UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Platformă inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea *titlului de Inginer* în domeniul *Electronică și Telecomunicații*
programul de studii *Microelectronică, Optoelectronică și Nanotehnologii*

Conducători științifici

Prof. Dr. Ing. Corneliu BURILEANU

Drd. Ing. Georgian NICOLAE

Absolvent

Anca-Maria PIȘCOCI

București, 2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **DCAE**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **PIŞCOCI A.Gh. Anca-Maria , 442E**

1. Titlul temei: Platformă inteligentă pentru sisteme embedded cu aplicaţii în terapia copiilor cu tulburări de spectru autist

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) şi specificaţii de proiectare:
Proiectul constă în dezvoltarea unei platforme software inteligente destinată sistemelor embedded, în cazul de față robotul umanoid NAO. Platforma este bazată pe o arhitectură de tip client-server. Se va implementa atât partea de client cât și partea de server.
Partea de client presupune dezvoltarea unei intefețe web prin intermediul căreia utilizatorul va avea la dispoziție o serie de comenzi ce pot fi transmise robotului.
Partea de server presupune implementarea unui serviciu REST cu rolul de a interfața ultilizatorul cu diferite proceduri parametrizabile de care dispune robotul NAO. Comenzile primite de la utilizator vor fi înregistrate într-o bază de date SQL cu scopul de a fi analizate ulterior de specialiștii în terapia copiilor cu tulburări de spectru autist.

3. Resurse folosite la dezvoltarea proiectului:
-Limbaje de programare: Python 2.7, HTML, CSS, JavaScript, SQL -Medii de devoltare: PyCharm

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
Programarea calculatoarelor, Structuri de date şi algoritmi, Programare Obiect-Orientată

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2018-11-28 21:47:58

Conducător(i) lucrare,
Prof. dr. ing. Corneliu BURILEANU

semnătura:.....

Drd. ing. Georgian NICOLAE

semnătura: .....

Student,

semnătura:.....

Director departament,
Prof. dr. ing Claudius DAN

semnătura:.....

Decan,
Prof. dr. ing. Cristian NEGRESCU

semnătura:.....

Cod Validare: **600f9976b4**

Declaraţie de onestitate academică

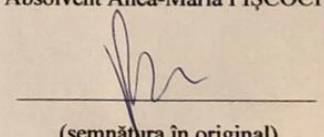
Prin prezenta declar că lucrarea cu titlul "*Platformă inteligentă pentru sisteme embedded cu aplicaţii în terapia copiilor cu tulburări de spectru autist*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Electronică, Telecomunicaţii şi Tehnologia Informaţiei*, programul de studii *Microelectronică, Optoelectronică şi Nanotehnologii* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, 2019

Absolvent Anca-Maria PIŞCOCI



(semnătura în original)

Cuprins

Listă de Figuri

Listă de Tabele

Listă de Acronime

API – Application Program Interface

ASP – Active Server Pages

CSS – Cascading Style Sheet

DCM – Device Communication Manager

DOM – Document Object Model

FSR – Force Sensitive Resistors

GUI – Graphical User Interface

HTML –HyperText Markup Language

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

IP – Internet Protocol

JS – JavaScript

JSON – JavaScript Object Notation

JSP – Java Server Page

PHP – Hypertext Preprocessor

REST – Representational State Transfer

SGBD – Sistemele de gestiune a bazelor de date

SOAP – Simple Object Access Protocol

SPA – Single Page Applications

SpeeD – Speech and Dialogue

SQL – Structured Query Language

TCP – Transmission Control Protocol

UI – User Interface

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

WEB – World Electronic Base

XML – Extensible Markup Language

XHTML - eXtensible HyperText Markup Language

1

INTRODUCERE

1.1 Motivația lucrării

Motivația principală în alegerea acestei teme de licență, *Platformă inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist*, se bazează pe interesul personal pentru interacțiunea om-robot care prin aplicații specifice poate contribui la îmbunatățirea societății și a mediului de trai. Dezvoltarea aplicațiilor de acest fel reprezintă o provocare specială deoarece implică îmbinarea componentei tehnologice cu o componentă psihologică, fiind necesară colaborarea specialiștilor din ambele domenii.

În ultimele decenii, foarte mulți oameni suferă de diferite boli ce au un efect dramatic asupra calității vieții. Din dorința de a veni în ajutorul acestor oameni să se integreze în societate, câteva tehnici ar trebui dezvoltate și luate în considerare pentru a le ușura viața. Printre tehnicile menționate anterior se numară și acelea dezvoltate cu intenția de a facilita progresul terapiei copiilor cu tulburări de spectru autist.

NAO Robotul Umanoid Autonom reprezintă nucleul acestui proiect, fiind creat special de Aldebaran Robotics pentru a fi folosit în procese de terapie. De obicei, copilul evită contactul cu specialistul, indiferent de funcția pe care acesta o ocupă, fie ca vorbim de educator, profesor sau psiholog. Pentru ca un copil să fie atras de întreaga activitate efectuată în timpul trapiei și pentru a putea învăța trăsăturile de bază ale comportamentului uman, trebuie descoperită o metodă interesantă și atragatoare ce poate fi aplicată. În acest context, NAO este un candidat ideal, acesta poate fi folosit pentru a ajuta pacientul să învețe cuvinte noi, să recunoască diferite tipare sau să reproducă anumite mișcări. Relația cu robotul NAO, comunicarea și tot ce are legătură cu interacțiunea se realizează într-o manieră naturală.

S-au demonstrat în foarte multe cazuri beneficiile utilizării terapeutice a roboților. Roboții terapeutici sunt roboți sociali folosiți pentru îngrijirea persoanelor cu afecțiunii comportamentale, emoționale sau motorii, precum și a bolilor cornice sau terapii de reabilitare. Pentru ca un robot să intercaționeze și să comunice cu persoanele în cauză, acesta trebuie să respecte câteva reguli comportamentale social umane. Un efect important al terapiei cu robotul umanoid NAO este că pacientul are parte de companie în timpul procesului dificil de terapie, iar specialistul poate fi înlocuit de robot în scopuri practice. Un factor important este faptul că robotul NAO este predictibil, fapt ce încurajează copilul să colaboreze. Aceasta face ca această metodă să fie deosebit de interesantă pentru tratamentul copiilor cu o mare reticență în vorbire.

1.2 Obiectivele principale

Scopul acestui proiect este dezvoltarea unei platforme software inteligente destinată sistemelor embedded, în cazul de față NAO Robotul Umanoid Autonom. Platforma este bazată pe o arhitectură de tip Client-Server. Prima parte a proiectului presupune dezvoltarea unei interfețe web prin intermediul căreia utilizatorul va avea la dispoziție o serie de comenzi ce pot fi transmise robotului. Partea de server presupune implementarea unui serviciu REST cu rolul de a interfața utilizatorul cu diferite proceduri parametrizabile de care dispune robotul NAO. Comenzile primite de la utilizator vor fi înregistrate într-o bază de date SQL cu scopul de a fi analizate ulterior de specialiști în terapia copiilor cu tulburări de spectru autist.

Obiectivul central al acestei lucrări de diplomă este de a expune utilizatorului sau terapeutului, prin intermediul unei interfețe web, diferite programe terapeutice realizate de către specialiști în domeniu ce îl au ca protagonist pe robotul umanid NAO. Se dorește ca aplicația să fie compatibilă cu dispozitivele mobile de tip Android, IOS și Windows, având o interfață accesibilă utilizatorilor fără cunoștințe tehnice (terapeuți, specialiști psihologi).

1.3 Sumar

În Capitolul 2 sunt prezentate sumar specificațiile robotului umanoid NAO, dar și resursele necesare rulării toolului și dezvoltării aplicației. Capitolul 3 abordează pașii ce trebuie parcurși pentru dezvoltarea aplicațiilor web. Sunt prezentate succint câteva din tehnologiile folosite în procesul dezvoltării aplicațiilor web, comunicarea client-server, dar și limbajele folosite în decursul defășurării acestei lucrări. Capitolul 4 va detalia sistemele de gestionare al bazelor de date pornind de la informațiile de bază folosite. În capitolul 5 se va descrie în detaliu aplicația Nao Terapia, respectiv platforma inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist. Pentru fiecare parte a aplicației vor fi prezentate arhitectura, modul de implementare și exemple concludente.

2
ROBOȚII UMANOIZI

2.1 Aspecte generale

Termenul 'Robot' a fost prima dată folosit de către Karel Čapek și Josef Čapek în lucrări science fiction. Karel Čapek a utilizat acest termen în piesa R.U.R., din anul 1921, pentru a denumi muncitorii cu asemănare umană, care se dezvoltă în rezerve. Pentru o lungă perioadă de timp roboții au fost irealizabili. Pentru a putea fi realizați, un număr de probleme urmau să fie rezolvate. Acești roboți trebuiau să acționeze și reacționeze autonom, însă mișcarea lor era restrânsă de cele două picioare ca locomoție. Mai mult, aceștia trebuiau să fie capabili să lucreze cu brațele. Problemele de bază au fost rezolvate începând cu anii 2000, când au apărut dezvoltări noi în acest domeniu.

Un robot umanoid este un robot cu forma corpului construit astfel încât să semene cu cea a corpului uman. În general, roboții umanoizi au un trunchi, un cap, două brațe și două picioare, deși unele forme de roboți umanoizi pot modela numai o parte a corpului, de exemplu, de la talie în sus. De asemenea, anumiți roboți umanoizi au fost concepuți pentru a reproduce caracteristicile feței umane, cum ar fi ochi și gură. Proiectarea unui robot umanoid se poate realiza în scopuri funcționale, cum ar fi interacțiunea cu instrumente și medii umane, în scopuri experimentale, cum ar fi studiul locomoției bipedice. Androizii sunt roboți umanoizi construiți pentru a se asemăna estetic cu oamenii, iar gynoidele sunt roboți umanoizi construiți pentru a semăna femeii.

Termenul autonom, în ceea ce privește roboții, se referă la capabilitățile acestora de a îndeplini funcții sub un anumit grad de autonomie. Autonomie înseamnă abilitatea de a se administra, mai exact independența controlului. Această caracteristică implică faptul că termenul definit mai sus este o proprietate a unei relații între doi agenți, în cazul de față, între designer și robotul autonom. Independența, învățarea și dezvoltarea sunt factori ce duc la ridicarea gradului de autonomie a unui agent. [1]

2.2 Aplicații ale roboților umanoizi

Roboții umanoizi sunt folosiți, în ziua de astăzi, ca instrumente de cercetare în multe domenii stiințifice. Oamenii de știință studiază structura și comportamentul organismului uman (biomecanica) pentru a construi roboți umanoizi. Pe de altă parte, încercarea de a imita corpul uman conduce la o întelegere mai bună a acestuia. Scopul initial al cercetării robotilor umanoizi s-a realizat pentru construcția ortezelor și a protezelor. Această cercetare a condus la transferul cunostințelor între ambele discipline.

