UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DOMENIUL: Calculatoare și tehnologia informației

SPECIALIZAREA:Tehnologia informației

**Virtual Assistant**

Student

Iordan Alexandru

**Cuprins**

[Introducere](#__RefHeading__5011_1794362497) 1

[Capitolul 1. Fundamentarea teoretică și documentarea bibliografică](#__RefHeading___Toc31704_503711699) 2

[1.1. Domeniul și contextul abordării temei](#__RefHeading___Toc389_797011268) 2

[1.1.1. Motivarea temei alese](#__RefHeading___Toc16433_2103660952) 3

[1.2. Tema aplicației](#__RefHeading___Toc16435_2103660952) 5

[1.2.1. Scopul și obiectivele aplicației](#__RefHeading___Toc16437_2103660952) 6

[1.3. Prezentarea comparativă a aplicațiilor concurente și analizarea acestora](#__RefHeading___Toc16439_2103660952) 6

[1.3.1. Virtual Assistant vs Siri sau Cortana](#__RefHeading___Toc7917_113963845) 7

[1.3.2. Virtual Assistant vs Google Asisstant sau Alexa](#__RefHeading___Toc7919_113963845) 8

[1.3.3. Concluzii](#__RefHeading___Toc7921_113963845) 9

[1.4. Specificațiile privind caracteristicile așteptate de la aplicație](#__RefHeading___Toc7923_113963845) 10

[Capitolul 2. Proiectarea aplicației](#__RefHeading___Toc7925_113963845) 12

[2.1. Analiza platformei hardware](#__RefHeading___Toc25839_113963845) 12

[2.2. Componente software](#__RefHeading___Toc25841_113963845) 13

[2.2.1. Diagrama ER](#__RefHeading___Toc25843_113963845) 13

[2.2.2. Diagrama de activitate](#__RefHeading___Toc32379_1590240391) 14

[2.2.3. Diagrama UML de clase](#__RefHeading___Toc25845_113963845) 15

[2.2.4. Tehnologia aleasă pentru implementare](#__RefHeading___Toc21546_2373635087) 17

[2.2.5. Descrierea claselor implementate](#__RefHeading___Toc21548_2373635087) 19

[Capitolul 3. Implementarea aplicației](#__RefHeading___Toc1561_3734190073) 24

[3.1. Descriere implementării](#__RefHeading___Toc8925_332029019) 24

[3.2. Dificultăți întâmpinate](#__RefHeading___Toc8927_332029019) 25

[3.3. Idei originale](#__RefHeading___Toc31598_332029019) 26

[3.4. Funcționalitatea sistemului](#__RefHeading___Toc31600_332029019) 27

[3.4.1. Interfața cu utilizatorul](#__RefHeading___Toc31602_332029019) 27

[3.4.2. Exemplu de raport primit pe mail](#__RefHeading___Toc31604_332029019) 27

[3.4.3. Exemplu de *i*nteracțiune cu *Virtual Assistant*](#__RefHeading___Toc31606_332029019) 28

[3.4.4. Exemplu rulare aplicație utilitară](#__RefHeading___Toc31608_332029019) 30

[Capitolul 4. Testarea aplicației și rezultate experimentale](#__RefHeading__6364_982623240) 31

[4.1. Lansarea aplicației](#__RefHeading___Toc40209_1879252158) 31

[4.2. Rezultate experimentale](#__RefHeading___Toc40211_1879252158) 32

[Concluzii](#__RefHeading___Toc40809_1879252158) 34

[Gradul de finalizare](#__RefHeading___Toc9468_1449576518) 34

[Comparație cu alte proiecte similare](#__RefHeading___Toc9470_1449576518) 34

[Posibile direcții de dezvoltare](#__RefHeading___Toc9472_1449576518) 34

[Bibliografie](#__RefHeading__6324_59671573) 35

[Anexe.](#__RefHeading__6326_59671573) 36

[Anexa 1. Ovum](#__RefHeading___Toc32381_1590240391) 36

[Anexa 2. IVR](#__RefHeading___Toc32383_1590240391) 36

[Anexa 3. DTMF](#__RefHeading___Toc32385_1590240391) 36

[Anexa 4. train\_chatbot.py](#__RefHeading___Toc32387_1590240391) 36

[Anexa 5. tutorial/views.py](#__RefHeading___Toc32389_1590240391) 39

[Anexa 6. decorators.py](#__RefHeading___Toc32391_1590240391) 46

[Anexa 7. forms.py](#__RefHeading___Toc32393_1590240391) 48

[Anexa 8. models.py](#__RefHeading___Toc32395_1590240391) 55

[Anexa 9. urls.py](#__RefHeading___Toc45950_1590240391) 57

[Anexa 10. text\_processing.py](#__RefHeading___Toc45952_1590240391) 58

[Anexa 11. Aplicație utilitară](#__RefHeading___Toc45954_1590240391) 72

Virtual Assistant

Iordan Alexandru

Rezumat

Virtual Assistant sau asistentul virtual reprezintă o aplicație web desemnată persoanelor cu anumite afecțiuni care sunt supravegheate de un îngrijitor sau un asistent medical. Acesta trebuie să își facă un cont pe website pentru a putea introduce pacienții. Pentru un plus de siguranță și simplitate, îngrijtorul este cel care se ocupă cu crearea contului și inserarea de date personale ale pacientului.

Ca principale afecțiuni monitorizate de către algoritmii din spatele website-ului sunt depresia, singurătatea și Alzheimerul. Acestea au în comun, printre altele, din punct de vedere al tratamentului, un stil de viață sănătos, combaterea singurătății și relaxarea părții cerebrale. Aplicația are rolul de a fi un companion inteligent în lupta împotriva singurătății a persoanelor în vârstă. Potrivit unor studii făcute de asociația CareMore Health0 , 43% din persoanele vârstnice se simt singuratice zi de zi, de asemenea există o rată de mortalitate de 45% a vârstnicilor care reportează că se simt singuri și nu în ultimul rând, singurătatea este mai periculoasă decât obezitatea și la fel de dăunătoare asupra sănătății cât consumul a 15 țigări pe zi.

Virtual Assistant este un proiect conceput de mine, inovativ din moment ce nu se găsesc alternative atât în mediul online cât și cel offline. Acesta are un modul special ce înregistrează, monitorizează și prezintă un raport cu privire la starea pacienților. Pentru început, pacienții trebuie să completeze un chestionar general cu privire la modul lor de viață, iar separat, în funcție de boală, au întrebări particulare care ajută algoritmul să știe mai multe legat de starea lor de sănătate. După ce au completat acest chestionar, rețin în baza de date rezultatele chestionarului alături de un punctaj care mă ajută să prelucrez mai ușor rezultatele. Acest chestionar se completează doar o singură dată, după care va fi accesibil un alt chestionar care va trebui completat din două în două săptămâni pentru a monitoriza starea pacientului în funcție de boala acestuia. La fiecare 3 chestionare completate (6 săptămâni) se va prezenta un raport complet asisitentului medical / îngrijitorului în care se vor preciza rezultatele chestionarelor alături de interpretarea acestora. Întrebările din ambele chestionare au fost atent selecționate, m-am ghidat după anumite site-uri guvernamentale sau aplicații oficiale și le-am integrat în Virtual Assistant.

Modulul care se ocupă special de combaterea singurătății are multe în comun cu servicii de asistență digitală precum Alexa, Google Assistant, Cortana sau Siri. Am ales să îl creez de la zero din diferite motive. Unul dintre acestea este faptul că, din punct de vedere al dezvoltatorului, nu ai transparență totală asupra aplicației, iar unele servicii pot fi achiziționate doar contra-cost. Printre funcționalitățile cheie ale modulului vocal se regăsesc capacitatea acestuia de a recunoaște intențiile utilizatorului folosindu-se de algoritmi de inteligență artificială, mai exact Machine Learning. Astfel, utilizatorul nu trebuie să spună cuvinte care să se regăsească într-un anumit pattern, algoritmul având capacitatea de a-și da singur seama de

intenția pacientului. Alte funcționalități care pot ușura viața utilizatorului sunt planificatorul de evenimente care folosește Google Calendar, prezentarea știrilor în funcție de interesele persoanei care controlează aplicația, un modul special creat discuțiilor între entitatea artificială și pacient, modul care folosește identificarea emoțiilor, am integrat și capacitatea aplicației de a deschide browsere sau alte programe, să caute pe Youtube, Google orice fel de informații.

Astfel, după două săptămâni de interacțiune cu asistentul virtual, pacientul completează cel de-al doilea chestionar care ar trebui să indice o stare mai bună de sănătate.

Alături de Virtual Assistant am creat o altă aplicație ce nu are legătură cu modulul web, ci e o aplicație utilitară care se bazează pe comunicarea între diferiți senzori conectați wireless la un gateway și în funcție de grosimea pereților prezenți în casă, se poate vizualiza dacă acel senzor este în aria de acoperire a gateway-ului sau nu.

Introducere

Trăim într-o lume în care accentul cade pe cei tineri, noi suntem viitorul, dar totodată, uităm de cei care au fost „viitorul” propriilor generații, de cei care ne-au ajutat să ajungem unde suntem astăzi. Deseori se întâmplă ca cei apropiați nouă, bunici, unchi, mătuși, chiar și părinți, să fie cu totul și cu totul depășiți de noțiunile ce țin de tehnologia actuală. Astfel, ei trec cu vederea anumite tehnologii, gadgeturi care le pot îmbunătăți viața considerabil.

Tot timpul am fost pasionat de cum tehnologia poate impacta viața individului. Tema licenței mele constă în atenția deosebită pe care o acord celor în vârstă care suferă de singurătate, Alzheimer, singurătate și depresie. Aceste boli sunt o consecință a singurătății apăsătoare de care mulți bătrâni au parte. Cei de la British Broadcasting Corporation au publicat o știre în anul 2011 pe 31 Ianuarie despre singurătate, supranumind-o „hidden killer of elderly”, ceea ce se traduce în asasinul ascuns al celor în vârstă, metafora făcând referire la bolile pe care aceasta le poate asunde. De asemenea, cei de la New York Times pe 25 Martie 2009 au citat următoarele: „singurătatea conduce la o stare precară a sănătății mintale și a celei fizice1.

Singurătatea este definită în moduri diferite. Conform Wikipediei, singurătatea sau solitudinea este o stare emoțională de izolare, când o anumită persoană simte lipsa relaționării cu ceilalți sau nu simte iubire. Poate fi rezultatul relațiilor rele cu altcineva, amorului nefericit, alegerii intenționate, bolii fizice sau psihice, depresiei, obișnuințelor personale antipatice sau altor condiții depinzând de situație. O altă definiție, dintr-o perspectivă medicală a celor de la romedic, descriu singurătatea ca fiind un sentiment subiectiv, nedorit, ce arată lipsa și pierderea relațiilor sociale. Aceasta apare atunci când există o discrepanță între relațiile sociale pe care le avem și cele pe care ni le dorim de fapt. Determinanții singurătății sunt adesea definiți pe baza a două modele cauzale. Primul model se referă la factorii externi, care lipsesc în relațiile sociale, ca rădăcină a singurătății. Al doilea model se referă la factorii interni, cum ar fi personalitatea individuală și factorii psihologici2.

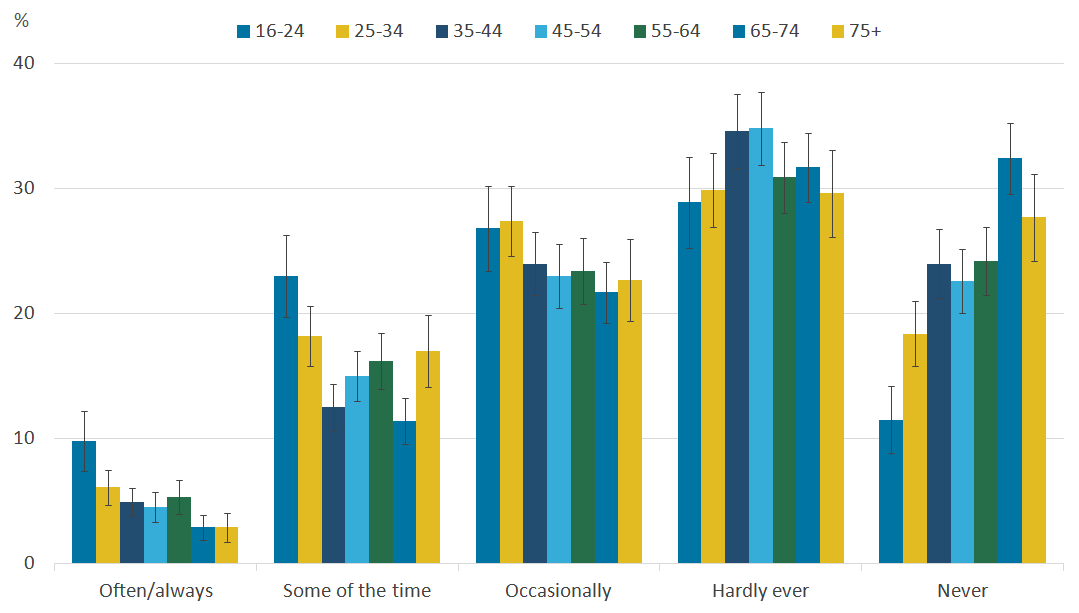
Din punct de vedere medical există două tipuri de singurătate și anume cea emoțională care se resimte atunci când ne e dor de compania unei anumite persoane (de cele mai multe ori este vorba despre soț, mama sau de cel mai bun prieten) și cea socială care apare atunci când ne lipsește o rețea socială mai largă sau un grup de prieteni. Singurătatea se poate manifesta ca fiind una ocazională, atunci când sentimentul apare doar în anumite momente (de exemplu de Crăciun) sau cronică, atunci când persoana se simte singură în majoritatea timpului.

1. Fundamentarea teoretică și documentarea bibliografică
   1. Domeniul și contextul abordării temei

Datorită progresului tehnologic exponențial din ultimele decenii, iar aici mă refer la aparatură medicală performanța, medicamente eficiente pentru diverse bolii, tratamende medicale mai ample, informații cu privire la un stil de viață sănătos, accesul la internet care facilitează cunoașterea și înțelegerea anumitor concepte care ne-ar putea prelungi viața (yoga, meditarea, sportul), populația vârstnică a ajuns să fie din ce în ce mai numeroasă.

Aceste persoane se confruntă cu numeroase schimbări fizice, psihologice și sociale. În consecință, de multe ori stima de sine poate scădea. O femeie în vârstă care rămâne singură, fără soț, iar copiii și-au făcut deja un rost, tinde să nu mai aibă încredere în propria persoana, să nu mai aibă un scop precis pentru care să trăiască. O astfel de persoană caută să ascundă starea de singurătate pentru a nu pune presiuni asupra celor dragi ca să compenseze lipsa prezenței. Odată cu înaintarea în vârstă este inevitabil ca oamenii să nu piardă legăturile de prietenie pe care le-au avut si să le fie dificil să inițieze noi relații, iar lipsa încrederii în sine joacă un rol esențial.

De asemenea, un alt grup afectat de singurătate este cel al tinerilor cu vârste cuprinse între 16-25 de ani. După cum se poate observa în graficul de mai jos, persoanele cele mai afectate de singurătate sunt tinerii și cei de peste 54 de ani3, potrivit celor de la biroul Național de statistică din United Kingdom. Acest studiu a fost desfășurat pe parcursul anilor 2016-2017 în Anglia.

  
Figura 1.1: Singurătatea pe grupe de vârstă

Principalele concluzii după ce s-a terminat perioada studiului, au fost următoarele:

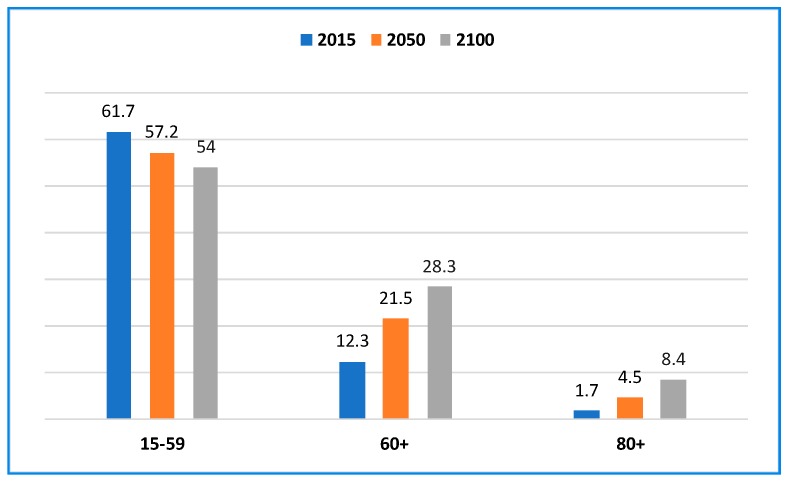
1) Femeile au raportat că se simt singure mai des decât bărbații

2) Cei singuri sau văduvi au declarat că se simt singuri mai des

3) Oamenii care n-au încredere în vecini au raportat că se simt singuri

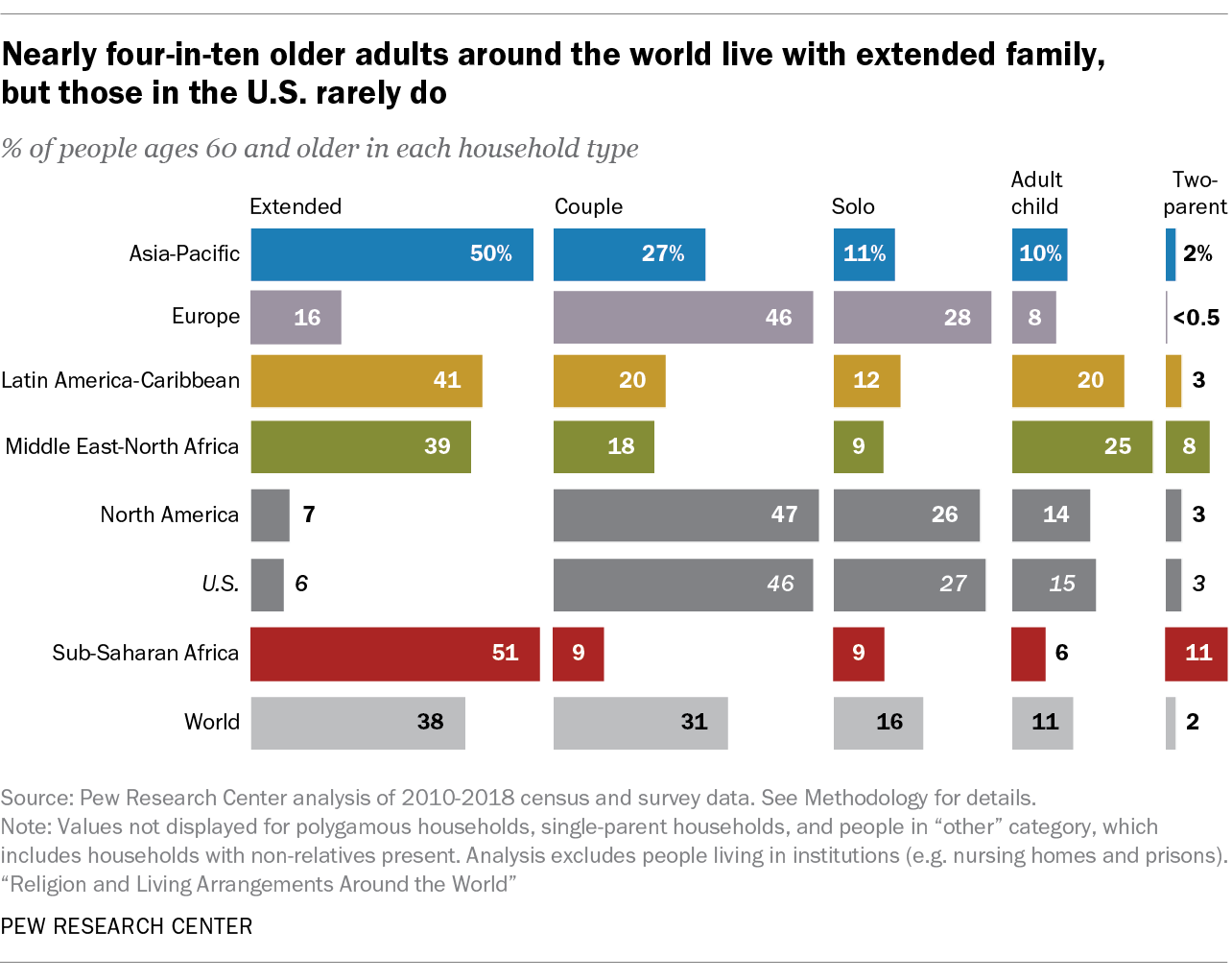
Pentru cea de-a doua aplicație, cea utilitară, am considerat că unele companii de supraveghere ar putea avea un modul special pentru persoanele singuratice și ar fi util ca diverși senzori să fie atașați la utilizator, senzori care monitorizează pulsul, nivelul zahărului din sânge pentru diabetici, senzori care detectează fum, gaz, lumini uitate aprinse sau alți factori ce ar putea genera costuri materiale consistente.

La nivel mondial, numărul total de persoane în vârstă crește mai rapid în comparație cu alte grupe de vârstă. În consecință, ponderea persoanelor în vârstă crește aproape peste tot. În 2015, una din opt persoane din întreaga lume avea vârsta de 60 de ani sau mai mult. Până în 2030, una din șase persoane va face parte din această grupă de vârstă la nivel global. În plus, persoanele în vârstă vor depăși copiii între 0 și 9 ani până în 2030. Până în 2050, aceștia pot depăși adolescenții și tinerii cu vârsta cuprinsă între 10 și 24 de ani. Procesul de îmbătrânire este mai avansat în țările cu venituri mari. Japonia are de departe cea mai mare vârstă a populației. În 2015, 33% din populație avea vârsta de 60 de ani sau mai mult. În ceea ce privește populația vârstnică, Japonia este urmată îndeaproape de Germania (28%), Italia (28%) și Finlanda (27%). Prin urmare, ritmul cu care populația mondială îmbătrânește crește în timp. Până în 2030, se anticipează că persoanele în vârstă vor reprezenta în mare măsură peste 25% din populațiile din Europa și America de Nord, 20% în Oceania, 17% în Asia și 6% în Africa. Așa cum se arată în figura 1, populația vârstnică (cu vârsta de peste 60 de ani) va crește mai repede ca procent din populația totală decât populația cu vârste cuprinse între 15 și 59 de ani. Dacă tendința continuă, nu vor fi suficiente persoane care să aibă grijă de persoanele în vârstă în viitorul îndepărtat. Prin urmare, în viitor vor fi necesare tehnologii de viață asistată pentru a avea grijă de persoanele în vârstă și a le ajuta să trăiască independent4.

  
Figura 1.2: Procentele de persoane de vârste diferite din lume în ani diferiți

* + 1. Motivarea temei alese

Am vrut ca proiectul meu să se focuseze în primul rând pe persoanele singure, indiferent de vârstă, urmând ca anumite particularități și o atenție specială să o acord celor în vârstă care suferă sau pot suferi de anumite afecțiuni. Multe studii arată faptul că numărul locuințelor în care trăiește o singură persoană crește de la an la an, iar grupul de vârstă care este majoritar sunt cei de peste 65 de ani. Unul din motive este decesul celor apropiați, iar aici mă refer la frați și surori, iar un alt motiv îl constituie mutarea copiilor la propria casă.

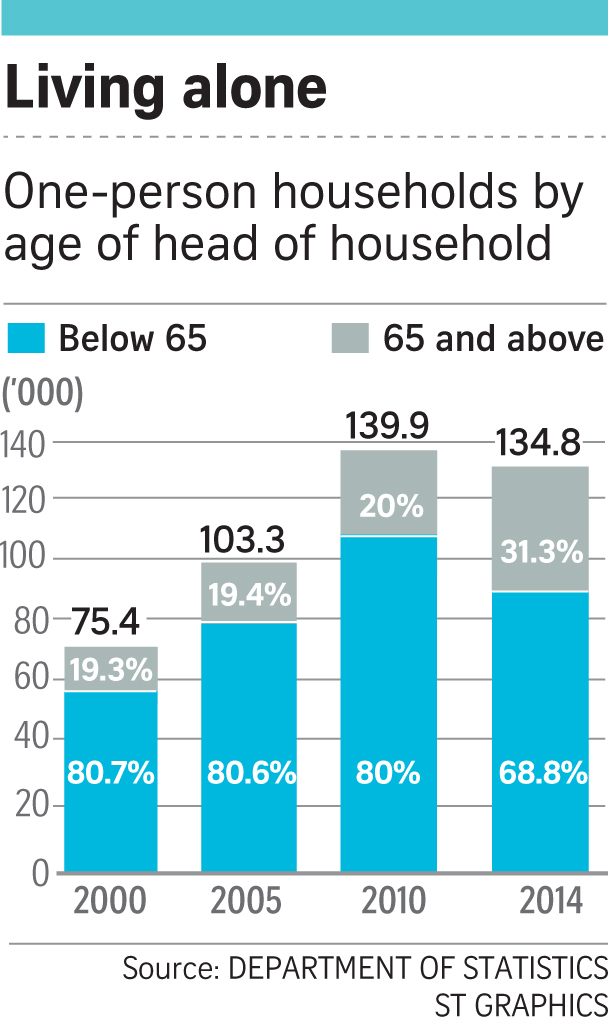
  
Figura 1.3: Procentajul persoanelor în vârstă cu privire la modul acestora de a locui

Statistica de mai sus cuprinde toate persoanele în vârstă de minim 60 de ani și cum locuiesc acestea din punct de vedere a membrilor familiei. După cum se poate observa, în Europa aproape 30% dintre persoanele care aparțin categoriei de vârstă menționate mai sus locuiesc singure, la egalitate aproape cu America de Nord5 De asemenea, putem observa o mare diferență față de Asia, unde în majoritatea locuințelor trăiesc familii extinse formate din copiii celor în vârstă și aceștia.

