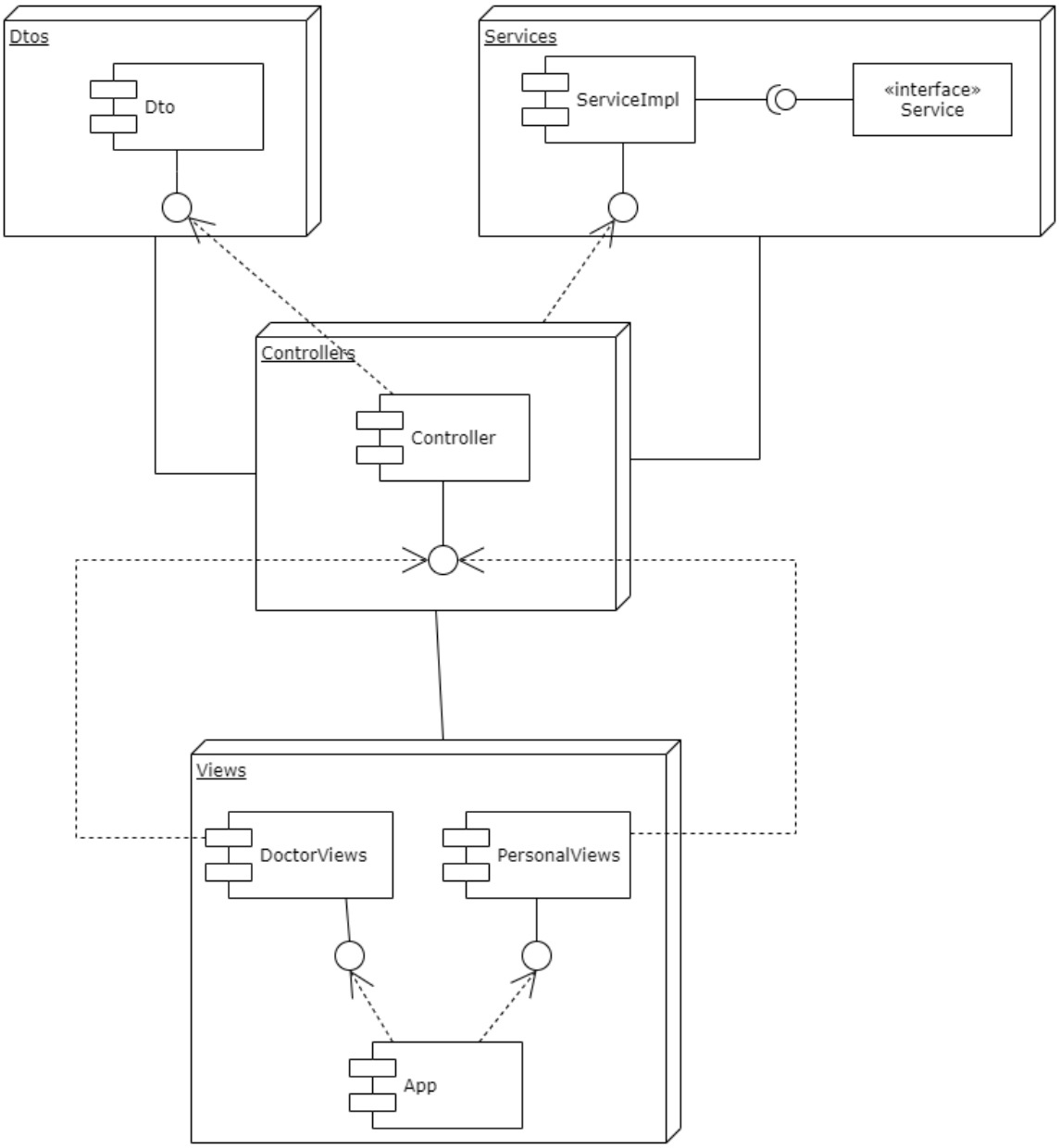


Figură 5.6 - Diagrama de deploy

Pe fond galben, sunt evidențiate componentele aplicației, care se regăsesc pe Cloud în containere diferite și care interacționează între ele prin request-response. Tot pe Cloud sunt stocate și informațiile, într-o bază de date de tip Firestore, în care se poate scrie și din care se poate citi de către componenta de backend. Nu există legătură directă între partea de frontend și baza de date pentru a preveni utilizatorii rău intenționați de la preluarea / scrierea / modificarea frauduloasă a datelor.

5.1.6. Diagrama de componente a aplicației Web

În figura 5.7 este prezentată interacțiunea și modul de organizare a componentelor. Există patru componente (una pentru partea de frontend și trei pentru partea de backend). Componenta principală App de frontend se folosește de DoctorViews și PersonalViews, în timp de acestea se folosesc de componenta de controller pentru a comunica cu backend-ul. Componenta de controller se folosește atât de componenta Dto cât și de cea de Service. Componenta de service conține două elemente: interfețele și serviciile care implementează interfețele respective.



Figură 5.7 - Diagrama de componente

5.2. Descrierea implementării modulelor

5.2.1. Conexiunea cu baza de date și blockchain

Pentru stocarea datelor s-a ales utilizarea unei baze de date de tip Firebase – Cloud Firestore. Pentru conexiunea cu aceasta s-a generat o cheie din Firebase Console care se utilizează pentru autentificarea aplicației. În figura 5.8 este prezentat modul în care se poate realiza inițializarea conexiunii, conexiune ce va fi ulterior utilizată pentru lucrul cu baza de date.

```java
@Component
@Data
@Service
public class FirebaseInitializer {

    @PostConstruct
    public void initialize(){
        try {
            InputStream serviceAccount = getClass().getResourceAsStream( name: "/cloudfirebase.json");

            FirebaseOptions options = new FirebaseOptions.Builder()
                    .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                    .setDatabaseUrl("https://mhealth-angeco.firebaseio.com")
                    .build();

            FirebaseApp.initializeApp(options);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figură 5.8 – Inițializarea conexunii cu baza de date

Apoi, pentru operațiile efectuate cu baza de date se utilizează un serviciu care folosește clientul inițializat.

Un alt aspect al bazei de date îl constituie și event listener-ele adăugate pe tabele. Deoarece o urgență publică poate fi raportată de mai mulți utilizatori, este de bun augur o combinare a tuturor informațiilor pentru o procesare mai precisă a datelor. Astfel, de fiecare dată când se adaugă o urgență, se verifică dacă pentru locația respectivă nu există deja un geofence în baza de date.

```java
FirebaseService.firestore.collection("Emergencies")
        .whereEqualTo("Type", "Public")
        .addSnapshotListener(new EventListener<QuerySnapshot>() {
            @Override
            public void onEvent(@Nullable QuerySnapshot snapshots,
                                @Nullable FirestoreException e) {
                if (e != null) {
                    System.err.println("Listen failed:" + e);
                    return;
                }
                ................
```

Dacă există, se marchează prezența în același cluster prin introducerea unui nou document în colecția *ClonedEmergencies*.

Urgențele închise sunt păstrate într-un blockchain, în conformitate cu standardele EHR. Pentru această parte, s-a realizat o structură proprie asemenea unui blockchain privat, care poate fi utilizat ulterior cu ușurință pentru dezvoltarea ulterioară a sistemului. S-a realizat un job automat care să mute zilnic, la miezul nopții, urgențele cu acest status în blockchain:

```
@Transactional
@Scheduled(cron = "0 0 0 * * *") //every midnight
public void checkEmergencyStatus(){
    List<EmergencyDto> closedEmergencies;
    closedEmergencies = emergencyService.getAllClosedEmergencies();
        for (EmergencyDto dto:closedEmergencies){
        System.out.println("Adding in blockchian...");
        Block newBlock = new Block(
                dto,
                blockchain.get(blockchain.size() - 1).getHash(),
                new Date().getTime());
        newBlock.mineBlock(prefix);
        blockchain.add(newBlock);
     }
}
```

Frecvența acestui job poate fi modificată cu ușurință în funcție de nevoile aplicației.

În ceea ce privește implementarea blockchain-ului, s-a realizat o clasă care să reprezinte blocul și care să accepte ca tip de date stocate un obiect de tip EmergencyDto. La prima rulare a aplicației se creează blockchain-ul sub forma unui ArrayList și se adaugă blocul de geneză. Informațiile sunt minate folosind o funcție de hashing (SHA-256) și un prefix. Prefixul este cel care dictează dificultatea funcției de hashing și durata minării. Deși este ușor să se genereze un hash, este dificil să se găsească hash-ul care are un prefix setat. Pentru acest lucru, s-a ales prefixul „0” și o lungime a sa de 4, deci minarea unui bloc presupune găsirea hash-ului care începe cu patru zerouri. Cu ajustări minime (schimbarea numărului de prefix), dificultatea poate fi crescută pentru sporirea siguranței. Figura. 5.9 prezintă funcțiile de calcul al valorii de hash și funcția de minare a blocurilor.

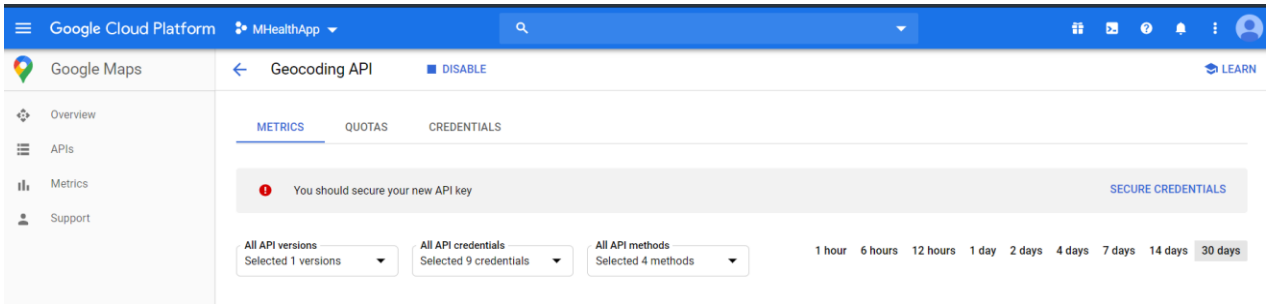
```
public String calculateBlockHash() {
    String dataToHash = previousHash
            + Long.toString(timeStamp)
            + Integer.toString(nonce)
            + data;
    MessageDigest digest = null;
    byte[] bytes = null;
    try {
        digest = MessageDigest.getInstance("SHA-256");
        bytes = digest.digest(dataToHash.getBytes(UTF_8));
    } catch (NoSuchAlgorithmException ex) {
    }
    StringBuffer buffer = new StringBuffer();
    for (byte b : bytes) {
        buffer.append(String.format("%02x", b));
    }
    return buffer.toString();
}