În afară de cercetare, roboții umanoizi sunt creați și pentru a îndeplini sarcini umane, cum ar fi asistența persoanală. Aceștia fiind capabili să asiste oameni bolnavi, vârsnici sau să înlocuiască personalul uman în locuri de muncă ce pun în pericol viața umană.

Seviciile prestare de roboții umanoizi se întind de la anumite vocații bazate pe procedură, cum ar fi administratori de recepție sau lucrători de linie de productie pentru automobile pana la relationarea cu clienții, citirea emoțiilor sau recunoașterea vocală. Roboții umanoizi pot fi folosiți de către organizații ce ajută persoanele ce suferă de tulburări de spectru autist, dar și pentru a face sport și a practica yoga. Toate aceste activități ce pot fi efectuate de un robot umanoid sunt dezvoltate prin cercetare în plan experimental.

2.3 NAO – Robot Umanoid Autonom

NAO este robotul umanoid, autonom și biped realizat de Aldebaran Robotics, o companie robotică franceză cu sediul la Paris, achiziționată de Grupul SoftBank în 2015. NAO este cel mai important și mai utilizat robot umanoid pentru educație, asistență medicală și cercetare. Acesta are o înălțime de 58 cm, este autonom și complet programabil, poate să se deplaseze, să vorbească, să asculte și chiar să recunoască fața umană.

Alcătuit dintr-o combinație unică de hardware și software, NAO conține senzori, motoare și componente software conduse de un sistem dedicat de operare – NAOqi. NAO are capacitatea de a-și detecta împrejurările datorită combinației de tehnologii anterior amintite. Folosindu-se de software-ul încorporat, acesta este capabil să interpreteze ceea ce detectează urmând să activeze răspunsuri programate.

Crearea componentelor complexe, accesarea datelor înregistrate de senzori și controlul robotului sunt posibile datorită librăriilor de mișcare, disponibile prin instrumente grafice cum ar fi software-ul de programare – Choregraphe.

Figura 2.1: Aspect exterior al robotului NAO [1]

2.3.1 Caracteristici Hardware

În cele ce urmează sunt prezentate detaliile tehnice ale robotului umanoid – NAO conform foilor de catalog.

Construcția

Robotul NAO este un robot de dimensiuni medii și greutate medie. Cei 574mm – înălțime, 275mm – lățime, 311mm – anvergura mâinilor, 4,5kg – greutate influențează în mod direct performanțele de mișcare ale robotului. Este construit din două tipuri de plastic rezistente la căldură, Policarbonatul-ABS (PC-ABS)/ Poliamida-66 (PA-66), și un tip de plastic rezistent la șocuri mecanice, XCF-30, ce are în componență și fibră de carbon în proporție de 30%. [2]

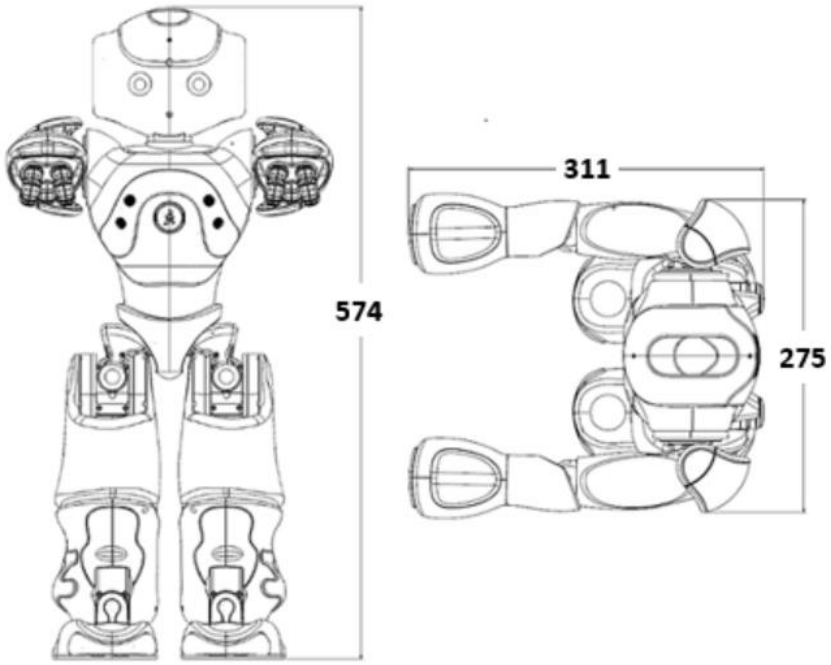


Figura 2.2: Dimensiunile robotului NAO [2]

Alimentarea

NAO este echipat cu o baterie Lithium-Ion, având o capacitate nominală de 2.25Ah. Durata de încărcare este mai puțin de 3 ore cu o autonomie de aproximativ 60 de minute. Robotul poate fi folosit în timp ce este la încărcat.

Unitățile de Comandă și Control

Robotul NAO (versiunile V5 și V6) dispune de două procesoare principale localizate unul la nivelul capului, iar celălalt la nivelul toracelui.

- UCP – ul de la nivelul capului este construit dintr-un procesor Intel x86, de tipul ATOM Z530 și are o memorie cache de 512KB . Având o viteză de ceas de 1.6 GHz, microcontrolerul este compatibil cu Intel System Controller Hub - Intel® SCH. Această componentă combină funcționalitatea plăcii grafice, ale interfeței procesorului și ale controlerului de memorie cu extensiile de intrare/ieșire, în vederea unui consum cât mai mic de energie.
- ATOM Z530 este un procesor cu un singur nucleu, conceput în principal pentru aplicații mobile și are avantajul eficienței energetice. În plus, arhitectura ATOM Z530 se bazează pe microarhitectura lui Bonnell fiind capabil să execute două instrucțiuni pe ciclu de ceas. Acesta face translația instrucțiunilor complexe de tip CISC în operații simple de tip RISC înainte de execuție.
- UCP – ul de la nivelul toracelui este un microprocesor de tipul ARM7TDMI, acesta controlează actuatoarele, distribuind informația de control tuturor microcontrolerelor locale de la nivelul fiecărui actuator. Mictrocontrolerul ARM7TDMI are un set de intrucțiuni pe 32 de biți, de tip RISC, de asemenea, oferă performanțe ridicate în ceea ce privește consumul redus de putere. Microcontrolerele locale de la nivelul actuatoarelor sunt Microchip pe 16 biți dsPICS și comunică cu UCP-urile secundare prin două magistrale de tipul RS-485, cu o viteză de 460 Kbps.

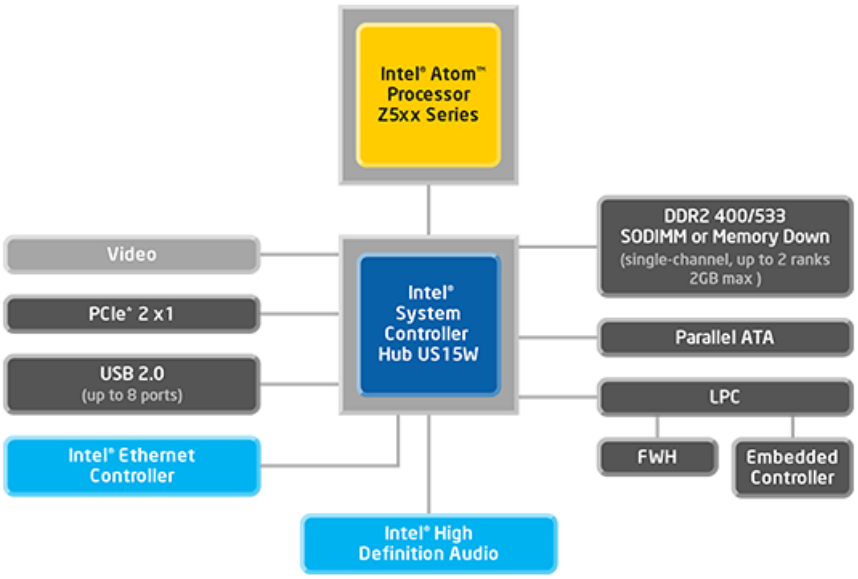


Figura 2.3: Diagrama seriile ATOM Z5xx [2]

Conectivitate

Robotul NAO suportă conexiuni prin două protocoale de comunicație: prima fiind de tip Ethernet, iar cealaltă prin Wi-Fi. Pentru conexiunea de tip Ethernet, robotul are la nivelul capului o mufă de tip RJ45, compatibilă cu 3 tipuri de adaptări: 10/100/1000 base T (viteze de 10Mbps, 1000Mbps sau 1 Gbps). Conexiunea prin cablu este cel mai des utilizată pentru setarea conexiunii prin WiFi. După conectarea cablului, robotului i se alocă o adresă IP locală, putând fi accesată printr-un browser. Dupa autentificare, robotul se va putea conecta la rețeaua wireless având parte de mobilitatea necesară, totodată primind o nouă adresă IP. [2]

Conexiuna la o rețea locală este necesară pentru programarea robotului prin intermediul programelor dedicate precum Choregraphe sau mediile clasice de programare (Python, C/C++, Java, etc), compatibil cu standardul IEEE 802.11 b/g/n.

LED-uri

Robotul NAO dispune de o multitudine de LED-uri folosite pentru notificări, animarea acestuia și răspunsurile la anumite scripturi rulate.