Un alt lucre ce trebuie luat în considerare este media la nivel global, cu mult sub cea europeană, unde mai bine de un sfert din bătrâni trăiesc singuri.

Într-o altă statistică, cea de mai jos, putem vedea clar că pe măsură ce timpul trece situația devine îngrijorătoare. Astfel, din anul 2000 până în 2005 nu s-au înregistrat schimbări în ceea ce privește modul în care cei în vârstă locuiesc, procentajul rămânând relativ același, 19.4%. În 2010 putem vedea o ușoară creștere în procentaj, dar o creștere semnificativă din punctul de vedere al numărului de persoane înregistrate. În 2014 creșterea numărului de persoane care locuiesc singure și au peste 65 de ani este de aproape 10%, ajungând la 31.3%. Este de la sine înțeles că avansul tehnologic crește speranța de viață peste tot în lume și de aceea trebuie să găsim o soluție pentru a-i integra și pe cei care rămân singuri.

Sociologii intervievați au spus că astfel de locuințe în care oamenii în vârstă trăiesc singuri sunt mai răspândite acum din cauza unor factori precum îmbătrânirea populației și schimbarea structurilor familiale rezultate în urma unei rate scăzute a natalității, a creșterii ratei divorțului și a relațiilor familiale înstrăinate. Alți oameni în vârstă doresc pur și simplu mai mult spațiu personal și independență.

  
Figura 1.4: Persoanele care trăiesc singure

Potrivit Biroului de recensământ al SUA, 11 milioane sau 28% dintre persoanele în vârstă de 65 de ani și mai mari, trăiau singuri la momentul recensământului6. Pe măsură ce oamenii îmbătrânesc, probabilitatea acestora de a trăi singuri crește. AARP ("American Association of Retired Persons" – „Asociația Americană a Pensionarilor) raportează că tot mai mulți adulți în vârstă nu au copii. Asta înseamnă că sunt mai puțini membri ai familiei care să ofere companie și îngrijire, deoarece acei adulți devin seniori.

Deși a trăi singur nu duce inevitabil la izolarea socială, cu siguranță poate fi un factor care contribuie la aceasta. Un alt factor ce trebuie luat în considerare este regularitatea persoanelor în vârstă de a se implica în activitățile sociale.

Statistici din Canada raportează că 80% dintre persoanele în vârstă participă la una sau mai multe activități sociale pe lună, rezultând astfel o cincime care este lăsată pe dinafară.

Contactele sociale tind să scadă pe măsură ce îmbătrânim din motive precum pensionarea, moartea prietenilor și a familiei sau lipsa de mobilitate. Indiferent de cauzele izolării a celor în vârstă, consecințele pot fi alarmante și dăunătoare. În plus, izolarea socială este o luptă pentru mulți oameni în vârstă. Din fericire, cercetările privind riscurile, cauzele și prevenirea singurătății la persoanele în vârstă au oferit cunoștințe cu privire la acest aspect în ultimele două decenii.

* 1. Tema aplicației

Consider că în următorii ani, pe măsură ce vom progresa din ce în ce mai mult din punct de vedere medical și tehnologic, vor fi din ce în ce mai mulți cei care aparțin categoriei de vârstă 65+. Așadar, după cum se vede și în graficele de mai sus, putem prezice că odată cu creșterea numărului de oameni din această categorie, o să crească și numărul celor care vor locui singuri în pragul bătrâneții. Tema aplicației constă în combaterea singurătății celor în vârstă prin diferite mijloace. Practic, am reușit să construiesc un asistent virtual capabil să dialogheze într-un mod eficient cu utilizatorul, să afle problemele care-l apasă pe acesta, să-l aducă la zi cu știri, informații din diferite surse, poate să-i „hrănească” curiozitatea cu privire la orice, atâta timp cât se regăsește pe Wikipedia, poate să-i recomande filme și să creeze evenimente în calendarul Google al utilizatorului. Pe scurt, acest asistent virtual e departe de a se compara cu ce oferă concurența, iar aici mă refer la Google Assistant, Alexa, Jarvis, Siri sau Cortana, dar ceea ce e special la asistentul meu virtual este interesul „artificial” pe care îl acordă utilizatorului.

* + 1. Scopul și obiectivele aplicației

Mai departe voi prezenta scopul aplicației și în ce constă mai exact acesta, dar mai întâi vreau să trec peste obiectivele pe care mi le-am propus.

Primul obiectiv pe care l-am îndeplinit cu succes este realizarea de rapoarte eficiente pe care le trimit asistentului medical sau celui care îngrijește pacientul. Acestea sunt sub forma unor chestionare pe care pacientul le completează o dată la două săptămâni, dar pentru a testa funcționalitatea acestora, din pricina timpului limitat, am exclus clauza timp. La fiecare 3 chestionare completate de către pacient, asistentul sau îngrijitorul va primi prin email un raport detaliat privind starea acestuia.

Menționez faptul că am gândit aplicația ca un fel de asistent virtual pentru oamenii care au anumite afecțiuni. Inițial, am vrut ca aplicația să poată oferi date cu privire la afecțiunea pacientului, cum să minimalizeze riscurile ca aceasta să se agraveze, iar ca principale afecțiuni am ales depresia, Alzheimerul și singurătatea. Documentându-mă mai bine, am realizat că nu am cunoștințe medicale suficiente astfel încât să pot recomanda pacientului sau să îi spun să evite anumite obiceiuri sau să îl pot îndruma. Astfel, am decis ca aplicația să aibă un rol strict de entertainment, deși îngrijitorul sau asistentul pacientului este obligat să specifice boala sau afecțiunea pe care o are acesta dintre cele menționate mai sus. Am tratat două cazuri în paralel, pacienții cu Alzheimer și cei cu depresie, iar pacienții care suferă de singurătate vor beneficia de modulul mai simplist, fără rapoarte, ci doar de interacțiunea acestora cu asistentul virtual.

Al doilea obiectiv îl reprezintă modulul asistentului virtual care are sarcina de a interacționa cu pacientul și de a reuși să-i facă viața mai ușoară. Scopul asistentului virtual este de a aduna date cu privire la starea psihologică a pacientului, de a înregistra informații și detalii despre acesta, iar aici mă refer la hobbie-uri, pasiuni, filme preferate și alte asemenea, cu ajutorul cărora entitatea artificială poate face dialogul mai intuitiv.

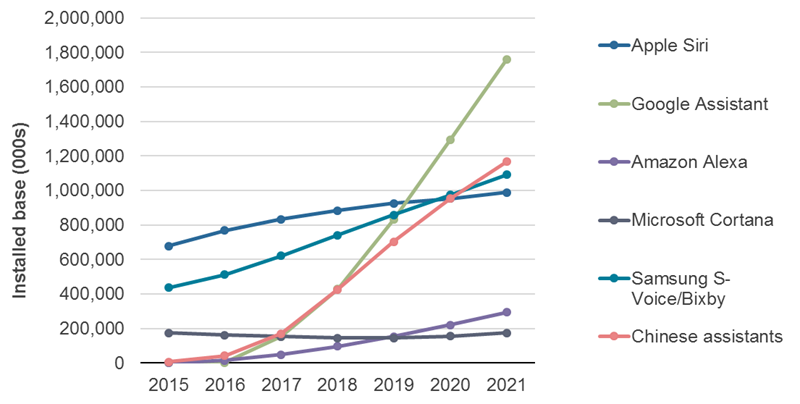
Având în vedere obiectivele de mai sus, scopul acestora este de a înlătura pe cât posibil singurătatea pe care o au de înfruntat mulți oameni în vârstă.

* 1. Prezentarea comparativă a aplicațiilor concurente și analizarea acestora

La nivel global, baza instalată de asistent digital este setată să depășească 7,5 miliarde de dispozitive active până în 2021, ceea ce reprezintă mai mult decât populația mondială, potrivit Biroului de recensământ american din 1 mai 2017. Google Assistant va domina piața de dispozitive AI-voce cu 23,3% cota de piață, urmată de Bixby-ul Samsung (14,5%), Siri-ul Apple (13,1%), Alexa-ul Amazon (3,9%) și Cortana Microsoft (2,3%)7.

Smartphone-urile și tabletele conduc în mod clar pe piața dispozitivelor inteligente vocale A.I., cu 3,5 miliarde de dispozitive active în 2016, majoritatea utilizând Google Now și Apple Siri. Cu toate acestea, utilizarea AI în combinație cu alte dispozitive crește mult implicarea consumatorilor și este setată să deblocheze noi oportunități, în special în casă. Ovum (vezi anexa 1 se așteaptă la o absorbție exponențială a capabilităților AI vocale printre dispozitivele noi, inclusiv dispozitivele portabile, smart home și TV, cu o bază instalată combinată de 1,63 miliarde de dispozitive active în 2021, o creștere de zece ori în 2016. În ciuda tuturor hype-urilor care înconjoară A.I, iar aici mă refer la difuzoare inteligente conectate8 dispozitive TV (adică televizoare inteligente, set-top box-uri și streamer-uri media) oferă o oportunitate mai mare, reprezentând 57% din baza instalată în 2021.

În continuare voi defini noțiunile de asistent digital și dispotive compatibile cu inteligența artificală tocmai pentru a ilustra mai bine diferența dintre aplicația mea și cele existente pe piață.

  
Figura 1.5: Trendul evoluției asistenților digitali

Astfel, dispozitivele compatibile cu AI sunt dispozitive conectate la Internet care au capacitatea de a capta și difuza conținut audio. Programul poate fi apoi adăugat pe dispozitivul respectiv pentru a crea funcții AI vocale prin care dispozitivul ar avea capacitatea de a auzi, de a înțelege, de a vorbi și de a răspunde la întrebările bazate pe voce. Dispozitivele care afișează numai răspunsuri printr-o interfață text și senzori / actuatori controlați vocal nu sunt incluși în această definiție. Este posibil ca un dispozitiv compatibil cu voce să nu aibă un asistent digital, în ciuda capacității sale de a face acest lucru. De exemplu, unele camere video conectate pot capta și transmite sunete, dar nu au un asistent digital. Totuși, unele dispozitive vocale care pot folosi AI pot avea mai mult de un asistent digital, cum ar fi Google Now9 și Samsung S-Voice pe smartphone-urile Galaxy.

Asistenții digitali prezintă un software de procesare a limbajului care folosește algoritmi de învățare automată pentru a extrage informații și pentru a efectua sarcini specifice în numele utilizatorului, pe baza unei conversații vocale sau a unui text. Asistenții care nu utilizează învățarea automată, inclusiv sistemele IVR (vezi anexa 2), sunt excluse din această definiție. Asistenții digitali pot fi, de asemenea, denumiți asistenți „inteligenți” sau „virtuali” de către industrie. Printre exemple se numără Google Assistant, Apple Siri, Microsoft Cortana și Alexa Amazon.

* + 1. Virtual Assistant vs Siri sau Cortana

Pornim de la premisa că Siri sau Cortana nu au module de inteligență artificială în spate.

Ce este„inteligent” la acestea este motorul de căutare vocală care are la bază un modul de procesare a limbajului natural, iar pentru oricine a folosit Siri știe că nu există nimic inteligent în acest sens, este doar o parsare a cuvintelor recunoscute de către aplicație10.

Ambii de mai sus sunt asistenți virtuali inteligenți. Asistentul virtual inteligent este un program de aplicație care poate înțelege limbajul natural (scris și / sau vorbit) și completează sarcini electronice pentru utilizatorul final. În cuvinte mai simple, sunt asistenți digitali capabili să execute anumite sarcini. Acestea includ preluarea, citirea mesajelor text sau e-mail cu voce tare, căutarea numerelor de telefon, căutarea pe un motor de căutare, plasarea apelurilor și reamintirea utilizatorului despre eventualele programări.

Virtual Assistant, neavând acces la dataseturile folosite pentru antrenarea rețelei neuronale ce stă la baza prelucrărilor de intenții, este un chatbot mai robust, construit pe un număr limitat de intenții. Acest fapt poate duce la interpretarea greșită sau chiar la neînțelegerea intenției utilizatorului, acest fapt reprezentând un minus.

O asemănare între cele trei servicii este utilizarea unui modul de procesare a limbajului natural, iar Virtual Assistant folosește un algoritm de Machine Learning capabil să clasifice intențiile după anumite cuvinte cheie, deci utilizatorul poate avea o libertate mai mare de exprimare, iar robotul va putea înțelege și clasifica intenția utilizatorului.

O altă asemănare o reprezintă modulul de executare a comenzilor prin asistență vocala. De exemplu, utilizatorul poate să execute anumite sarcini prin intermediul asistentului virtual. Printre acestea se enumeră deschiderea browserului, căutarea pe web, citirea informațiilor de pe anumite site-uri.

* + 1. Virtual Assistant vs Google Asisstant sau Alexa

Google Assistant este un asistent virtual bazat pe inteligență artificială dezvoltat de Google, care este disponibil în principal pe dispozitive mobile și inteligente. Spre deosebire de asistentul virtual anterior al companiei, Google Now, asistentul Google se poate angaja în conversații cu două sensuri11.

Utilizatorii interacționează în primul rând cu asistentul Google cu ajutorul vocii, deși este acceptată și introducerea tastaturii. În aceeași natură și manieră ca Google Now, asistentul este capabil să caute pe internet, să planifice evenimente și alarme, să regleze setările hardware pe dispozitivul utilizatorului și să afișeze informații din contul Google al utilizatorului. De asemenea, Google a anunțat că asistentul va putea identifica obiecte și colecta informații vizuale prin camera dispozitivului și va sprijini achiziționarea de produse și trimiterea de bani, precum și identificarea melodiilor.

Amazon Alexa, cunoscută și sub denumirea simplă de Alexa, este o tehnologie de asistență virtuală asistată dezvoltată de Amazon, utilizată pentru prima dată în boxele inteligente Amazon Echo dezvoltate de Amazon. Este capabil să interacționeze vocea, să redea muzică, să facă liste de activități, să stabilească alarme, să transmită podcast-uri, să joace cărți audio și să ofere vreme, trafic, sport și alte informații în timp real, cum ar fi știri12. De asemenea, Alexa poate controla mai multe dispozitive inteligente folosindu-se ca sistem de automatizare casnică. Utilizatorii sunt capabili să extindă capabilitățile Alexa instalând „abilități” (funcționalitate suplimentară dezvoltată de furnizorii terți, în alte setări mai frecvent numite aplicații precum programe meteorologice și funcții audio).

Majoritatea dispozitivelor cu Alexa permit utilizatorilor să activeze dispozitivul folosind un cuvânt de veghe (cum ar fi Alexa sau Amazon); alte dispozitive (precum aplicația mobilă Amazon de pe iOS sau Android și Amazon Dash Wand) impun utilizatorului să apese un buton pentru a activa modul de ascultare al Alexa, deși, unele telefoane permit utilizatorului să spună o comandă, cum ar fi „Alexa” sau „Trezire Alexa”. În prezent, interacțiunea și comunicarea cu Alexa sunt disponibile numai în engleză, germană, franceză, italiană, spaniolă, portugheză, japoneză și hindi.

Am ales ca Virtual Assistant să folosească un cuvânt de veghe pentru a activa aplicația. Pentru simplitate am ales să fie la fel ca în cazul aplicației dezvoltate de Amazon și anume cuvântul „Alexa” să pornească modulul Asistentului Virtual.

O altă noțiune similară este faptul că toate trei serviciile sunt facilitează utilizarea de acasă, nu ai nevoie de un smartphone, ci de un device special cumpărat de la Google, respectiv Amazon. Ceea ce e diferit însă și un mare plus, Asistentul Virtual e un modul complet desemnat interacțiunii pe web, nefiind nevoie să stocheze nimic la adresa utilizatorului. Pacientul (utilizatorul) va trebui să aibă o conexiune stabilă la internet pentru a putea folosi modulul și evident, să fie logat și înregistrat în baza de date ca pacient.

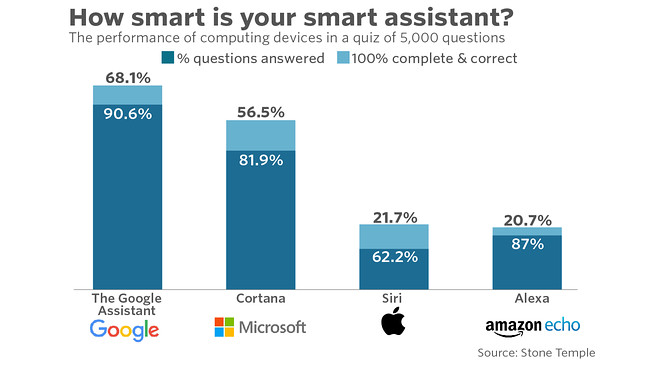
Un alt plus pentru pacienții care acuză anumite probleme ce țin de starea lor psihică este faptul că aplicația are un modul inteligent care oferă diferite soluții cu privire la problemele lor. Aceste soluții sunt parsate în mod inteligent, folosind motorul de căutare Google. De asemenea, datele sensibile sunt trimise prin e-mail asistentului medical pentru a putea supraveghea și interveni atunci când este cazul.

Un posibil minus al modulului vocal este faptul că singura limbă acceptată deocamdată este limba engleză și de asemenea, alte caracteristici mai avansate prezente la Google Assistant sau Amazon Alexa nu sunt implementate în Virtual Assistant.

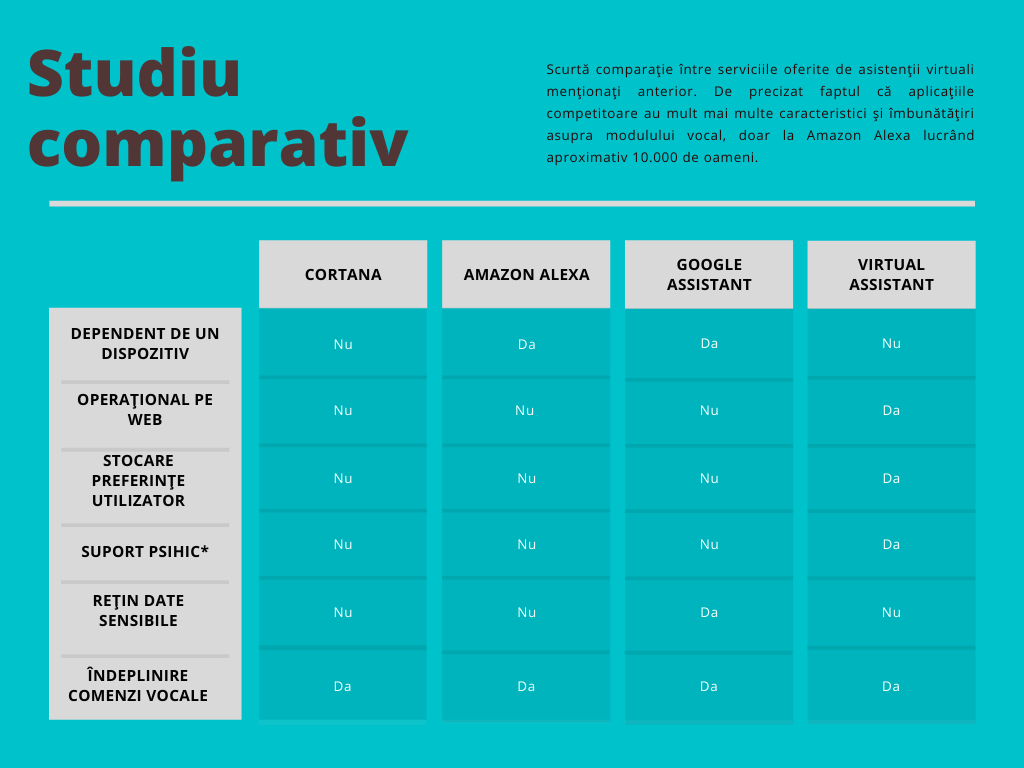
* + 1. Concluzii

Pentru o mai bună comunicare între pacient și asistentul digital, am vrut să integrez unul din asistenții virtuali de mai sus în aplicația mea pentru a putea oferi din experiența și inteligența cu care aceștia parsează informația. De asemenea, trebuie menționat faptul că doar Google Assistant reușește să aibă cea mai mare precizie în ceea ce privește interacțiunea acestuia cu utilizatorul, fiind capabil să răspundă la aproape 68.1% din 5000 de întrebări13.

Ca limbaje de programare, Google Assistant folosește în mare parte C++, iar Amazon Alexa o combinație între Java, Python și C#. Aplicația mea folosește în totalitate Python.

  
Figura 1.6: performanța asistenților virtuali

Din păcate, toți acești asistenți virtuali au un acord de licențiere ce nu permite decât integrarea acestora ca un tot în aplicația sau proiectul tău. Eu aș fi vrut să pot prelucra datele recepționate de aceștia și să pot stoca în propria bază de date anumite informații. Tocmai de aceea m-am axat pe dezvoltarea cât mai amplă a propriului meu asistent virtual. În studiul comparativ de mai jos am ales să ilustrez mai bine avantajele Virtual Assistentului pe care le are acesta față de concurență.

  
Figura 1.7: Studiu comparativ privind aplicațiile concurente

* 1. Specificațiile privind caracteristicile așteptate de la aplicație

Caracteristicile aplicației mele nu diferă cu mult față de cele ale asistenților virtuali precizați mai sus. Voi enumera mai jos câteva caracteristici cheie ale modulului vocal și voi începe cu cele care fac virtual assistentul unic.

1. Monitorizează boala pacientului și trimite un feedback asistentului sau îngrijitorului;
2. Poate îmbunătăți starea fizică și psihică a pacientului cu ajutorul modulului de entertainment;
3. Folosește „natural language processing” sau „procesare de limbaj natural”, ceea ce conferă un plus de umanitate robotului
4. Poate aduce la zi utilizatorul cu știri din întreaga lume din zeci de surse de încredere;
5. Poate să creeze un eveniment în calendarul google al utilizatorului și să customizeze detaliile acestuia;
6. Modul integrat Wikipedia. Utilizatorul poate întreba orice despre orice, iar acest modul va încerca să găsească un articol;
7. Poate recomanda filme dacă utilizatorul și-a setat filmul preferat în modulul de discuție. Virtual Assistent-ul poate să caute de asemenea top 3 filme bazat pe un gen și să revină cu un titlu și o scurtă descriere;
8. Modulul de discuție, poate cel mai complex modul, folosește identificarea emoției din text și își dă seama când pacientul are o problemă mai gravă, după care încearcă să parseze automat căutările de pe google cu privire la problema sa și să revină cu un răspuns sau o posibilă soluție. Acest modul pune bazele preferințelor utilizatorului (cartea preferată, filmul preferat, pasiuni, etc);
9. Poate să caute pe web într-un mod inteligent, parsând căutarea după cuvinte cheie;
10. Poate deschide aplicații variate aflate în calculatorul personal, de exemplu browsere, word, excel, aplicații care sunt de bază pentru un calculator;
11. Poate căuta pe YouTube o anumită melodie.
12. Proiectarea aplicației
    1. Analiza platformei hardware

Virtual Assistant este o aplicație web de sine stătătoare. Asta înseamnă că interacțiunea cu partea hardware este minimă. Aplicația web folosește un framework numit Django care rulează pe un server local.

Mai jos voi prezenta câteva dintre specificațiile calculatorului personal pe care rulează serverul.

|  |  |  |
| --- | --- | --- |
| Specificații sistem | Sistem de operare | Windows 10 Pro 64-bit |
| Producător Sistem | ASUSTek Computer INC. |
| Model | GL703GM |
| Procesor | Intel(R) Core(TM) i7-8750H CPU @ 2.2GHz |
| Memorie | 16384MB RAM |
| Specificații display | Nume | NVIDIA Geforce GTX 1060 |
| Memorie totală | 14205 MB |
| Memorie afișare (VRAM) | 6052MB |

Virtual Assistant are două module generale. Unul care are scopul de a înregistra și trimite date îngrijitorului pacientului sau asistentului acestuia, iar celălalt modul are funcția de a binedispune pacientul folosind diferite mijloace de entertainment. Ambele module sunt implementate pe framework-ul Django și nu necesită permisiuni speciale sau stocare de date pe calculatorul utilizatorului. Modulul vocal se folosește de anumite date pe care pacientul le inserează cu ajutorul unor chestionare în modulul care se ocupă cu manipularea datelor.

Un prim avantaj pe care îl constituie această interacțiune între module este faptul că sunt complet web-designed. Utilizatorul poate accesa paginile asupra cărora are permisiuni de vizualizare, imediat după ce se loghează. Astfel, modulul vocal este accesibil doar categoriei de tip pacient, iar modulul de inserare de date personale este accesibil doar asistentului sau îngrijitorului.

Un dezavantaj ține de comunicarea pe server. Modulul vocal trebuie să trimită fișiere de format mp3 care necesită o lățime de bandă mai mare decât pentru fișierele tip text. De asemenea trebuie stocat pe server și partea ce ține de antrenarea rețelei neuronale care se ocupă cu detecția de intenții.