public String mineBlock(int prefix) {
    String prefixString = new String(new char[prefix]).replace( oldChar: '\0',  newChar: '0');
    while (!hash.substring(0, prefix).equals(prefixString)) {
        nonce++;
        hash = calculateBlockHash();
    }
    return hash;
}
```

Figură 5.9 – Minarea blocurilor

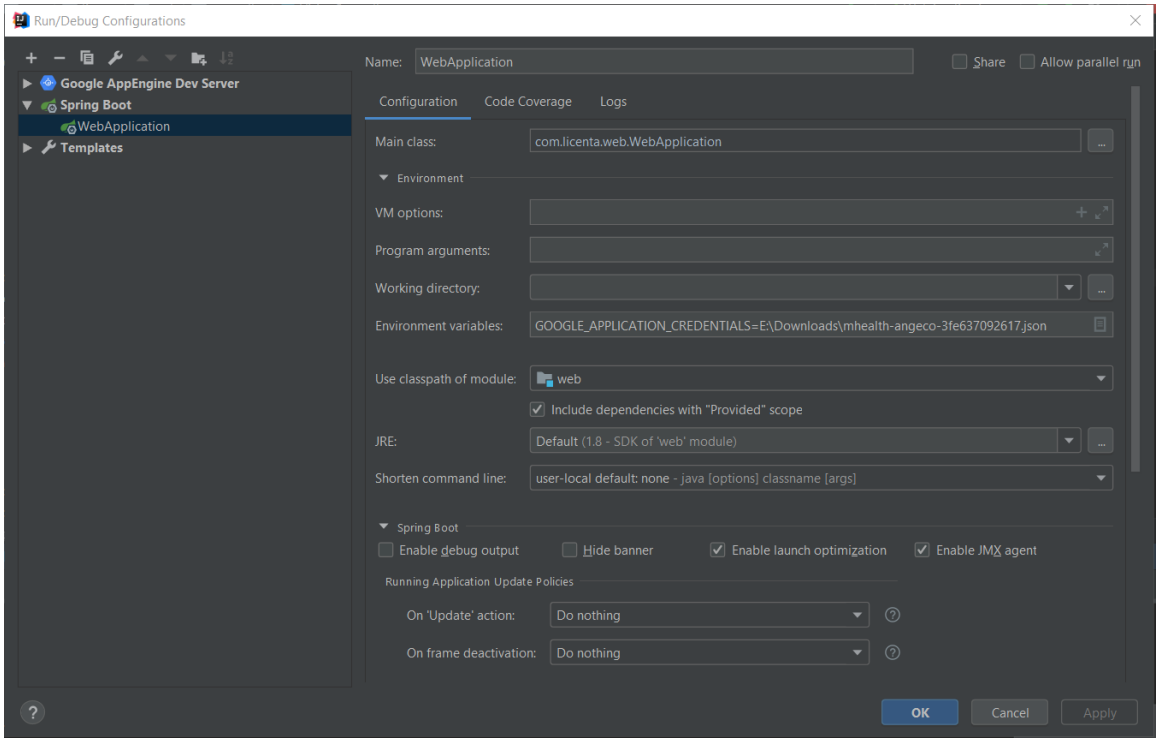
5.2.2. *Proiectul de Cloud și implementarea serviciilor*

Pentru integrarea serviciilor Cloud oferite de Google s-a creat un cont nou, pe care s-au activat serviciile de facturare. Astfel, s-a obținut un credit inițial de $300 pentru acoperirea costurilor din perioada implementării. Figura 5.10 prezintă un exemplu de activare al serviciilor.



Figură 5.10 – Activare Google APIs

Odata activate, pentru a le putea utiliza, este necesara crearea unui Service Account. Acesta se generează sub forma unui JSON și trebuie configurat ca variabilă de mediu în IDE-ul ales, precum se poate observa în figura 5.11.



Figură 5.11 - Setarea variabilei de mediu

La același proiect de *Cloud* s-a legat și baza de date și proiectul *App Engine* necesar pentru realuzarea deploy-ului.

Un prim serviciu integrat a fost **Geocoding API** care să permită transformarea din coordonate în adrese și vice versa. Pentru utilizarea acestui API tot ce trebuie făcut este un apel de genul:

```
GeocodingApi.newRequest(context).latlng(location).await();
```

Unde *location* este obiectul de tip *LatLng* care se dorește transformat în locație, iar contextul este setat anterior folosind cheia generată la activarea serviciului din *consola Google*.

Mână în mână cu acest API funcționează și **Google Maps API**. Folosind aceeași cheie s-a realizat și funcționalitatea de hartă interactivă. La încărcarea paginii respective, se realizează citirea din baza de date a tuturor punctelor de interes și se face marcarea acestora pe hartă folosind câte un *Marker*.

Un alt serviciu activat este **Vision API**. Acesta este folosit pentru procesarea imaginilor preluate din baza de date. Pentru acest lucru, s-a folosit agentul pre antrenat pus la dispoziție prin intermediul API-ului. S-a inițializat un obiect de tipul *CloudVisionTemplate* și s-a apelat metoda analyzeImage având ca parametri imaginea și tipul de procesare *Feature.Type.LABEL_DETECTION*. Rezultatul este returnat sub forma unei liste de etichete descoperite în imagine si un scor (intre 0 si 1) care reprezintă gradul de încredere cu care s-a făcut predicția. Un exemplu de apel este prezentat în Capitolul 6, la secțiunea de testare de acceptanță.

Pentru transformarea din audio în text s-a activat **Speech-to-Text API.** Acesta permite preluarea unei intrări audio de maxim un minut (durata suficientă pentru aplicația MHealth) și returnarea transcrierii acesteia. Mai întâi, trebuie realizată configurarea obiectului de configurare audio după cum urmează:

```
RecognitionConfig config =
        RecognitionConfig.newBuilder()
                .setEncoding(RecognitionConfig.AudioEncoding.AMR_WB)
                .setSampleRateHertz(16000)
                .setLanguageCode("en-US")
                .setEnableAutomaticPunctuation(true)
                .build();
RecognitionAudio audio =
RecognitionAudio.newBuilder().setUri(fileName).build();
```

Provocarea a fost găsirea unui tip de AudioEncoder care să fie suportat și de aplicația mobilă (de unde se înregistrează sunetul și se pune în baza de date) și de API. S-a ales *AMR_WB* deoarece permite o calitate ridicată a sunetului datorită lățimii de bandă mai mari.

Rezultatul este returnat apelând metoda *recognize* având ca parametri fișierul audio și configurația definită anterior.

Această transcriere este afișată pe pagina de detalii a urgențelor și este adăugată pentru procesare alături de descrierea text. Pentru aceste procesări s-a folosit Natural **Language API**. Deoarece acest API nu returna ceva suficient de detaliat / specific pentru aplicația *MHealth*, s-a folosit componenta sa de **AutoML**.

Pentru început, a fost necesară găsirea unui set de date potrivit pentru scopul aplicației. S-au căutat seturi mari de date, disponibile gratuit, dar care să conțină

informație precisă și relevantă pentru *MHealth*. Cele mai bune astfel de seturi de date au fost disponibile pe site-ul Kaggle[46].

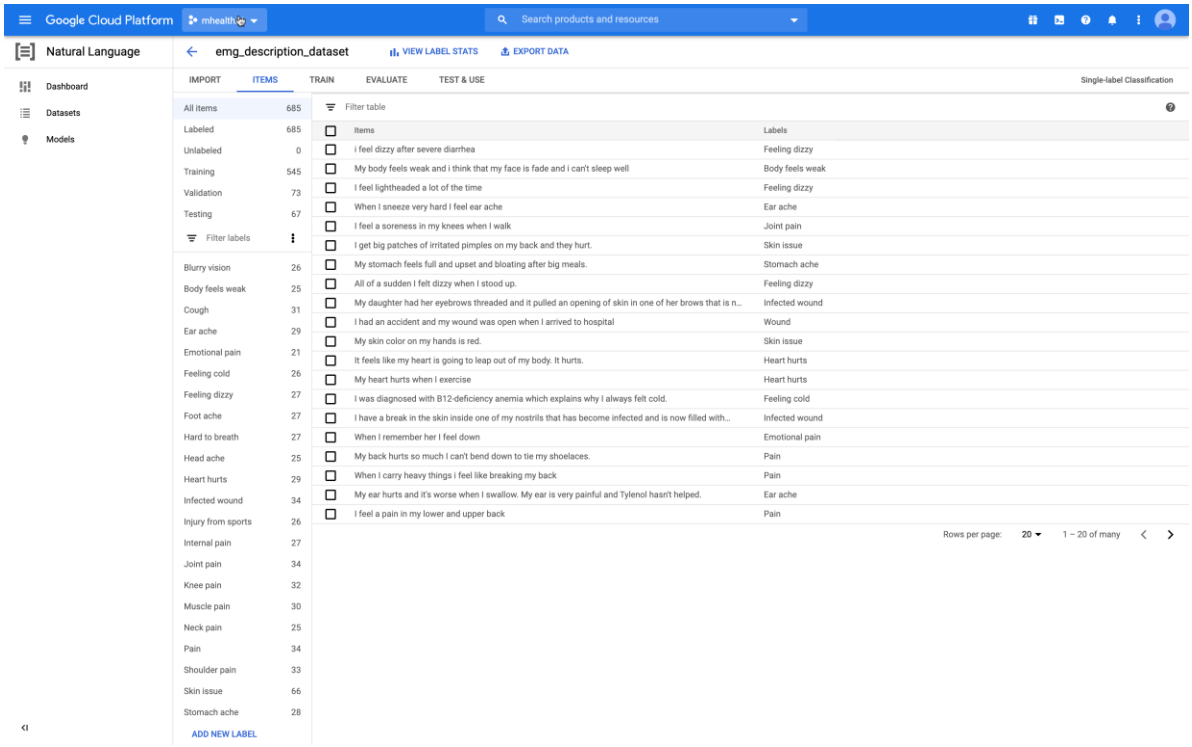
Modelul antrenat face o clasificare de tip **Single Label Classification**, deci preia o intrare și îi asignează o singură etichetă. Așadar, am ales doua seturi de date relevante: Medical Sppech, Transcription and Intent[47] (set de date care conține transcrierile unor înregistrări de discuții între pacienți și doctori) și Medical Transcriptions[48] (care conține descrieri de afecțiuni și diagnosticul). Deoarece seturile de date aveau mai multe variabile după care se făceau predicțiile, am prelucrat datele astfel încât să rămână doar valorile unice și valorile pentru care se poate face o asignare de etichetă doar folosind descrierea, fără alți factori.

Astfel, în consola Google pentru AutoML s-a încărcat pentru antrenat un set de date care conține 685 valori unice și prezice 23 de etichete:

- Blurry Vision
- Body felling weak
- Cough
- Ear ache
- Emotional pain
- Feeling cold
- Feeling dizzy
- Foot ache
- Hard to breath
- Head ache
- Heart hurts
- Infected wound
- Injury from sports
- Internal pain
- Joint pain
- Knee pain
- Muscle pain
- Neck pain
- Pain
- Shoulder pain
- Skin issue
- Stomach ache
- Wound

Detalii și exemple se regăsesc în figura 5.12.

[46] https://www.kaggle.com
[47] https://www.kaggle.com/paultimothymooney/medical-speech-transcription-and-intent#overview-of-recordings.csv
[48] https://www.kaggle.com/tboyle10/medicaltranscriptions#mtsamples.csv



Figură 5.12 – Antrenarea modelului pe setul de date

După efectuarea antrenării prin învățare, testare și validare, modelul a fost *deployed* pentru a putea fi folosit în apeluri REST.

Apelul modelului se face setand id-ul proectului de *Cloud* și al modelului antrenat prin intermediul unui obiect de tip *ModelName*. Apoi se pregătește textul prin încapsularea lui într-un obiect de tip TextSnippet. Cererea este de asemenea împachetată într-un obiect de tip *PredictRequest* care conține modelul și textul dat. În final, acest obiect este folosit ca parametru al metodei *predict* a clientului creat.

Aceste detalii de implementare se regăsesc în figura 5.13, unde se poate observa și adăugarea textului preluat din transcrierea audio la aceasta procesare.

Rezultatele procesărilor audio, foto și text sunt afișate în interfața utilizatorului sub forma unor *grafice*. Pe axa verticală se află valorile de la 0 la 1 ale gradului de încredere cu care s-a făcut prezicerea, iar pe axa orizontală se află etichetele prezise ordonate descrescător după gradul de încredere.

Datele obținute în urma procesării și detaliile introduse de către doctor se transmit prin email utilizatorilor aplicației mobile care marchează participarea la caz. Transmiterea de face cu ajutorul **JavaMailSender** conform figurii 5.14. Mesajele se transmit de la adresa de mail de care aparține și proiectul de *Cloud*, adresă setată în *application.properties* unde este inclusă și parola contului.