Localizare	Cantitate	Descriere
Pe cap	1 x 12	16 nivele de Albastru
Ochi	2 x 8	Toate culorile pe baza RGB
Urechi	2 x 10	16 nivele de Albastru
Butonul de pe piept	1 x 1	Toate culorile pe baza RGB
Picioare	2 x 1	Toate culorile pe baza RGB

Tabelul 2.1: LED-urile de pe Robotul NAO [2]

Audio

Pentru a interacţiona cu NAO la nivel audio, acesta dispune de un sistem stereo de difuzare format din două difuzoare plasate de o parte şi de alta a capului. Tot pe cap se află și 4 microfoane, având o senzitivitate de la 20mV/Pa +/-3dB până la 1 KHz şi o gamă de frecvenţă de la 300 Hz până la 8 KHz. [5]

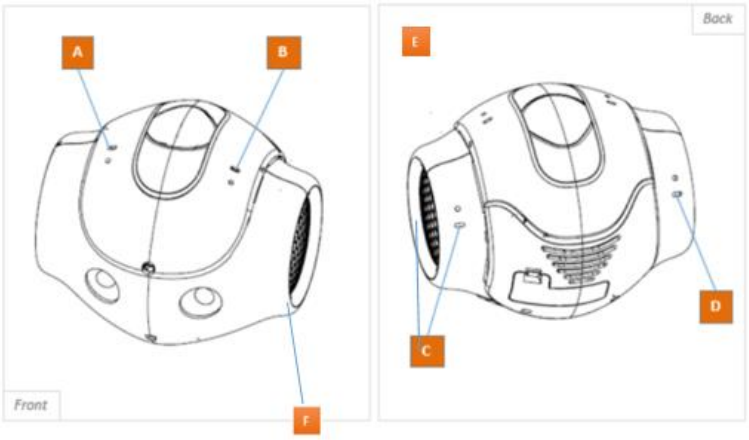


Figura 2.4: Dispunere Audio pe Robotul NAO [2]

Părti	**Localizare**		**Nume**
A	Fată	Dreapta	MicroFR
B		Stânga	MicroFL
C	Spate	Stânga	MicroRL
D		Dreapta	MicroRR
E		Stânga	LoudSpeakerLeft
F		Dreapta	LoudSpeakerRight

Tabelul 2.2: Dispunere Audio pe Robotul NAO [2]

Camere Video

Pe partea din față a lui Nao sunt plasate două camere video identice, care oferă imagini de 1280x960 rezoluție, la 90 de cadre pe secundă. Ele sunt folosite pentru a identifica obiectele din jur și spațiul în care se află Nao când se mișcă.

Robotul dispune de camera de tip SOC Image Sensor, modelul MT9M114, cu o rezoluție de 1.22 Mp, formatul optic de 1/6 inch, pixeli activi(HxV) 1288x968, cu mărimea unui pixel de 1.9µm*1.9µm, gama dinamică de 70dB. Semnal/Zgomot ratio de 37dB responsivitatea - 2.24V/Lux-sec (550 nm). [2]

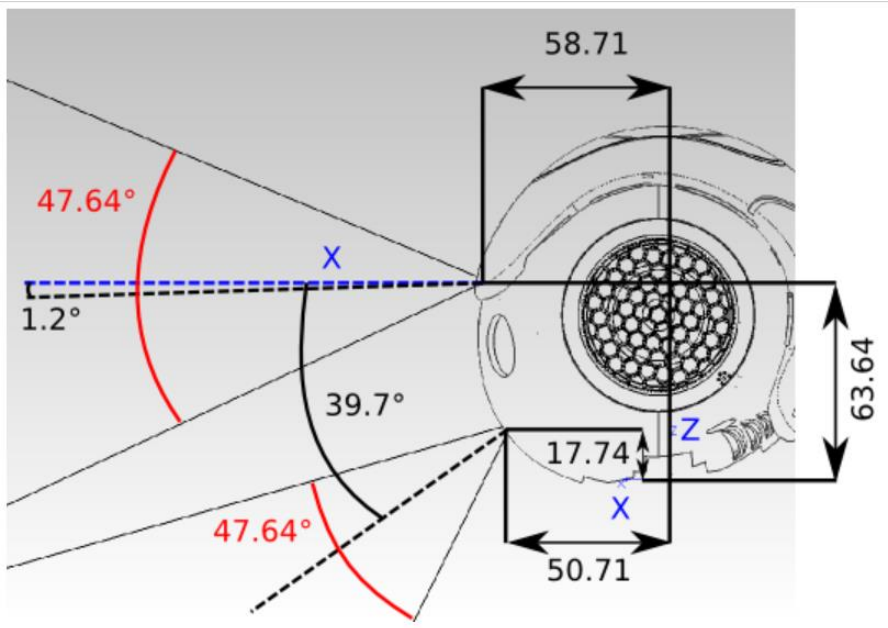


Figura 2.5: Apertura camerelor frontale [2]

Dispozitive Infra-Roșii și sonare

În ochii robotului NAO sunt prezente două dispozitive infra-roșii, acestea prezintă un emițător și un receptor, funcționarea realizându-se pe o lungime de undă de 940 mm. Comunicarea între roboți se realizează folosind dispozitivele infra-roșii în aplicații de comunicație. Pentru ca robotul să poată detecta obstacolele, implicit să le poată ocoli, acesta dispune de sonare ce funcționează la o frecventă de 40kHz având au un domeniu de detecție între 0.25mm și 2.55m. Dacă între robot si obstacol este o distanță mai mică de 0.25mm nici un fel de informație legată de distanță nu este returnată de către robot, știind totuși de existența obstacolului. [2]

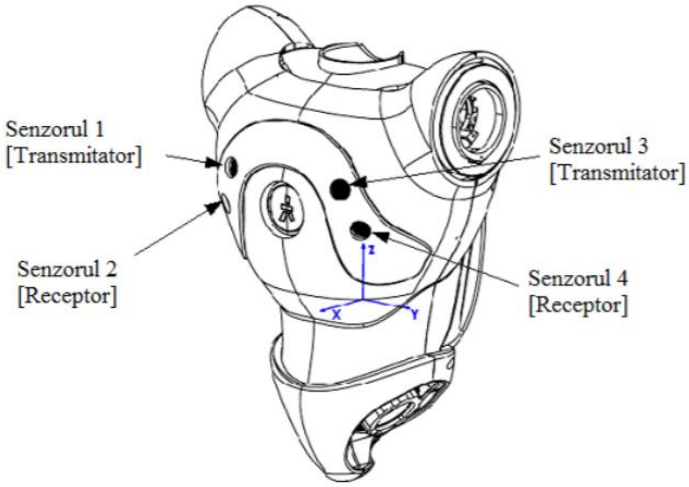


Figura 2.6: Senzorii de tip Sonar [2]

Girometrul si Accelerometrul

Aceste două tipuri de dispozitive constituie unitatea inerțială a robotului, acestea oferă informații despre poziția în mers și informații despre stabilitatea lui. Girometrul, bazat pe aflarea vitezei unghiulare date de 2 axe spațiale, măsoară orientarea (rad/s). Accelerometrul măsoară forța de apăsare g(m/) sau accelerația proprie.

Rezistențe sensibile la forță (Force Sensitive Resistors)

FSR-urile sunt senzori ce măsoară schimbarea de rezistență ținând cont de presiunea aplicată. FSR-urile sunt localizate în cele două picioare ale robotului NAO. Sunt în numar de 8 astfel de senzori, localizați câte patru pe fiecare talpă, având un domeniu de funcționare între 0 N și 25 N. Acest tip de senzori sunt compuși dintr-un polimer conductiv ce poate să-și modifice rezistența într-o manieră predictibilă, în funcție de presiunea aplicată pe suprafața sa.

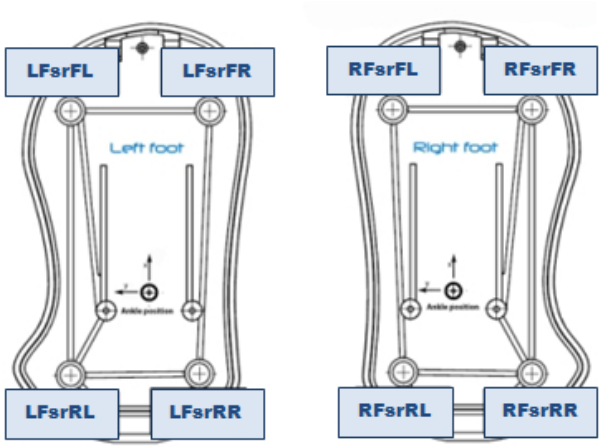


Figura 2.7: Poziționarea celor 8 senzori FRS în tălpile robotului NAO [2]

Senzorii tactili și de contact

Senzorii de contact descriu dacă cineva sau ceva atinge robotul NAO. Aceștia sunt poziționați în multe puncte ale robotului: butonul din mijlocul toracelui, trei senzori tactili pe cap, doi senzori tactili pe mâini și doi senzori la nivelul picioarelor ce au rol de amortizoare. Butonul din mijolcul toracelui are multe funcționalități de bază, pornind de la pornirea robotului, până la aflarea adresei IP. În funcție de dorința utilizatorului, fiecărui senzor I se poate atribui diverse funcionalități.

Senzori de poziție

Robotul NAO prezintă senzori expliciți de poziție denumiți generic encodere de rotație magnetice (ERM). Funcționalitatea acestora se bazează pe Efectul Hall, mișcările ce sunt efectuate se resimt prin străpungerea unui câmp magnetic ce poate determina o diferență în potențial prelucrată ulterior obținându-se astfel informații despre localizarea în spațiu.

Actuatoare si grade de libertate

Actuatoarele sunt motoarele robotului ce asigură mișcarea liberă a acestuia. Motoarele lui NAO sunt în număr de 25, plasate în diferite poziții la nivelul articulațiilor oferind o mobilitate cât mai bună și emularea cât mai corectă a mișcărilor corpului uman. Actuatoarele ce se regăsesc pe robotul NAO sunt motoare cu perii de carbon, datorită prețului redus și a vitezei reconfigurabile de către utilizator. Aceste motoare generează și în cuplu direct de la sursa de alimentare, fiind posibile datorita magneților staționari și a electromagneților rotativi.

Motoarele de tipul 1 sunt cele mai puternice. Datorită cuplului maxim ce se dezvoltă în blocaj, acestea se folosesc la picioare, reușind să mențină robotul stabil în mers și pe loc. Pentru restul articulațiilor, cum ar fi capul, coatele și umerii sunt utilizate motoare cu viteza de rotație mai mare. Pentru că sunt utilizate în activități ce implică multă mișcare se folosesc motoare de tipul 1 și 2.

	Motor tip 1	**Motor tip 2**	**Motor tip 3**
Model	22NT82213P	17N88208E	16GT83210E
Viteză	8 300 rpm ±10%	8 400 rpm ±12%	10 700 rpm ±10%
Cuplu	68 mNm ±8%	9.4 mNm ±8%	14.3 mNm ±8%
Cuplu nominal	16.1 mNm	4.9 mNm	6.2 mNm

Tabelul 2.3: Caracteristicile tipurilor de motoare [2]

Gradele de libertate exprimă, în mecanică, fiecare dintre mărimile scalare independente ce sunt necesare pentru a determina starea unui sistem. Numărul minim de parametri prin care se poate specifica starea unui sistem este reprezentat de gradele de libertate ale acestuia. Spre exemplu, pentru a determina poziția punctului material în spațiu este nevoie de coordonatele x,y și z, stiind că fiecare legătură scade numărul gradelor de libertate.