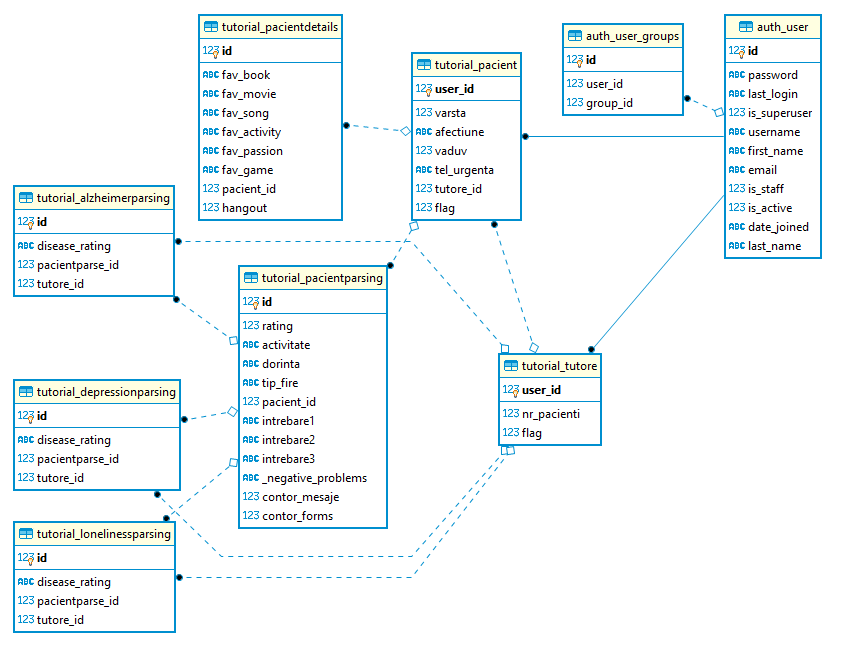
Virtual Assistent-ul este destul de limitat în ceea ce privește modul de interacțiune cu acesta. Utilizatorul trebuie să vorbească clar, să aibă un microfon performant și să se rezume la caracteristicile existente pe care le oferă acesta. Un dezavantaj este faptul că acesta poate interpreta greșit intenția utilizatorului sau să nu înțeleagă deloc ce spune pacientul, rezultând într-o eroare care îl va întoarce pe utilizator la interfața de bază.

Un dezavantaj al acestui tip de implementări îl constituie faptul că nu am suficiente date, așadar a trebuit eu să construiesc fișierul cu intenții din care învață entitatea artificială. Acest model este rigid, atunci când utilizatorul are formulări mai complexe, algoritmul aruncă o excepție de tipul intent not found și revine la funcționalitatea de bază, adică așteaptă ca utilizatorul să folosească keyword-ul „Alexa” și o comandă.

* 1. Componente software
     1. Diagrama ER

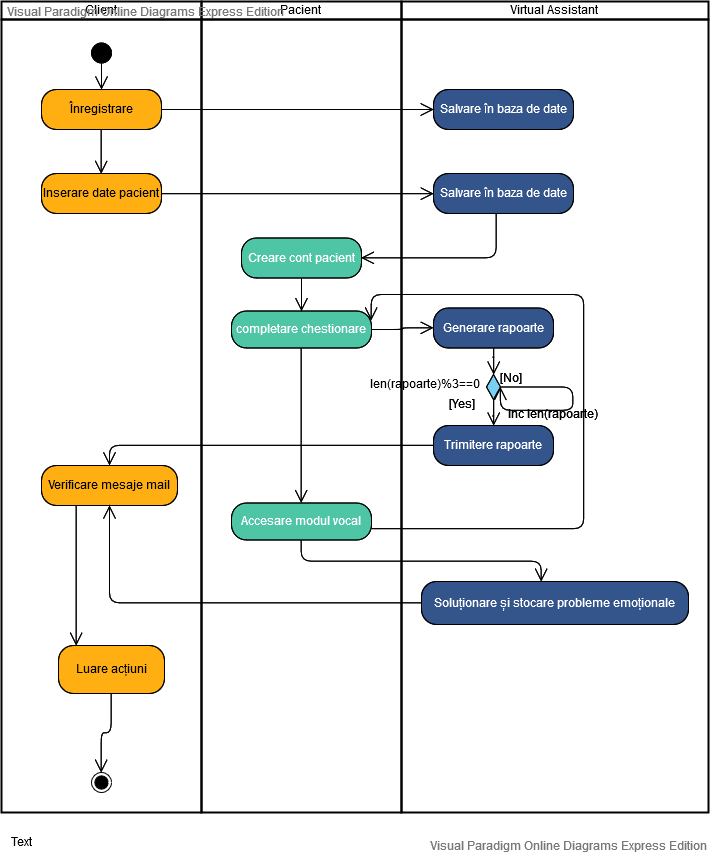
O diagrama de relații de entitate (ERD) arată relațiile seturilor de entități stocate într-o bază de date. O entitate în acest context este un obiect, o componentă a datelor. Un set de entități este o colecție de entități similare. Aceste entități pot avea atribute care îi definesc proprietățile. Definind entitățile, atributele lor și arătând relațiile dintre ele, o diagramă ER ilustrează structura logică a bazelor de date. Diagramele ER sunt utilizate pentru a schița proiectarea unei baze de date.

Ca bază de date am folosit SQLite3, aceasta este baza de date implicită a framework-ului Django. SQLite este o mică bibliotecă în C care implementează un motor de baze de date SQL încapsulat, oferă posibilitatea de a-l introduce în diverse sisteme și necesită zero-configurare. Mai în jos este descrisă diagram ER a bazei de date pe care am folosit-o.

  
Figura 2.1: Diagrama ER a bazei de date SQLite

* + 1. Diagrama de activitate

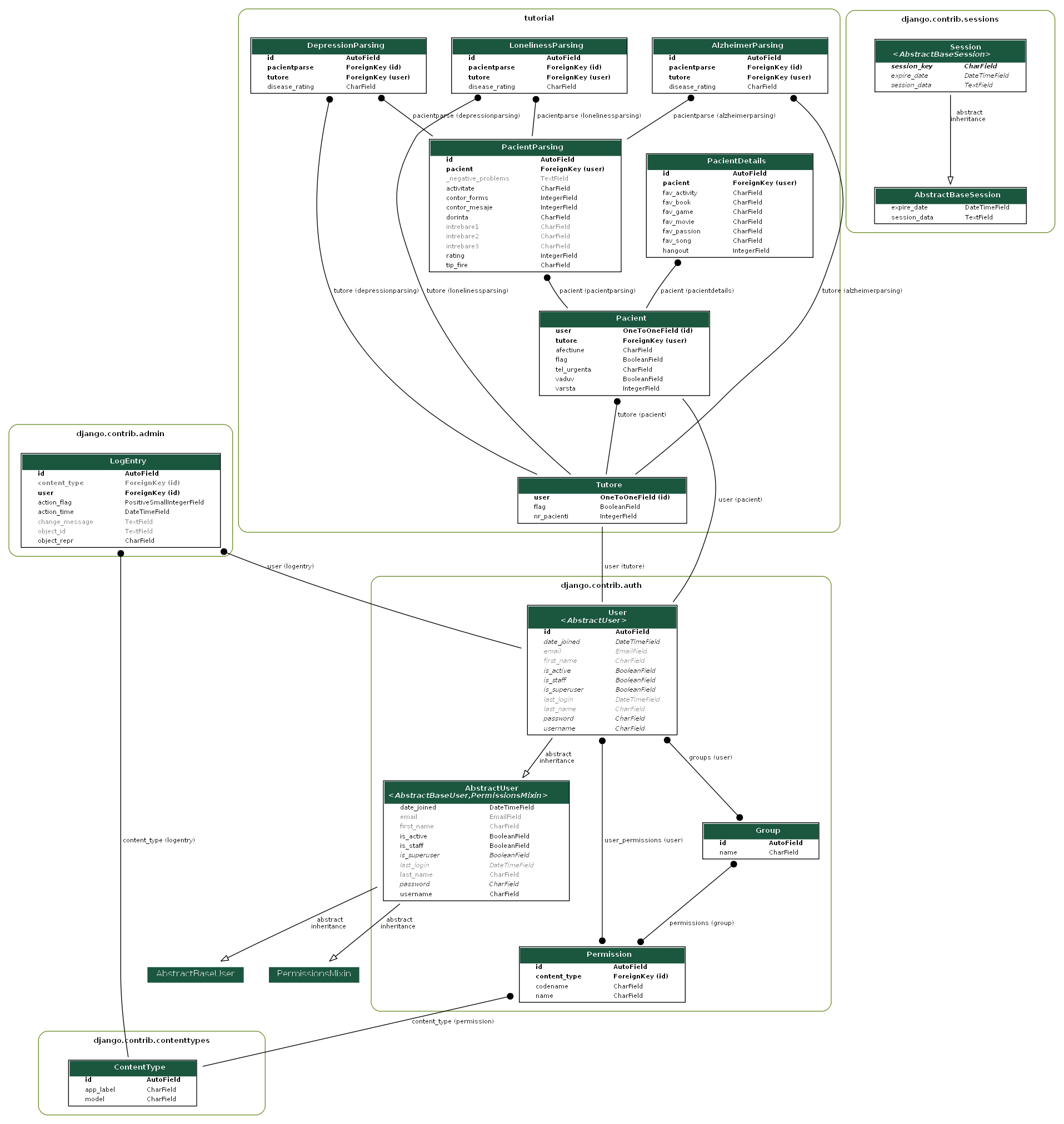
Mai jos puteți observa diagrama de activitate care cuprinde relațiile între client sau îngrijitorul pacientului, pacientul și aplicația Virtual Assistant. Astfel, se vede că pentru a accesa orice funcționalitate a aplicației, trebuie să ai un cont. Atunci când îți faci unul, automat ești trecut ca tutore sau asistent medical. Trebuie să inserezi pacienți și doar aceștia pot accesa modulul vocal pentru a evita confuziile cu privire la rapoarte. Un raport se întocmește o dată la 6 săptămâni sau altfel spus, atunci când sunt 3 chestionare completate la o frecvență de două săptămâni, timp în care pacientul accesează chatbotul. Doar persoana care are grijă de acesta poate lua decizii în funcție de rapoartele primite pe mail.

  
Figura 2.2: Diagrama de activitate a aplicației

* + 1. Diagrama UML de clase

Diagrama clasei UML este o notație grafică folosită pentru a construi și vizualiza sisteme orientate pe obiect. O diagramă de clasă în limbajul unificat de modelare (UML) este un tip de diagramă de structură statică care descrie structura unui sistem prin clase, atributele acestora operațiile sau metodele și relațiile dintre obiecte.

Clasele principale în Django sunt denumite view-uri. Potrivit documentației Django, un view este o funcție apelabilă care preia o solicitare și returnează un răspuns. Acesta poate fi mai mult decât o simplă funcție, iar Django oferă un exemplu al unor clase care pot fi utilizate ca view. Mai jos puteți observa clasele principale, atributele și relațiile dintre acestea.

Având în vedere complexitatea proiectului, diagrama de clase completă în ceea ce privește modulul de date și manipularea acestora este cea de mai sus. Pentru modulul ce ține strict de procesarea naturală de limbaj, am ales o diagramă de secvențe pentru a ilustra mai bine cum utilizatorul poate interacționa cu Virtual Assistent-ul.

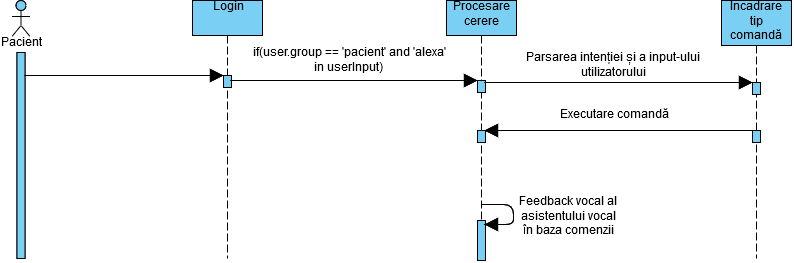
  
Figura 2.3: Diagrama de secvențe Chatbot

Diagrama de secvențe este una generală, indiferent de tipul comenzii. Astfel, cererile se pot împărți în două categorii și anume comenzi care deschid anumite aplicații, comenzi simple și comenzi complexe care interacționează mai mult cu utilizatorul pentru a afla diverse informații care sunt esențiale pentru executarea comenzii. Aici mă refer, de exemplu, la purtarea unei discuții în care pacientul spune despre cum i-a fost ziua, iar asistentul virtual poate interpreta și reveni cu un feedback bazat pe input-ul utilizatorului cu ajutorul unor algoritmi de recunoaștere a sentimentului.

Pacientul nu are acces din păcate la o interfață cu utilizatorul demnă de așa aplicație fiindcă nu mă pricep așa bine în JavaScript. În mod normal conversația între asistent și pacient ar trebui să fie vizibilă, iar log-urile clare. Astfel, diagrama de secvențe ar avea o ramură în plus, dar toate feedback-urile asistentului virtual sunt vocale, deci imediat după ce se termină de procesat cererea sau comanda, pacientul este informat.

* + 1. Tehnologia aleasă pentru implementare

Am ales ca tehnologie de implementare limbajul de programare Python împreună cu un framework desemnat interacțiunii pe web, Django. Acest mod de lucru mi-a permis să adaug o mulțime de funcționalități și o documentație vastă pe care o pot accesa și învăța.

Python pune accentul pe lizibilitatea și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca [C](https://ro.wikipedia.org/wiki/C)/C++. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul *object-oriented*, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu „gunoier” (*garbage collector*). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

Multe din bibliotecile sau API-urile folosite în dezvoltarea acestei aplicații sunt disponibile doar în acest mediu de programare. A fost nevoie de cunoștințe detaliate privind implementarea acestora folosind framework-ul Django, mai specific, a trebuit să găsesc metode și să îmi creez propriul design ca majoritatea API-urilor și a librăriilor precum și conceptele de prelucrarea naturală de limbaj să pot fi folosite într-un mediu web.

Django este un framework pentru dezvoltarea aplicațiilor web gratuit și cu sursă deschisă, scris în Python, care urmează modelul arhitectural Model-View-Controller.

Scopul principal al acestui soft cadru pentru dezvoltarea aplicațiilor web este de a facilita crearea de website-uri complexe, fundate pe baze de date. Django pune accent pe reutilizarea codului, pe modularitate, dezvoltare rapidă a site-urilor web, ghidându-se după principiul "nu te repeta" (en. Don't repeat yourself - DRY). Django este codat de la un capăt la altul în Python, chiar și fișierele de configurare și modelele de date sunt implementate în acest limbaj de programare. Django oferă și un panou administrativ, care, deși vine preinstalat, este opțional, prin intermediul acestuia se pot crea, citi, actualiza și șterge cu ușurință informații din baza de date. Acest panou de adminstrare este generat dinamic prin introspecție (prin analizarea tabelelor din baza de date) și poate fi ușor configurat prin modelele administrative de date.

Câteva website-uri bine cunoscute care utilizează Django sunt Pinterest, Instagram, Mozilla, The Washington Times, Disqus, Public Broadcasting Service și Bitbucket.

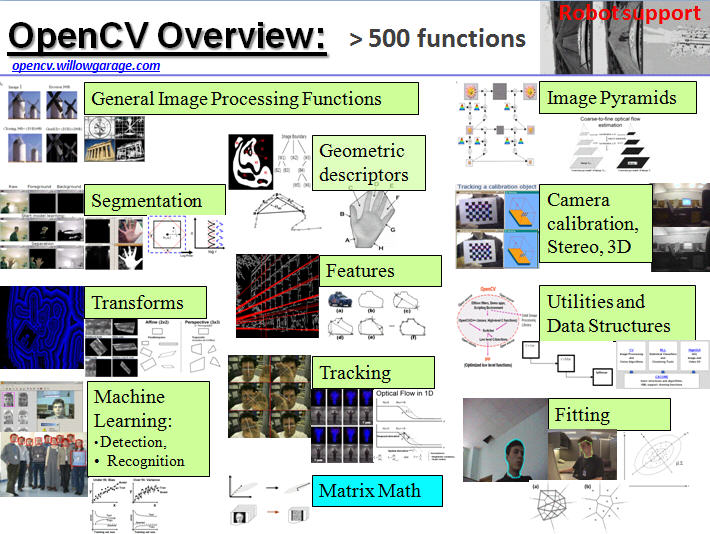
Mai sus puteți vedea câteva dintre caracteristicile cheie ale acestui framework, iar în mare parte reprezintă motivarea alegerii acestei metode de implementare. Conform DataFlair, autorul imaginii și al unui articol amănunțit14, aceste trăsături ale framework-ului sunt de-a dreptul unice.

  
Figura 2.4: Caracteristici cheie Django

Documentația foarte amănunțită și ușor de înțeles, poate fi folosit într-un limbaj de programare ușor și de asemenea foarte documentat (Python), faptul că modulele de securitate în ceea ce privesc utilizatorii sunt foarte bine implementate și ușor accesibile, pot crea utilizatori și clase derivate din modulul de bază „User” tocmai pentru a păstra nivelul de securitate foarte ridicat, acești factori sunt cei care m-au convins să aleg Django în ceea ce privește aplicația mea.

În privința modulului ce ține de procesarea de limbaj am ales librăria NLTK (Natural Language ToolKit) ce conține toate metodele de care am nevoie pentru a parsa în mod eficient input-ul utilizatorului. De asemenea, pentru procesul de învățare automată am inclus librăria TensorFlow, iar mai specific modulul keras ce include modele, layere și optimizatoare pentru a-mi crea modelul capabil să învețe dintr-un fișier json cu intenții.

Pentru cea de-a doua aplicație, cea utilitară, am ales să fie implementată utilizând OpenCV. Potrivit Wikipedia, OpenCV (vedere computerizată cu sursă deschisă) este o bibliotecă de funcții informatice specializată pe vedere computerizată în timp-real. Elaborată inițial de Intel, a fost dezvoltată ulterior de Willow Garage, apoi de Itseez (care a fost achiziționată mai târziu de Intel). OpenCV are peste 500 de funcții și o gamă foarte largă de aplicabilitate după cum se poate observa și în figura de mai jos, imagine preluată dintr-un articol scris de cei de la skyose15.

  
Figura 2.5: Caracteristici cheie OpenCV

Am ales acest program pentru a implementa cea de-a doua aplicație tocmai din cauza ușurinței și a lizibilității codului. Deși am avut dificultăți în a procesa imaginile, documentația a fost tot timpul arhisuficientă și am reușit să învăț foarte multe lucruri despre lucrul cu imaginile.

* + 1. Descrierea claselor implementate

Sunt două module principale, iar la fiecare dintre ele vă voi prezenta succint ce face fiecare clasă.

Modulul de antrenare a robotului conversațional este prezentat în anexa 4 și se ocupă cu crearea unui model cu ajutorul bibliotecilor de procesare naturală de limbaj (nltk) și de învățare automată (tensorflow). Acesta este un modul de sine stătător și creează un model pe care îl folosesc în fișierul text\_processing.py descris în anexa 10. Încep prin a parsa intențiile din fișierul cu intenții și să creez listele pe care le voi folosi la generarea modelului. Aplic un algoritm de lemmatizare (aduc la forma de bază cuvintele) pe toate cuvintele dacă acestea nu sunt caractere speciale și simultan, folosind metoda set, înlătur toate duplicatele. La fel fac și pentru clasele acestora (patterns). Pentru fiecare document pun toate cuvintele lemmatizate într-un „sac”, termen folosit în parsarea naturală a limbajului și creez lista de antrenare. Având toate acestea puse la punct, încep să creez modelul secvențial la care adaug multiple straturi activate de funcția de activare „relu” (rectified linear unit) după care încadrez între 0 și 1 probabilitățile ca o intenție să fie corectă, iar cea mai mare valoarea reprezintă predicția entității artificiale.

Cel de-al doilea modul care se ocupă cu procesul de înregistrare, logare și relațiile dintre utilizatori, dar și de raportarea statusului pacienților este unul mult mai complex decât modulul anterior. Django oferă o multitudine de clase chiar de la început, ce trebuiesc implementate pe parcursul dezvoltării aplicației. Acest modul conține la rândul său două aplicații Django. Prima se numește speech\_bot și are ca scop integrarea asistentului virtual alături de baza de date și funcționalitățile website-ului, funcționalități care sunt implementate în cea de-a doua aplicație care se numește tutorial. Denumirea celei din urmă indică faptul că am urmat diverse tutoriiale pe care le-am dezvoltat ulterior și mi-am construit proiectul de licență pe acestea. Clasele principale în care se găsește logica din backend sunt denumite views.py și în continuare vă voi descrie sumar principalele secvențe de cod din acestea.

În primul rând, Django are nevoie de propriul environment. Asta înseamnă că își creează de la bun început toate clasele necesare dezvoltării de aplicații. Pe lângă aplicații există un folder cu diferite clase care au ca scop managementul aplicațiilor. De exemplu în fișierul urls.py se scriu toate rutele url ale aplicațiilor.

Aplicația de bază denumită „tutorial” reprezintă modul de utilizare a site-ului de către un utilizator. Acesta poate să se înregistreze pe site, să se logheze și să vizualizeze anumite pagini în funcție de accesul pe care-l deține. Astfel, am restricționat accesul la pagina de înregistrare dacă ești deja logat, la fel și pentru pagina de login. Pentru înregistrarea unui pacient se va apela o funcție de callback denumită pacient\_create\_view. Aceasta apelează și înglobează două chestionare în pagina web și parsează rezultatele (vezi anexa 5). Astfel, verific dacă chestionarele sunt valide și incrementez cu 1 numărul pacienților pe care un tutore sau un asistent îl are. Salvez toate aceste date în baza de date și ader în grupul pacienți pacientul abia introdus. Această pagină poate fi accesată doar de tutore sau asistenți și doar ei pot adăuga pacienți noi, introducând datele personale ale acestora pentru o obiectivitate crescută.

O altă metodă asemănătoare celei de mai sus este pacient\_general\_form\_view care preia rezultatele unui formular destinat pacientului și calculează un rating pentru a afla situația inițială. La fel ca mai sus, stochez toate aceste date în baza de date și redirecționez utilizatorul la pagina cu asistentul virtual.

Pentru a păstra logica între diferitele clase de utilizatori, iar aici mă refer la asistent, pacient și administrator, am creat un fișier numit decorators.py (vezi anexa 6) care are drept scop setarea restricțiilor asupra unui grup de utilizatori. O astfel de logică contribuie la optimizarea aplicației, rolurile și interacțiunile dintre acestea fiind foarte bine definite. Fiecare metodă definită în acest modul conține o funcție de sine stătătoare numită wrapper function sau funcție învelitoare care returnează un răspuns HTTP sau un redirect spre pagina principală în caz că utilizatorul nu are suficiente privilegii sau o redirectare asupra paginii căutate dacă toate cerințele sunt îndeplinite.

O metodă mai complexă o reprezintă pagina web care ar trebui să fie accesată o dată la două săptămâni pentru a completa un chestionar de stare al pacientului. Aceasta metodă preia boala pacientului și în funcție de aceasta îi dă un chestionar la care se calculează un punctaj, iar acest punctaj ar trebui să fie unul din ce în ce mai bun. Folosesc o fabrică de chestionare care îmi returnează un chestionar specific bolii pacientului, iar dacă chestionarul este valid, calculez rating-ul în baza unor formule regăsite pe anumite site-uri guvernamentale și oficiale cu tematică asemănătoare. Metoda creation\_factory stochează separat rezultatele pacienților și trimite mail asistenților cu ultimele noutăți legat de pacienții lor alături de o interpretare a acesttora.

Chestionarele sunt definite într-un fișier separat numit forms.py (vezi anexa 7) în care am diferite metode care sunt modelate după modelele din fișierul models.py. Astfel, o metodă conține toate câmpurile necesare unui chestionar, iar în view pot schimba logica, reține date sau modela datele, iar după să le pot introduce în baza de date. Fiecare chestionar apelează la anumite structuri de date ce conțin câmpurile aferente, în cazul de față mă folosesc de liste predefinite. Sunt unele excepții în schimb în care utilizatorul trebuie să introducă numere sau string-uri în căsuțele chestionarului. Am creat chestionare specifice bolilor Alzheimer și depresie, dar și chestionare hibride ce îmbină întrebări pe care le adresez tuturor pacienților împreună cu un set de întrebări specifice afecțiunii acestora.

Modelele după care funcționează toată aplicația tutorial sunt descrise în anexa 8. Aici mă folosesc de modelul principal al celor de la Django, modelul User de la care moștenesc atributele și îmi creez propriile modele. Astfel, asistenții se încadrează în mod prestabilit în modelul User, iar pacienții și tutorele aduc câmpuri în plus modelului User precum afecțiune, vârsta și diverse chei străine. Fiecare parsare inteligentă a informației are propriul model în care se încadrează tocmai pentru a putea accesa mai facil detalii cheie ale utilizatorului.

În cele din urmă, fișierul urls.py prezentat în anexa 9 prezintă rutele URL-urilor aplicației. Fiecare metodă descrisă în views.py returnează cererea HTTP, pagina web care conține logica și un dicționar cu datele stocate în interiorul view-ului pentru a le putea afișa în front-end. Astfel, în urls.py există path-uri care duc la diferite URL-uri care apelează view-urile aferente acestora și opțional pot avea un nume pentru a le putea apela mai ușor în pagina HTML.