```
try (PredictionServiceClient client = PredictionServiceClient.create()) {
    ModelName name = ModelName.of(projectId,  location: "us-central1", modelId);

    TextSnippet textSnippet =
            TextSnippet.newBuilder()
                    .setContent(emergencyDto.getDescription()
                            + " "
                            + speechToTextService.trascribeFromAudio(emergencyDto.getAudio()))
                    .setMimeType("text/plain")
                    .build();
    ExamplePayload payload = ExamplePayload.newBuilder().setTextSnippet(textSnippet).build();
    PredictRequest predictRequest =
            PredictRequest.newBuilder().setName(name.toString()).setPayload(payload).build();

    PredictResponse response = client.predict(predictRequest);

    for (AnnotationPayload annotationPayload : response.getPayloadList()) {
        System.out.format("Predicted class name: %s\n", annotationPayload.getDisplayName());
        System.out.format(
                "Predicted sentiment score: %.2f\n\n",
                annotationPayload.getClassification().getScore());
        if (annotationPayload.getClassification().getScore() > 0.05){
            MLLabelDto mlLabelDto = new MLLabelDto();
            mlLabelDto.setDescription(annotationPayload.getDisplayName());
            mlLabelDto.setScore( annotationPayload.getClassification().getScore());
            result.add(mlLabelDto);
        }
    }
}
```

Figură 5.13 – Apelarea angentului antrenat