Robotul NAO dispune de 25 astfel de grade de libertate: 11 pentru partea inferioară (incluzând pelvisul și picioarele), 14 pentru partea superioară a trunchiului (inclusiv capul). În tabel sunt prezentate numărul gradelor de libertate dispuse pe robotul NAO.

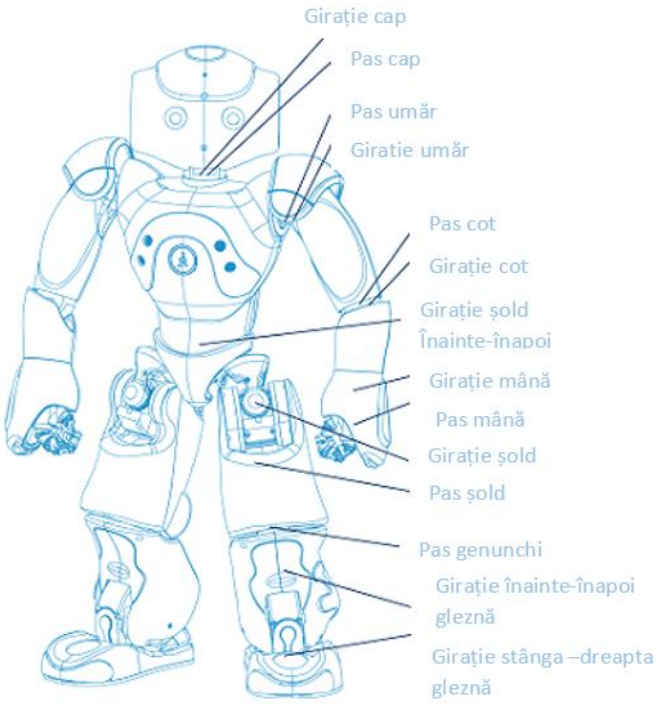


Figura 2.8: Articulațiile active robot NAO [2]

Componentă	Grade de libertate
Cap	2
Braț(x2)	5
Mână(x2)	1
Picior(x2)	5
Pelvis	1

Tabelul 2.4: Dispunerea gradelor de libertate pentru robotul NAO [2]

2.3.2 Caracteristici Software

NAOqi Framework

Robotul NAO dispune de un sistem de operare OpenNAO, ce are la bază un nucleu Linux/Gentoo ce rulează o serie de instrumente software denumite generic NAOqi prin care se asigură controlul și autonomia funcțiilor robotului. Această versiune este o versiune integrată, dezvoltată special pentru roboții umanoizi concepuți de Aldebaran. Acest sistem de operare asigură și rulează un numar mare de librării și programe. De asemenea, pentru a facilita interacțiunea cu robotul NAO, NAOqi este un sistem modular și distribuit, fiind capabil să lucreze cu un număr mare de fișiere executabile în funcție de preferințele utilizatorului. Arhitectura este bazată pe paralelism, sincronizare și evenimente, oferind posibilitatea utilizatorului să acceseze resursele de care are nevoie în timp ce împarte informația între acestea.

Avantajele sistemului distribuit sunt multe și diferite. Utilizatorul poate rula comportamente ale robotului locat cât și de la distanță. Funcționalitățile robotului (mișcarea, vorbirea, vederea etc.), pot fi grupate în funcție de cerințele aplicației țintă. Același cod poate fi compilat pe platforme diferite, dezvoltarea aplicațiilor fiind facilitată pe astfel de medii distribuite.

Funcționalitățile arhitecturii NAOqi sunt:

- Suportă diferite limbaje de programare. Aplicațiile pot fi scrise utilizând limbaje precum: C++, Java și Python.
- Execuția proceselor se poate realiza în paralel, secvențial sau prin apeluri de evenimente.
- Managementul simplu al memoriei partajate.
- Programarea aplicațiilor (sau API) facilitat de managementul interfețelor.
- Modularitatea – este la alegerea utilizatorului cum alege sa compileze aplicația: ca o librărie dinamică sau ca un fișier executabil.
- Platforme suportate: Linux, Windows sau MAC OS.

Arhitectura NAOqi este divizată în trei părți, din punct de vedere structural:

- NAOqi sau OpenNAO – sistemul de operare menționat anterior
- Biblioteca NAOqi, este divizată în diferite acțiuni pe care robotul este capabil să le execute. Avantajul bibliotecii este nivelul de abstractizare, robotul poate fi programat de către utilizator fără acces efectiv la această bibliotecă.
- Managementul de comunicații al dispozitivelor (DCM-Device Communication Manager), este modulul ce se ocupă de controlul comunicației dintre dispozitivele electronice de care dispune robotul, excepție fac senzorii audio și camera video. DCM este văzut că o legătură între nivelul software superior (compus din module) și nivelul software inferior (care este integrat în placile electronice). Are rolul de a prelua informații de la dispozitivele electronice, folosindu-se de unitatea de prelucrare de la nivelul toracelui.

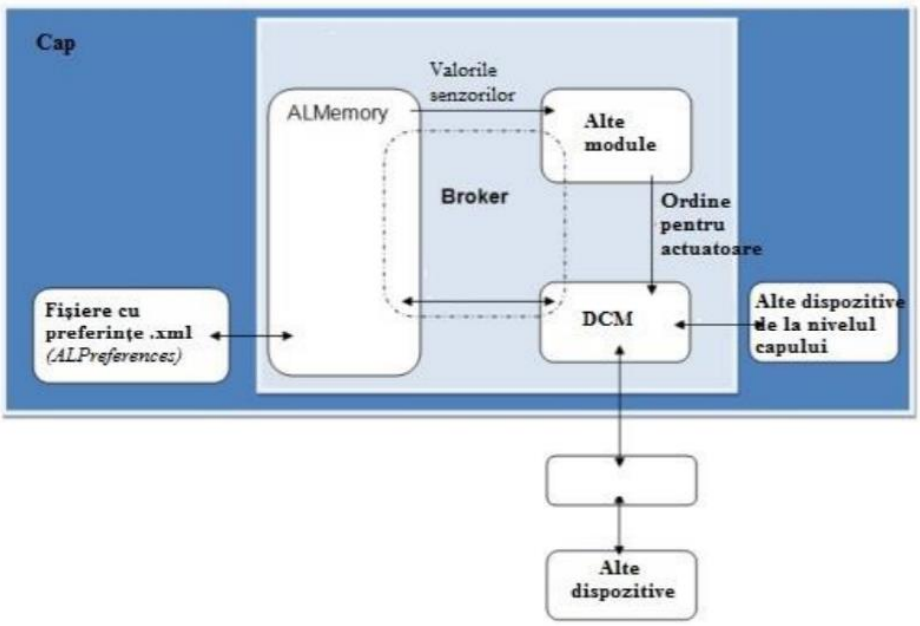


Figura 2.9: Legăturile realizate de DCM între nivelul software inferior și cel superior. [1]

Arhitectura NAOqi este orientată pe obiect, folosindu-se de trei tipuri principale de obiecte:

I. Broker (Agent) – este un obiect ce are două roluri: să ofere servicii de director (oferă posibilitatea utilizatorului să găsească modulele și metodele necesare) și să lucreze transparent. Pentru a putea accesa un modul, este necesar ca acesta să fie legat de un broker. Agenții sunt cei ce controlează rețeaua de comunicație. Un agent poate fi copilul unui alt agent, astfel formându-se un arbore distribuit. (Figura 2.).
II. Modul – metodele definite de utilizator.
III. Proxy – este un obiect ce are comportamentul modulului pe care îl reprezintă. Spre exemplu, crearea unui proxy ALPreferences, crează o copie a acestuia ce poate fi accesat de utilizator.

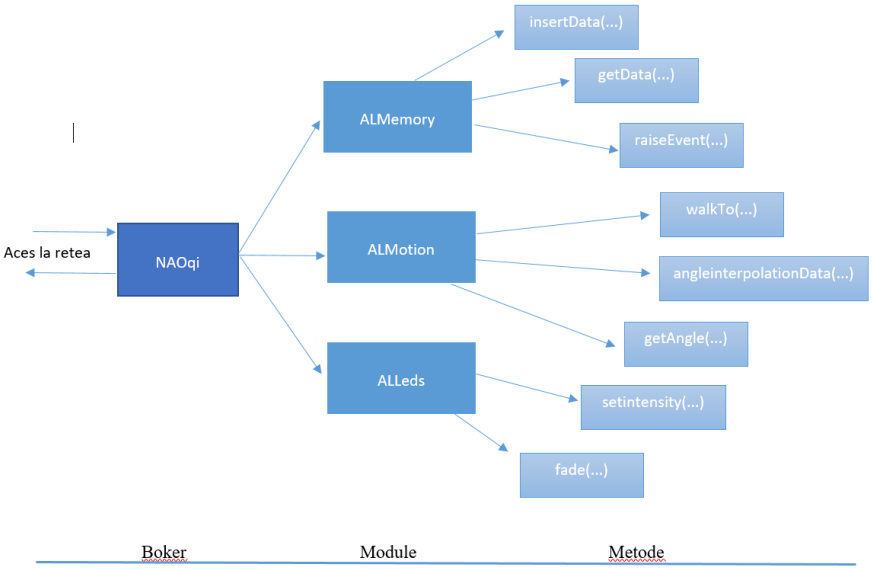


Figura 2.10: Relația dintre agenți, module și metodele corespunzătoare [1]

Choregraphe

Choregraphe este o aplicație desktop dezvoltată de Aldebaran Robotics, care permite programarea robotului. Acest mediu grafic permite utilizatorului să facă conexiuni de comportamente la nivel înalt cu ușurință, prin utilizarea unor cutii de comportamente specifice. În afară de aceasta, îi dă posibilitatea de reglare fină a mișcărilor comune. De asemenea, această aplicație permite programare în Python.

Choregraphe oferă utilizatorilor funcțiile NAOqi, într-un mod prietenos. Astfel, utilizatorul poate executa anumite comportamente predefinite, prin legarea unor casete specifice. De exemplu, dacă utilizatorul dorește ca robotul să execute anumite acțiuni în lanț, trebuie să lege secvențial secțiunile corespunzătoare; dacă se dorește să execute mai multe comportamente în același timp, casetele trebuie să fie legate paralel. Un exemplu de serie și legare paralelă, pentru a obține un comportament mai complex este prezentat în Figuri 2. și 2.