În cea de-a doua aplicație, speech\_bot sunt două clase principale, cea care se ocupă cu interfața cu utilizatorul, una simplistă, și cea care are ca scop interacțiunea dintre virtual assistant și pacient descrisă în anexa 10. În metoda speech\_to\_text din views folosesc biblioteca Google Text to Speech și Speech Recognition pentru a valida input-ul utilizatorului (input vocal și nu alte sunete). Dacă nu intră în nicio excepție algoritmul, atunci pornesc un thread separat care se ocupă cu modulul de asistență vocală. Acest modul apelează cea de-a doua clasă, iar anume text\_processing.py în care am principalele caracteristici ale asistentului virtual. Aici metoda assistant\_speaks are scopul de a utiliza librăria Google Text to Speech și a furniza un mp3 cu răspunsul asistentului virtual. Metoda search\_web caută pe web diferite informații sau interogări. De exemplu, pacientul poate deschide Youtube sau să caute un articol pe Wikipedia sau pur și simplu să navigheze pe internet. Metoda open\_application deschide o aplicație sau browser și se folosește de librăria internă os (capabilă să pornească diverse aplicații). Metoda bow returnează un vector în care se regăsesc valorile 0 dacă în cuvintele din intents nu se găsesc cuvintele utilizatorului și 1, invers. Metoda predict\_class are ca parametru mesajul și modelul și prezice un intent și un mesaj aleator bazat pe input-ul utilizatorului. Metoda get\_audio preia cuvintele rostite de utiilizator și se folosește de librăria speech recognition. Metoda parse\_movies redă un top 3 al filmelor în funcție de preferințele utilizatorilor. Metoda process\_text procesează textul în funcție de cuvintele utilizatorului și este folosită de asistentul virtual pentru a acționa și a activa diferite module.Printre acestea se enumeră recomandarea filmelor, crearea de evenimente folosind Google Calendar, modulul de asistență vocală care permite pacientului o discuție liberă cu utilizatorul, informarea acestuia cu cele mai recente știri, poate să caute pe web sau să deschidă diverse aplicații, etc.

Modulul de procesare de text începe cu o variabilă declarată global care are scopul de a fi numele fișierului mp3 care rezultă după input-ul utilizatorului, mai explicit este răspunsul asistentului virtual. Astfel salvez output-ul asistentului într-un fisier numit număr.mp3 unde număr reprezintă valoarea incrementată de-a lungul interacțiunii asistentului cu robotul după care redau acest fișier mp3, iar în final sterg fișierul pentru a nu ocupa spațiu.

În metoda search\_web mă folosesc de pachetul selenium, mai specific librăria webdriver pentru controlul ferestrelor web. Maximizez fereastra web și în funcție de cuvântul regăsit în input-ul utilizatorului, rețin indexul cuvântului youtube și construiesc un query pentru cuvintele care urmează după acesta, iar după apelez metoda driver.get pentru a deschide acel url salvat sub formă de string. Fac la fel și pentru wikipedia sau google.

Pentru a deschide diverse aplicații, folosesc pachetul os care interacționează direct cu sistemul utilizatorului și în funcție de keywordul regăsit în input, deschid aplicația din pc.

Pentru construirea anumitor query-uri sau diverse căutări pe web am decis să optimizez cuvintele din căutare trecându-le printr-un filtru de cuvinte tip stopwords și le aduc la forma de bază folosind un algoritm de lematizare. Metoda clean\_up sentence are fix rolul acesta, de a lematiza fiecare cuvânt dintr-o propoziție.

Metoda bow utilizează metoda clean\_up pentru o propoziție și pentru fiecare cuvânt din propoziție îl caut în fișierul words.pkl pentru a crea o listă în care voi avea 1 pe pozițiile în care se regăsesc cuvintele din input-ul utilizatorului. Astfel, dacă eu am 50 de cuvinte în fișierul words, lista bag va avea 50 de zerouri, iar dacă utilizatorul spune în propoziția sa 5 cuvinte care se regăsesc în fișierul words, acei indecși se vor updata și vor primi valoarea 1. Funcția asta ajută la calcularea probabilității intent-ului.

Funcția predict\_class prezice folosind modelul deja antrenat care este intenția unei propoziții. Astfel, variabila p reține lista returnată de funcția bow, cea explicată mai sus, iar variabila res conține rezultatul predicției propoziției. Fiecare rezultat îl trec printr-un threshold ales de mine tocmai pentru a nu exista erate foarte mari în condițiile în care propoziția spusă de utilizator are probabilitate foarte mică de a se regăsi într-un intent. Astfel, după ce filtrez aceste probabilități, le ordonez descrescător și le adaug într-o listă.

Metoda getResponse alege un mesaj oarecare din intenția cu cea mai mare probabilitate și îl returnează în metoda chatbot\_response care asamblează toate funcțiile de mai. Astfel, această metodă returnează răspunsul și intenția utilizatorului.

Pentru a prelua input-ul utilizatorului mă folosesc de metoda get\_audio și de microfonul acestuia și o librărie numită speech recognition. Pentru a utiliza această librărie, utilizatorul trebuie să vorbească limba engleză, iar recunoașterea cuvintelor din input o fac cu ajutorul altei librării dezvoltate de Google. Dacă intenția utilizatorului este de a părăsi programul în orice stadiu s-ar afla, fie că ascultă știrile sau ascultă o melodie, opresc execuția acestuia.

Metoda parse\_text este o metodă ajutătoare care parsează textul preluat de la un articol de pe Wikipedia. Astfel, mă folosesc de anumite cuvinte care identifică în mod corect tema pe care vrea utilizatorul să o cerceteze, iar API-ul celor de la Wikipedia îmi va returna cel mai relevant rezultat și folosind metodele descrise mai devreme, mai exact assistant\_speaks, redau rezumatul acelui articol precum și titlul.

În ceea ce privește aplicația utilitară descrisă în anexa 11, am definit static dreptunghiurile care formează o schiță de apartament 2D. Primul pas a fost să desenez o imagine neagră, folosind vectori numpy și am pus peste dreptunghiurile care formează planul apartamentului. După aceea m-am folosit de o funcție de callback care se invocă atunci când dau click. În interiorul funcției am tratat diferite cazuri. Astfel, când utilizatorul dă click dreapta, setez originele cercului și pentru fiecare cerc creez în funcția draw\_circles linii drepte care indică raza de acoperire a gateway-ului și atașez imaginea în centrul cercului. Pentru a vedea dacă o linie se intersectează cu un contur ce reprezintă un perete, am folosit funcția check\_contours care îmi returnează numărul de pereți prin care trece linia aferentă unui cerc.

Pentru click stânga, funcția de callback desenează un senzor și în funcție de raza gateway-ului acesta este colorat în roșu sau în verde, fapt ce indică dacă acesta este în raza de acoperire sau nu. Funcția check\_senzor verifiică dacă senzorul este în aria de acoperire sau nu, iar funcția draw\_walls desenează pereții încăperii după ce trasez liniile aferente cercului.

Pentru a șterge toate elementele am mai adăugat o funcție de callback pentru rotița de la mouse, așadar dacă o apăs, listele cu senzorii și cu cercurile devin vide.

Și nu în ultimul rând, pentru a comuta între senzori folosesc rotița de la mouse.

1. Implementarea aplicației
   1. Descriere implementării

Am conceput această aplicație ca fiind una web de sine stătătoare. Cel mai util framework web este Django, din punctul de vedere al funcționalităților sale și a securității. Astfel, am început prin a crea o simplă interfață cu utilizatorul, folosind bootstrap pentru un aspect mai plăcut la care am adăugat modulele de înregistrare și logare, iar pentru un plus de securitate, am moștenit clasa User implementată de Django și am creat profilul asistent sau tutore și pacient. Acești utilizatori au acces la diferite pagini în funcție de grupul din care fac parte. Nivelul de acces și logica din spate am defint-o într-un fișier numit „decorators.py”. Cei care fac parte din grupul de tutore sau asistenți pot adăuga maxim cinci pacienți, dar se poate interveni cu o logică, tutorii sau îngrijitorii pacienților să aibă maxim doi pacienți, iar asistenții medicali să poată avea în grijă un număr mai mare de pacienți. Acesta este obligat să introducă datele pacientului precum și un nume de utilizator și parola datorită faptului că obiectivitatea și autenticitatea datelor sunt esențiale. După ce completează formularul, pacientul se poate loga și accesa diferite pagini ale site-ului. Primul chestionar pe care trebuie să-l completeze pacientul este unul hibrid, ce conține întrebări generale, dar și despre statutul afecțiunii sale. Acest chestionar va oferi un indice cu privire la starea inițială a pacientului. După ce a fost completat, chestionarul va deveni inaccesibil, dar un altul va trebui completat o dată la două săptămâni pentru a monitoriza eficient starea acestuia. Cel de-al doilea formular va genera un raport o dată la 3 completări ale acestuia, adică la 6 săptămâni un raport complet va fi trimis pe mail-ul tutorelui sau asistentului care are grijă de pacient. Împreună cu acest raport, tutorele va putea analiza datele și progresia sau regresia afecțiunii pacientului împreună cu interpretarea rezultatelor.

Între timp, pacientul poate accesa modulul de asistență vocală pentru a-și alina singurătatea. Acesta folosește tehnici de procesare naturală de limbaj. Astfel, input-ul vocal al utilizatorului îl preiau cu ajutorul unei librării numite speech\_recognition și îl analizez. Pentru ca robotul să asculte comenzile utilizatorului, acesta trebuie să spună la început keyword-ul „Alexa”. Pentru a recunoaște textul am folosit o metodă a aceleiași librării și anume recognize\_google. Textul din păcate trebuie să fie obligatoriu în limba engleză. Acest modul este compus și el de asemenea din două părți. Una în care antrenez robotul cu diferite intenții și răspunsuri și cea de testare care practic este interacțiunea utilizatorului cu acesta. Modulul de antrenare îmi generează un fișier tip model care va fi încărcat pe server. Se vor face request-uri la acesta atunci când trebuie să prezică intenția utilizatorului, iar serverul va oferi un răspuns vocal sub forma unui mp3.

Pentru a prezice intenția utilizatorului am folosit concepte de prelucrare naturală de limbaj, iar mai specific, am un model pe care îl antrenez cu datele din fișierul cu intenții în care se află anumite tag-uri care reprezintă intenția și mai multe răspunsuri. Modelul îl antrenez și îl salvez pe server folosind o rețea neuronală care tokenizează cuvintele din intenții și practic, prelucrez input-ul ce-mi va deservi la antrenarea modelului. După ce am procesat cuvintele le trec printr-un algoritm de lemmatize care le aduce la forma de bază, salvez clasele și cuvintele în fișiere tip pickle pentru a le putea accesa mai târziu și din alte module și încep antrenarea modelului. Modelul este unul de tip secvențial, adică am o rețea neuronală în care voi activa neuronii de la stânga la dreapta. Adaug un prim strat dens care are în componență 128 de neuroni și ca funcție de antrenare folosesc rectified liniar unit care practic, valorile negative vor fi compresate și vor indica 0, iar valorile pozitive le va reduce pe o scară de la (0,1]. Adaug un strat care are un hiperparametru numit dropout care adaugă o nouă interpretare asupra rețelei neuronale prin scoaterea unor neuroni din funcțiune și restabilind conexiunile. Fac acest lucru pentru a mări capacitatea modelului de a prezice noi intenții. Prin experimente succesive am constatat că cele mai bune rezultate le am atunci când am trei straturi neuronale, primele două au ca funcție de activare relu, iar ultima are ca funcție de activare softmax ceea ce compresează rezultatele între 0 și 1, ceea ce înseamnă că toate intențiile vor avea suma probabilităților maxim 1, iar cea cu cea mai mare probabilitate va fi predicția modelului. Între aceste straturi am alte două straturi cu hiperparametrul dropout pentru a facilita predicția modelului și a reduce overfittingul. Documentându-mă mai mult, am observat că unele aplicații care au la bază un chatbot folosesc Stochastic Gradient Descent cu gradientul accelerat Nesterov și am decis să compilez modelul folosind aceeași metodă. Numărul de epoci este unul relativ, acesta dând rezultate foarte bune atunci când este cuprins între 150 și 250.

În ceea ce privește aplicația utilitară, implementarea acesteia este una mult mai matematică. Astfel, datorită faptului că multe apartamente sau case au un plan 2D, am decis să construiesc eu propriul plan al unui apartament în OpenCV, acesta fiind format din multiple dreptunghiuri cu coordonate date. Încep prin a afișa aceste dreptunghiuri, iar interacțiunea cu utilizatorul se realizează prin mouse events. Astfel, atunci când utilizatorul apasă click dreapta, acesta plasează un gateway care are o anumită rază de acțiune sub forma unui disc. Această rază poate fi micșorată de pereții cu care intră în contact, fapt ce influențează conectarea unui senzor la acesta. Pentru a schimba senzorii utilizatorul se poate folosi de rotița de la mouse, iar pentru a șterge elementele de pe schiță, utilizatorul trebuie să apese pe rotiță. Toate datele ce țin de grosimea pereților sau de ariile de acoperire ale gateway-urilor sunt variabile. Cel care va folosi aplicația va trebui să facă niște cercetări în prealabil pentru a afla cu câți decibeli atenuează un tip de perete și să consulte specificațiile router-ului sau al gateway-ului pentru a vedea aria de acoperire. Astfel, după ce plasez un router, creez un cerc invizibil la coordonatele x și y ale click-ului după care trasez razele cercului cu linii verzi ce reprezintă aria de acoperire. Lista de senzori poate fi și ea customizată în funcție de dorințele utilizatorului, iar la click stânga plasez un senzor ce va avea în loc de culoarea neagră a pictogramei, roșu dacă nu este în aria de acoperire a niciunui router sau verde dacă este în proximitatea unuia.

* 1. Dificultăți întâmpinate

În primul rând, una dintre problemele pe care le-am avut la început a fost înțelegerea framework-ului Django. Acesta fiind o noutate pentru mine, a trebuit să mă documentez mult atât din tutoriale pe YouTube cât și din propria lor documentație. Astfel, am reușit să leg conexiunile necesare între asistent, pacient și administrator.

O altă provocare a fost moștenirea modelelor menționate mai sus direct de la modelul User care este implementat și securizat de Django. Faptul că trebuia să creez conexiunile doar într-o anumită manieră în SQLite, folosind legături OneToOne și ștergerea în cascadă a utilizatorilor au fost noțiuni care cu greu mi-am dat seama ordinea și importanța acestora.

Chestionarele au fost și ele un punct sensibil mai ales fiindcă nu știam cum pot îmbina două chestionare într-unul și să stochez în tabele diferite rezultatele sau cum să le pot face accesibile doar unui grup de utilizatori. Însă cel mai greu a fost să preiau datele unui individ ce moștenește clasa User ce aparține unui grup și să le updatez cu un set nou de date rezultat în urma completării unui chestionar care are câmpuri în plus față de User. A trebuit să trec prin toată documentația Django ca să aflu astfel de noțiuni.

Nu există la momentul actual (din câte m-am informat și am căutat eu) module de inteligență artificială, iar mai specific mă refer la ramura de procesare naturală de limbaj care să fie online, de sine stătătoare, accesibile la un click distanță. Există chatbots mult mai inteligenți, dar care nu interacționează cu utilizatorul și nu rețin date despre aceștia în baza lor de date. Sunt făcuți doar pentru o simplă interacțiune. Integrarea unui modul vocal pe un server a reprezentat de departe cea mai mare provocare din punct de vedere arhitectural, chit că am acces la toată documentația disponibilă pe internet, cu greu am reușit să pun la punct un design capabil să ruleze în mediul online cu module de inteligență artificială și să fie antrenat în mod constant cu noi date. Faptul că Django nu oferă niciun suport pentru acest tip de interacțiune a fost un punct în minus față de Flask care asigură compatibilitate între Alexa și acesta. Ideal ar fi fost să pot integra modulul Alexa în aplicația mea, dar termenii și condițiile ale celor de la Amazon nu permit reverse engineering, așa că am creat propria entitate artificială, iar conexiunea acesteia cu Django se realizează într-un thread diferit.

În ceea ce privește aplicația utilitară principala problemă a fost design-ul logic urmat de matematică. Prima provocare a fost găsirea formulei matematice corecte de generare a razelor cercului calculate sub formă de radiani în funcție de cele 4 cadrane. Alte provocări au constat în generarea unor cercuri invizibile care au culoarea verde pentru a putea încadra senzorii în aria acestora de acoperire sau nu. Pentru asta am schimbat culoarea neagră ce aparține unor iconițe cu verde, implicit roșu pentru a indica dacă se află sau nu în cercul verde invizibil. În locul acestui cerc eu trasez doar raze verzi din centrul cercului (x,y) până la extremitatea sa. O altă problemă a fost de logică, iar aici mă refer la modul de construcție a schemei, cum să șterg elementele și ce imagini să suprascriu.

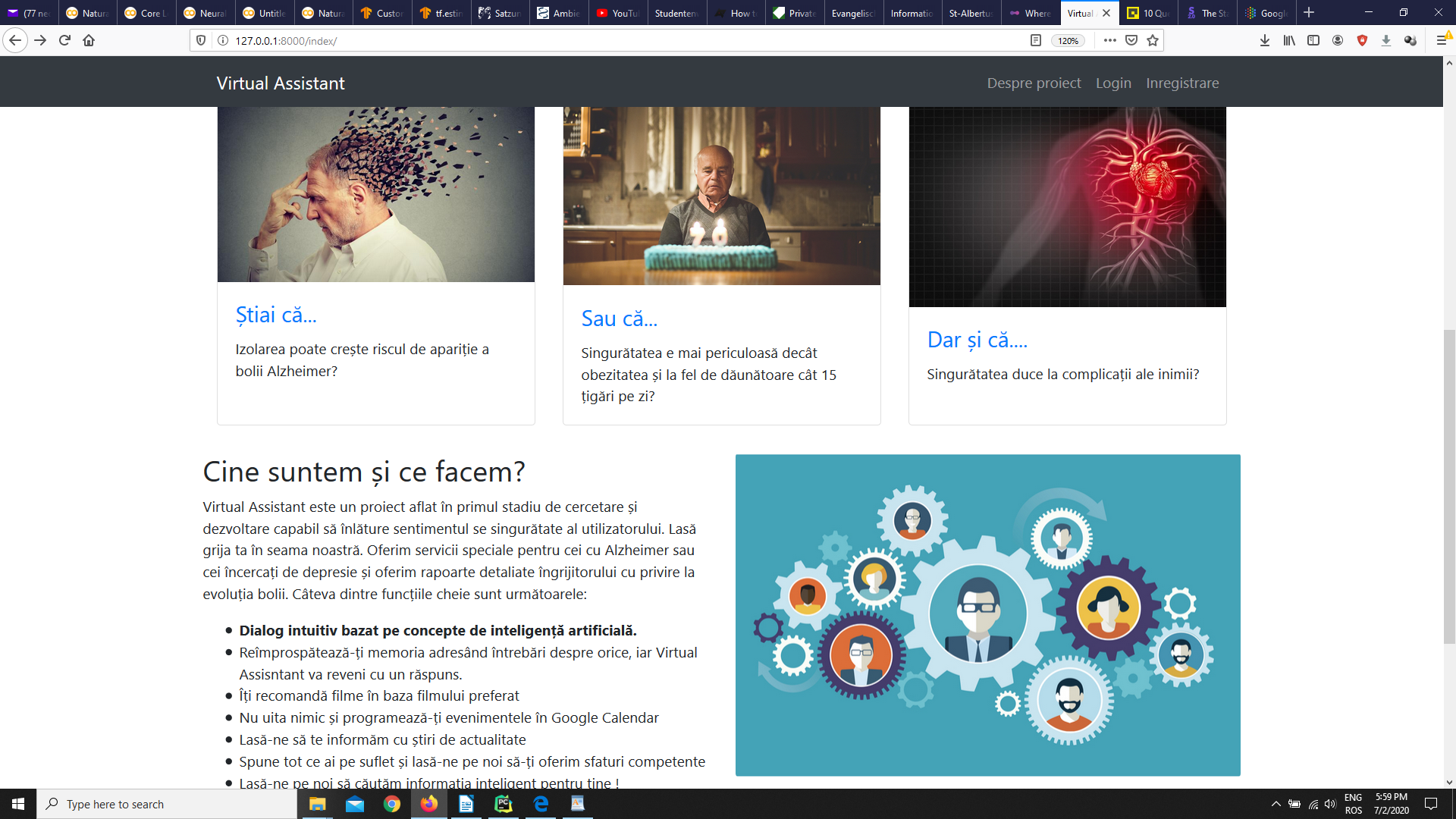
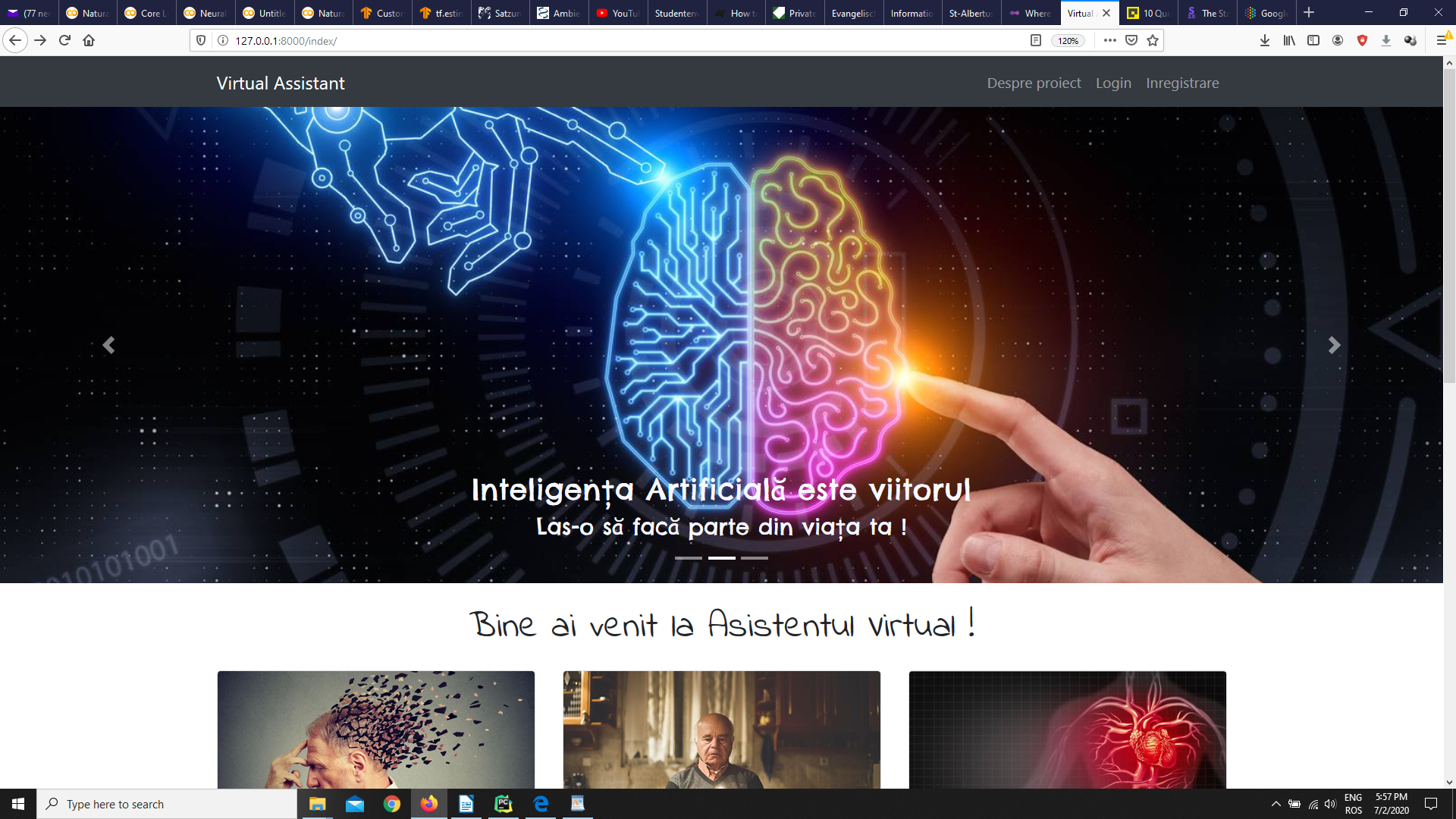
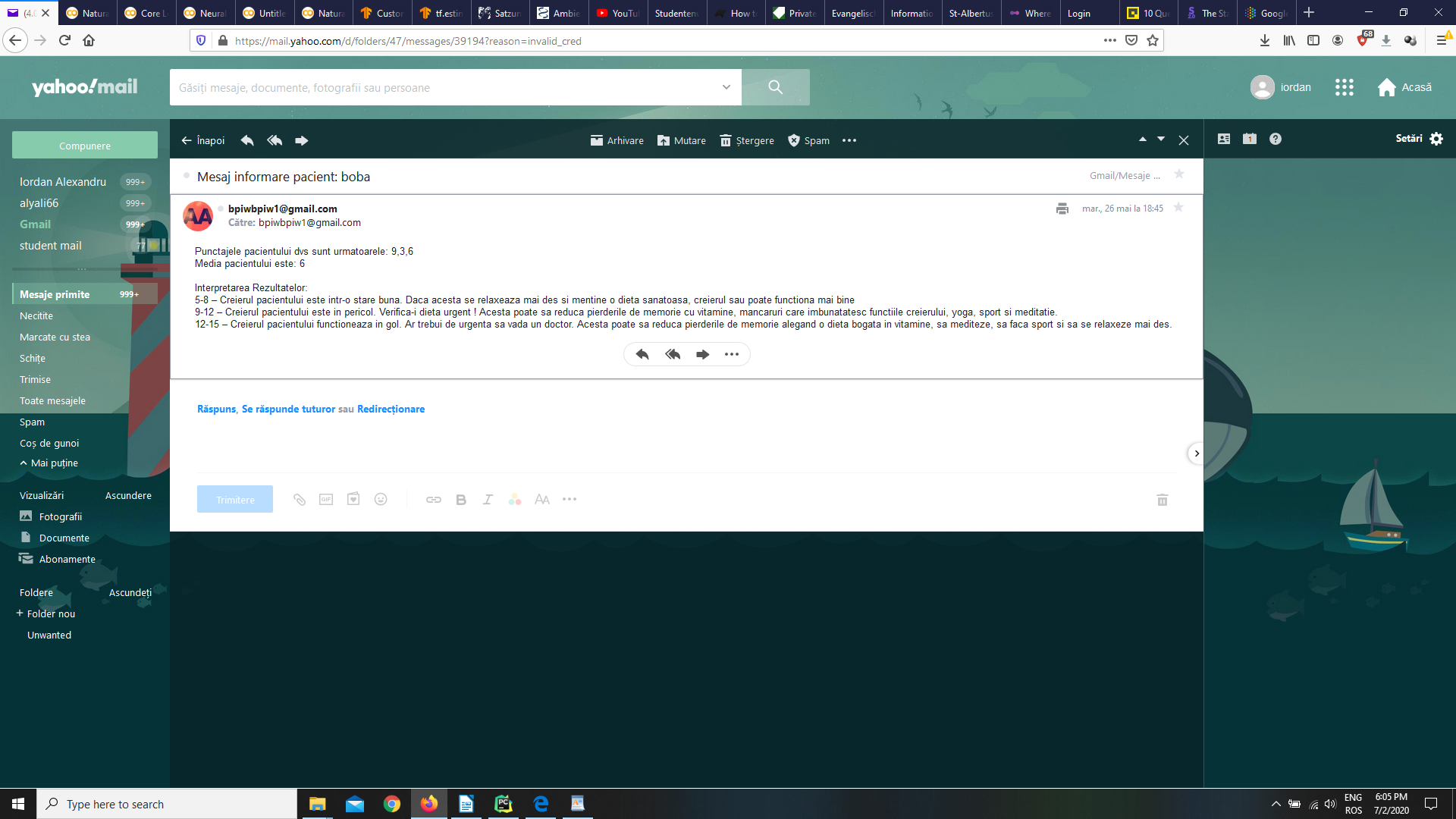
* 1. Idei originale