```
public void sendEmailFromDoctor(String email, String mailData) throws MailException
{
    SimpleMailMessage mail = new SimpleMailMessage();
    mail.setTo(email);
    mail.setSubject("Emergency Info");
    mail.setText("This is some helpful information for your intervention. \n" + mailData);
    System.out.println(mailData);
    javaMailSender.send(mail);
}
```

Figură 5.14 – Transmiterea de email

5.2.3. Deploy pe Cloud

Pentru deploy-ul pe Google Cloud s-a creat un proiect de tip App Engine. Acesta a fost setat ca server principal în zona europe-west-3, server localizat în Germania, care poate deservi cu succes cererile, situându-se în zona centrală a Europei.

O aplicație App Engine poate să aibă mai multe servicii și mai multe versiuni ale fiecărui serviciu. Un serviciu care nu poate lipsi este cel default, serviciu în care se face deploy dacă nu se menționează un altul.

Așadar, s-a decis utilizarea serviciului default pentru backend și utilizarea unui alt serviciu pentru frontend.

Pentru realizarea acestor operații folosind terminalul, s-a instalat și inițializat Google Cloud SDK Shell[49], folosind contul de gmail cu care s-a creat și proiectul de Cloud.

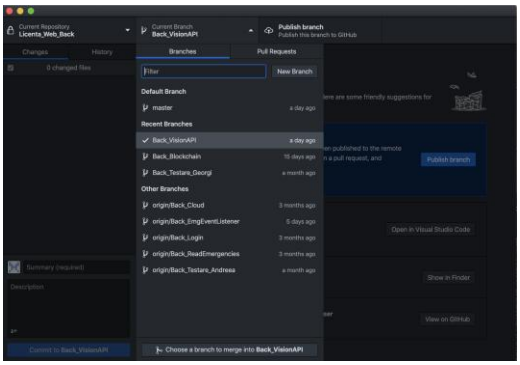
Pentru a se face deploy la backend, acesta trebuie împachetat într-un jar (folosind comanda **./mvnw package**), jar care ajunge în proiectul App Engine. Deploy-ul propriu zis se face prin comanda **gcoud app deploy** împreună cu **numele jar-ului**. Pentru a deschide aplicația în browser se poate utiliza comanda **gcloud app browse**.

În ceea ce privește partea de frontend, s-a adăugat o clasă .yaml, unde s-a specificat tipul de mediu (flexibil – pentru ca scalarea să fie realizată în mod automat) și numele serviciului (app – identificator unic al serviciului). Din terminal s-a rulat apoi doar comanda de **gcloud app deploy**.

5.2.4. Management de proiect

Sistemul MHealth este dezvoltat în colaborare cu Andreea Ifrim și Cosmin-Ștefan Dascălu, având două module (unul web și unul mobil) a câte două componente (cel de gestionare al urgențelor și cel personal). Astfel că, a fost necesară introducerea unor mecanisme prin care funcționalitățile implementate individual să ajungă împreună, ca un întreg.

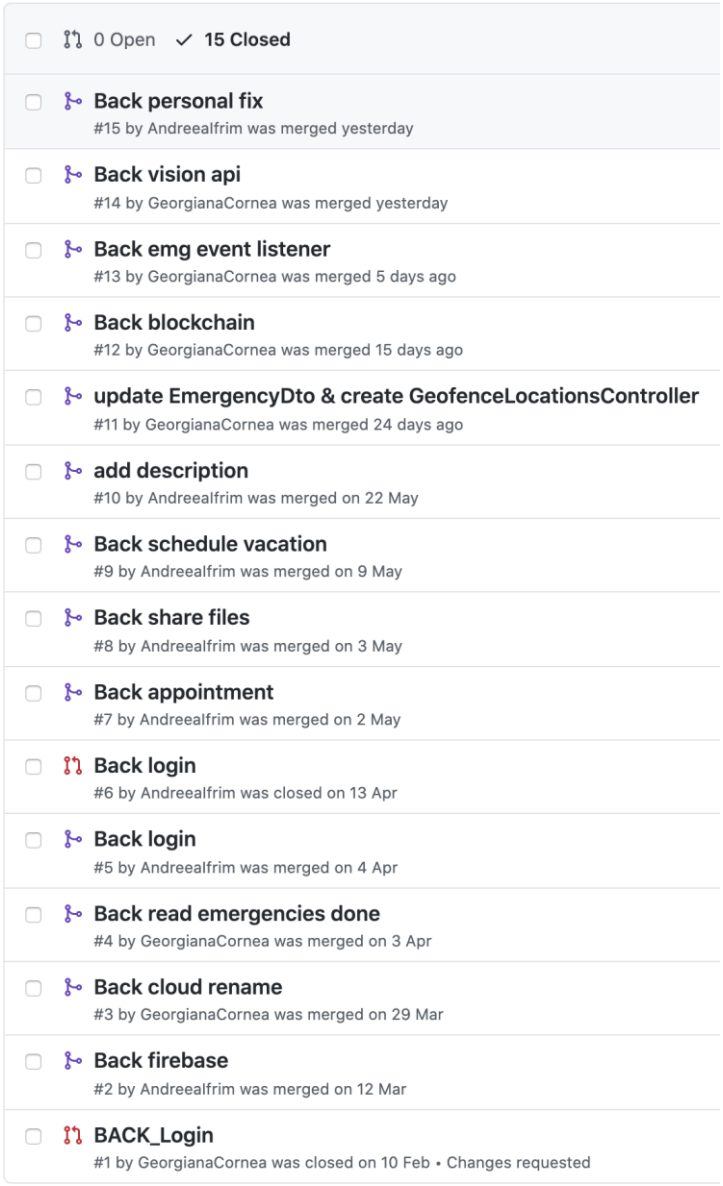
Cel mai usor mod de a realiza asta, este utilizarea unei platforme precum **Github** (împreuna cu Github Desktop). Acolo, s-a creat un *repository* pentru aplicația mobilă și două pentru aplicația web (unul pentru backend și altul pentru frontend), repository-uri la care au avut acces toți membrii echipei. Progresul este ușor de urmărit datorită modului de lucru pe *branch*-uri, unde fiecare funcționalitate este implementată pe o altă ramură, iar în ramura principală (*master*) se păstrează mereu ultima versiune stabilă, perfect funcțională, precum este vizibil în figura 5.15.



Figură 5.15 – Github Desktop – ramuri

[49] https://cloud.google.com/sdk/docs/quickstart-windows

Odată implementată o funcționalitate, se făcea un *pull request* pentru a face merge branch-ului respectiv în master. Ceilalți doi membri verificau funcționalitatea respectivă și solicitau schimbările necesare sau aprobau request-ul, iar codul ajungea în master. Figura 5.16 prezintă toate aceste pull request-uri de pe modulul web, componenta de backend.



Figură 5.16 – Github – pull requests

Tot pentru a putea urmări progresul mai ușor, atât pe partea de implementare cât și pe cea de documentare și alegere a tehnologiilor, s-a conceput o fișă de evoluție comună. După implementarea fiecărei funcționalități și / sau documentarea unor aspecte, aceasta era actualizată. Fișa de evoluție este prezentată în Anexa 1.

Capitolul 6. Testare și Validare

Testarea unei aplicații este un pas ce nu ar trebui să lipsească indiferent de tipul sau natura unei aplicații, cu atât mai mult în cadrul unei aplicații cu caracter medical precum sistemul *MHealth*.

Principala dificultate întâlnită la testare este datorată faptului că se folosește un proiect Google Cloud și servicii oferite de aceeași platformă. Astfel, pentru unele cazuri de test (Vision API, Geocoding API, Speech-to-Text API), s-au privit rezultatele din perspectiva unui client (pentru că se folosea un servicu oferit de Google, fără să se cunoască în detaliu modul de implementare al acelor metode) și s-a realizat doar „acceptance testing". De asemenea, componentele de Machine Learning care au folosit agenți antrenați de către mine (Natural Language Auto ML) au fost testate prin intermediul consolei și al UI-ului pus la dispoziție de către Google Cloud.

Așadar, s-au ales pentru testarea din perioada implementării doar cazuri de test care au ținut complet de metode nou implementate, unde puteam anticipa rezultatul așteptat.

6.1. Cazuri și metode de testare

Testarea funcțională presupune verificarea cerințelor funcționale și se bazează în principal pe teste de accesibilitate și de funcționare a unor cazuri de bază, esențiale pentru sistem.

În cadrul acestui tip de testare se pot aminti:

- Unit testing
- Integration testing
- System testing
- Acceptance testing
- Regression testing

Unit testing este elementul de *bază* al testării din timpul implementării. Testele sunt *scurte*, pe cazuri mici și urmăresc funcționalități de bază. Cazurile fiind mici, erorile sunt ușor de detectat.

Fiind o aplicație care are ca scop principal *afișarea* unor date existente în baza de date și a unor *rezultate* de prelucrări, s-a ales testarea unor metode de tip *get*. Pentru asta, am utilizat teste *Junit* care au fost posibile prin introducerea dependințelor de spring-boot-starter-test si junit-jupiter-api, versiunea 5.

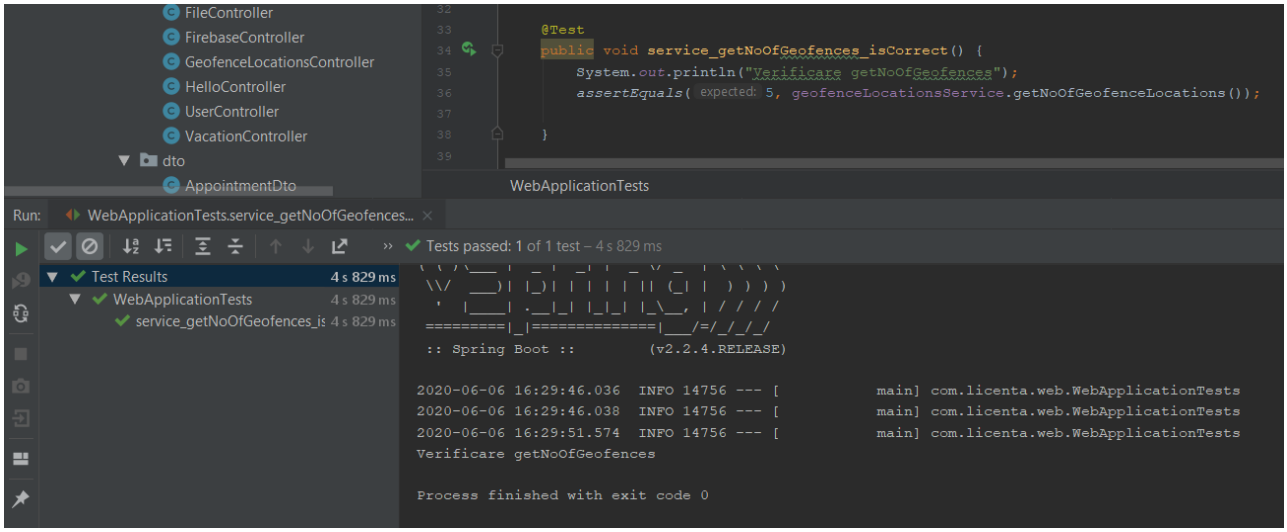
```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <version>2.1.6.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.3.2</version>
    <scope>test</scope>
</dependency>
```

Figură 6.1 - Dependițe necesare

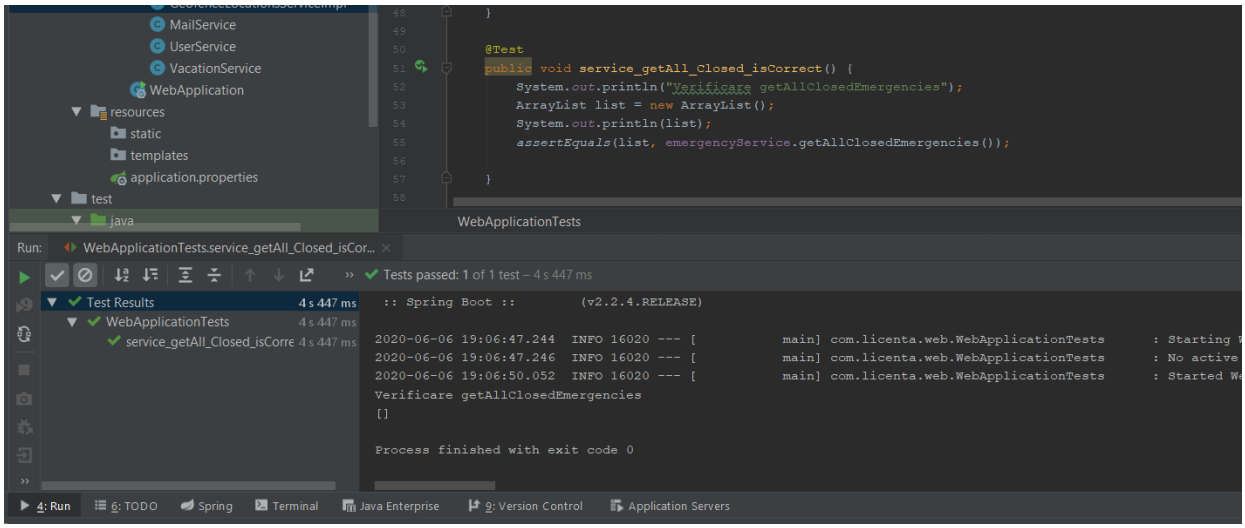
Cazuri de test:

- Afișarea numărului de geofence-uri din baza de date: am verificat câte sunt în baza de date la momentul rulării testului și am asertat acea valoare (5) la rezultatul metodei care se ocupă de asta; rezultatul se regăsește în figura 6.2.



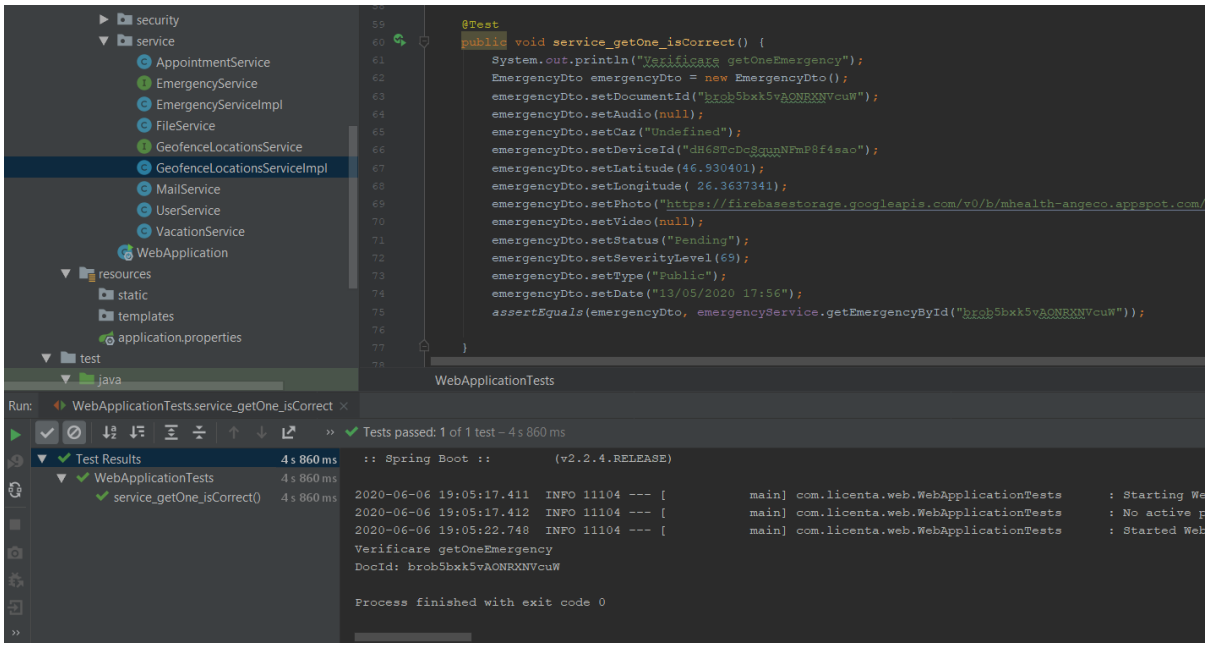
Figură 6.2 - Rezultatul testării

- Afișarea urgențelor cu un anumit status (closed): am verificat că metoda nu returnează niciun element, în conformitate cu ce se regăsea în baza de date; rezultatul se regăsește în figura 6.3.



Figură 6.3 - Rezultatul testării

- Afișarea unei anumite urgențe: am creat în cadrul testului o urgență care știam că se regăsește în baza de date și am asertat-o rezultatului metodei care caută urgența după id; rezultatul se regăsește în figura 6.4.



Figură 6.4 - Rezultatul testării

S-a considerat un test ca încheiat cu *succes* când rezultatul returnat a corespuns cu ce se găsea în baza de date.

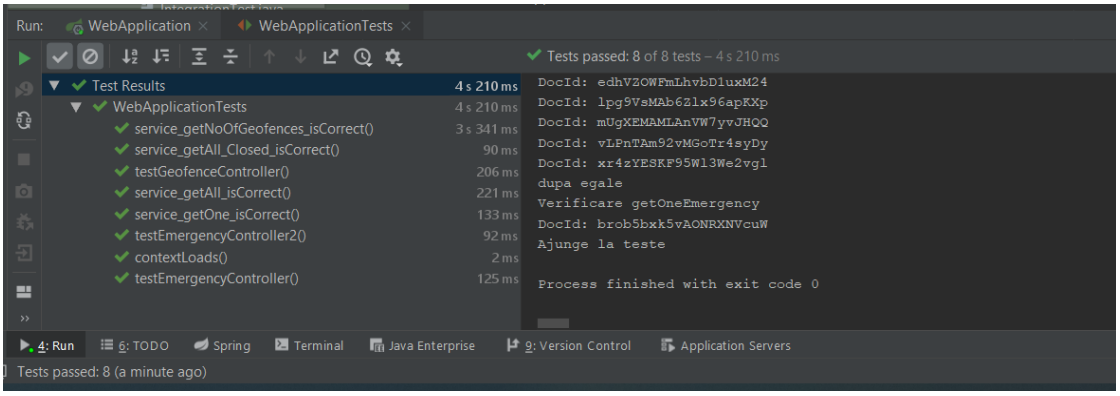
Când vine vorba de un sistem format din mai multe layere și mai multe module **Integration testing** joacă un rol esențial. Trecerea de acest test ne indică faptul că bucățile de cod deja testate funcționează bine și *împreună* și nu apar inconsistențe la comunicarea între nivele sau module. Datorită faptului că aplicația folosește servicii REST, s-a ales să se testeze *controllerele*. Figurile 6.5 si 6.6 prezintă aceste teste și rezultatele lor.