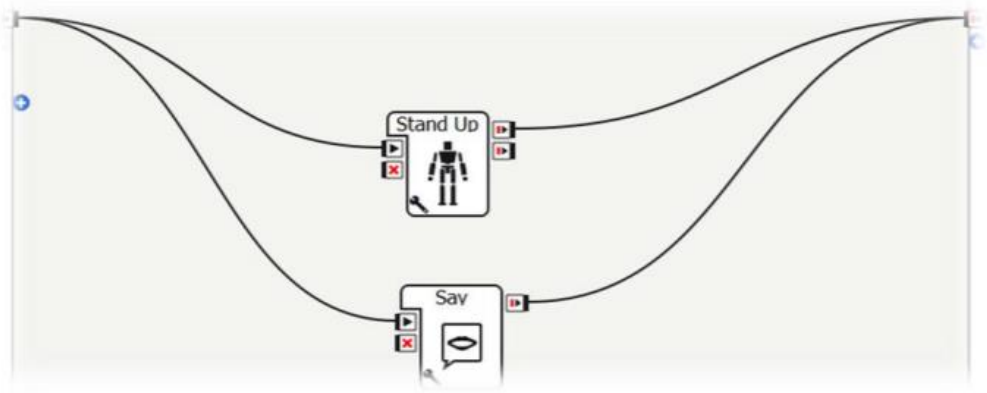


Figura 2.11: Acțiuni executate în paralele de către Nao în Choregraphe

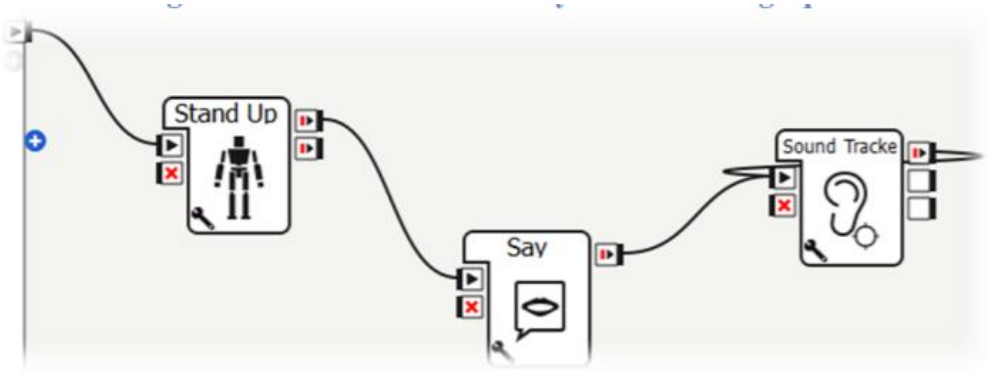


Figura 2.12: Acțiuni executate în paralele de către Nao în Choregraphe

3

TEHNOLOGII SOFTWARE UTILIZATE

Dezvoltarea web este un termen care se referă la procesul de creare a unui site web și poate varia de la dezvoltarea unei pagini simple la o serie de pagini complexe. Dezvoltarea web cuprinde mai multe acțiuni sau practici și unele dintre ele includ: web design, crearea de conținut, programare, sarcini de securitate a rețelei, precum și scripting pe partea clientului sau serverului.

În ultimii ani, dezvoltarea web a luat definiția creării sistemelor de gestionare a conținutului, care reprezintă etapa intermediară între utilizator și baza de date. Unul dintre cele mai importante lucruri ce trebuie luate în considerare sunt limbajele de programare.

Unul dintre aspectele cele mai importante ale dezvoltării web este programarea web care este realizată cu ajutorul limbajelor de programare/editare. Limbajele de dezvoltare web sunt platformele prin care instrucțiunile sunt comunicate unei mașini și acțiunile sunt urmărite.

Un limbaj de programare este, în general, împărțit în două componente: semantică și sintaxă. Cele trei caracteristici principale ale unor astfel de limbaje de programare sunt:

- Abstractizarea – majoritatea limbajelor de programare au anumite reguli care ajută la definirea stucturilor de date, precum și manipularea modului în care aceste comenzi sunt executate. Aceste reguli sunt denumite abstracții, iar fiecare limbaj de programare trebuie să conțină suficiente abstractizări, acestea bazându-se pe principiul abstractizării.
- Funcționalitatea și ținta – când sunt folosite limbaje de programare în dezvoltarea web este nevoie de ajutorul sistemului de operare care efectuează activitatea de calcul sau controlează algoritmul. Definiția completă a unui limbaj de programare include o descriere, o mașină sau un procesor care a fost idealizat pentru acea limbă.
- Puterea expresivă – în știința calculatoarelor, puterea expresivă (numită și expresivitate) a limbajului este determinată de multitudinea de idei care pot fi reprezentate și comunicate în acea limbă. Cu cât limbajul este mai expresiv, cu atât mai mare este varietatea și cantitatea de idei pe care le poate reprezenta.

3.1 Aplicațiile WEB

O aplicație web este orice program care realizează o funcție specifică prin utilizarea unui browser web într-o arhitectură client-server folosind tehnologiile deschise World Wide Web. Ele înlocuiesc modelele în care atât serverul cât și clientul rulează tehnologii proprii, mentenanța aplicațiilor de pe partea de client fiind prea complexă, costisitoare și susceptibilă la erori. În schimb, omniprezența browserelor web și comoditatea de a le folosi drept client conduce la eliminarea acestei mari probleme. Astfel, au evoluat serverele de aplicații, iar pe lângă limbajele de programare au apărut frameworkuri și tehnologii dedicate programării acestora.[3]

Termenul “Client” este folosit pentru mediul client-server pentru a face referire la programul pe care utilizatorul îl folosește pentru a putea rula o aplicație. Un mediu client-server este caracterizat de mai multe computere ce partajează informații, spre exemplu: introducerea informațiilor într-o bază de date. “Client” este aplicația folosită pentru a introduce informațiile, iar “Serverul” este aplicația utilizată pentru stocarea acestora.

Beneficiile utilizării aplicațiilor Web

O aplicație web scutește dezvoltatorul de responsabilitatea de a construi un client pentru un anumit tip de sistem de operare, astfel încât oricine poate folosi aplicația dacă are acces la internet.

Aplicațiile web utilizează în mod obișnuit o combinație de script-uri de pe server (ASP, PHP etc.) și de script-uri de la client (HTML, Javascript etc.) pentru a dezvolta aplicația. Scriptul de pe partea clientului se ocupă de prezentarea informațiilor, în timp ce scriptul de pe server se ocupă de stocarea și extragerea informațiilor.

Interacțiunea serviciilor Web cu aplicațiile Web

Un serviciu Web este o componentă software ce poate fi găsită pe Internet printr-un mechanism simplu de căutare. Pentru a se putea face disponibilă, folosește un sistem de mesaje XML standardizat. O altă parte importantă a serviciilor web este reprezentată de faptul că nu ar trebui să fie legate de limbajele de programare sau sistemele de operare, dar fiind totuși disponibile să realizeze un schimb de date între aplicații și sistemele respective.

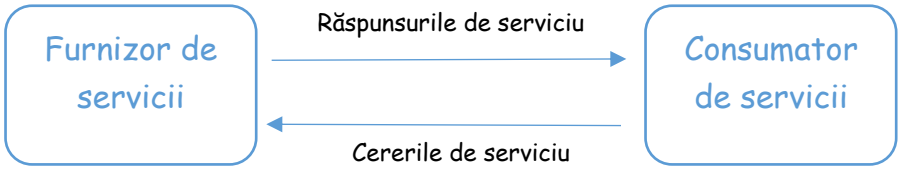


Figura 3.1: Interacțiunea dintre furnizorul de servicii și consumatorul de servicii Web

3.1.1 Javascript

JavaScript a fost pentru prima dată cunoscut sub numele de LiveScript, dar Netscape a schimbat numele în JavaScript, probabil datorită excitării generate de Java. JavaScript a apărut prima dată în Netscape 2.0 în 1995 cu numele LiveScript. Nucleul general al limbajului de programare a fost încorporat în Netscape, Internet Explorer și alte browsere web. [4]

JavaScript® (adesea redus la JS) este un limbaj de programare dinamic, interpretat, orientat pe obiect, cu funcții de primă clasă. Este cel mai bine cunoscut sub numele de limbaj de scripting pentru paginile Web, fiind folosit și în medii non-browser. Este un limbaj de scripting interpretat cu capabilități orientate pe stiluri de programare orientate pe obiect, imperative și funcționale.

JavaScript este un limbaj dinamic de scriere care susține construcția de obiecte bazate pe prototipuri. Sintaxa de bază este intenționat similară atât cu Java cât și cu C ++ pentru a reduce numărul de concepte noi necesare pentru a învăța limbajul.

JS poate funcționa atât ca limbaj procedural, cât și ca un limbaj orientat pe obiect. Obiectele sunt create în mod programabil în JavaScript, prin atașarea de metode și proprietăți la alte obiecte golite la timpul de execuție, spre deosebire de definițiile claselor sintactice comune în limbile compilate precum C++ și Java. Odată ce un obiect a fost construit, el poate fi folosit ca un model (sau prototip) pentru a crea obiecte similare.

Partea de Client JavaScript

JavaScript este una dintre cele mai populare și dinamice limbaje de programare utilizate pentru crearea și dezvoltarea site-urilor web. Aceast limbaj rulează pe partea de client a web-ului și poate fi folosit pentru a proiecta / programa modul în care paginile web se comportă la apariția unui eveniment. JavaScript este un limbaj ușor de învățat și, de asemenea, puternic de scripting, folosit pe scară largă pentru controlul comportamentului paginii web. [4]

Avantaje:

- Interacțiune limitată cu serverul – se pot valida comenzile utilizatorului înainte de a le trimite la server. Acest lucru economisește traficul serverului, rezultând o viteză mai mare.
- Feedback imediat către utilizator – aceștia nu trebuie să reacceseze pagina dacă au omis anumite informații.
- Interactivitate sporită – se pot crea interfete interactive, facilitând interacțiunea cu utilizatorul.
- Interfețe complexe

Limitări:

- JavaScript-ul de pe partea clientului nu permite citirea sau scrierea de fișiere. Acest lucru a fost păstrat din motive de securitate.
- JavaScript nu poate fi utilizat pentru aplicații de rețea, deoarece nu există un astfel de suport disponibil.
- JavaScript nu are capabilități multithreading sau multiprocesor. JavaScript nu este un limbaj de programare complet, acesta permite doar dezvoltarea paginilor HTML statice.