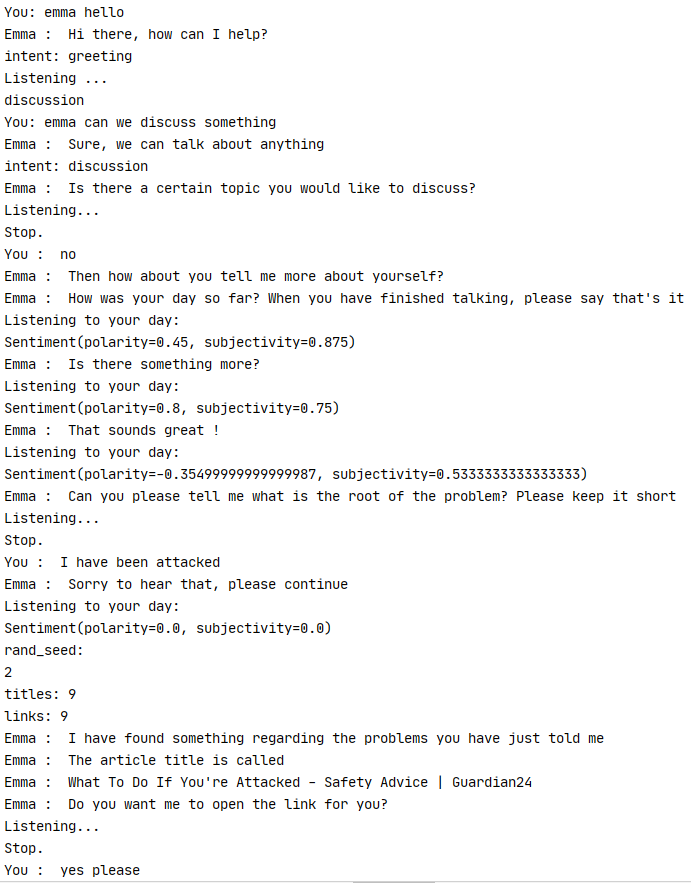
În primul rând, toată aplicația constă într-o idee originală. Faptul că singurele firme care se apropie de ideea de bază ce stă în spatele aplicației sunt niște giganți în domeniul IT, iar aici mă refer la Amazon și Google, unde numai la Amazon Alexa lucrează 10 mii de oameni, înseamnă că proiectul meu de licență este unul actual, capabil să concureze aceștia într-o mică măsură și să vină cu concepte noi în ceea ce ține de relația dintre om și entitate artificială.

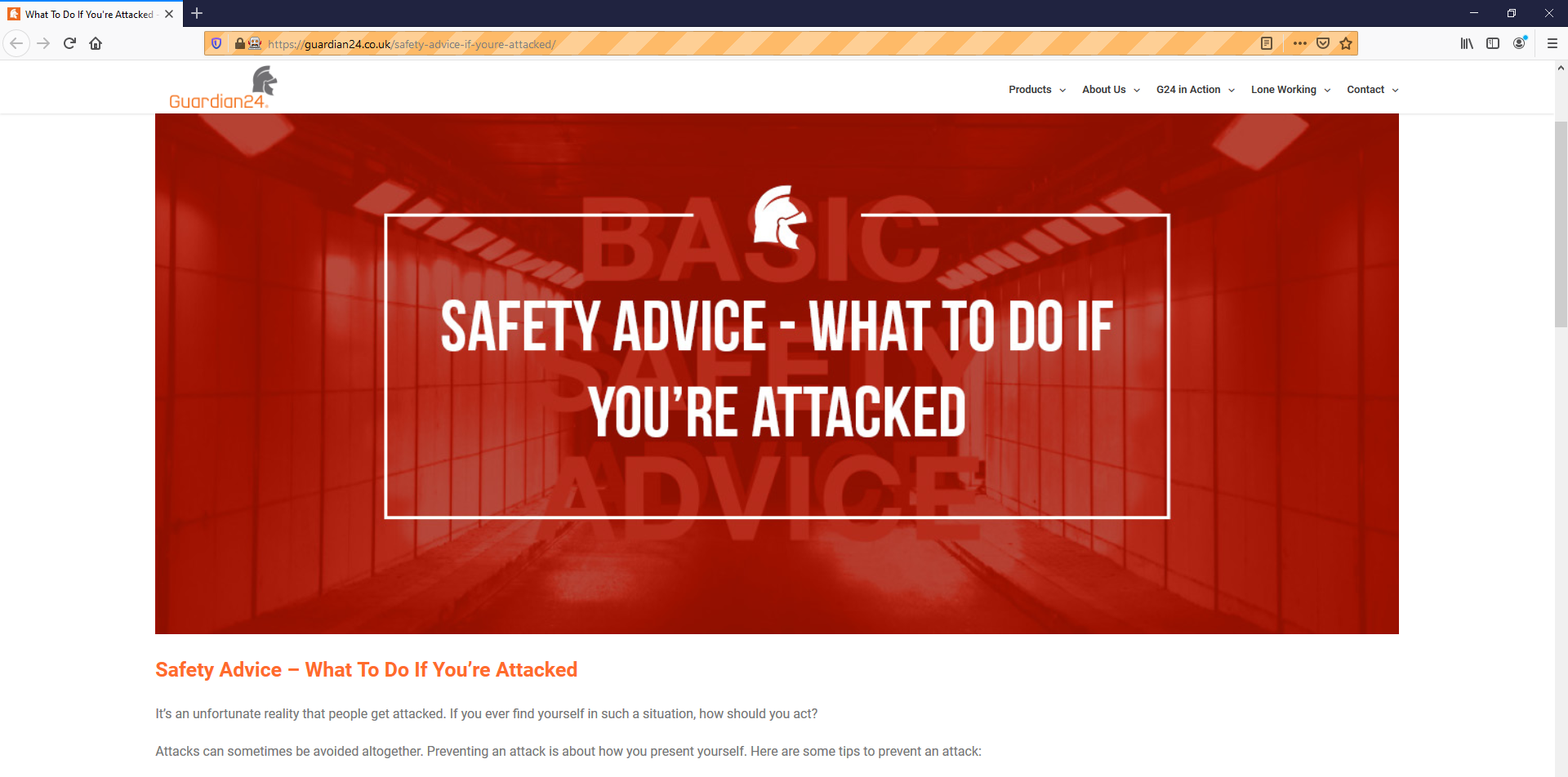
Amazon Alexa și Google Assistant vin la pachet cu device-uri pe care să ruleze, de exemplu Echo Dot, boxe inteligente sau alte sisteme precum Raspberry PI. Acest website permite tuturor oamenilor cu o conexiune la internet să acceseze module asemănătoare cu cele regăsite pe Amazon Alexa sau Google Assistant. Singura diferență majoră și totodată un punct slab al acestei aplicații o reprezintă lipsa de date de antrenare, astfel pentru formulări mai complexe, robotul este indus în eroare și revine la starea inițială. Pentru a se putea evita acest inconvenient, ar trebui introduse concepte asemănătoare skill-urilor ale lui Amazon Alexa. Aceste module pot fi de sine stătătoare și să fie declanșate doar la anumite intenții ale utilizatorului. Eu n-am creat proiectul în această manieră, ci într-una mai ineficientă și am inclus toate tipurile de intenții în același fișier formatat tip json.

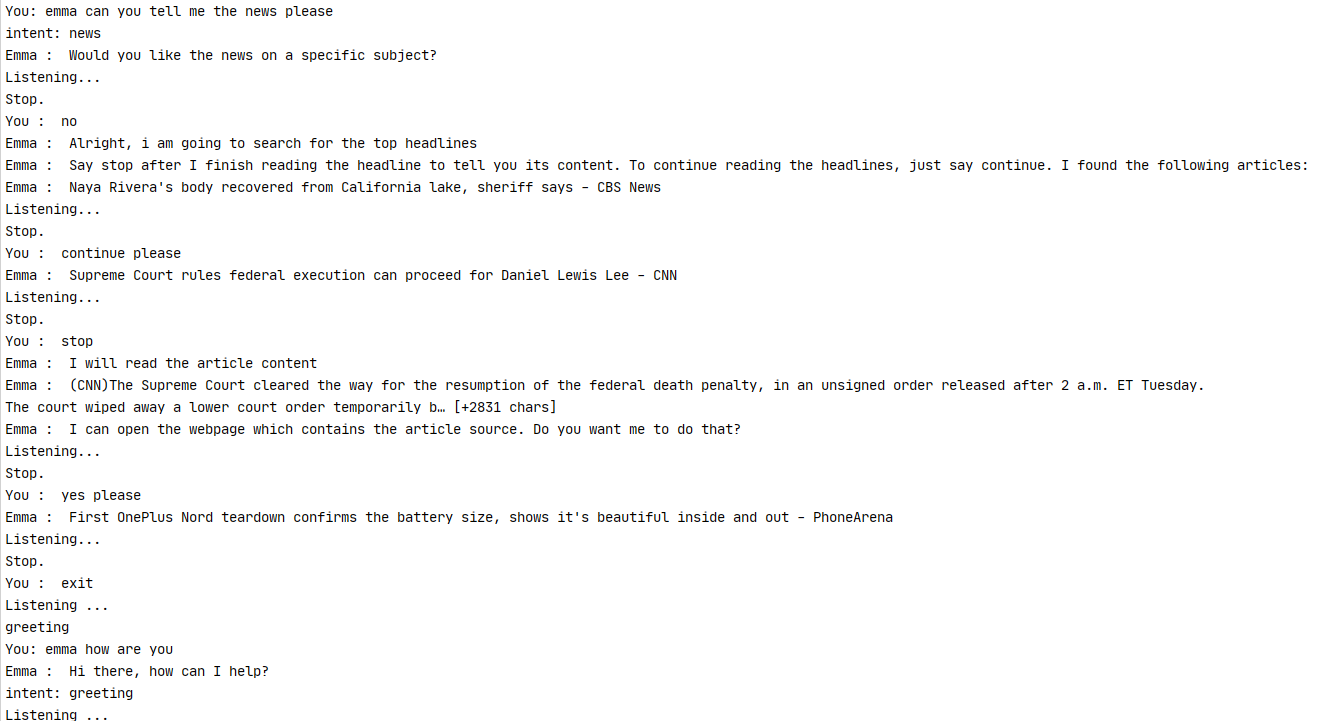
O altă idee originală o reprezintă modul în care se face raportul asistentului sau îngrijitorului. O dată la două săptămâni, cam atât timp ia unui pacient să-și îmbunătățească rezultatele, acesta trebuie să completeze un formular care va fi ulterior parsat de server și trimis asistentului împreună cu interpretarea rezultatelor. Practic, în cele două săptămâni ar trebui să se vadă o îmbunătățire dacă pacientul interacționează atunci când simte el nevoia cu Virtual Assistant.

Asistentul Virtual ar trebui să poată găsi o modalitate inteligentă de comparare a rezultatelor chestionarelor cu un rezultat intern al dialogurilor purtate între entitatea artificială și pacient, astfel încât să nu fie căi simple de a păcăli chestionarul. O astfel de parsare inteligentă a rezultatelor ar putea fi implementată cu ajutorul unor întrebări puse la anumite momente, acestea repetându-se tocmai ca utilizatorul să poată răspunde cât mai sincer.

* 1. Funcționalitatea sistemului
     1. Interfața cu utilizatorul
     2. Exemplu de raport primit pe mail
     3. Exemplu de *i*nteracțiune cu *Virtual Assistant*

  
Figura 3.1: Interacțiune pacient

  
Figura 3.2: îndrumare pacient

  
Figura 3.3: Redare știri externe

* + 1. Exemplu rulare aplicație utilitară

  
Figura 3.4: Setare două gateway-uri și 4 senzori

1. **Testarea aplicației și rezultate experimentale**
   1. Lansarea aplicației

Pentru a putea lansa aplicația este nevoie ca următoarele biblioteci să le aveți instalate:

|  |  |
| --- | --- |
| **Numele bibliotecii** | **Versiunea Bibliotecii** |
| gTTS | 2.1.1 |
| selenium | 3.141.0 |
| SpeechRecognition | 3.8.1 |
| nltk | 3.4.5 |
| Django | 3.0.1 |
| numpy | 1.18.1 |
| pandas | 1.0.3 |
| wikipedia | 1.4.0 |
| tmdbv3api | 1.6.0 |
| datefinder | 0.7.0 |
| newsapi\_python | 0.2.6 |
| google\_auth\_oauthlib | 0.4.1 |
| textblob | 0.15.3 |
| playsound | 1.2.2 |
| fake\_useragent | 0.1.11 |
| requests | 2.23.0 |
| beautifulsoup4 | 4.9.1 |
| django\_crispy\_forms | 1.9.1 |
| django\_jsonfield | 1.4.0 |
| djangorestframework | 3.11.0 |
| google\_api\_python\_client | 1.9.3 |
| **newsapi** | **0.1.1** |

Următorul pas este generarea modelului. Pentru asta este nevoie de un fișier cu intenții cu extensia json care are următoarele câmpuri: tag, patterns, responses și context. Tag-ul este eticheta, pattern-ul reprezintă o listă cu ce ar putea spune utilizatorul, iar response conține tot o listă cu răspunsuri posibile. Câmpul context atribuie o descriere mai detaliată asupra etichetei.

Un exemplu de fișier json cu două intenții este următorul:

{"intents": [  
 {"tag": "greeting",  
 "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],  
 "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],  
 "context": [""]  
 },  
 {"tag": "goodbye",  
 "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],  
 "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],  
 "context": [""]  
 }

]

}

După crearea unui astfel de fișier cu cât mai multe intenții, putem începe să construim modelul. Pentru asta dăm Build la train\_chatbot.py.

Acum că avem modelul creat, e timpul să rulăm aplicația. Comanda ***python manage.py runserver*** executată în terminal este specifică mediului de dezvoltare Django care deschide un server local pe care rulează aplicația. Totodată, în consolă apare un mesaj de genul: *Starting development server at* [*http://127.0.0.1:8000/*](http://127.0.0.1:8000/)*.* Ceea ce înseamnă că aplicația noastră poate fi accesată la adresa menționată anterior.

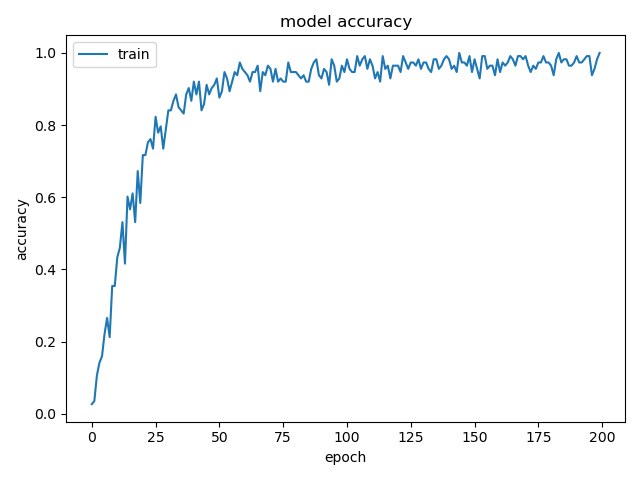
* 1. Rezultate experimentale

Potrivit librăriei tensorflow, antrenarea modelului se face utilizând placa video unde am o viteză de bandă de aproximativ 179 GB/s.



În ceea ce privește acuratețea modelului, m-am folosit de librăria MatPlotLib pentru a realiza câteva proiecții grafice cu scopul de a ilustra mai bine curba antrenării modelului.

Se poate observa că acuratețea maximă este atinsă în jurul epocii 125 din 200. Se pot face multe ajustări acestor hiperparametri astfel încât algoritmul să aibă o acuratețe cât mai mare. M-am folosit și de layere tip Dropout care deconectează anumiți neuroni împreună cu conexiunile acestora atunci când se face antrenarea tocmai pentru a reduce overfitting-ul.

  
Figura 4.1: Acuratețea modelului

În ceea ce privește loss-ul modelului, cu alte cuvinte penalizarea în cazul unei predicții proaste, putem observa că și acesta se stabilizează și tinde spre 0 începând cu epoca 80. Loss-ul este un număr care arată cât de proastă este predicția modelului asupra unui exemplu. Dacă predicția este perfectă atunci loss-ul este 0, iar dacă nu, atunci acesta crește.

Concluzii

Gradul de finalizare

La aplicația celor de la Amazon Alexa lucrează lunar peste zece mii de persoane. Ceea ce am reușit eu să construiesc în câteva luni, nu reprezintă nici pe departe aplicația în stadiul final. Virtual Assistant este un proiect în stadiul alfa, funcțional, cu bug-uri minore, dar care poate fi îmbunătățit și optimizat mult mai mult. Am reușit să duc la bun final partea de logică, de interacțiune între pacient și asistent în care pacientul completează chestionarele, iar asistentul primește un raport cu privire la starea pacientului.

Comparație cu alte proiecte similare

În timp ce mă documentam, am dat peste mai multe proiecte similare care au doar modulul de interacțiune vocală cu un chatbot. Acestea nu erau implementate în Django, ofereau un suport pentru procesarea naturală a limbajului și am urmat multiple tutoriale ca să înțeleg cum ar trebui puse cap la cap toate metodele ca să pot integra în proiectul meu acest concept. Doar din acest punct de vedere există o oarecare similaritate, mai exact modul de antrenare a datelor și configurarea fișierului text cu intenții16.

Posibile direcții de dezvoltare

Acest proiect se poate dezvolta foarte ușor într-un proiect foarte competitiv pentru Amazon Alexa sau Google Assistant doar cu un set de antrenare a datelor mult mai mare alături de o parsare mai optimă a intențiilor.

Principala direcție de dezvoltare a acestui proiect este cea medicală. Astfel, proiectul poate oferi asistență, consiliere pacientului, iar în alte cuvinte, poate deveni un prieten al acestuia. Dacă partea ce ține de procesarea naturală de limbaj ar fi bine pusă la punct, iar aici mă refer la mai multe utilități și API-uri incluse în proiect, iar pacientul să le poată accesa folosind un limbaj cât se poate de natural, acest proiect ar putea să fie folosit de milioane de oameni în vârstă care suferă fie de Alzheimer, depresie sau se simt singuratici. Totodată, această direcție de dezvoltare oferă infinit de multe posibilități. Atunci când se lucrează cu oamenii, intervine partea emoțională de care m-am ocupat într-o oarecare măsură cu ajutorul unor biblioteci care se ocupă cu recunoașterea emoției și am reușit să identific corect ce simte în anumite momente pacientul. Astfel, scopul acestui proiect e să devină pe cât se poate de umană interacțiunea dintre chatbot și pacient, site-ul web să fie ușor de accesat de oriunde și cel mai important să ofere rapoarte și rezultate concrete asistenților care au în grijă pacienți.

Bibliografie

[0] , The "Loneliness Epidemic" [Online], Disponibil la adresa: https://www.hrsa.gov/enews/past-issues/2019/january-17/loneliness-epidemic, Accesat: 2019.

[1] Ardeleanu Loredana, Singurătatea la bătrânețe [Online], Disponibil la adresa: https://www.romedic.ro/singuratatea-la-batranete-0P33211, Accesat: .

[2] Ardeleanu Loredana, Singurătatea la bătrânețe [Online], Disponibil la adresa: , Accesat: .

[3] , Office for National Statistics [Online], Disponibil la adresa: https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/articles/lonelinesswhatcharacteristicsandcircumstancesareassociatedwithfeelinglonely/2018-04-10, Accesat: .

[4] Md. Zia Uddin,\* Weria Khaksar, and Jim Torresen, [Online], Disponibil la adresa: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6068532/, Accesat: 2018 Jun 25.

[5] PEW research center, [Online], Disponibil la adresa: https://www.pewresearch.org/fact-tank/2020/03/10/older-people-are-more-likely-to-live-alone-in-the-u-s-than-elsewhere-in-the-world/, Accesat: .

[6] Daniel B. Kaplan, Barbara J. Berkman, „Older Adults Living Alone”, , , pp. , May 2019.

[7] Ronan De Renesse , Virtual digital assistants to overtake world population by 2021 [Online], Disponibil la adresa: https://www.omdia.com/resources/product-content/virtual-digital-assistants-to-overtake-world-population-by-2021, Accesat: 17 May 2017.

[8] From Wikipedia, Virtual Assistant [Online], Disponibil la adresa: https://wiki2.org/en/Virtual\_assistant, Accesat: .

[9] From Wikipedia, the free encyclopedia, [Online], Disponibil la adresa: https://en.wikipedia.org/wiki/Google\_Now, Accesat: .

[10] Jon Martindale , [Online], Disponibil la adresa: https://www.digitaltrends.com/computing/cortana-vs-siri-vs-google-now/, Accesat: 2020.

[11] , Google Assistant [Online], Disponibil la adresa: https://simple.wikipedia.org/wiki/Google\_Assistant, Accesat: .

[12] Wikipedia, Amazon Alexa wiki [Online], Disponibil la adresa: https://en.wikipedia.org/wiki/Amazon\_Alexa, Accesat: .

[13] Sachin Maini andTarun Wadhwa, [Online], Disponibil la adresa: https://www.marketwatch.com/story/heres-how-smart-speakers-need-to-get-smarter-2018-09-20, Accesat: Sept. 20, 2018.

[14] DataFlair Team, [Online], Disponibil la adresa: https://data-flair.training/blogs/django-features/, Accesat: 19 mar 2019.

[15] Kr3dHead (username), [Online], Disponibil la adresa: https://skyose.com/full-opencv-review/, Accesat: 30 Mar 2020.

Anexe.

* + - * 1. Ovum

Ovum este o firmă de cercetare și consultanță lider pe piață, axată pe a ajuta furnizorii de servicii digitale și partenerii lor de tehnologie să prospere în economia digitală conectată. Prin intermediul a 150 de analiști și consultanți din întreaga lume, oferă analize de experti și perspective strategice în industria IT, telecomunicații și mass-media. Fondată în 1985, Ovum are una dintre cele mai experimentate echipe de analiști din industrie și este o sursă de îndrumare respectată pentru liderii de afaceri, CIO-uri, furnizori, furnizori de servicii și autorități de reglementare care caută date de piață complete, exacte și interesante, cercetare și consultanță. Cu 23 de birouri pe șase continente, Ovum oferă o perspectivă cu adevărat globală asupra piețelor tehnologiei, comunicațiilor și mass-media și oferă clienților informații despre instrumente de flux de lucru, previziuni, sondaje, evaluări ale pieței, audituri tehnologice și avize.

* + - * 1. IVR

Răspunsul vocal interactiv (IVR în engleză) este o tehnologie care permite unui computer să interacționeze cu oamenii prin utilizarea vocalelor și a tonurilor DTMF (vezi anexa 3) introduse prin intermediul unei tastaturi.

* + - * 1. DTMF

Semnalizarea cu mai multe frecvențe cu ton dublu (DTMF) este un sistem de semnalizare de telecomunicații care folosește banda de frecvență vocală peste liniile telefonice între echipamentele telefonice și alte dispozitive de comunicații și centrele de comutare.