```
@Test
public void testGeofenceController() {
    ResponseEntity<String> responseEntity = this.restTemplate
            .getForEntity( url: "http://localhost:8080/getNoOfGeofenceLocations", String.class);
    System.out.println( responseEntity.getStatusCodeValue());
    assertEquals( expected: 200, responseEntity.getStatusCodeValue());
}

@Test
public void testEmergencyController() {
    ResponseEntity<String> responseEntity = this.restTemplate
            .getForEntity( url: "http://localhost:8080/getAllEmergencies", String.class);
    assertEquals( expected: 200, responseEntity.getStatusCodeValue());
}

@Test
public void testEmergencyController2() {
    ResponseEntity<String> responseEntity = this.restTemplate
            .getForEntity( url: "http://localhost:8080/getEmergencyById/brob5bxk5vAONRXNVcuW", String.class);
    assertEquals( expected: 200, responseEntity.getStatusCodeValue());
}
```

Figură 6.5 - Testele aplicate



Figură 6.6 - Rezultatele testelor

System testing este o testare de tip *black-box* care verifică funcționarea sistemului ca întreg. În mod ideal, se aleg cazuri de test care să acopere cât mai mult din aplicație. Astfel, s-a discutat la nivel de echipă și s-a ales un caz de test potrivit care să acopere cât mai multe *interacțiuni* între aplicații și module.

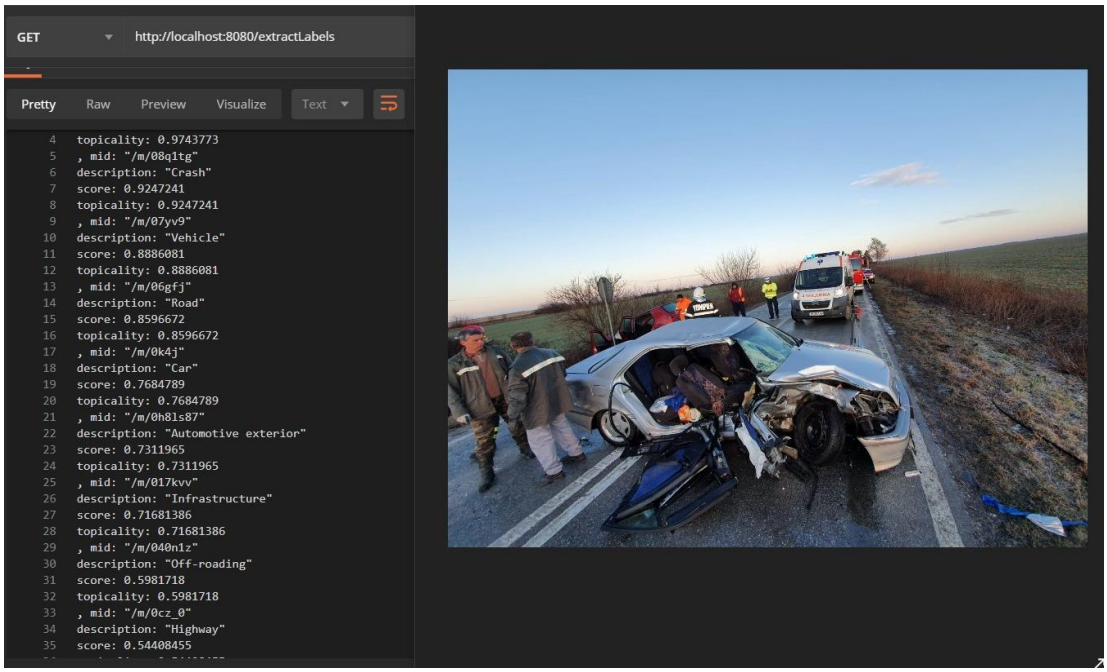
Exemplu de caz:

Un utilizator obișnuit al aplicației mobile, victimă sau martor a unui incident se decide să trimită un raport însoțit de imagine și descriere text. Urgența, împreună cu locația sa ajung să fie prelucrate de aplicația web. Medicul vede rezultatele prelucrării și adaugă pașii ce trebuie urmați de către persoanele de la fața locului. Simultan raportării urgenței, un utilizator cu pregătire medicală, aflat în vecinătatea urgenței raportate, manifestă interes pentru a oferi suport. Astfel, primește un mail cu datele necesare și locația la care trebuie să se prezinte. În acest timp, alți utilizatori comunică cu medici, prin intermediul componentei de chat.

Cazul de test s-a încheiat cu *succes*, dovedind astfel că modulele și componentele interacționează fără probleme.
Totodată, testele de sistem validează și *comunicarea* aplicației între toate straturile acesteia (de la frontend, prin backend, către baza de date și invers).
Aplicația este *găzduită* pe Cloud, iar aceste teste au funcționat și furnizează aceleași rezultate atât înainte de deploy, cât și după.

Acceptance testing se referă la o testare din perspectiva *clientului*. Este tot o testare de tip *black-box* care verifică dacă aplicația respectă toate cerințele clientului.
Deși aplicația este găzduită pe Cloud, încă nu are utilizatori reali, deci nu poate fi evaluată de către persoane, astfel că un acceptance test în adevăratul sens al cuvântului nu poate fi realizat.

Tot din această perspectivă am testat si unele dintre API-urile integrate, precum *Vision API*. Deoarece o testare la nivel de unitate nu a putut fi realizată necunoscând modul de implementare al metodelor, am privit aceste API-uri din perspectiva unui client și am decis dacă sunt adecvate folosirii. De exemplu, figura 6.7 prezintă ce returnează API-ul pentru imaginea respectivă.



Figură 6.7 - Răspuns Vision API

În ceea ce privește agenții antrenați, testarea se face în mod automat, păstrându-se valori din setul de date cu care se face antrenarea. După aflarea preciziei cu care agentul face etichetări, s-a decis daca mai trebuie adăugate noi seturi de date pentru imbunătățirea acurateții sau daca sistemul este suficient de precis. Totodată, este oferită posibilitatea evaluării modelului din interfață.

6.2. Validarea sistemului

Sistemul a trecut toate testele automate și manuale realizate, fiind astfel realizată validarea sa.

Totodată, în cazul în care apar erori, acestea vor fi raportate în consola Google, la secțiunea de *Error Reporting,* împreună cu data și ora la care au apărut. De asemenea, se poate analiza și graficul de solicitări pe fiecare serviciu și timpii de acces și de răspuns de la servere.

Astfel, la momentul deploy-ului, aplicația funcționează în parametri optimi, nefiind întâlnite erori sau latențe, iar natura aplicației permite urmărirea și depistarea unor noi probleme într-un mod facil.

Capitolul 7. Manual de Instalare si Utilizare

În acesta secțiune se prezinta modul corect de instalare si utilizare a aplicației Web. Se descriu resursele necesare pentru instalarea întregului sistem cu accent pe componenta Web si se realizează manualul de utilizare pentru doctor.

7.1. Resurse necesare pentru instalare

Deși modulul web a fost realizat utilizând tehnologii si servicii noi, singura resursa de care un utilizator are nevoie este un dispozitiv cu conexiune la internet.

Toate operațiile se realizează pe *Cloud*, astfel că nu este necesar un server local care sa suporte toate aceste servicii.

Totodată, design-ul aplicației este unul de tip *responsive* care se adaptează în funcție de mărimea ecranului, deci utilizatorul poate accesa și utiliza cu aceeași ușurință modulul independent de mărimea ecranului.

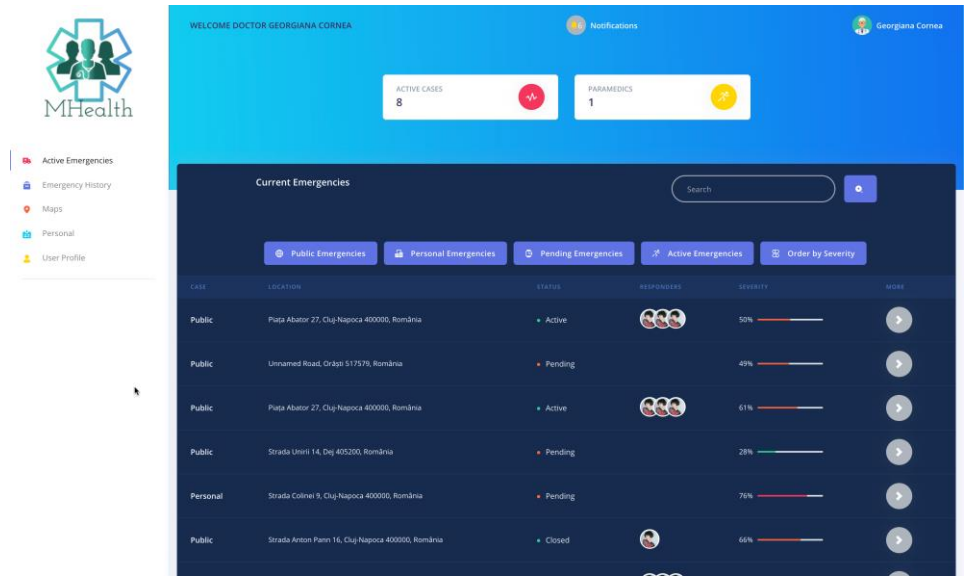
7.2. Manual de utilizare pentru doctor

Pentru început, medicul trebuie sa acceseze sistemul prin link-ul corespunzător.

Folosind pagina din figura 7.1, utilizatorul se autentifică in sistem folosidu-și credențialele: username si parolă.

Figură 7.1 – Pagina de Login

Apoi, este redirecționat către pagina sa principala, precum arata figura 7.2

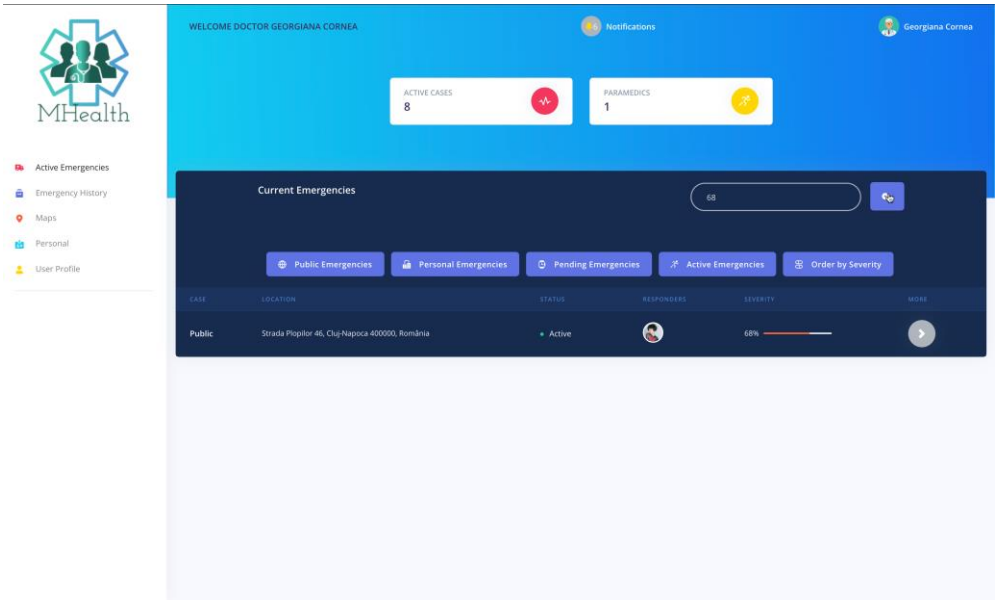


Figură 7.2 - Pagina principală a doctorului

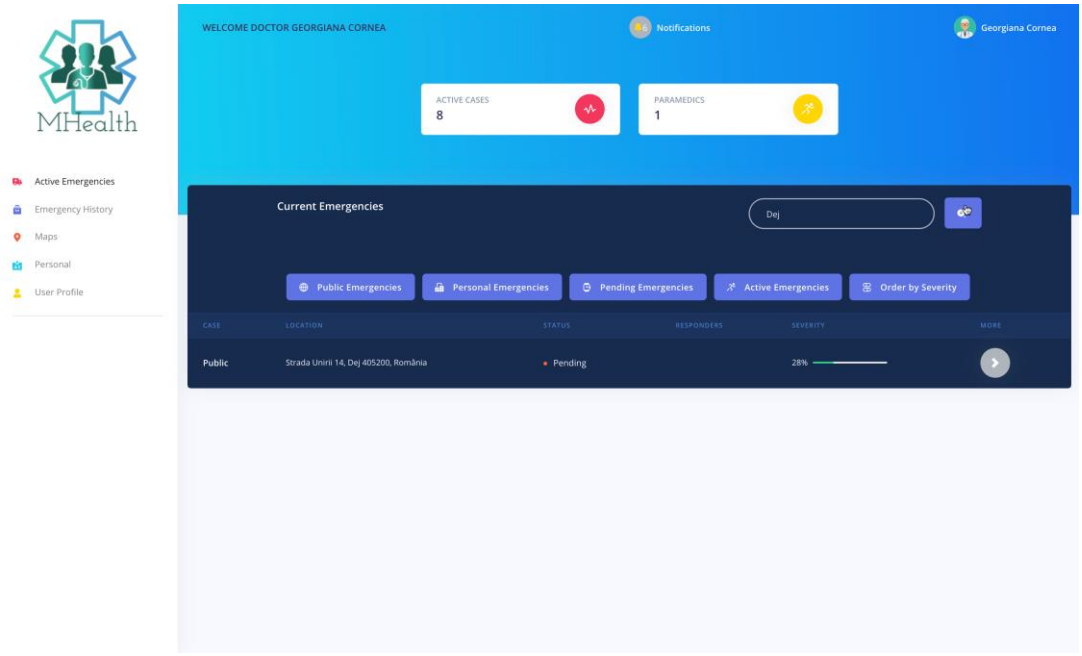
De pe această pagină, medicul poate realiza toate acțiunile legate de gestionarea urgențelor.

În partea laterală este meniul principal, din care doctorul poate naviga spre restul paginilor sau spre modulul *Personal*.