3.1.2 HTML, CSS, XML, JSON

HTML (Hypertext Markup Language) și CSS (Cascading Style Sheets) sunt două dintre tehnologiile de bază pentru construirea de pagini web. HTML oferă structura paginii, iar CSS aspectul vizual și audio, pentru o varietate de dispozitive. Alături de grafică și scripting, HTML și CSS sunt elementele de bază în realizarea de pagini și aplicații web. [3]

HTML este codul care stă la baza paginilor web. HTML este un limbaj de editare utilizat în crearea structurilor paginilor web ce pot fi afișate într-un browser sau navigator. Termenul HTML se traduce ca Limbajul de marcare a hipertextului. Scopul HTML este prezentarea informațiilor precum paragrafe, fonturi, tabele și altele. Acest limbaj nu descrie semantica documentului. W3C(World Wide Web Consort)se ocupă de specificațiile HTML.

HTML este un format text ce poate fi citit și editat de către utilizator, acesta din urmă putând utiliza un simplu editor de text pentru modificare. HTML poate fi generat direct utilizând tehnologii de codare precum: PHP, JSP sau ASP.

Ca orice alt limbaj, HTML a cunoscut o dezvoltare continuă. Astfel au apărut specificațiile 2.0, 3.0, 4.0. Este important că orice versiune ulterioară susține prelucrarea documentelor create în versiunile precedente. Specificarea curentă a limbajului poate fi obținut oricând pe serverul http://www.w3.org/.ium. [9]

Paginile HTML

Paginile HTML sunt formate din etichete sau tag-uri, având extensia ".htlm" sau ".htm". Etichetele au două părți: una de deschidere <eticheta> și alta de închidere </eticheta>. Browserul sau navigatorul web interpretează etichetele, afișându-le pe ecran.

CSS (sau "Cascading Style Sheets") este un standard pentru formatarea elementelor HTML. Este un limbaj pentru descrierea prezentării paginilor web, inclusiv culori, aspect și fonturi. Acesta permite adaptarea prezentării la diferite tipuri de dispozitive, cum ar fi ecrane mari, ecrane mici sau imprimante. CSS este independent de HTML și poate fi folosit cu orice limbaj bazat pe XML. Separarea codului HTML de CSS facilitează menținerea site-urilor, partajarea foilor de stil între pagini și adaptarea paginilor la medii diferite. Aceasta se numește separarea structurii sau a conținutului de prezentare. [7]

CSS a rezolvat o mare problemă a HTML-ului. Acesta nu intenționa niciodată să conțină etichete pentru formatarea unei pagini web. HTML a fost creat pentru a descrie conținutul unei pagini web, cum ar fi:

<h1> Acesta este un titlu </ h1>

<p> Acesta este un paragraf. </ p>

Atunci când etichete precum <*font*> și atributele de culoare au fost adăugate la specificația HTML 3.2, dezvoltatarea web a devenit foarte complicată. Dezvoltarea de site-uri mari, în care fonturile și informațiile de culoare au fost adăugate pe fiecare pagină, au devenit un proces lung și costisitor. Pentru a rezolva această problemă, World Wide Web (W3C) a creat CSS. Acesta a eliminat formatarea stilului din pagina HTML. Stilurile se pot atașa elementelor HTML prin intermediul unor fișiere externe sau în cadrul documentului, prin elementul <*style*> și/sau atributul *style*.

XML este un meta limbaj de marcare folosit pentru a crea limbaje de marcare precum: XHTML, RDF, SVG. Un limbaj de marcare este format dintr-un set de tag-uri plasate într-un text astfel încât să fie posibilă demarcarea anumitor componente ale documentului.

Putem spune că XML este doar informație cuprinsă în etichete, tag-uri. Datorită faptului că nu efectuează algoritmi sau operații computaționale, XML nu poate fi considerat un limbaj de programare. XML a fost conceput pentru a stoca și a transporta date.

JSON este un format de date minimal, ușor de citit, utilizat ca alternativă la XML pentru stocarea și transmiterea datelor între serverul web și aplicația web. JSON este text și putem converti orice JSON într-un obiect JavaScript și invers. JSON este realizat din două părți: chei și valori. JSON este o colecție de perechi cheie-valoare. Cheia este întotdeauna un șir, în timp ce valoarea poate fi un șir, un număr, un boolean sau chiar o matrice. [8]

De exemplu:

{"nume": "Maria", "vârstă":22, "oraș": "Londra"}

Dacă vom compara JSON cu XML, putem spune că în ceea ce privește asemănările, atât JSON cât și XML sunt citite de oameni,sunt ierarhice (adică avem valori în valori), pot fi analizate de majoritatea limbajelor de programare și pot fi extrase folosind solicitări HTTP.

De asemenea, putem observa unele diferențe, cum ar fi faptul că doza JSON nu folosește etichete finale, este mai scurtă, mai rapidă de citit și scris și poate folosi matrice. Cea mai mare diferență dintre XML și JSON este modul în care acestea sunt analizate. XML este analizat folosind un parser XML, în timp ce JSON este analizat cu o funcție JavaScript. JSON are avantajul că este mai ușor de analizat decât XML și că JSON este analizat într-un obiect JavaScript disponibil pentru utilizare.

3.1.3 Protocolul HTTP

Hypertext Transfer Protocol, abreviat HTTP, este un protocol la nivelul aplicaţie care stă la baza comunicaţiei în World Wide Web. Hypertext reprezintă un set de obiecte care construiesc conexiuni între noduri prin legături logice, hyperlinks. HTTP este protocolul care permite transferul unor astfel de obiecte.

HTTP este un protocol de tip cerere-răspuns în modelul arhitectural client-server. Un client, cel mai des o pagină web, trimite o cerere HTTP către un server web, care întoarce un răspuns. Acest răspuns conţine o stare al completării cererii şi, în caz de succes, informaţia cerută.

De obicei, comunicarea se realizează prin protocolul TCP/IP, dar putând fi folosite și alte tipuri de protocoale. Portul folosit implicit este 80, dar se pot folosi de asemenea și alte porturi.

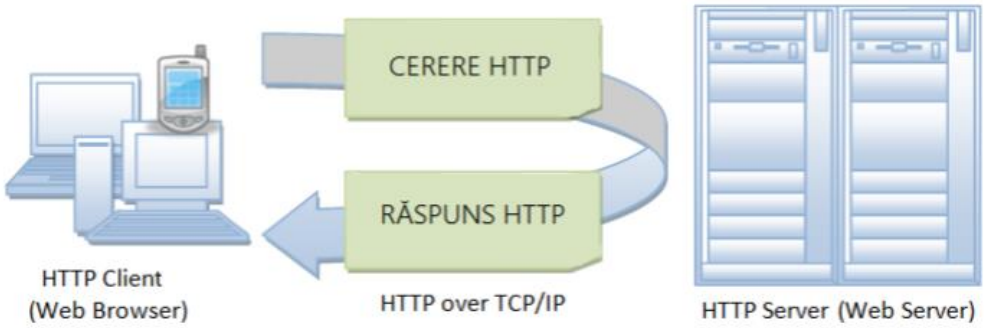


Figura 3.2: Comunicarea client-server[9]

Conexiunea client-server pentru protocolul HTTP este menținută doar pe parcursul cererii curente, după aceasta conexiunea fiind închisă. Clientul realizează o conexiune TCP cu serverul și trimite o comandă de cerere, urmând ca serverul să trimită înapoi răspunsul și să închidă conexiunea. Ultima versiune HTTP 1.1 nu mai prezintă problema supraîncărcării pe care versiunile anterioare le aveau, fișierele multiple putând fi descărcate prin aceeași conexiune.

Resursele HTTP sunt identificate şi localizate pe reţea prin Uniform Resource Locators (URLs) folosind schema http URI definită în RFC 3986 astfel:

<nume_schema> : <porţiunea_ierarhică> [? <query>] [# <fragment>]

În cazul HTTP numele schemei este http. Partea ierarhică începe cu ‚//', urmat de un nume al autorităţii, fiind un nume de domeniu sau un IP, urmat apoi de un port opţional, urmat de ‚:'. După autoritate urmează o cale construită ca o succesiune de segmente, asemănător unei structuri de directoare, caracterul ce separă structurile de directoare fiind ‚/'. Porţiunea de ‚query' este opţională, conţine informaţia adiţională care nu este ierarhică, ci organizată tipic prin perechi de tip *<cheie>=<valoare>*, cu perechile separate prin ‚;' sau ‚&'. Partea *<fragment>* este opţională, începe cu "#" şi conţine o directivă către o resursă secundară, cel mai des un atribut id al resursei principale, ca în cazul documentelor HTML.

O formă simplă și constantă a URL-ului este prezentată în urmatoarea figură (Figura 3.3). Protocolul este http, dar poate să fie https pentru comunicații sigure. Portul utilizat implicit este 80, dar poate fi ales și un altul. Calea către resursă este o cale locală a resursei de pe server.

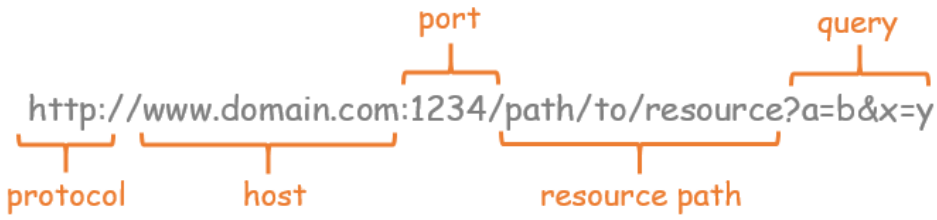


Figura 3.3: Structură URL [10]

HTTP defineşte nouă metode care indică acţiunea dorită asupra resursei accesate. Aceste nouă metode sunt: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT şi PATCH. Cele mai utilizate sunt: GET, prin care se citeşte resursa, şi cele de POST, PUT şi DELETE, prin care se pot modifica resursele. Metodele prin care se dorește doar citirea informației, fără modificarea stării serverului sunt considerate metode sigure. Acestea sunt HEAD, GET, OPTIONS şi TRACE.

O cerere HTTP conţine următoarele:

- O linie de request (cerere), spre exemplu GET /diverse/car.jpg HTTP/1.1 care formulează o cerere pentru resursa /diverse/car.jpg de la server.
- Antete HTTP, cum ar fi Accept-Language: en-US, Accept-Charset: utf-8, etc.
- O linie goală.
- Un corp al mesajului, optional.