* + - * 1. train\_chatbot.py

Conținutul fișierului train\_chatbot este afișat mai jos.

import nltk  
nltk.download(**'punkt'**)  
nltk.download(**'wordnet'**)  
from nltk.stem import WordNetLemmatizer  
lemmatizer = WordNetLemmatizer()  
import json  
import pickle  
  
import numpy as np  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Activation, Dropout  
from tensorflow.keras.optimizers import SGD  
import random  
  
import matplotlib.pyplot as plt  
  
words=[]  
classes = []  
documents = []  
ignore\_words = [**'?'**, **'!'**]  
data\_file = open(**'intents.json'**).read()  
intents = json.loads(data\_file)  
  
*#json tip dict*  
for intent in intents[**'intents'**]:  
 for pattern in intent[**'patterns'**]:  
 *# take each word and tokenize it -> am toate cuvintele intr-o lista*  
w = nltk.word\_tokenize(pattern)  
 words.extend(w)  
 *# adding documents -> de forma asta [(['Hi', 'there'], 'greeting'), (['How', 'are', 'you'], 'greeting')] și tot adaug*  
documents.append((w, intent[**'tag'**]))  
 *# adding classes to our class list -> clasele vor contine doar intent-urile*  
if intent[**'tag'**] not in classes:  
 classes.append(intent[**'tag'**])  
  
*#aduc cuvintele la forma de baza, le ordonez crescator si inlatur duplicatele*  
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]  
words = sorted(list(set(words)))  
*# la fel fac si pentru clase*  
classes = sorted(list(set(classes)))  
  
print (len(documents), **"documents"**)  
*#*  
print (len(classes), **"classes"**, classes)  
*#*  
print (len(words), **"unique lemmatized words"**, words)  
  
  
pickle.dump(words,open(**'words.pkl'**,**'wb'**))  
pickle.dump(classes,open(**'classes.pkl'**,**'wb'**))  
  
*# initializing training data*  
training = []  
output\_empty = [0] \* len(classes)  
for doc in documents:  
 *# initializing bag of words*  
bag = []  
 *# list of tokenized words for the pattern*  
pattern\_words = doc[0] *# cuvintele din interiorul unui pattern: hi there*  
 *# lemmatize each word - create base word, in attempt to represent related words*  
pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]  
 *# create our bag of words array with 1, if word match found in current pattern*  
 *# practic pun 1 daca gasesc cuvant din pattern curent in lista totala de cuvinte*  
for w in words:  
 bag.append(1) if w in pattern\_words else bag.append(0)  
  
 *# output is a '0' for each tag and '1' for current tag (for each pattern)*  
output\_row = list(output\_empty)  
 output\_row[classes.index(doc[1])] = 1  
  
 training.append([bag, output\_row])  
*# shuffle our features and turn into np.array*  
*# 109 de documente (tag + pattern)*  
random.shuffle(training)  
training = np.array(training)  
*# create train and test lists. X - patterns, Y - intents*  
train\_x = list(training[:,0])  
train\_y = list(training[:,1])  
  
*# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons*  
*# equal to number of intents to predict output intent with softmax*  
*# acestia se numesc hiperparametri. Tot ce poate fi schimbat, functia de activare, loss function, optimizatorul,*  
*# se pot schimba ca sa fie un alt fel de output*  
model = Sequential()  
model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation=**'relu'**))  
*#adaug un dropout pentru a raspunde bine la datele de evaluare/test ca sa evit overfittingul*  
*#adaug o "interpretare" noua asupra retelei atunci cand fac dropout. Practic fac drop la un nod si la conexiunile sale*  
model.add(Dropout(0.5))  
model.add(Dense(64, activation=**'relu'**))  
*#adaug dropout si la celalalt strat*  
*#bias e pt a activa neuroni diferiti, ajustez greutatile conform bias-ului pt a avea diferite rezultate*  
model.add(Dropout(0.5))  
*# ultimul strat are lungimea tuturor intentiilor antrenate /labels. ca sa poata clasifica*  
*# softmax se asigura ca toate valorile sunt cuprinse intre 0 si 1 si adaugate dau maxim 1. Sigmoid nu face la fel?*  
model.add(Dense(len(train\_y[0]), activation=**'softmax'**))  
  
*# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model*  
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
model.compile(loss=**'categorical\_crossentropy'**, optimizer=sgd, metrics=[**'accuracy'**])  
  
*#fitting and saving the model*  
epochs = 200  
history = model.fit(np.array(train\_x), np.array(train\_y), epochs=epochs, batch\_size=5, verbose=1)  
model.save(**'chatbot\_model.h5'**, history)  
  
  
loss\_train = history.history[**'accuracy'**]  
loss\_val = history.history[**'loss'**]  
epochs = range(1,epochs+1)  
plt.plot(epochs, loss\_train, **'g'**, label=**'Training accuracy'**)  
plt.plot(epochs, loss\_val, **'b'**, label=**'validation accuracy'**)  
plt.title(**'Training and Validation accuracy'**)  
plt.xlabel(**'Epochs'**)  
plt.ylabel(**'Accuracy'**)  
plt.legend()  
plt.show()  
print(**"model created"**)

* + - * 1. tutorial/views.py

Mai jos este prezentat conținutul fișierului views.py al aplicației tutorial

from django.core.mail import send\_mail  
from django.http import JsonResponse  
from .models import Pacient, Tutore, PacientParsing, DepressionParsing, AlzheimerParsing, User  
from django.contrib.auth.models import Group  
from django.views.generic import View  
from django.shortcuts import render, redirect  
from django.contrib import messages  
from .decorators import restrict\_unauthenticated\_user, restrict\_pacient\_general\_form, restrict\_tutore\_insert\_patient, tutore\_and\_admin\_only, pacient\_and\_admin\_only, restrict\_authenticated, restrict\_pacient\_mood  
from .forms import UserRegisterForm, PacientForm, GeneralForm, DepressionMoodForm, AlzheimerForm, LonelinessForm,DepressionForm, AlzheimerMoodForm  
from django.contrib.auth import authenticate, login, logout  
from rest\_framework.views import APIView  
from rest\_framework.response import Response  
import numpy as np  
  
  
class mood\_results(View):  
 def get(self, request, \*args, \*\*kwargs):  
 return render(request, **'website/mood\_results.html'**)  
  
  
def get\_data(request, \*args, \*\*kwargs):  
  
 data = {  
 **"user"**: **'alex'**,  
 **"nr\_pacienti"**:111,  
 }  
 return JsonResponse(data)  
  
  
class ChartDataAPI(APIView):  
  
 authentication\_classes = []  
 permission\_classes = []  
  
 def get(self, request, format=None):  
 print(**'aaaaaaaaaaaaaaaaaaaaaaaaaaa'**)  
 print(self.request.user)  
 users\_no = User.objects.all().count()  
 labels = [**"users"**, **"blue"**, **"yellow"**, **"green"**, **"purple"**,**"orange"**]  
 default\_items = [users\_no, 34,21,32,15,22]  
 data = {  
 **"labels"**:labels,  
 **"default"**:default\_items,  
 }  
 return Response(data)  
  
  
def index(request):  
 context = {}  
 if request.user.is\_authenticated and request.user.groups.filter(name=**'pacient'**).exists():  
 pacient = Pacient.objects.get(user=request.user)  
 context = {**'afectiune'**: pacient.afectiune, **'flag'**:pacient.flag}  
 return render(request, **'website/index.html'**, context)  
  
  
def logoutPage(request):  
 logout(request)  
 return redirect(**'tutorial:website\_index'**)  
  
  
def about(request):  
 return render(request, **'website/about.html'**, {**'title'**: **'About'**})  
  
@restrict\_authenticated  
def registerPage(request):  
 if request.method == **'POST'**:  
 form = UserRegisterForm(request.POST)  
 if form.is\_valid():  
 user = form.save(commit = False)  
 user.save()  
 group = Group.objects.get(name=**'tutore'**)  
 user.groups.add(group)  
 username = form.cleaned\_data.get(**'username'**)  
 Tutore.objects.create(user=user,nr\_pacienti=0)  
 messages.success(request, **f'Account created for** {username}**!'**)  
 return redirect(**'tutorial:website\_loginPage'**)  
 else:  
 form = UserRegisterForm()  
 return render(request, **'website/register.html'**, {**'form'**: form})  
  
@restrict\_authenticated  
def loginPage(request):  
 if request.method==**'POST'**:  
 username = request.POST.get(**'username'**)  
 password = request.POST.get(**'password'**)  
  
 user = authenticate(request, username = username, password = password)  
 if user is not None:  
 login(request, user)  
 if user.groups.filter(name=**'tutore'**).exists():  
 t = Tutore.objects.get(user=request.user)  
 if t.flag == False:  
 t.nr\_pacienti = 0 *# change field*  
t.flag = True  
 t.save() *# this will update only*  
return redirect(**'tutorial:website\_index'**)  
 else:  
 messages.info(request, **'userul sau parola este incorecta !'**)  
 context = {}  
 return render(request, **'website/login.html'**, context)  
  
  
@restrict\_unauthenticated\_user  
@tutore\_and\_admin\_only  
@restrict\_tutore\_insert\_patient  
def pacient\_create\_view(request):  
 instanta = Tutore.objects.get(user = request.user)  
 nr\_pac = instanta.nr\_pacienti  
 if request.method == **'POST'**:  
 form = PacientForm(request.POST)  
 form\_user = UserRegisterForm(request.POST)  
 if form.is\_valid() and form\_user.is\_valid():  
 fu = form\_user.save(commit=False)  
 fs = form.save(commit=False)  
 if request.user.is\_authenticated:  
 t = Tutore.objects.get(user = request.user)  
 t.nr\_pacienti += 1  
 t.save()  
 varsta = form.cleaned\_data[**'varsta'**]  
 afectiune = form.cleaned\_data[**'afectiune'**]  
 tel = form.cleaned\_data[**'tel\_urgenta'**]  
 fu.save()  
 fs.tutore = Tutore.objects.get(user = request.user)  
  
 Pacient.objects.create(user=fu,  
 varsta= varsta,  
 afectiune=afectiune,  
 tutore = fs.tutore,  
 tel\_urgenta= tel  
 )  
  
 group = Group.objects.get(name=**'pacient'**)  
 fu.groups.add(group)  
 return redirect(**'tutorial:website\_index'**)  
 else:  
 form = PacientForm()  
 form\_user = UserRegisterForm  
 return render(request, **'website/update.html'**, {**'form'**: form, **'form\_user'**:form\_user, **'nr\_pac'**:nr\_pac})  
  
  
  
def parse\_disease(request):  
 form\_disease = GeneralForm(request.POST)  
 date\_pac = Pacient.objects.get(user=request.user)  
  
 if str(date\_pac.afectiune) == **'alzheimer'**:  
 form\_disease = AlzheimerForm()  
 if request.method == **'POST'**:  
 form\_disease = AlzheimerForm(request.POST)  
  
 elif str(date\_pac.afectiune) == **'singuratate'**:  
 form\_disease = LonelinessForm()  
 if request.method == **'POST'**:  
 form\_disease = LonelinessForm(request.POST)  
  
 elif str(date\_pac.afectiune) == **'depresie'**:  
 form\_disease = DepressionForm()  
 if request.method == **'POST'**:  
 form\_disease = DepressionForm(request.POST)  
 return form\_disease  
  
  
def calculate(param, type, question=None):  
 rat=0  
 if type==**'tip1'**:  
 if question == **'q1'**:  
 if param==**'1'**:  
 rat += 30  
 if param == **'2'**:  
 rat +=20  
 if param == **'3'**:  
 rat +=10  
 else:  
 if param==**'1'**:  
 rat += 10  
 if param == **'2'**:  
 rat += 5  
 if param == **'3'**:  
 rat += 3  
 if type == **'tip2'**:  
 if param==**'1'**:  
 rat += 20  
 if param == **'2'**:  
 rat +=10  
 if param == **'3'**:  
 rat +=5  
 return rat  
  
  
def calculate\_rating(request, fd, r1, r3): *# e nevoie sa reverific boala pt rating*  
r4 = fd.cleaned\_data[**'c4'**]  
 r5 = fd.cleaned\_data[**'c5'**]  
 r6 = fd.cleaned\_data[**'c6'**]  
 r = [r4,r5,r6]  
 rating1 = [calculate(x,**'tip2'**) for x in r]  
 rating2\_q1 = calculate(r1,**'tip1'**, **'q1'**)  
 rating2\_q2 = calculate(r3,**'tip1'**)  
 rating = np.sum(rating1)+rating2\_q1+rating2\_q2  
 return rating  
  
@pacient\_and\_admin\_only  
@restrict\_pacient\_general\_form  
def pacient\_general\_form\_view(request):  
 if request.method == **'POST'**:  
 form = GeneralForm(request.POST)  
 form\_disease = parse\_disease(request)  
  
 if form.is\_valid() and form\_disease.is\_valid():  
 instance = form.save(commit=False)  
 fd = form\_disease.save(commit = False)  
 instance.pacient = Pacient.objects.get(user = request.user)  
 r1 = form.cleaned\_data[**'c1'**] *#general form parsing*  
 *# r2 = form.cleaned\_data['dorinta'] # tratata separat*  
r3 = form.cleaned\_data[**'c3'**]  
  
 instance.rating = calculate\_rating(request,form\_disease,r1,r3)  
  
  
 r2 = form.cleaned\_data[**'dorinta'**]  
  
 instance.activitate = r1  
 instance.dorinta = r2  
 instance.tip\_fire = r3  
  
 intr1 = form\_disease.cleaned\_data[**'c4'**]  
 intr2 = form\_disease.cleaned\_data[**'c5'**]  
 intr3 = form\_disease.cleaned\_data[**'c6'**]  
 pac\_pars = PacientParsing.objects.create(pacient = instance.pacient,  
 rating = instance.rating,  
 activitate = instance.activitate,  
 dorinta = instance.dorinta,  
 tip\_fire = instance.tip\_fire,  
 intrebare1 = intr1,  
 intrebare2= intr2,  
 intrebare3= intr3,  
 )  
 instanta\_pacient = Pacient.objects.get(user=request.user)  
 instanta\_pacient.flag= True  
 instanta\_pacient.save()  
 return redirect(**'tutorial:website\_index'**)  
 else:  
 form = GeneralForm()  
 form\_disease = parse\_disease(request)  
 return render(request, **'website/pacient\_general\_form.html'**, {**'form'**: form, **'form\_disease'**:form\_disease})  
  
  
def calcul\_mood\_form(form, afectiune):  
 rating = 0  
 if afectiune == **'depresie'**:  
 for key, value in form.cleaned\_data.items():  
 rating = rating + int(value)  
 if afectiune == **'alzheimer'**:  
 for key, value in form.cleaned\_data.items():  
 rating = rating + int(value)  
 return rating  
  
  
  
def parse\_mood\_form(request, afectiune):  
 form = None  
 if afectiune == **'alzheimer'**:  
 form = AlzheimerMoodForm()  
 if request.method == **'POST'**:  
 form = AlzheimerMoodForm(request.POST)  
 if afectiune == **'depresie'**:  
 form = DepressionMoodForm()  
 if request.method == **'POST'**:  
 form = DepressionMoodForm(request.POST)  
 return form  
  
  
  
def Average(lst):  
 return sum(lst) / len(lst)  
  
  
def creation\_factory(user, pac\_pars, afectiune, rating):  
 p = Pacient.objects.get(user = user)  
 if afectiune == **'alzheimer'**:  
 try:  
 instanta = AlzheimerParsing.objects.get(pacientparse=pac\_pars)  
 old\_rating = instanta.disease\_rating  
 instanta.disease\_rating = old\_rating + **','** + str(rating)  
 total\_ratings = instanta.disease\_rating.split(**','**)  
 lista = []  
 for x in total\_ratings:  
 lista.append(int(x))  
 media = Average(lista)  
 if len(total\_ratings) % 3 == 0:  
 send\_mail(  
 **'Mesaj informare pacient: '** + str(p),  
 **'Punctajele pacientului dvs sunt urmatoarele: '** + str(instanta.disease\_rating) + **'**\n**Media pacientului este: '** + str(round(media))  
 + **'**\n\n**'** +**'Interpretarea Rezultatelor:**\n**5-8 – Creierul pacientului este intr-o stare buna.'**  
 **' Daca acesta se relaxeaza mai des si mentine o dieta sanatoasa,'**  
 **' creierul sau poate functiona mai bine**\n**'**  
 **'9-12 – Creierul pacientului este in pericol. Verifica-i dieta urgent ! Acesta poate sa reduca '**  
 **'pierderile de memorie cu vitamine, mancaruri care imbunatatesc functiile creierului, yoga, sport si meditatie.**\n**'**  
 **'12-15 – Creierul pacientului functioneaza in gol. Ar trebui de urgenta sa vada un doctor. '**  
 **'Acesta poate sa reduca pierderile de memorie alegand o dieta bogata in vitamine, sa mediteze, sa faca sport si sa se relaxeze mai des.**\n**'**  
,  
 **'virtual\_assistant@gov.com'**,  
 [**'bpiwbpiw1@gmail.com'**],  
 fail\_silently=False,  
 )  
 instanta.save()  
 except AlzheimerParsing.DoesNotExist:  
 AlzheimerParsing.objects.create(pacientparse=pac\_pars, disease\_rating=rating, tutore=p.tutore)  
 elif afectiune == **'depresie'**:  
 try:  
 instanta = DepressionParsing.objects.get(pacientparse=pac\_pars)  
 old\_rating = instanta.disease\_rating  
 instanta.disease\_rating = old\_rating + **','** + str(rating)  
 total\_ratings = instanta.disease\_rating.split(**','**)  
 lista = []  
 for x in total\_ratings:  
 lista.append(int(x))  
 media = Average(lista)  
 if len(total\_ratings) % 3 == 0:  
 send\_mail(  
 **'Mesaj informare pacient: '** + str(p),  
 **'Punctajele pacientului dvs sunt urmatoarele: '** + str(instanta.disease\_rating) + **'**\n**Media puntajelor este '** + str(round(media)) + **'**\n**Interpretarea rezultatelor:**\n**Scor 1-4: Forme de depresie minima**\n**'**  
 **'Scor 5-9: Depresie usoara**\n**'**  
 **'Scor 10-14: Depresie moderata**\n**'**  
 **'Scor 15-19: Depresie moderata severa**\n **'**  
 **'Scor 20-27: Depresie severa**\n**'**,  
 **'virtual\_assistant@gov.com'**,  
 [**'bpiwbpiw1@gmail.com'**],  
 fail\_silently=False,  
 )  
 instanta.save()  
 except DepressionParsing.DoesNotExist:  
 DepressionParsing.objects.create(pacientparse=pac\_pars, disease\_rating=rating, tutore = p.tutore)  
  
  
@restrict\_pacient\_mood  
@pacient\_and\_admin\_only  
def MoodFormView(request):  
 if request.method == **'POST'**:  
 pacient = Pacient.objects.get(user=request.user)  
 form = parse\_mood\_form(request, pacient.afectiune)  
 if form.is\_valid():  
 rating = calcul\_mood\_form(form, pacient.afectiune)  
 pacient\_parsing = PacientParsing.objects.get(pacient = pacient)  
 creation\_factory(request.user, pacient\_parsing,pacient.afectiune,rating)  
  
 else:  
 pacient = Pacient.objects.get(user=request.user)  
 form = parse\_mood\_form(request, pacient.afectiune)  
 context = {  
 **'form'**: form,  
 **'afectiune'**: pacient.afectiune  
 }  
 return render(request, **'website/mood\_form.html'**, context)

* + - * 1. decorators.py

Conținutul fișierului decorators.py

from collections import defaultdict  
from django.http import HttpResponse  
from django.shortcuts import redirect  
from .models import Pacient, Tutore, User, PacientParsing  
  
  
def restrict\_unauthenticated\_user(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 user = request.user  
 if user.is\_authenticated:  
 return view\_func(request, \*args, \*\*kwargs)  
 else:  
 return redirect(**'tutorial:website\_index'**)  
 return wrapper\_func  
  
def restrict\_pacient\_general\_form(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 user = request.user  
 pacient = Pacient.objects.get(user = request.user)  
 if pacient.flag == False:  
 return view\_func(request, \*args, \*\*kwargs)  
 else:  
 return redirect(**'tutorial:website\_index'**)  
 return wrapper\_func  
  
def allowed\_users(allowed\_roles = []):  
 def decorator(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 group = None  
 if request.user.groups.exists():  
 group = request.user.groups.all()[0].name  
 if group in allowed\_roles:  
 return view\_func(request, \*args, \*\*kwargs)  
 else:  
 return HttpResponse(**'Nu esti autorizat sa vezi pagina asta !'**)  
 return wrapper\_func  
 return decorator  
  
def tutore\_and\_admin\_only(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 group = None  
 if request.user.groups.exists():  
 group = request.user.groups.all()[0].name  
 if group == **'tutore'** or group == **'admin'**:  
 return view\_func(request, \*args, \*\*kwargs)  
 else:  
 return redirect(**'tutorial:website\_index'**)  
 return wrapper\_func  
  
def pacient\_and\_admin\_only(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 group = None  
 if request.user.groups.exists():  
 group = request.user.groups.all()[0].name  
 if group == **'pacient'** or group == **'admin'**:  
 return view\_func(request, \*args, \*\*kwargs)  
 else:  
 return redirect(**'tutorial:website\_index'**)  
 return wrapper\_func  
  
def restrict\_authenticated(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 user = request.user  
 if user.is\_authenticated:  
 return redirect(**'tutorial:website\_index'**)  
 else:  
 return view\_func(request, \*args, \*\*kwargs)  
 return wrapper\_func  
  
def restrict\_pacient\_mood(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 user = request.user  
 pacient = Pacient.objects.get(user = user)  
 flag = pacient.flag  
 if flag == False:  
 return redirect(**'tutorial:website\_index'**)  
 else:  
 return view\_func(request, \*args, \*\*kwargs)  
 return wrapper\_func  
  
def restrict\_tutore\_insert\_patient(view\_func):  
 def wrapper\_func(request, \*args, \*\*kwargs):  
 username\_get = request.user  
 selected\_user = Tutore.objects.get(user = username\_get)  
  
 print(**'!!!!!!!!!!!!'**)  
 print(selected\_user)  
 print(selected\_user.nr\_pacienti)  
 if selected\_user.nr\_pacienti > 100:  
 return HttpResponse(**"Ne pare rau, ati introdus deja numarul maxim de pacienti !<br>"**  
 **"pentru a putea adauga mai multe persoane, va rugam consultati pachetele premium"**)  
 return view\_func(request, \*args, \*\*kwargs)  
 return wrapper\_func