In partea superioara a paginii se regăsesc date statistice precum numărul de cazuri active și numărul de utilizatori cu pregătire medicala. Mai jos, sunt prezentate urgențele sub forma unui tabel. Prin câmpul de căutare se poate realiza o căutare după gradul de severitate (dacă se introduce un număr întreg), conform figurii 7.3 sau o căutare după locație (dacă se introduce numele unei locații), conform figurii 7.4.

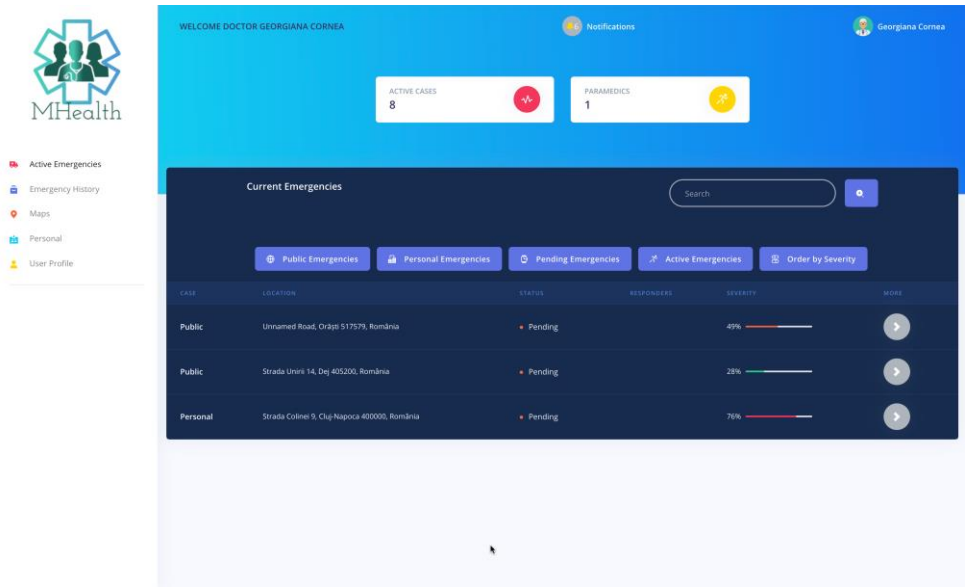


Figură 7.3 - Căutare după gradul de severitate



Figură 7.4 - Căutare după locație

Folosind butoanele, doctorul poate realiza o sortare a urgențelor după criteriile prezentate: Public Emergencies (urgențele cu caracter public), Personal Emergencies (urgențe raportate o singură dată, cu caracter personal), Pending Emergecies (urgnțele care sunt raportate, dar niciun utilizator mobil nu a marcat participarea), Active Emergencies (cazurile care au deja *responderi* asignați), Order By Severity (ordonare descrescătoare a urgențelor după nivelul severității). Un exemplu de sortare este prezentat în figura 7.5.



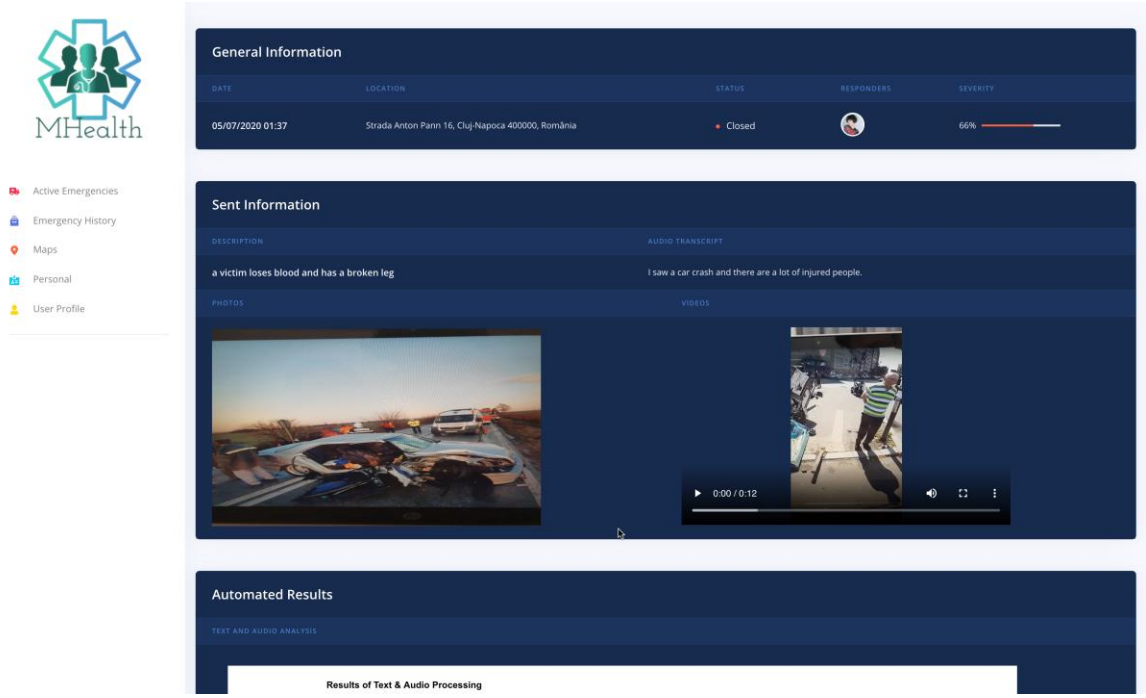
Figură 7.5 - Afișarea urgențelor cu status de Pending

Pentru mai multe detalii legate de urgențe, pentru afișarea rezultatelor procesării automate și pentru introducerea de sfaturi suplimentare, doctorul trebuie să acționeze butonul de *More*.

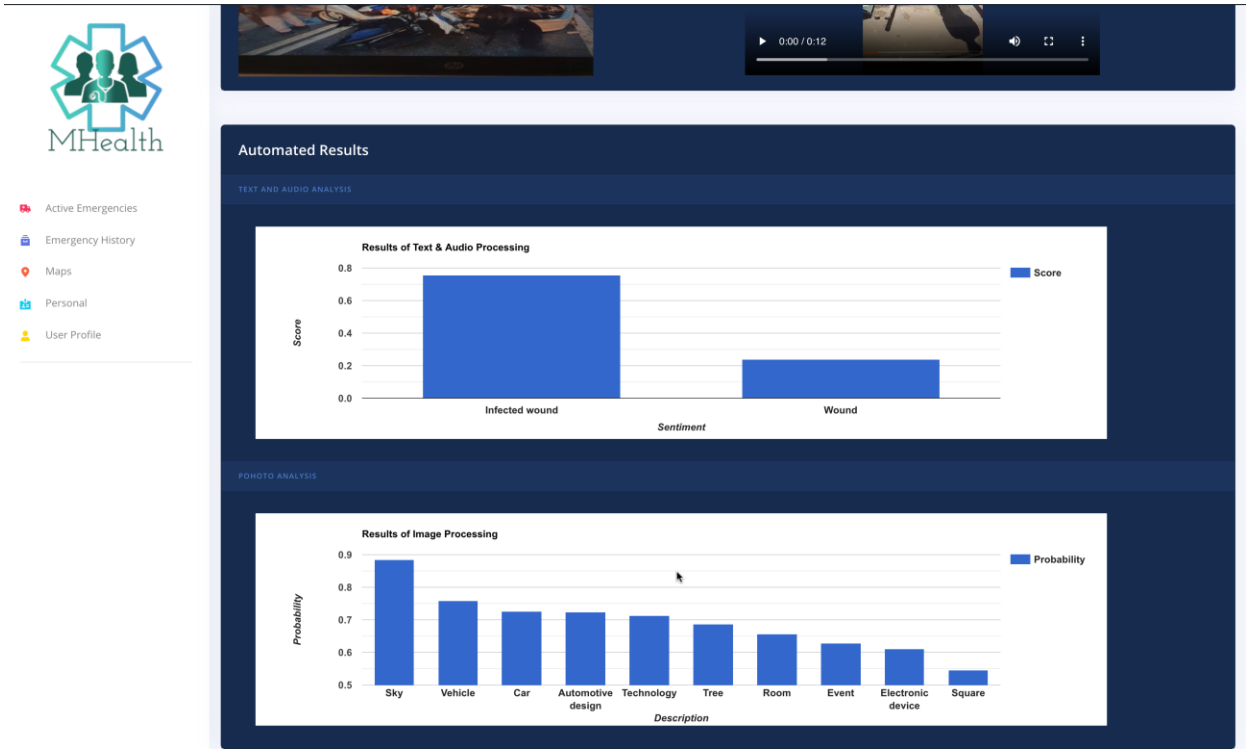
Pagina deschisă este împărțită în patru secțiuni principale:

- **General Information** care prezintă datele generale despre caz, prezentate și în previzualizarea acestora.
- **Sent Information** unde sunt afișate toate datele transmise de către utilizator: descriere text, transcrierea înregistrării audio, imaginile și secvențele video.
- **Automated Results** în care sunt afișate rezultatele procesărilor sub forma de grafice.
- **Your Advice** unde doctorul poate să scrie propriile observații legate de caz.

Aceste detalii sunt prezentate in figurile 7.6, 7.7 și 7.8.



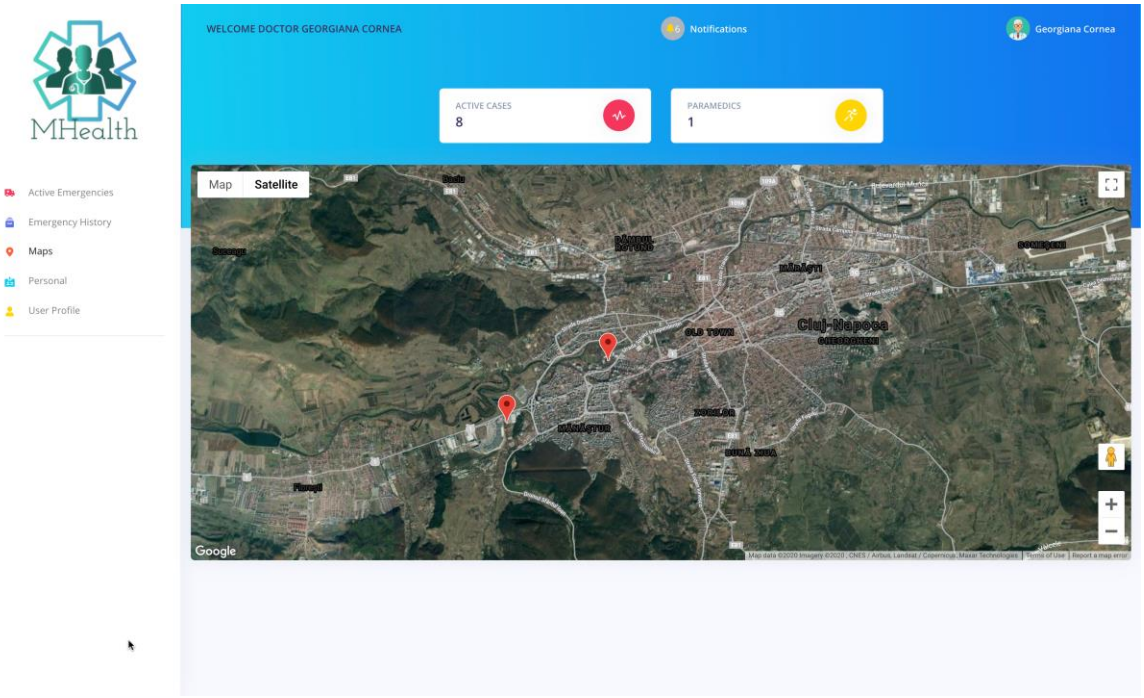
Figură 7.6 - Secțiunile General Information și Sent Information



Figură 7.7 - Secțiunea Automated Results

Figură 7.8 - Secțiunea Your Advice

Pentru vizualizarea hărții interactive, se acționează butonul de Maps din meniul lateral. Pagina afișată se regăsește in figura 7.9. De asemenea, se poate schimba modul de vizualizare al hărții.



Figură 7.9 - Pagina de Maps

Capitolul 8. Concluzii

În acest capitol se prezintă concluziile lucrării. Pentru început se descriu contribuțiile personale și se prezintă rezultatele obținute, continuându-se cu o lista de posibile dezvoltări ulterioare.

8.1. Contribuții personale

Contribuția personală constă în mare parte în alegerea temei, a modului de implementare și a setului de servicii disponibile care pot deservi scopul aplicației.

Ideea a pornit observând numărul mare de incidente care are loc și faptul că echipajele de intervenție sunt tot mai aglomerate, timpul de răspuns devenind uneori foarte mare. Există numeroase persoane care ar dori sau ar putea ajuta în situații limită, dar care fie nu au expertiza necesară, fie nu sunt martori direcți ai incidentului. Astfel, un sistem care să poată notifica persoanele specializate și să trimită informații ajutătoare, poate face diferența dintre viață și moarte.

Necesitatea procesării automate a datelor vine tocmai în completarea ideii că timpul este crucial. Până când utilizatorul vede notificarea și se îndreaptă spre locul incidentului datele sunt deja procesate și un răspuns detaliat este pregătit pentru a putea informa corespunzător utilizatorul.

Totodată, faptul că întreg sistemul este fie mobil, fie găzduit pe Cloud face interacțiunea cu un număr maxim de utilizatori posibilă. Rularea pe Cloud nu condiționează sau limitează utilizatorii în vreun mod, permițând folosirea aplicației indiferent de componentele hardware pe care le au aceștia la dispoziție.

8.2. Rezultate obținute

În final, s-a obținut un sistem complex, capabil de gestiunea unor situații medicale atât de natură personală cât și cu caracter urgent. Utilizarea serviciilor Cloud face sistemul rapid și de încredere, iar împărțirea funcționalităților între componente și module face interacțiunea utilizatorilor cu aplicația mult mai ușoară și prietenoasă.

Astfel, sistemul nu își propune sa înlocuiască, ci să sprijine și să completeze serviciile medicale existente deja, facilitând conectarea persoanelor cu medicii și scoțând maximul din orice secundă în care cineva are nevoie de ajutor.

8.3. Dezvoltări ulterioare

Deși sistemul îndeplinește toate funcționalitățile gândite pentru conceptul prezentat, există un set de funcționalități ce ar putea extinde platforma pentru o și mai bună asistență în situații limită.

8.3.1. Dezvoltarea ulterioară a sistemului MHealth

Sistemul MHealth, ca întreg, poate integra cu ușurință și alte module datorită arhitecturii propuse.

Astfel că, o posibilă dezvoltare ulterioară ar fi integrarea unui modul care să gestioneze în mod explicit urgențele naturale precum cutremure, inundații, incendii

naturale etc.; acestea ar fi raportate similar din aplicația mobilă și prelucrate în cea web pentru a putea oferi asistență sau pentru a veni în ajutorul autorităților și a locatarilor din aceeași arie.

O alta funcționalitate de care ar putea beneficia întreg sistemul este introducerea unui asistent vocal. Acesta ar putea fi de mare folos pentru persoane cu dizabilități care doresc să folosească aplicația sau pentru a ușura interacțiunea copiilor cu interfața în situații limită. Pe partea de web, un chatbot ar putea oferi răspunsuri la întrebări sau seta remindere pentru doctori.

8.3.2. Dezvoltarea ulterioară a modulului Web

În ceea ce privește componenta web de gestiune a urgențelor, există de asemenea, o listă de posibile dezvoltări ulterioare:

- **antrenarea sistemului cu un număr mult mai mare de cazuri pentru a se putea transmite mail-uri cu detalii chiar înainte de aprobarea medicului**

Antrenarea s-ar putea face adăugând predicțiile corecte ale sistemului ca date de antrenament sau prin verificarea constantă a site-urilor de specialitate pentru găsirea unui nou set de date care sa completeze datele existente sau găsirea unei actualizări pentru setul de date existent.