Un răspuns HTTP conţine următoarele:

- Linie de status, spre exemplu HTTP/1.1 200 OK, care indică finalizarea cu succes a cererii clientului. Un status 3xx indică necesitatea unei redirectări, 4xx – eroare la client, 5xx- eroare la server.
- Antete HTTP, cum ar fi Content-Type: text/html.
- O linie goală.
- Un corp al mesajului, opțional.

Antetele HTTP sunt perechi "cheie: valoare", scrise în text clar şi separate prin succesiunea de caractere “carriage return” (CR) şi “line feed” (FD). Aceste antete definesc parametrii de operare a unei tranzacţii HTTP.

Cu ajutorul URL-ului un client poate realiza cereri către un server. Serverul răspunde cu anumite coduri de stare și mesaje utile. Protocolul HTTP are coduri specifice pentru diferite tipuri de răspuns. Aceste coduri sunt prezentate în continuare.

- 1xx descrie clasa codurilor de informare.
- 2xx descrie clasa codurilor pentru cereri ce se realizează cu succes. Codul 200 este cel mai frecvent cod folosit, acesta ne informează că cererea a fost procesată cu succes.
- 3xx descrie clasa codurilor pentru redirecționare. Se presupune că un client ia acțiuni suplimentare, una dintre cele mai comune este cea în care acesta să folosească un alt URL pentru a putea accesa o resursă.
- 4xx descrie clasa erorilor provenite de la client. Codurile acestea sunt folosite atunci când serverul consideră ca o eroare provine de la client, fie acesta dorește să aibă acces la o resursă inexistentă, sau realizează o cerere eronată. Codul 404, cel mai folosit, indică faptul că resursa nu este disponibilă.
- 5xx descrie clasa erorilor provenite de la server. Aceste coduri sunt folosite pentru a indica o eroare ce survine de la server în timpul procesării cererii. Codul 500 este cel mai utilizat, acesta informează că există o problemă internă produsă de server.

3.2 Servicii Web RESTful

Representational State Transfer (REST) reprezintă un stil arhitectural bazat pe standardele web şi pe protocolul HTTP, pentru sisteme distribuite ca World Wide Web (WWW). REST s-a evidenţiat în ultimii ani ca modelul de design predominant al web-ului, datorită simplităţii sale.

Roy T. Fielding a definit REST-ul ca fiind un set coordonat de constrângeri arhitecturale ce își dorește o minimizare a latenței și a comunicării din interiorul rețelei. Totodată, își dorește o maximizare a independenței și scalabilității implementării componentelor. REST se folosește de stabilirea și refolosirea interacțiunilor, componentele sunt înlocuite dinamic, iar acțiunile se realizează de către intermediari, rezultând că se ridică la nevoile sistemului numit Internet. [11]

Într-o arhitectură bazată pe REST, totul este o resursă. O resursă este accesată printr-o interfaţă comună bazată pe metodele standard HTTP. Într-o arhitectură REST, există tipic un server REST care asigură accesul la resurse şi clienţi REST care accesează sau modifică resursele. Fiecare resursă ar trebui să suporte operaţiile standard HTTP (GET, POST, PUT, DELETE). Resursele sunt identificate prin ID-uri globale – URI-uri.

"Limbajul" REST este bazat pe substantive şi verbe şi pune accentul pe lizibilitate. Mai mult, REST nu necesită parsare XML şi nu necesită un antet al mesajului de la/către un furnizor de servicii, ceea ce duce la reducerea lățimii de bandă consumată.

Motivația acestei arhitecturi constă în faptul ca serviciile RPC și cele bazate pe SOAP oferă o complexitate mărită. Simplitatea WEB-ului este reprezentată de ideea de bază a serviciilor web REST. REST aplică arhitectura web prin intermediul serviciilor web. Chiar dacă REST nu este un standard, acesta utilizează următoarele standarde:

- HTTP (HyperText Transfer Protocol)
- URI (Uniform Resource Identifier)
- XML

Cu alte cuvinte, REST reprezintă construirea unui serviciu web, utilizând HTTP, XML și URI exact așa cum a fost construit web-ul. Fiecare resursă are o adresă cunoscută sub numele de Uniform Resource Identifier (URI). Pentru a accesa o resursă, trebuie să utilizați o metodă HTTP. Un avantaj imens al acestei arhitecturi este faptul că prima linie a unei cereri HTTP conține toate informațiile necesare pentru a accesa o resursă ("GET / example / index HTTP / 1.1").

Arhitectura Client – server: o interfaţă uniformă separă clienţii de servere. Separarea preocupărilor înseamnă că, spre exemplu, clienţii nu se ocupă de stocarea datelor, aşa cum serverul nu se ocupă cu interfaţa utilizatorilor sau cu starea clienţilor. Astfel, serverele şi clienţii pot fi dezvoltaţi independent, interfaţa rămânând constantă. [11]

- Stateless – acest principiu asigură că niciun context al clientului nu este memorat pe server între cereri. Fiecare cerere efectuată conţine toate informaţiile necesare, şi orice informaţie de stare a sesiunii este stocată pe client. Aceasta este o caracteristică de rezistenţă la erori a serverelor.
- Cacheable – clienţii pot reţine în memoria cache răspunsurile. O memorie cache bine întreţinută poate îmbunătăţi scalabilitatea şi performanţa sistemului, prin reducerea numărul de cereri de la clienţi la server.
- Layered – un client nu-şi poate da seama prin interfaa comună dacă este conectat direct la server sau la un proxy intermediar. Serverele intermediare pot îmbunătăţi scalabilitatea sistemului prin load-balancing sau prin partajarea unei memorii cache.
- Code on demand (opţional) – serverele pot să extindă temporar funcţionalitatea clientului prin transferul codului - exemple fiind Java applets sau limbajele client-side scripting ca JavaScript.

Un Serviciu Web RESTful se bazează pe metodele HTTP şi pe conceptele arhitecturii REST. Acesta defineşte tipic un URI de bază pentru resurse şi tipurile MIME suportate (XML, Text, JSON, Protocol Buffers, etc) şi setul de operaţii HTTP. Metodele standard sunt [6]:

- GET – defineşte un acces la citirea unei resurse, fără efecte secundare. Resursa nu este niciodată alterată în urma unei cereri GET.
- PUT – crează o nouă resursă.
- DELETE – şterge o resursă, operaţia trebuie să fie idempotentă, o repetare a cererii nu trebuie să producă efecte suplimentare primei cereri.
- POST – actulizează resursa existentă sau crează o nouă resursă.

3.3 Framework-ul ANGULAR

Angular este o platformă, un *framework* pentru crearea de aplicații client în HTML și TypeScript. Angular este scris în TypeScript și implementează funcționalitatea de bază și opțional un set de biblioteci TypeScript ce sunt importate în aplicații.

Angular este un framework de dezvoltare web cu sursă deschisă ce are la bază limbajul TypeScript. Proiectul a fost dezvoltat de Echipa Angular de la Google si de comunitățile de utilizatori individuali și companii. AngularJS este frameworkul de la baza lui Angular. De fapt, Angular este varianta rescrisă complet a frameworkului AngularJS. [12]

Blocurile de bază ale unei aplicații Angular sunt NgModules, oferind un context de compilare pentru componente. NgModulele colectează codul asociat în seturi funcționale; o aplicație Angular este definită de un set de NgModule. O aplicație are întotdeauna cel puțin un modul rădăcină și mai multe module de caracteristici.

- Componentele definesc vizualizările. Acestea sunt un set de elemente ale ecranului pe care le poate alege și modifica Angular potrivit logicii programului și datelor.
- Componentele folosesc servicii. Acestea oferă funcționalități specifice care nu sunt în legătură directă cu vizualizările. Furnizorii de servicii pot fi injectați in componente, numindu-se dependențe. Aceste dependențe conduc la un cod modular și eficient.

Atât componentele, cât și serviciile sunt simple clase cu decorator ce marchează tipul acestora și oferă metadate care transmit către Angular cum să le folosească.

- Metadatele definesc o componentă a clasei asociate cu un șablon (template) care definește vizualizarea. Un șablon combină cod HTML cu directivele Angular și marcajele de legătură (binding markup) ce îi permit să modifice codul HTML înainte de a îl afișa.
- Metadatele pentru clasele de servicii sunt prevăzute cu informațiile de care are nevoie Angular pentru a face disponibile componentele prin dependență (dependency injection (DI)).

Componentele unei aplicații definesc de obicei multe vizualizări, aranjate ierarhic. Angular furnizează serviciul Router pentru a putea defini căile de navigare între vizualizări. Router-ul oferă capabilități sofisticate de navigare în browser.

Arhitectura aplicațiilor Angular

Angular este un framework conceput pentru a dezvolta aplicații de o singură pagină (single page applications – SPAs), iar cea mai mare parte a designului său de arhitectură este orientată pe eficiență.

Aplicațiile pe o singură pagină (SPA) sunt aplicații ce sunt accesate de browserele web la fel ca alte site-uri web, dar oferă interacțiuni dinamice mai asemănătoare aplicațiilor mobile și desktop. Cea mai notabilă diferență dintre un site web regulat și SPA este cantitatea redusă de reîmprospătare a paginii. Tipic, 95% din codul SPA rulează în browser, restul de cod rulează în server atunci când utilizatorul are nevoie de informații noi sau trebuie să execute operațiuni securizate, cum ar fi autentificarea. Ca rezultat, procesul de redare a paginilor se realizează mai ales pe partea clientului.

Modulele

Angular NgModules diferă și complementează modulele JavaScript. Un NgModule declară un context de compilare pentru un set de componente care este dedicat unui domeniu de aplicație, unui flux de lucru sau unui set de capabilități strâns legate. Un NgModule poate asocia componentele sale cu codul asociat, cum ar fi serviciile, pentru a forma unități funcționale.

Fiecare aplicație Angular are un modul rădăcină (root module), numit convențional AppModule, fiind prevăzut cu un mecanism de bootstrap ce lansează aplicația. O aplicație conține multe module funcționale. La fel ca modulele JavaScrip, NgModules pot importa funcționalități de la alte NgModules, permițându-le să-și exporte propria funcționalitate și să fie folosite de alte NgModules.