* + - * 1. forms.py

Conținutul fișierului forms.py

from crispy\_forms.helper import FormHelper  
from crispy\_forms.layout import Layout, Submit  
from django import forms  
from django.contrib.auth.models import User  
from django.contrib.auth.forms import UserCreationForm  
from django.contrib.auth.models import Permission  
from django.core.validators import MaxValueValidator, MinValueValidator  
from .models import Pacient, Tutore, PacientParsing  
from django.core.validators import RegexValidator  
from django.utils.safestring import mark\_safe  
  
  
class UserRegisterForm(UserCreationForm):  
 class Meta:  
 model = User  
 fields = [**'first\_name'**,**'last\_name'**,**'username'**, **'email'**, **'password1'**, **'password2'**]  
  
class TutoreForm(forms.ModelForm):  
 class Meta:  
 model = Tutore  
 fields = [**'nr\_pacienti'**]  
  
  
selectie\_boli= [  
 (**'alzheimer'**, **'Alzheimer'**),  
 (**'singuratate'**, **'Singuratate'**),  
 (**'depresie'**, **'Depresie'**),  
 ]  
  
  
class PacientForm(forms.ModelForm):  
 varsta = forms.IntegerField(validators=[MinValueValidator(0), MaxValueValidator(100)])  
 afectiune = forms.CharField(label=**'Afectiune:'**, widget=forms.Select(choices=selectie\_boli))  
 vaduv = forms.BooleanField(label=**'Vaduv:'**, required = False)  
 tel\_urgenta = forms.CharField(max\_length=10, validators=[RegexValidator(**r'^\d{0,10}$'**)])  
  
  
 class Meta:  
 model = Pacient  
 fields = [  
 **'varsta'**,  
 **'afectiune'**,  
 **'vaduv'**,  
 **'tel\_urgenta'**,  
 ]  
  
alegere\_stil\_viata= [  
 (**'3'**, **'Nu fac sport si mananc des alimente nesanatoase'**),  
 (**'2'**, **'Mici scapari la dulciuri / fastfood'**),  
 (**'1'**,**'Fac sport si mananc sanatos regulat'**),  
 ]  
alegere\_fire=[  
 (**'1'**, **'Sunt extrovertit'**),  
 (**'2'**, **'Sunt introvertit'**),  
 (**'3'**, **'Am o stare depresiva uneori'**),  
]  
  
alegere\_dorinta= [  
 (**'conectare'**, **'Mi-ar placea sa fiu mai conectat cu tehnologia'**),  
 (**'sanatate'**, **'As vrea sa imi pot schimba stilul de viata'**),  
 (**'socializare'**,**'Vreau sa socializez mai mult'**),  
 (**'normal'**,**'Nu vreau sa schimb nimic la mine'**),  
 ]  
  
  
class GeneralForm(forms.ModelForm):  
  
 c1 = forms.ChoiceField(label=**'Care este stilul tau de viata?'**,choices=alegere\_stil\_viata,  
 widget=forms.RadioSelect)  
 dorinta = forms.ChoiceField(label=**'Ce-ti doresti cel mai mult acum?'**,choices=alegere\_dorinta,  
 widget=forms.RadioSelect)  
 c3= forms.ChoiceField(label=**'Ce tip de fire esti?'**,choices=alegere\_fire,  
 widget=forms.RadioSelect)  
 rating = forms.IntegerField(widget = forms.HiddenInput(), required = False)  
 class Meta:  
 model = PacientParsing  
 fields = [  
 **'c1'**,  
 **'dorinta'**,  
 **'c3'**,  
 **'rating'**,  
 ]  
  
  
alegere\_greutate= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_stres= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_dieta= [  
 (**'1'**, **'Da'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Nu'**),  
 ]  
  
class LonelinessForm(forms.ModelForm):  
 c4 = forms.ChoiceField(label=**'De obicei ai încredere în tine?'**, choices=alegere\_greutate,  
 widget=forms.RadioSelect)  
 c5 = forms.ChoiceField(label=**'Te simți constant obosit?'**, choices=alegere\_stres,  
 widget=forms.RadioSelect)  
 c6 = forms.ChoiceField(label=**'Ții mult la lucrurile materiale?'**, choices=alegere\_dieta,  
 widget=forms.RadioSelect)  
 class Meta:  
 model = Pacient  
 fields = [  
 **'c4'**,  
 **'c5'**,  
 **'c6'**  
]  
  
alegere\_stresAlz= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_confuzie= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_concentrare= [  
 (**'1'**, **'Da'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Nu'**),  
 ]  
  
class AlzheimerForm(forms.ModelForm):  
 c4= forms.ChoiceField(label=**'Te lasi stresat usor?'**,choices=alegere\_stresAlz,  
 widget=forms.RadioSelect)  
 c5= forms.ChoiceField(label=**'Ai cateodata stari de confuzie?'**,choices=alegere\_confuzie,  
 widget=forms.RadioSelect)  
 c6 = forms.ChoiceField(label=**'Reusesti cu greu sa te concentrezi?'**,choices=alegere\_concentrare,  
 widget=forms.RadioSelect)  
 class Meta:  
 model = Pacient  
 fields = [  
 **'c4'**,  
 **'c5'**,  
 **'c6'**  
]  
  
  
alegere\_somn= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_izolare= [  
 (**'1'**, **'Nu'**),  
 (**'2'**, **'Cateodata'**),  
 (**'3'**,**'Da'**),  
 ]  
alegere\_dietaDepr= [  
 (**'1'**, **'Da'**),  
 (**'2'**, **'Uneori'**),  
 (**'3'**,**'Nu'**),  
 ]  
  
  
class DepressionForm(forms.ModelForm):  
 c4 = forms.ChoiceField(label=**'Ai probleme cu somnul?'**, choices=alegere\_somn,  
 widget=forms.RadioSelect)  
 c5 = forms.ChoiceField(label=**'Ai momente cand te simti izolat?'**, choices=alegere\_izolare,  
 widget=forms.RadioSelect)  
 c6 = forms.ChoiceField(label=**'Ai o dieta echilibrata?'**, choices=alegere\_dietaDepr,  
 widget=forms.RadioSelect)  
 class Meta:  
 model = Pacient  
 fields = [  
 *# 'stare',*  
**'c4'**,  
 **'c5'**,  
 **'c6'**  
]  
  
alegere\_depr\_mood= [  
 (**'0'**, **'Deloc'**),  
 (**'1'**, **'Cateva zile'**),  
 (**'2'**,**'Mai mult decat jumatate din zile'**),  
 (**'3'**,**'Aproape in fiecare zi'**),  
 ]  
  
class DepressionMoodForm(forms.ModelForm):  
 c1 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>n-ai avut stare, incat simteai nevoia de a te misca mai des decat de '**  
 **'obicei?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c2 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>vorbeai sau te miscai foarte incet incat alti oameni au '**  
 **'observat acest lucru?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c3 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>aveai probleme concentrandu-te asupra lucrurilor minore precum '**  
 **'cititul sau vizionatul unor emisiuni</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c4 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>aveai o stima scazuta de sine - ca si cum ai fi un esec sau '**  
 **'te-ai dezamagit pe tine sau familia?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c5 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>n-ai mancat deloc sau ai mancat prea mult?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c6 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>te simteai obosit sau nu aveai energie?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c7 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>ai probleme cu somnul, adormi greu, dormi prea mult sau dormi'**  
 **' prea putin?</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c8 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong>aveai o stare depresiva, te simteai prost sau fara speranta?'**  
 **'</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 c9 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani, cat de des ai fost deranjat de faptul ca '**  
 **'<strong> nu aveai interes sau placere in a face anumite lucruri'**  
 **'</strong>'**), choices=alegere\_depr\_mood,  
 widget=forms.RadioSelect)  
 class Meta:  
 model = PacientParsing  
 fields = [  
 **'c1'**,  
 **'c2'**,  
 **'c3'**,  
 **'c4'**,  
 **'c5'**,  
 **'c6'**,  
 **'c7'**,  
 **'c8'**,  
 **'c9'**,  
 ]  
  
alegere\_alzheimer\_mood = [  
 (**'1'**,**'Da'**),  
 (**'0'**,**'Nu'**),  
]  
class AlzheimerMoodForm(forms.ModelForm):  
 c1 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>uiti in ce zi din saptamana'**  
 **' esti?</strong>'**), choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c2 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani<strong>ai cautat vreun lucru si ai uitat ce anume cauti?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c3 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>uiti vreun nume de-al prietenilor'**  
 **' tai? </strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c4 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>aduni numere cu doua cifre '**  
 **'si sa iti fie greu sa le calculezi din minte?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c5 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat ca <strong>rareori sa te simti energic?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c6 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani <strong>te-au suparat mai des problemele minore decat de'**  
 **' obicei?</strong> '**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c7 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa iti fie <strong>greu sa te concentrezi '**  
 **'pentru macar o ora?</strong> '**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c8 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>gasesti cheile undeva si nu-ti'**  
 **' amintesti faptul ca le-ai pus acolo?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c9 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>te repeti de mai multe ori?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c10 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>te pierzi undeva unde ai mai'**  
 **' fost inainte?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c11 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat ca <strong>prietenii sau familia sa observe'**  
 **' ca esti mai uituc decat de obicei?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c12 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>ratezi anumite intalniri /'**  
 **' programari din cauza ca ai uitat?</strong> '**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c13 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa<strong> uiti punctul de vedere '**  
 **'pe care voiai sa il spui?</strong>'**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c14 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa<strong> depinzi de cafeina / bauturi '**  
 **'energizante pentru a te putea concentra?</strong> '**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
 c15 = forms.ChoiceField(label=mark\_safe(**'In ultimele doua saptamani ti s-a intamplat sa <strong>trebuiasca mai mult timp sa '**  
 **'inveti lucruri care in mod normal ti-ar lua putin timp?</strong> '**),  
 choices=alegere\_alzheimer\_mood,  
 widget=forms.RadioSelect)  
  
 class Meta:  
 model = PacientParsing  
 fields = [  
 **'c1'**,  
 **'c2'**,  
 **'c3'**,  
 **'c4'**,  
 **'c5'**,  
 **'c6'**,  
 **'c7'**,  
 **'c8'**,  
 **'c9'**,  
 **'c10'**,  
 **'c11'**,  
 **'c12'**,  
 **'c13'**,  
 **'c14'**,  
 **'c15'**,  
 ]

* + - * 1. models.py

Conținutul fișierului models.py este descris mai jos.

from django.db import models  
from django.contrib.auth.models import User  
import json  
from django.core.validators import RegexValidator  
from django.core.validators import MaxValueValidator, MinValueValidator  
from django.core.validators import int\_list\_validator  
  
  
class Tutore(models.Model):  
 user = models.OneToOneField(  
 User,  
 on\_delete=models.CASCADE,  
 primary\_key=True,  
 related\_name=**'tutore'**  
)  
  
 nr\_pacienti = models.IntegerField(default=0)  
 flag = models.BooleanField(default=False)  
  
 def \_\_str\_\_(self):  
 return self.user.username  
  
  
class Pacient(models.Model):  
 user = models.OneToOneField(  
 User,  
 on\_delete=models.CASCADE,  
 primary\_key=True,  
 related\_name=**'pacient'**  
)  
  
 varsta = models.IntegerField(validators=[MinValueValidator(0), MaxValueValidator(100)])  
 afectiune = models.CharField(max\_length=20)  
 vaduv = models.BooleanField(default = False)  
 tutore = models.ForeignKey(Tutore, on\_delete=models.CASCADE)  
 tel\_urgenta = models.CharField(max\_length=10, validators=[RegexValidator(**r'^\d{0,10}$'**)])  
 flag = models.BooleanField(default=False)  
 def \_\_str\_\_(self):  
 return self.user.username  
  
class PacientParsing(models.Model):  
 pacient = models.ForeignKey(Pacient, on\_delete=models.CASCADE)  
 rating = models.IntegerField() *#general\_rating*  
contor\_forms = models.IntegerField(default=0)  
 activitate = models.CharField(max\_length=50)  
 dorinta= models.CharField(max\_length=50)  
 tip\_fire= models.CharField(max\_length=50)  
 intrebare1 = models.CharField(max\_length=200, blank=True)  
 intrebare2 = models.CharField(max\_length=200, blank=True)  
 intrebare3 = models.CharField(max\_length=200, blank=True)  
 contor\_mesaje = models.IntegerField(default=0)  
 \_negative\_problems = models.TextField(default=**'[]'**, blank=True)  
 @property  
 def problems(self):  
 return json.loads(self.\_negative\_problems)  
 @problems.setter  
 def problems(self,value):  
 self.\_negative\_problems = json.dumps(self.problems)  
  
 def \_\_str\_\_(self):  
 return str(self.pacient)  
  
  
class PacientDetails(models.Model):  
 pacient = models.ForeignKey(Pacient,on\_delete=models.CASCADE)  
 fav\_book = models.CharField( max\_length=200, default=**''**)  
 fav\_movie = models.CharField(max\_length=200, default=**''**)  
 fav\_song = models.CharField(max\_length = 200, default=**''**)  
 fav\_activity= models.CharField(max\_length=200, default=**''**)  
 fav\_passion = models.CharField(max\_length= 200, default=**''**)  
 fav\_game = models.CharField(max\_length = 200, default=**''**)  
 hangout = models.IntegerField( default = 0)  
  
 def \_\_str\_\_(self):  
 return str(self.pacient)  
  
  
class DepressionParsing(models.Model):  
 pacientparse = models.ForeignKey(PacientParsing, on\_delete=models.CASCADE)  
 disease\_rating = models.CharField(default=0, validators=[int\_list\_validator], max\_length=100)  
 tutore = models.ForeignKey(Tutore, on\_delete=models.CASCADE, null=True)  
 def \_\_str\_\_(self):  
 return str(self.pacientparse)  
  
  
class AlzheimerParsing(models.Model):  
 pacientparse = models.ForeignKey(PacientParsing, on\_delete=models.CASCADE)  
 disease\_rating = models.CharField(default=0, validators=[int\_list\_validator], max\_length=100)  
 tutore = models.ForeignKey(Tutore, on\_delete=models.CASCADE, null=True)  
 def \_\_str\_\_(self):  
 return str(self.pacientparse)  
  
  
class LonelinessParsing(models.Model):  
 pacientparse = models.ForeignKey(PacientParsing, on\_delete=models.CASCADE)  
 disease\_rating = models.CharField(default=0, validators=[int\_list\_validator], max\_length=100)  
 tutore = models.ForeignKey(Tutore, on\_delete=models.CASCADE, null=True)  
 def \_\_str\_\_(self):  
 return str(self.pacientparse)

* + - * 1. urls.py

Conținutul fișierului urls.py este redat mai jos.

*"""website URL Configuration*  
  
*The `urlpatterns` list routes URLs to views. For more information please see:*  
 *https://docs.djangoproject.com/en/3.0/topics/http/urls/*  
*Examples:*  
*Function views*  
 *1. Add an import: from my\_app import views*  
 *2. Add a URL to urlpatterns: path('', views.home, name='home')*  
*Class-based views*  
 *1. Add an import: from other\_app.views import Home*  
 *2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')*  
*Including another URLconf*  
 *1. Import the include() function: from django.urls import include, path*  
 *2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))*  
*"""*  
from django.contrib import admin  
from django.urls import path  
from django.contrib.auth.views import LoginView  
from . import views  
  
  
  
app\_name = **"tutorial"**  
urlpatterns = [  
 path(**'update/'**, views.pacient\_create\_view, name=**'website\_update'**),  
 path(**'about/'**, views.about, name=**'website\_about'**),  
 path(**'pacient\_details/'**, views.pacient\_general\_form\_view, name=**'website\_general\_form'**),  
 path(**'register/'**, views.registerPage, name=**'website\_registerPage'**),  
 path(**'login/'**, views.loginPage, name=**'website\_loginPage'**),  
 path(**''**, views.index, name=**'website\_index'**),  
 path(**'index/'**, views.index, name=**'website\_index'**),  
 path(**'logout/'**, views.logoutPage, name=**'website\_logoutPage'**),  
 path(**'mood/'**, views.MoodFormView, name=**'website\_mood\_form'**),  
 path(**'api/data/'**, views.get\_data),  
 path(**'results/'**, views.mood\_results.as\_view()),  
 path(**'chartdata/'**, views.ChartDataAPI.as\_view()),  
  
 *# path('login/', LoginView.as\_view(template\_name='website/login.html'), name='website\_login'),*  
]

* + - * 1. text\_processing.py

import sys  
import re  
import time  
import requests  
from django.shortcuts import render  
from newsapi import NewsApiClient  
*# from key import b71542370a6247d493860e6b01d0d713*  
import datetime as dt  
import pandas as pd  
import nltk  
from django.core.mail import send\_mail  
import speech\_recognition as sr *# importing speech recognition package from google api*  
*# from pygame import mixer*  
import playsound *# to play saved mp3 file*  
from gtts import gTTS *# google text to speech*  
from selenium import webdriver *# to control browser operations*  
from textblob import TextBlob  
from textblob import Word  
from fake\_useragent import UserAgent  
from nltk.stem import WordNetLemmatizer  
  
lemmatizer = WordNetLemmatizer()  
import pickle  
import numpy as np  
from bs4 import BeautifulSoup  
import os  
  
os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**] = **'3'**  
from tensorflow.keras.models import load\_model  
  
model = load\_model(**'static/chatbot\_model.h5'**)  
import json  
import random  
from tutorial.models import PacientParsing, Pacient, AlzheimerParsing, DepressionParsing, LonelinessParsing, \  
 PacientDetails  
from datetime import datetime, timedelta  
import datefinder  
import wikipedia  
from tmdbv3api import TMDb, Discover  
  
tmdb = TMDb()  
tmdb.api\_key = **'530ff67defeb9213256896802945b0d2'**  
  
intents = json.loads(open(**'static/intents.json'**).read())  
words = pickle.load(open(**'static/words.pkl'**, **'rb'**))  
classes = pickle.load(open(**'static/classes.pkl'**, **'rb'**))  
  
num = 1  
  
  
def assistant\_speaks(output):  
 global num  
 num += 1  
 print(**"Alexa : "**, output)  
 toSpeak = gTTS(text=output, lang=**'en-US'**, slow=False)  
 file = str(num) + **".mp3"**  
toSpeak.save(file)  
 *# mp3\_fp = BytesIO()*  
 *# toSpeak = gTTS(output, 'en', slow=False)*  
 *# toSpeak.write\_to\_fp(mp3\_fp)*  
 *# os.system("D:\PeRSon\\audio\spoken.mp3")*  
**'''mixer.init()**  
 **mixer.music.load('D:\PeRSon**\\**audio\spoken.mp3')**  
 **mixer.music.play()**  
 **time.sleep(5)**  
 **mixer.music.stop()'''**  
*# song = AudioSegment.from\_file(mp3\_fp, format="mp3")*  
 *# playsound.playsound(mp3\_fp)*  
playsound.playsound(file, True)  
 os.remove(file)  
  
  
def search\_web(input):  
 driver = webdriver.Firefox()  
 driver.implicitly\_wait(1)  
 driver.maximize\_window()  
 if **'youtube'** in input.lower():  
 assistant\_speaks(**"Opening in youtube"**)  
 indx = input.lower().split().index(**'youtube'**)  
 query = input.split()[indx + 1:]  
 driver.get(**"http://www.youtube.com/results?search\_query="** + **'+'**.join(query))  
 return  
  
 elif **'wikipedia'** in input.lower():  
 assistant\_speaks(**"Opening Wikipedia"**)  
 indx = input.lower().split().index(**'wikipedia'**)  
 query = input.split()[indx + 1:]  
 driver.get(**"https://en.wikipedia.org/wiki/"** + **'\_'**.join(query))  
 return  
 else:  
 if **'google'** in input:  
 indx = input.lower().split().index(**'google'**)  
 query = input.split()[indx + 1:]  
 driver.get(**"https://www.google.com/search?q="** + **'+'**.join(query))  
 elif **'search'** in input:  
 indx = input.lower().split().index(**'google'**)  
 query = input.split()[indx + 1:]  
 driver.get(**"https://www.google.com/search?q="** + **'+'**.join(query))  
 else:  
 driver.get(**"https://www.google.com/search?q="** + **'+'**.join(input.split()))  
 return  
  
  
def open\_application(input):  
 if **"firefox"** in input or **"mozilla"** in input:  
 assistant\_speaks(**"Opening Mozilla Firefox"**)  
 os.startfile(**'C:\Program Files\Mozilla Firefox**\\**firefox.exe'**)  
 return  
 elif **"word"** in input:  
 assistant\_speaks(**"Opening Microsoft Word"**)  
 os.startfile(**'C:**\\**Program Files**\\**Microsoft Office**\\**root**\\**Office16**\\**WINWORD.EXE'**)  
 return  
  
 else:  
 assistant\_speaks(**"Application not available"**)  
 return  
  
  
negative\_phrases = []  
with open(**'static/stopwords.txt'**, **'r'**) as file:  
 stopwords = file.read().replace(**'**\n**'**, **''**)  
  
  
def parse\_stopwords(text):  
 result = **''**  
text\_split = text.split(**' '**)  
 result = [word for word in text\_split if word not in stopwords]  
 result = **' '**.join(result)  
 assistant\_speaks(**'Opening Firefox'**)  
 return result  
 *# search\_web('google '+result)*  
  
  
def clean\_up\_sentence(sentence):  
 sentence\_words = nltk.word\_tokenize(sentence)  
 sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]  
 return sentence\_words  
  
  
*# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence*  
*# we dont really care what the words mean / the frequency so 0 and 1 is the best for intent recognition*  
*# forma cuvintelor din bag este urmatoarea:*  
*# text = "this is a test to see if this test will work is is test a a"*  
*# {1: 2, 2: 3, 3: 3, 4: 3, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1} -> 9 cuvinte unice*  
*# {'this': 1, 'is': 2, 'a': 3, 'test': 4, 'to': 5, 'see': 6, 'if': 7, 'will': 8, 'work': 9}*  
def bow(sentence, words, show\_details=True):  
 *# tokenize the pattern*  
sentence\_words = clean\_up\_sentence(sentence)  
 *# bag of words - array of N words, vocabulary matrix*  
bag = [0] \* len(words)  
 for s in sentence\_words:  
 for i, w in enumerate(words):  
 if w == s:  
 *# assign 1 if current word is in the vocabulary position*  
bag[i] = 1  
 if show\_details:  
 print(**"found in bag: %s"** % w)  
 return (np.array(bag))  
  
  
def predict\_class(sentence, model):  
 *# filter out predictions below a threshold*  
p = bow(sentence, words, show\_details=False)  
 res = model.predict(np.array([p]))[0]  
 ERROR\_THRESHOLD = 0.25  
 results = [[i, r] for i, r in enumerate(res) if r > ERROR\_THRESHOLD]  
 *# sort by strength of probability*  
results.sort(key=lambda x: x[1], reverse=True)  
 *# print(results)*  
return\_list = []  
 for r in results:  
 return\_list.append({**"intent"**: classes[r[0]], **"probability"**: str(r[1])})  
  
 return return\_list  
  
  
*# preiau un raspuns random din lista cu intentii*  
def getResponse(ints, intents\_json):  
 tag = ints[0][**'intent'**]  
 list\_of\_intents = intents\_json[**'intents'**]  
 for i in list\_of\_intents:  
 if (i[**'tag'**] == tag):  
 result = random.choice(i[**'responses'**])  
 break  
 return result  
  
  
def chatbot\_response(msg):  
 ints = predict\_class(msg, model)  
 intent = next(iter(ints[0].values()))  
 res = getResponse(ints, intents)  
 return res, intent  
  
  
def get\_audio():  
 r = sr.Recognizer()  
 audio = **''**  
with sr.Microphone() as source:  
 print(**"Listening..."**)  
 audio = r.listen(source, phrase\_time\_limit=50)  
 print(**"Stop."**)  
 try:  
 text = r.recognize\_google(audio, language=**'en-US'**)  
 print(**"You : "**, text)  
 res, intent = chatbot\_response(text)  
 if intent == **'goodbye'**:  
 assistant\_speaks(res)  
 sys.exit()  
 return text  
 except:  
 assistant\_speaks(**"Could not understand what you said. Going back to the main module"**)  
  
  
def parse\_details(ans):  
 result = **''**  
ans = ans.split()  
 for word in ans:  
 if word not in stopwords:  
 result += word + **' '**  
return result  
  
  
def parse\_text(input, keyword):  
 if keyword in input:  
 ans = input  
 indx = ans.split().index(keyword)  
 query = ans.split()[indx + 1:]  
 result = wikipedia.page(**' '**.join(query))  
 assistant\_speaks(result.title)  
 assistant\_speaks(result.summary[0:400])  
 assistant\_speaks(**'Do you want me to continue?'**)  
 ans = get\_audio()  
 res, intent = chatbot\_response(ans)  
 if intent == **'yes'**:  
 assistant\_speaks(result.summary[400:])  
 elif intent == **'no'**:  
 assistant\_speaks(**'Exiting wikipedia module'**)  
 else:  
 suggestion = wikipedia.suggest(input)  
 result = wikipedia.page(suggestion)  
 assistant\_speaks(result.title)  
 assistant\_speaks(result.summary[0:400])  
  
  
def parse\_movies(genre\_id, discover, movieInstance, genre\_string):  
 assistant\_speaks(**"The top 3 best "** + genre\_string + **" movies are the following"**)  
 movie = discover.discover\_movies({  
 **'with\_genres'**: genre\_id,  
 **'sort\_by'**: **'vote\_average.desc'**  
})  
 assistant\_speaks(**'I will read the title and the description'**)  
 for mov in movie[0:3]:  
 mov = str(mov)  
 searchMovie = movieInstance.search(mov)  
 result = searchMovie[0]  
 assistant\_speaks(mov)  
 assistant\_speaks(**'description is '** + result.overview)  
 assistant\_speaks(**'exiting movie module'**)  
  
  
def process\_text(input, intent, request=None):  
 try:  
 if intent == **'word'**:  
 open\_application(**'word'**)  
 if intent == **'firefox'**:  
 open\_application(**'firefox'**)  
 if intent == **'youtube'**:  
 assistant\_speaks(**'Sure. Tell me what do you want to search for'**)  
 ans = get\_audio()  
 result = parse\_stopwords(ans)  
 search\_web(**'youtube '** + result)  
 if intent == **'knowledge'**:  
 if **'about'** in input:  
 parse\_text(input, **'about'**)  
 if **'what is'** in input:  
 parse\_text(input, **'is'**)  
 if **'who was'** in input:  
 parse\_text(input, **'was'**)  
 if intent == **'web\_search'**:  
 if **'about'** in input:  
 ans = input  
 indx = ans.split().index(**'about'**)  
 query = ans.split()[indx + 1:]  
 string\_query = **' '**.join(query)  
 result = parse\_stopwords(string\_query)  
 search\_web(**'google '** + result)  
 if **'for'** in input:  
 ans = input  
 indx = ans.split().index(**'for'**)  
 query = ans.split()[indx + 1:]  
 string\_query = **' '**.join(query)  
 result = parse\_stopwords(string\_query)  
 search\_web(**'google '** + result)  
 assistant\_speaks(**'Going back to the main interface'**)  
 movie\_list\_intents = [**'movie'**, **'horror'**, **'action'**, **'comedy'**, **'popular'**, **'thriller'**]  
 if intent in movie\_list\_intents:  
 from tmdbv3api import Movie  
  
 movie = Movie()  
 discover = Discover()  
 if intent == **'popular'**:  
 pop\_movie = discover.discover\_movies({  
 **'sort\_by'**: **'popularity.desc'**  
})  
 assistant\_speaks(**"The most popular 5 movies are the following"**)  
 pop\_movies = **", "**.join(str(x) for x in pop\_movie[0:5])  
 assistant\_speaks(pop\_movies)  
 if intent == **'horror'**:  
 parse\_movies(27, discover, movie, **'horror'**)  
 if intent == **'action'**:  
 parse\_movies(28, discover, movie, **'action'**)  
 if intent == **'comedy'**:  
 parse\_movies(35, discover, movie, **'comedy'**)  
 if intent == **'thriller'**:  
 parse\_movies(53, discover, movie, **'thriller'**)  
 if intent == **'movie'**:  
 assistant\_speaks(**'Do you want a movie recommendation?'**)  
 ans = get\_audio()  
 if **'yes'** in ans:  
 pacient = Pacient.objects.get(user=request.user)  
  