- **primirea de notificări asincrone**

Primirea de notificări pe un canal care sa fie deschis in permanenta (printr-un socket) pentru ca doctorul sa fie notificat mereu, nu doar la logarea in sistem.

- **prelucrarea mai amănunțita a secvențelor video și returnarea unui „rezumat” al acestora**

Secvențele video ar putea fi prelucrate cadru cu cadru pentru a putea estima modul in care situația evoluează. Totodată se pot adauga subtitrări.

- **prelucrarea urgențelor din același cluster pentru eliminarea duplicatelor**

Se poate realiza prin verificarea imaginilor si a descrierilor pentru a elimina informațiile redundante si care nu aduc o noua valoare.

- **sesiuni live între doctor și persoana de la fața locului**

Aceasta îmbunătățire ar putea fi de folos când manevrele executate ar trebui supravegheata sau când explicațiile oferite de doctor ar fi mult mai ușor de înțeles oferite pas cu pas. Pentru integrarea unui astfel de modul se pot utiliza librarii externe precum WebRTC[50].

- **realizarea funcționalităților pentru suportul mai multor tipuri de utilizatori web**

Utilizatorii obișnuiți ar putea utiliza un modul web pentru raportarea urgentelor de tip personal.

[50] https://webrtc.org

Bibliografie

[1] S. B. Edillo, P. J. E. Garrote, L. C. C. Domingo, A. G. Malapit și B. S. Fabito, „A mobile based emergency reporting application for the Philippine National Police Emergency Hotline 911: A case for the development of i911," *International Conference on Mobile Computing and Ubiquitous Network (ICMU),* Toyama, 2017.
DOI: 10.23919/ICMU.2017.8330110
Available at: https://ieeexplore.ieee.org/document/8330110/authors#authors

[2] B. Gómez și C. Juiz, „Emergency Web App for Accessing the Medical Emergency Services," în *eTELEMED 2014: The Sixth International Conference on eHealth, Telemedicine and Social Medicine,* Palma de Mallorca, 2014.
ISBN: 978-1-61208-327-8
Available at:
https://www.researchgate.net/publication/311706039_Emergency_Web_App_for_Accessing_the_Medical_Emergency_Services

[3] M. Antal, C. Pop, D. Moldovan, T. Petrican, C. Stan, I. Salomie, T. Cioara și I. Anghel, „Distributed Systems – Laboratory Guide", Cluj-Napoca: *UTPRESS Cluj-Napoca,* 2018.
ISBN: 978-606-737-329-5
Available at: https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/329-5.pdf

[4] A. A. Monrat, „A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities," *IEEE Access,* 2019.
DOI: 10.1109/ACCESS.2019.2936094
Available at: https://ieeexplore.ieee.org/document/8805074

[5] V. Greavu-Șerban, „Cloud Computing - Caracteristici și modele", București: *Editura ASE,* 2015.
ISBN: 978-606-505-982-5
Available at:
https://www.researchgate.net/publication/289952478_Cloud_Computing_Caracteristici_si_Modele

[6] L. DeNardis, „E-health Standards and Interoperability," *ITU-T Technology Watch,* 2012.
Available at:
https://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000170001PDFE.pdf

[7] M. Eichelberg, J. Riesmeier, T. Aden, A. Dogac și G. Laleci, „Electronic Health Record Standards - A Brief Overview," *Research Gate,* 2006.
DOI: 10.1109/ITICT.2006.358222
Available at:
https://www.researchgate.net/publication/267975385_Electronic_Health_Record_Standards_-_A_Brief_Overview

[8] E. Tîrziu și M. Gheorghe-Moisii, „Caracteristici de calitate ale aplicațiilor eHealth," *Romanian Journal of Information Technology and Automatic Control,* 2018.
Available at: https://rria.ici.ro/wp-content/uploads/2018/07/art.3-Tarziu-Moisii-2018.pdf

[9] M. Ding, D. C. Nguyen, A. Seneviratne și P. N. Pathirana, „Blockchain for Secure EHRs Sharing of Mobile Cloud based E-health Systems," *Research Gate,* 2019.
DOI: 10.1109/ACCESS.2019.2917555
Available at:
https://www.researchgate.net/publication/333179363_Blockchain_for_Secure_EHRs_Sharing_of_Mobile_Cloud_based_E-health_Systems

[10] J. P. Dias, L. Reis, H. Sereno Ferreira și A. Martins, „Blockchain for Access Control in e-Health Scenarios," *Research Gate,* 2018.
Available at:
https://www.researchgate.net/publication/325483874_Blockchain_for_Access_Control_in_e-Health_Scenarios

[11] A. Botha, M. Weiss și M. Herselman, „Towards a Taxonomy of mHealth," *Research Gate,* 2018.
DOI: 10.1109/ICABCD.2018.8465427
Available at: https://ieeexplore.ieee.org/document/8465427

Anexa 1 – Fișă de evoluție

Data	Membru	Feature
01.11.2019	-	Emiterea temei
05.11.2019	-	Documentare inițială în legătură cu contextul temei
12.12.2019	-	Analiza funcționalități sistem posibile
05.01.2020	-	Împărțirea aplicației în componente și module
16.02.2020	-	Crearea proiectelor inițiale (Spring, React, Android)
19.02.2020	-	Implementarea interfeței inițiale a aplicației web și mobile
20.02.2020	-	Setarea resurselor și uneltelor necesare pentru managementul aplicației (GitHub + GitHubDesktop + Google Drive)
01.03.2020	Andreea	Integrarea temei pe frontend
02.03.2020	Georgiana	Crearea proiectului Google Cloud
03.03.2020	Cosmin	Integrare servicii Firebase în proiect
05.03.2020	Andreea	Login - implementare și legatura backend-frontend
07.03.2020	Cosmin	Login în aplicația Android folosind senzorul de amprentă
12.03.2020	Cosmin	Funcțiile de înregistrare video/audio, realizare fotografie în aplicația Android
14.03.2020	Georgiana	Deploy backend pe Cloud
16.03.2020 18.03.2020	-	Realizarea capitolului 3
20.03.2020	-	Schema bazei de date (draft)
21.03.2020	Georgiana	Integrare Firebase la aplicația de pe Cloud
22.03.2020	Andreea	Adăugare funcționalități la Login pe Web (create account și forget password)
27.03.2020	Georgiana	Deploy frontend pe Cloud

28.03.2020	Andreea	Create account backend și legatura cu frontend
29.03.2020	Cosmin	Completare funcționalitate “Report Emergency” (locație, descriere text, nivel de severitate), stocare foto/video/audio pe firebase cloud storage, salvare urgență în firebase cloud firestore
29.03.2020	-	Actualizarea schemei bazei de date
31.03.2020	Andreea	Reset password cu trimitere de email și cod de securitate + redirecționare pentru confirmare parola nouă (back + front)
01.04.2020	Georgiana	Funcții backend pentru citirea urgențelor din baza de date; Redeploy backend
02.04.2020	Georgiana	Enable Geocoding API pentru transformarea din coordonate in adresa exactă (frontend)
04.04.2020	Cosmin	Folosire ML Kit pentru traducerea textului folosit ca descriere pentru urgență
05.04.2020	-	Realizare capitolelor 1 și 2
07.04.2020	Cosmin	Realizare creare cont
12.04.2020	Cosmin	Folosire ML Kit pentru preluarea datelor de pe poza cu buletinul
17.04.2020	Andreea	Implementare frontend și backend adăugare de appointment
18.04.2020	-	Propunere Cuprins
19.04.2020	-	Actualizare TF și adăugare NF
20.04.2020	-	Project Management
22.04.2020	Georgiana	Integrare Google Maps
23.04.2020	-	Diagrame UseCase
24.04.2020	-	Diagrama Conceptuală a sistemului
25.04.2020	-	Flowcharts
26.04.2020	Cosmin	Informare despre structura CI

		https://en.wikipedia.org/wiki/National_identity_cards_in_the_European_Economic_Area
27.04.2020	-	Informare EHR Standards
28.04.2020	-	Subfoldere LucrareLicenta
29.04.2020	Andreea	View, delete, update appointment backend și frontend
01.05.2020	Georgiana	Găsire set de date și antrenare + deploy model Language AutoML pentru procesarea textului
03.05.2020	Cosmin	Gestionare servicii foreground cu event listeners pentru tabelul de geofences
06.05.2020	Andreea	Implementare funcționalitate share files pe partea de web
07.05.2020	Georgiana	Implementare procesare imagine cu Vision API pentru extragerea etichetelor și a textului din imaginile stocate în baza de date
08.05.2020	Cosmin	Implementare Geofencing Api și Geocoding Api pentru anunțarea unei urgențe în apropiere și afișarea unei notificări cu adresa
09.05.2020	Andreea	Adăugare vacanță și asignare doctor înlocuitor backend și frontend
10.05.2020	Cosmin	Implementare funcționalitate Medical Practitioner
11.05.2020	Georgiana	Analiza cont Cloud pentru billing (servicii din free tier / servicii always free)
12.05.2020	-	Documentare parțială CF / TF
14.05.2020	Andreea	Implementare share files Android cu salvare in Firebase Storage + modificare salvare fisiere pe web
16.05.2020	Georgiana	Adăugare eventListener pentru verificarea urgențelor inserate (marcarea lor ca și clone în alt tabel)
17.05.2020	Cosmin	Îmbunătățiri Android (repornire servicii după restart smartphone, resetare componenta SharedPreferences etc)
19.05.2020	Andreea	CRUD appointment Android cu redirecționare la alt doctor în caz de indisponibilitate la data aleasă
21.05.2020	Georgiana	Implementare Speech-to-Text API

23.05.2020	-	Împărțire diagrame existente și adăugarea lor în lucrări + diferențiere cuprins
24.05.2020	-	Migrare text din latex in word (TF Cloud si Studiu bibliografic)
27.05.2020	-	Diagrame de pachete, secvențiere
29.05.2020	-	Diagrama de componente, deployment
31.05.2020 01.06.2020	-	Scris în lucrarea de licenta: CF, NF, arhitectura conceptuală, cazuri de utilizare
05.06.2020 06.06.2020	-	Realizarea capitolului 6 - Testare
08.06.2020	-	Modificare diagrame Descrierea tehnologiilor
10.06.2020	Georgiana	Documentare si implementare blockchain
11.06.2020	Andreea	Documentare și implementare chat în componenta web
12.06.2020 13.06.2020	-	Realizare capitolului 4
14.06.2020	Andreea	Documentare și implementare chat în componenta mobile
15.06.2020	Georgiana	Implementare funcțiilor de sortare și trimitere mail
17.06.2020	-	Selectarea referințelor și materialelor potrivite pentru studiul bibliografic
20.06.2020	-	Verificarea funcționalității sistemului
21.06.2020 22.06.2020 23.06.2020	-	Realizarea capitolului 5 și 7
24.06.2020	Cosmin	Optimizare și refactorizare componente Android
25.06.2020	Andreea	Acoperirea unor noi funcționalități pe web
26.06.2020	Georgiana	Deploy-ul versiunilor finale de backend și frontend
28.06.2020	-	Realizarea capitolului 8
01.07.2020	-	Finalizarea temei

Anexa 2 – Tabel de figuri

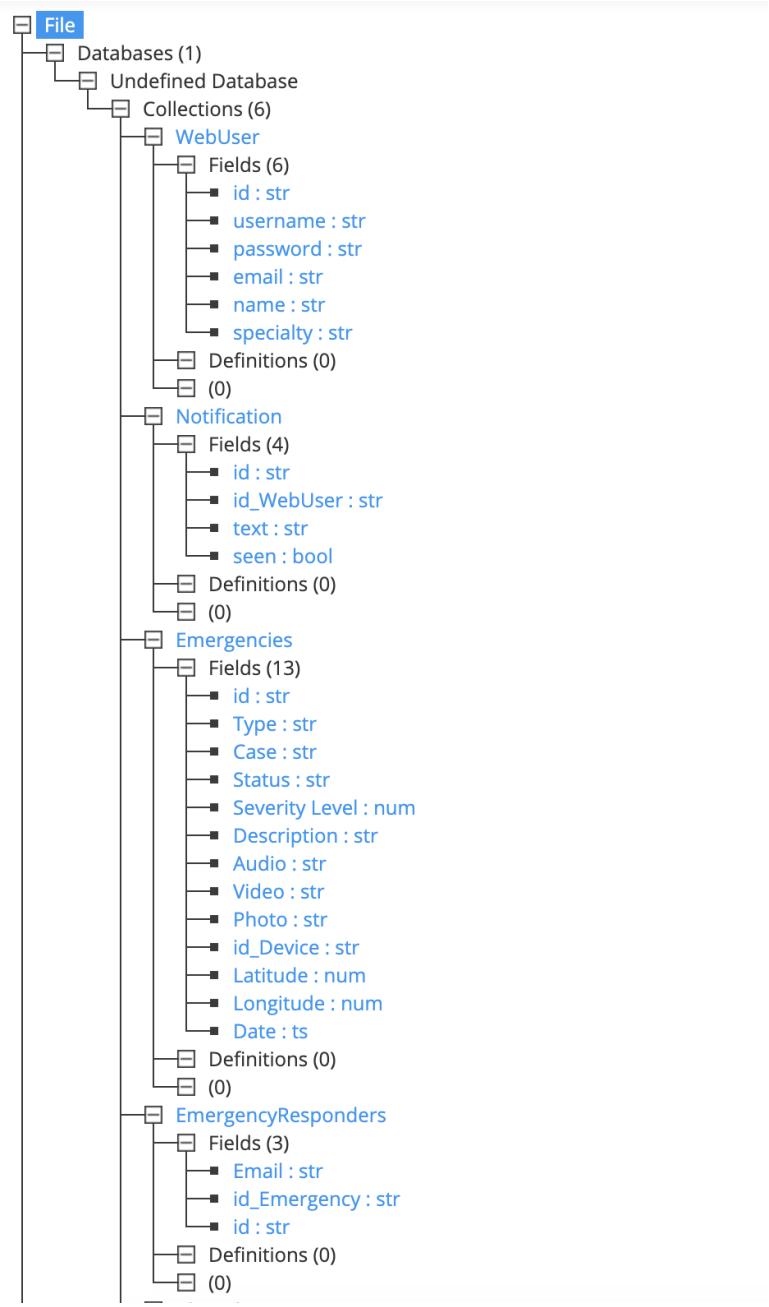
Anexa 3 – Listă de tabele

Anexa 4 – Glosar de termeni

Termen	Scurtă descriere
MHealth	Denumirea platformei implementate
GCC	Google Cloud Console – consola pusă la dispoziție de Google pentru vizulizarea resurselor si proiectelor Cloud
GCP	Google Cloud Project – proiectul de Cloud
App Engine	Server de deployment pentru proiectele Google Cloud
API (Vision API, Speech-to-Text API, Natural Language API, Geocoding API)	Application Prograamming Interface – set de metode care pot fi apelate fara a cunoaste modul de implemnetare
Auto ML	Componenta Google pentru Machine Learning care permite antrenarea si deploy-ul unui model
REST	Representational State Transfer – protocol pentru transferul resurselor folosind URI
Blockchain	Structura de date pentru stocarea informatiilor intr-un mod sigur, dar transparent
NO-SQL	Tip de baza de date ne-relațională
Firebase Console	Consola Firebase petntru gestionarea si vizualzarea bazei de date si a drepturilor asupra acesteia
EHR	Electronic Health Record – un protocol de strangere si stocare a datelor medicale
SDK	Software Development Kit – set de instrumente software pentru dezvoltarea si integrarea aplictiilor
JVM	Java Virtual Machine – masina virtula pe care este compilat si rulat codul Java

Anexa 5 – Modelarea bazei de date cu Hackolade

Pentru modelarea bazei de date te tip NO-SQL s-a gasit un instrument performant, lansat recent, Hackolade[51], care permite realizarea diagramelor cu usurinta si previzuaizarea structurii sub forma de JSON.



[51] https://hackolade.com

- ClonedEmergencies
  - Fields (3)
    - id_NewEmg : str
    - id_OriginalEmg : str
    - id : str
  - Definitions (0)
  - (0)
- GeofenceLocations
  - Fields (3)
    - id : str
    - Latitude : num
    - Longitude : num
  - Definitions (0)
  - (0)
- Relationships (4)
  - fk WebUser.id to Notification.id_WebUser (Foreign Key)
  - fk Emergencies.id to Responders.id_Emergency (Foreign Key)
  - fk Emergencies.id to ClonedEmergencies.id_OriginalEmg (Foreign Key)
  - fk Emergencies.id to ClonedEmergencies.id_NewEmg (Foreign Key)
- Definitions (0)