 pac\_details = PacientDetails.objects.get(pacient=pacient)  
 fav\_movie = pac\_details.fav\_movie  
 search\_movie = movie.search(fav\_movie)  
 assistant\_speaks(**'I will read top three recommended movies based on your favorite movie'**)  
 res = search\_movie[0]  
 recommendations = movie.recommendations(movie\_id=res.id)  
 cnt = 0  
 for recommendation in recommendations:  
 if cnt >= 3:  
 break  
 else:  
 assistant\_speaks(recommendation.title)  
 assistant\_speaks(recommendation.overview)  
 cnt += 1  
  
 assistant\_speaks(**'Exiting movie module'**)  
 else:  
 assistant\_speaks(  
 **'I can give you the top movies based on a genre. Just tell me what are you looking for'**)  
 ans = get\_audio()  
 res, ints = chatbot\_response(ans)  
 process\_text(ans, ints)  
  
 pacient = Pacient.objects.get(user=request.user)  
 pac\_pars = PacientParsing.objects.get(pacient=pacient)  
 if intent == **'event'**:  
 from googleapiclient.discovery import build  
 from google\_auth\_oauthlib.flow import InstalledAppFlow  
 scopes = [**'https://www.googleapis.com/auth/calendar'**]  
 flow = InstalledAppFlow.from\_client\_secrets\_file(**"client\_secret.json"**, scopes=scopes)  
 credentials = flow.run\_console()  
 pickle.dump(credentials, open(**"token.pkl"**, **"wb"**))  
 credentials = pickle.load(open(**"token.pkl"**, **"rb"**))  
 service = build(**"calendar"**, **"v3"**, credentials=credentials)  
  
 calendarlist = service.calendarList().list().execute()  
 calendar\_id = calendarlist[**'items'**][0][**'id'**]  
 result = service.events().list(calendarId=calendar\_id, timeZone=**"Europe/Bucharest"**).execute()  
  
 timp\_event = get\_audio()  
  
 assistant\_speaks(**"What about the name of the event?"**)  
 name = get\_audio()  
  
 assistant\_speaks(**"would you like to add a description?"**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 assistant\_speaks(**"please tell me the description"**)  
 desc = get\_audio()  
 assistant\_speaks(**"should i add a location too?"**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 assistant\_speaks(**"Go ahead, tell me the location"**)  
 location = get\_audio()  
 create\_event(service, timp\_event, name, 1, desc, location)  
 elif sub\_intent == **'no'**:  
 create\_event(service, timp\_event, name, 1, desc)  
 elif sub\_intent == **'no'**:  
 create\_event(service, timp\_event, name)  
 assistant\_speaks(**'Event '** + name + **' created.'**)  
 assistant\_speaks(**'Exiting event module'**)  
 if intent == **'web'**:  
 ans = get\_audio()  
 result = parse\_stopwords(ans)  
 search\_web(**'google '** + result)  
 assistant\_speaks(**'Exiting web module'**)  
 if intent == **'discussion'**:  
 assistant\_speaks(**'Is there a certain topic you would like to discuss?'**)  
 ans = get\_audio()  
 *# print(ans)*  
sub\_resp, sub\_intent = chatbot\_response(ans)  
 *# print(sub\_intent)*  
if sub\_intent == **'no'**:  
 assistant\_speaks(**'Then how about you tell me more about yourself?'**)  
 try:  
 pac\_details = PacientDetails.objects.get(pacient=pacient)  
 except PacientDetails.DoesNotExist:  
 pac\_details = PacientDetails.objects.create(pacient=pacient)  
 if pac\_details.fav\_activity == **''**:  
 assistant\_speaks(**'Tell me your favorite activity'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 *# print(ans)*  
pac\_details.fav\_activity = ans  
 pac\_details.save()  
 if pac\_details.fav\_movie == **''**:  
 assistant\_speaks(**'what about your favorite movie?'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 pac\_details.fav\_movie = ans  
 pac\_details.save()  
 if pac\_details.fav\_game == **''**:  
 assistant\_speaks(**'Tell me your favorite game'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 pac\_details.fav\_game = ans  
 pac\_details.save()  
 if pac\_details.fav\_passion == **''**:  
 assistant\_speaks(**'Do you have a favorite passion?'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 pac\_details.fav\_passion = ans  
 pac\_details.save()  
 if pac\_details.fav\_song == **''**:  
 assistant\_speaks(**'What is your favorite song?'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 pac\_details.fav\_song = ans  
 pac\_details.save()  
 if pac\_details.fav\_book == **''**:  
 assistant\_speaks(**'And your favorite book is?'**)  
 ans = get\_audio()  
 if **"don't"** not in ans or **'no'** not in ans:  
 ans = parse\_details(ans)  
 pac\_details.fav\_book = ans  
 pac\_details.save()  
 assistant\_speaks(**"How was your day so far? When you have finished talking, please say that's it"**)  
 r = sr.Recognizer()  
 list = []  
 happy\_list = [**'That sounds great !'**, **'Wow, I am glad for you'**, **'Good job!'**, **'This sounds awesome'**]  
 neutral\_list = [**'Okay, continue'**, **'Understood'**, **'What next?'**, **'Is there something more?'**]  
 sad\_list = [**'What is the specific reason that made you feel this way? Please keep it short'**,  
 **'Can you please tell me what is the root of the problem? Please keep it short'**,  
 **'what disturbed you that much? Please keep it short'**]  
 with sr.Microphone() as source:  
 while (1):  
 try:  
 print(**'Listening to your day: '**)  
 audio = r.listen(source)  
 text = r.recognize\_google(audio, language=**'en-US'**)  
 blob1 = TextBlob(text)  
 blob1 = blob1.correct()  
 text = text.lower()  
 print(format(blob1.sentiment))  
 if **"that's it"** in text:  
 break  
 if blob1.polarity < 0:  
 assistant\_speaks(random.choice(sad\_list))  
 motiv = get\_audio()  
 pac\_pars.\_negative\_problems += motiv + **'**\n**'**  
pac\_pars.contor\_mesaje += 1  
 if pac\_pars.contor\_mesaje % 3 == 0:  
 pac\_pars.contor\_mesaje = 0  
 send\_mail(  
 **'Mesaj informare pacient '** + str(pacient),  
 **'Urmatoarele probleme par sa-l afecteze pe pacient: '** + pac\_pars.\_negative\_problems,  
 **'virtual\_assistant@gov.com'**,  
 [**'bpiwbpiw1@gmail.com'**],  
 fail\_silently=False,  
 )  
 list.append(motiv)  
 pac\_pars.save()  
 if blob1.polarity > 0.5:  
 assistant\_speaks(random.choice(happy\_list))  
 elif blob1.polarity <= 0.5 and blob1.polarity >= 0:  
 assistant\_speaks(random.choice(neutral\_list))  
 except:  
 continue  
  
 motiv = random.choice(list)  
 research\_later = **"what+to+do+when+"** + motiv  
 ua = UserAgent()  
 google\_url = **"https://www.google.com/search?q="** + research\_later  
 response = requests.get(google\_url, {**"User-Agent"**: ua.random})  
 soup = BeautifulSoup(response.text, **"html.parser"**)  
 result\_div = soup.find\_all(**'div'**, attrs={**'class'**: **'ZINbbc'**})  
  
 links = []  
 titles = []  
 descriptions = []  
 for r in result\_div:  
 *# Checks if each element is present, else, raise exception*  
try:  
 link = r.find(**'a'**, href=True)  
 title = r.find(**'div'**, attrs={**'class'**: **'vvjwJb'**}).get\_text()  
 description = r.find(**'div'**, attrs={**'class'**: **'s3v9rd'**}).get\_text()  
  
 *# Check to make sure everything is present before appending*  
if link != **''** and title != **''** and description != **''**:  
 links.append(link[**'href'**])  
 titles.append(title)  
 descriptions.append(description)  
 *# Next loop if one element is not present*  
except:  
 continue  
 to\_remove = []  
 clean\_links = []  
 for i, l in enumerate(links):  
 clean = re.search(**'\/url\?q\=(.\*)\&sa'**, l)  
  
 *# Anything that doesn't fit the above pattern will be removed*  
if clean is None:  
 to\_remove.append(i)  
 continue  
 clean\_links.append(clean.group(1))  
 *# Remove the corresponding titles & descriptions*  
 *# for x in to\_remove:*  
 *# print(titles[x])*  
 *# print(descriptions[x])*  
 *# del titles[x]*  
 *# del descriptions[x]*  
random\_seed = random.randint(0, 3)  
 print(**'rand\_seed: '**)  
 print(random\_seed)  
 print(**'titles: '** + str(len(titles)))  
 print(**'links: '** + str(len(clean\_links)))  
 assistant\_speaks(**"I have found something regarding the problems you have just told me"**)  
 assistant\_speaks(**"The article title is called"**)  
 assistant\_speaks(titles[random\_seed])  
 assistant\_speaks(**"Do you want me to open the link for you?"**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 driver = webdriver.Firefox()  
 driver.implicitly\_wait(1)  
 driver.maximize\_window()  
 driver.get(clean\_links[random\_seed])  
 assistant\_speaks(**'I have opened the browser for you. Exiting discussion module'**)  
 else:  
 assistant\_speaks(**'Exiting discussion module'**)  
 return  
  
 if intent == **'news'**:  
 assistant\_speaks(**"Would you like the news on a specific subject?"**)  
 ans = get\_audio()  
 newsapi = NewsApiClient(api\_key=**'b71542370a6247d493860e6b01d0d713'**)  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 assistant\_speaks(**'What would you like me to search for? Please be as specific as you can.'**)  
 ans = get\_audio()  
 data = newsapi.get\_everything(q=ans, language=**'en'**, page\_size=5)  
 articles = data[**'articles'**]  
 assistant\_speaks(  
 **'You could choose one article by saying stop when i finish reading the headline. To continue '**  
 **'reading the headlines, just say continue. I found the following articles: '**)  
 for article in articles:  
 title = article[**'title'**]  
 url = article[**'url'**]  
 content = article[**'content'**]  
 assistant\_speaks(title)  
 ans = get\_audio()  
 if **'continue'** in ans:  
 continue  
 elif **'stop'** in ans:  
 assistant\_speaks(**'I will read the article content'**)  
 assistant\_speaks(content)  
 assistant\_speaks(  
 **'I can open the webpage which contains the article source. Do you want me to do that? '**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 driver = webdriver.Firefox()  
 driver.implicitly\_wait(1)  
 driver.maximize\_window()  
 driver.get(url)  
 r = sr.Recognizer()  
 assistant\_speaks(  
 **'I have opened your browser. To resume the articles read, just say resume. '**  
 **'If you want me to stop, just say stop'**)  
 with sr.Microphone() as source:  
 while (1):  
 print(**'Listening ...'**)  
 audio = r.listen(source)  
 try:  
 text = r.recognize\_google(audio, language=**'en-US'**)  
 if **'resume'** in text:  
 break  
 elif **'stop'** in text:  
 return  
 except:  
 continue  
 elif sub\_intent == **'no'**:  
 assistant\_speaks(**'would you like me to continue reading the next articles?'**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 continue  
 elif sub\_intent == **'no'**:  
 assistant\_speaks(**'If you want to find out more, just let me know. Exiting news module'**)  
 break  
 elif sub\_intent == **'no'**:  
 assistant\_speaks(**'Alright, i am going to search for the top headlines'**)  
 url = (**'http://newsapi.org/v2/top-headlines?'**  
 **'country=us&'**  
 **'apiKey=b71542370a6247d493860e6b01d0d713'**)  
 response = requests.get(url).json()  
 articles = response[**'articles'**]  
 assistant\_speaks(  
 **'Say stop after I finish reading the headline to tell you its content. To continue reading '**  
 **'the headlines, just say continue. I found the following articles: '**)  
 for article in articles:  
 title = article[**'title'**]  
 url = article[**'url'**]  
 content = article[**'content'**]  
 assistant\_speaks(title)  
 ans = get\_audio()  
 if **'continue'** in ans:  
 continue  
 elif **'stop'** in ans:  
 assistant\_speaks(**'I will read the article content'**)  
 assistant\_speaks(content)  
 assistant\_speaks(**'I can open the webpage which contains the article source. Do you want me'**  
 **' to do that? '**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 driver = webdriver.Firefox()  
 driver.implicitly\_wait(1)  
 driver.maximize\_window()  
 driver.get(url)  
 elif sub\_intent == **'no'**:  
 assistant\_speaks(**'would you like me to continue reading the next articles?'**)  
 ans = get\_audio()  
 sub\_resp, sub\_intent = chatbot\_response(ans)  
 if sub\_intent == **'yes'**:  
 return  
 elif sub\_intent == **'no'**:  
 assistant\_speaks(**'If you want to find out more, just let me know. Exiting news module'**)  
 break  
 elif **'exit'** in ans:  
 break  
 except Exception as e:  
 print(e)  
 assistant\_speaks(**"I don't understand, Can you please repeat?"**)  
 ans = get\_audio()  
  
  
def create\_event(service, start\_time\_str, summary, duration=1, description=None, location=None):  
 matches = list(datefinder.find\_dates(start\_time\_str))  
 if len(matches):  
 start\_time = matches[0]  
 end\_time = start\_time + timedelta(hours=duration)  
  
 event = {  
 **'summary'**: summary,  
 **'location'**: location,  
 **'description'**: description,  
 **'start'**: {  
 **'dateTime'**: start\_time.strftime(**"%Y-%m-%dT%H:%M:%S"**),  
 **'timeZone'**: **'Europe/Bucharest'**,  
 },  
 **'end'**: {  
 **'dateTime'**: end\_time.strftime(**"%Y-%m-%dT%H:%M:%S"**),  
 **'timeZone'**: **'Europe/Bucharest'**,  
 },  
 **'reminders'**: {  
 **'useDefault'**: False,  
 **'overrides'**: [  
 {**'method'**: **'email'**, **'minutes'**: 24 \* 60},  
 {**'method'**: **'popup'**, **'minutes'**: 10},  
 ],  
 },  
 }  
 return service.events().insert(calendarId=**'primary'**, body=event).execute()

* + - * 1. Aplicație utilitară

*# importing cv2*  
import cv2  
import numpy as np  
import math  
from numpy import ones,vstack  
from numpy.linalg import lstsq  
  
  
*# rect1*  
start\_point = (100, 100)  
end\_point = (300, 320)  
*# rect2*  
start\_point2 = (300, 100)  
end\_point2 = (450, 500)  
*# rect3*  
start\_point3 = (450, 200)  
end\_point3 = (700, 350)  
*# rect4*  
start\_point4 = (450, 350)  
end\_point4 = (800, 580)  
common\_walls = [((300,100),(300,320)),((450,200),(450,350)),((450,350),(700,350))]  
  
  
  
rectangle\_points = [(start\_point, end\_point), (start\_point2, end\_point2), (start\_point3, end\_point3),  
 (start\_point4, end\_point4)]  
  
  
  
def ccw(A, B, C):  
 return (C[1] - A[1]) \* (B[0] - A[0]) > (B[1] - A[1]) \* (C[0] - A[0])  
  
  
*# Return true if line segments AB and CD intersect*  
def intersect(A, B, C, D):  
 return ccw(A, C, D) != ccw(B, C, D) and ccw(A, B, C) != ccw(A, B, D)  
  
  
*# Reading an image in default mode]*  
  
  
def segm\_check(origin, point, rectangle, skip\_line=[]):  
 global common\_walls  
 max\_x = max(rectangle[0][0], rectangle[1][0])  
 min\_x = min(rectangle[0][0], rectangle[1][0])  
  
 max\_y = max(rectangle[0][1], rectangle[1][1])  
 min\_y = min(rectangle[0][1], rectangle[1][1])  
  
 if intersect((point[0], point[1]), (origin[0], origin[1]), (min\_x, min\_y), (min\_x, max\_y)):  
 if ((min\_x, min\_y), (min\_x, max\_y)) not in common\_walls and ((min\_x, max\_y), (min\_x, min\_y)) not in common\_walls:  
 print((min\_x, min\_y), (min\_x, max\_y))  
 print((min\_x, max\_y), (min\_x, min\_y))  
 print(**'a'**)  
 *#if len(skip\_line)<2:*  
return True  
 if intersect((point[0], point[1]), (origin[0], origin[1]), (min\_x, max\_y), (max\_x, max\_y)):  
 if ((min\_x, max\_y), (max\_x, max\_y)) not in common\_walls and ((max\_x, max\_y), (min\_x, max\_y)) not in common\_walls:  
 print((min\_x, max\_y), (max\_x, max\_y))  
 print(**'b'**)  
 *#if len(skip\_line)<2:*  
return True  
 if intersect((point[0], point[1]), (origin[0], origin[1]), (max\_x, max\_y), (max\_x, min\_y)):  
 if ((max\_x, max\_y), (max\_x, min\_y)) not in common\_walls and ((max\_x, min\_y),(max\_x, max\_y)) not in common\_walls:  
 print((max\_x, max\_y), (max\_x, min\_y))  
 print(**'c'**)  
 *#if len(skip\_line) < 2:*  
return True  
 if intersect((point[0], point[1]), (origin[0], origin[1]), (max\_x, min\_y), (min\_x, min\_y)):  
 if ((max\_x, min\_y), (min\_x, min\_y)) not in common\_walls and ( (min\_x, min\_y),(max\_x, min\_y)) not in common\_walls:  
 print((max\_x, min\_y), (min\_x, min\_y))  
 print(**'d'**)  
 *#if len(skip\_line) < 2:*  
return True  
 else:  
 return False  
  
  
def check\_point(point, rectangle):  
 max\_x = max(rectangle[0][0], rectangle[1][0])  
 min\_x = min(rectangle[0][0], rectangle[1][0])  
  
 max\_y = max(rectangle[0][1], rectangle[1][1])  
 min\_y = min(rectangle[0][1], rectangle[1][1])  
 *#schimbat semne*  
if point[0] < min\_x and point[0] > max\_x and point[1] < min\_y and point[1] > max\_y:  
 return True  
 else:  
 return False  
  
  
def check\_contours(circle\_origins, point, rectangle\_list):  
 cnt = 0  
 skip\_lines = []  
 for rect in rectangle\_list:  
 *#daca iese din dreptunghiul tinta in altul*  
 *#cauza bug: doua dreptunghiuri unul peste celalalt*  
  
if segm\_check(circle\_origins, point, rect,skip\_lines):  
 *# if not check\_point(point, rect):*  
cnt += 1  
 *#daca iese din dreptunghi in afara casei*  
 *# if check\_point(point, rect):*  
 *# cnt += 1*  
  
return cnt  
  
  
def new\_dest(origine, old\_dest):  
 new\_destination = []  
 if origine[0] < old\_dest[0]:  
 sinx = math.sin((2 \* math.pi) \* i / 36)  
 cosx = math.cos((2 \* math.pi) \* i / 36)  
 dest\_point\_x = int(x + cosx \* 160)  
 dest\_point\_y = int(y + sinx \* 160)  
 new\_destination.append()  
  
senzor\_fum = cv2.imread(**"smoke\_sensor.png"**)  
senzor\_temp = cv2.imread(**"temperature\_sensor.jpg"**)  
senzor\_temp = cv2.resize(senzor\_temp,(50,50))  
senzor\_miscare = cv2.imread(**"motion\_senzor.jpg"**)  
senzor\_miscare = cv2.resize(senzor\_miscare,(50,50))  
  
senzor\_temp = cv2.resize(senzor\_temp, (50, 50))  
circle\_origins = []  
senzor\_origins = []  
senzor\_list = [senzor\_fum, senzor\_temp, senzor\_miscare]  
senzor\_index = 0  
  
  
*# https://eyenetworks.no/en/wifi-signal-loss-by-material/*  
*# https://blog.ibwave.com/a-closer-look-at-attenuation-across-materials-the-2-4ghz-5ghz-bands/*  
constanta\_material = [**'caramida'**, **'beton'**, **'rigips'**, **'lemn'**]  
  
  
def draw\_circle(x, y, rez, line\_width=2):  
  
 for i in range(62):  
 sinx = math.sin((2 \* math.pi) \* i / 62)  
 cosx = math.cos((2 \* math.pi) \* i / 62)  
 *# conditie daca linia intersecteaza conturul*  
dest\_point\_x = int(x + cosx \* 200)  
 dest\_point\_y = int(y + sinx \* 200)  
 cnt = check\_contours((x, y), (dest\_point\_x, dest\_point\_y), rectangle\_points)  
 if cnt:  
 dest\_point2\_x = int(x + cosx \* (200 - 20 \* cnt))  
 dest\_point2\_y = int(y + sinx \* (200 - 20 \* cnt))  
 cv2.line(rez, (x, y), (dest\_point2\_x, dest\_point2\_y), (0, 200, 0), line\_width)  
 else:  
 cv2.line(rez, (x, y), (dest\_point\_x, dest\_point\_y), (0, 200, 0), line\_width)  
 img = cv2.imread(**'gateway.jpg'**)  
 img = cv2.resize(img, (50,50))  
 rez[(y-25):(y-25) + img.shape[0],  
 x-25:x-25 + img.shape[1]] = img  
  
  
def check\_senzor(rez, senzor\_coords, circle\_coords, senzor\_img):  
 found = False  
 for circle\_coord in circle\_coords:  
 draw\_circle(circle\_coord[0], circle\_coord[1], rez, 40)  
 if rez.item(senzor\_coords[1], senzor\_coords[0], 1) == 200:  
 found = True  
 break  
  
 return found  
  
  
def draw\_senzor(rez, senzor\_coords, circle\_coords):  
  
 for senzor\_coord in senzor\_coords:  
 rez\_copy = rez.copy()  
 senzor\_img = senzor\_coord[2]  
 if check\_senzor(rez\_copy, senzor\_coord, circle\_coords, senzor\_img):  
 senzor\_cp = senzor\_img.copy()  
 *#convertirea culorilor*  
red = np.where((senzor\_cp[:, :, 0] == 0) & (senzor\_cp[:, :, 1] == 0) & (senzor\_cp[:, :, 2] == 0))  
 senzor\_cp[red] = (0, 255, 00)  
 rez[senzor\_coord[1]:senzor\_coord[1] + senzor\_img.shape[0],  
 senzor\_coord[0]:senzor\_coord[0] + senzor\_img.shape[1]] = senzor\_cp  
 else:  
 senzor\_cp = senzor\_img.copy()  
 green = np.where((senzor\_cp[:, :, 0] == 0) & (senzor\_cp[:, :, 1] == 0) & (senzor\_cp[:, :, 2] == 0))  
 senzor\_cp[green] = (0, 0, 255)  
 rez[senzor\_coord[1]:senzor\_coord[1] + senzor\_img.shape[0],  
 senzor\_coord[0]:senzor\_coord[0] + senzor\_img.shape[1]] = senzor\_cp  
  
  
def draw\_walls(image):  
 image = cv2.rectangle(image, start\_point, end\_point, color, thickness)  
 image = cv2.rectangle(image, start\_point2, end\_point2, color, thickness)  
 image = cv2.rectangle(image, start\_point3, end\_point3, color, thickness)  
 image = cv2.rectangle(image, start\_point4, end\_point4, color, thickness)  
  
  
def click\_event(event, x, y, flags, param):  
 global rectangle\_points  
 global circle\_origins  
 global senzor\_origins  
 global senzor\_index  
 replace = image.copy()  
 original\_image = image.copy()  
 alpha = 0.4  
 rez = cv2.addWeighted(replace, alpha, image, 1 - alpha, 0)  
  
 if event == cv2.EVENT\_RBUTTONDOWN:  
 *# cv2.circle(rez, (x, y), 200, (255, 127, 0), 5)*  
  
circle\_origins.append((x, y))  
 for origin in circle\_origins:  
 draw\_circle(origin[0], origin[1], rez)  
 draw\_senzor(rez, senzor\_origins, circle\_origins)  
 draw\_walls(rez)  
 cv2.imshow(**'image\_window'**, rez)  
  
 if event == cv2.EVENT\_LBUTTONDOWN and circle\_origins is not None:  
 *# cv2.circle(rez, (circle\_origins[0], circle\_origins[1]), 160, (255,127, 0), 5)*  
for origin in circle\_origins:  
 draw\_circle(origin[0], origin[1], rez)  
 senzor\_origins.append((x, y, senzor\_list[senzor\_index]))  
 draw\_senzor(rez, senzor\_origins, circle\_origins)  
 draw\_walls(rez)  
 cv2.imshow(**'image\_window'**, rez)  
  
 if event == cv2.EVENT\_MBUTTONDOWN:  
 circle\_origins = []  
 senzor\_origins = []  
 cv2.imshow(**'image\_window'**, rez)  
  
 if event == cv2.EVENT\_MOUSEWHEEL:  
 print(senzor\_index)  
 senzor\_index += 1  
 if senzor\_index >= len(senzor\_list):  
 senzor\_index = 0  
  
  
image = np.zeros((800, 1000, 3), np.uint8)  
*# Window name in which image is displayed*  
window\_name = **'image\_window'**  
  
*# Blue color in BGR*  
color = (255, 0, 0)  
  
*# Line thickness of 2 px*  
thickness = 4  
  
*# Using cv2.rectangle() method*  
*# Draw a rectangle with blue line borders of thickness of 2 px*  
image = cv2.rectangle(image, start\_point, end\_point, color, thickness)  
image = cv2.rectangle(image, start\_point2, end\_point2, color, thickness)  
image = cv2.rectangle(image, start\_point3, end\_point3, color, thickness)  
image = cv2.rectangle(image, start\_point4, end\_point4, color, thickness)  
  
cv2.imshow(window\_name, image)  
cv2.setMouseCallback(window\_name, click\_event)  
cv2.waitKey()  
  
cv2.destroyAllWindows()