Modulele ajută la organizarea unei aplicații în blocuri de funcționalitate prin împachetarea componentelor, conductelor, directivelor și serviciilor. Aplicațiile Angular sunt modulare. Fiecare aplicație Angular are cel puțin un modul - modulul rădăcină(root-module), denumit convențional *AppModule*. Modulul rădăcină poate fi singurul modul dintr-o aplicație mică, dar majoritatea aplicațiilor au mai multe module.

Orice modul Angular este o clasă cu decoratorul @NgModule. Decoratorii sunt funcții care modifică clasele JavaScript. Sunt utilizate în principal pentru atașarea metadatelor la clase, astfel încât să cunoască configurația acestor clase și modul în care ar trebui să funcționeze. Proprietățile decoratorului @NgModule care descriu modulul sunt:

- Declarații – Clasele ce aparțin acestui modul și sunt conectate cu vizualizările. Există trei clase în Angular ce pot conține vizualizări(views): componente, directive și conducte(pipes).
- Exporturi (Exports) – Clasele care ar trebui să fie accesibile altor componente ale modulelor.
- Importuri (Imports) – Modulele claselor ce sunt necesare componentelor acestui modul.
- Furnizori (Providers) – Servicii prezente în unul dintre modulele care urmează să fie utilizate în celelalte module sau componente. Odată ce un serviciu este inclus în furnizori, acesta devine accesibil în toate părțile acelei aplicații.
- Bootstrap – Componenta rădăcină care este vizualizarea principală a aplicației. Numai modulul rădăcină are această proprietate și indică componenta care va fi bootstrapată.
- inputComponents (entryComponents) – O componentă de intrare este orice componentă care se încarcă imperativ (ceea ce înseamnă că nu se reface în șablon), după tip.

Componentele

Fiecare aplicație Angular are cel puțin o componentă, componenta rădăcină (root component) ce conectează o ierarhie a componentelor cu modelul de obiect document de pagină (document object model DOM). Fiecare componentă definește o clasă ce conține date aplicative și logică. Este asociată cu un șablon HTML care definește o vizualizare ce trebuie afișată într-un mediu țintă.

@Component () identifică clasa imediat inferioară acesteia ca o componentă și furnizează șablonul și metadatele specifice componentelor aferente.

Componentele reprezintă cel mai elementar element de construcție al aplicațiilor UI și Angulare. O componentă controlează una sau mai multe secțiuni de pe ecran. Aceasta este autonomă și reprezintă o bucată reutilizabilă de cod, este de obicei constituită din trei părți importante:

- O parte de cod html cunoscut sub numele de vizualizare.
- O clasă care încapsulează toate datele și interacțiunile disponibile pentru această vizualizare printr-un API de proprietăți și metode, construite de Angular.
- Elementul html cunoscut și ca selector.

Folosind decoratorul Angular *@Component* oferim metadate suplimentare care determină modul în care componenta trebuie procesată, instanțiată și utilizată în timpul rulării. De exemplu, setăm șablonul HTML asociat vizualizării, apoi setăm selectorul html pe care îl vom folosi pentru acea componentă și setăm foile de stil(stylesheets) pentru componenta respectivă.

O componentă trebuie să aparțină unui *NgModule* pentru ca acesta să poată fi utilizabilă de o altă componentă sau aplicație. Pentru a specifica faptul că o componentă este membră a unui *NgModule*, ar trebui menționată în proprietatea de declarații a acelui *NgModule*.

Șabloane, directive și legare de date (Templates, directives, and data binding)

Un șablon combină cod HTML cu marcaje Angular ce pot modifica elementele HTML înainte de a le afișa. Șabloanele directive realizează logica programului, iar marcajele de legătură conectează datele aplicației și DOM. Există două tipuri de date de legătură:

- Legarea evenimentului permite aplicației să răspundă la intrarea utilizatorului în mediul țintă prin actualizarea datelor din aplicație.
- Legarea proprietății permite interpolarea valorile calculate de la datele aplicației în codul HTML.

Înainte ca o vizualizare să fie afișată, Angular evaluează directivele și rezolvă sintaxele de legătură în șabloanele pentru a modifica elementele HTML și DOM, potrivit programului de date și logicii.

Servicii și injecție de dependență

Pentru datele sau logica care nu sunt asociate cu o anumită vizualizare și care trebuie partajate între ele, există o clasă de servicii. Clasa de servicii este imediat precedată de decoratorul @Injectable (). Decoratorul furnizează metadatele care permit altor furnizori să fie injectați ca dependențe în clasa ta.

Dependența de injectare (DI) permite păstrarea clasele componentelor slabe și eficiente. Nu prelucrează date de la server, nu validează intrarea utilizatorului și nu se înregistrează direct în consolă; ele deleagă astfel de sarcini serviciilor.

Rutare

Routerul Angular oferă un serviciu care permite definirea unei căi de navigare între diferitele stări ale aplicației și vizualizarea ierarhiile din aplicație. Este modelat pe convențiile de navigare a browserului:

- Se introduce o adresă URL în bara de adrese, iar utilizatorul navighează către o pagină corespunzătoare.
- Se face click pe link-ul din pagină și browser-ul navighează către o pagină nouă.
- Se face click pe butoanele din spate și înainte ale browserului, iar utilizatorul navighează înapoi și înainte în istoricul paginilor ce pot fi văzute.

Următoarea diagramă (Figura 3.4) arată modul în care sunt legate aceste bucăți de bază.

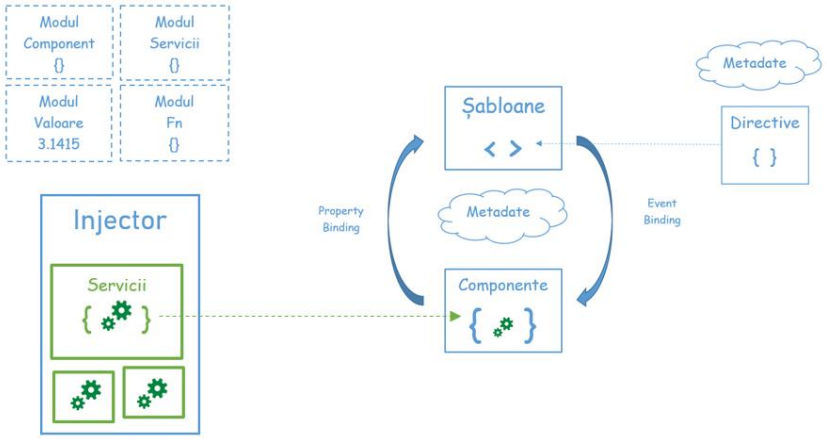


Figura 3.4: Diagrama de interconectare a componentelor de baza într-o aplicație Angular

3.3.1 TypeScript

TypeScript este un limbaj de programare dezvoltat si actualizat de Microsoft. Este un superset sintactic al limbajului JavaScript. Acest limbaj de programare este proiectat pentru dezvoltarea aplicațiilor de mari dimensiuni. Pentru ca TypeScript este un superset peste JavaScript, programele JavaScript deja existente sunt programe valide TypeScript. Poate fi utilizat pentru dezvoltarea aplicațiilor de tip JavaScript pentru partea de client, dar și pentru partea de server.

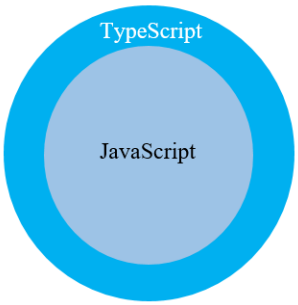


Figura 3.5: Relația de legătură dintre limbajul de program are JavaScript și TypeScript

Caracteristicile TypeScript

Pe măsură ce tehnologiile hardware au avansat, s-a dezvoltat și nevoia de aplicații software mai avansate pentru a funcționa pe diferite tipuri de dispozitive. JavaScript este soluția firească pentru un model de tipul „platformă încrucișată” (în engleză: cross-platform), deoarece aproape toate browserele moderne sunt capabile să o ruleze. Crearea de aplicații mari în JavaScript și structurarea codului într-o formă care poate fi întreținută poate fi dificilă folosind JavaScript. TypeScript își propune să conducă dezvoltarea JavaScript la nivelul următor. [13]

JavaScript a fost original creat ca un limbaj de scripting, ceea ce înseamnă că a fost proiectat ca un limbaj ce poate interacționa și comunica cu alte limbaje de programare. Pe măsură ce nevoia de aplicații software mai mari a

crescut, a devenit necesar și ca aceste aplicații să conțină mai multe componente și biblioteci reutilizabile. JavaScript este un limbaj dinamic ce poate să acționeze atât ca un limbaj functional, cât și ca un limbaj orientat pe obiect pentru anumite aplicații. Codul obiect-orientat se focusează pe conceptul de reutilizare a codului.

TypeJava oferă o gamă de tipuri de obiecte și accesibilitate pe nivele asemănătoarea cu programarea obiect-orientată. Moștenirea, încapsularea și abstractizarea sunt simplificate în TypeScript. JavaScript este axat pe valoarea funcțiilor și a moștenirii bazate pe prototip, ceea ce înseamnă că și TypeScript respectă aceste reguli. Prin utilizarea compilatorului său, TypeScript introduce o serie de concept importante din diferite limbaje orientate pe obiect. Acestea sunt: încrucișarea/scriere statică (în engleză: Static typing), clase, interfețe și module.

TypeScript adaugă un strat de scriere statică peste JavaScript, care este apoi rulat cu un compilator. Compilatorul analizează codul TypeScript și îl convertește în JavaScript simplu. Introducerea claselor și a modulelor permite dezvoltarea aplicațiilor de scară înaltă mai ușor. Introducerea interfețelor și a generalizării în sistemul de scriere ne permite să creem cu ușurință componente și biblioteci ce pot fi folosite pentru o varietate de obiecte. [13]

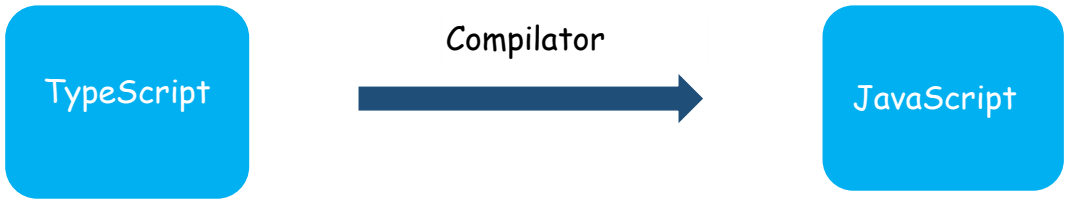


Figura 3.6: Relație de legătură: TypeScript – JavaScript

TypeScript introduce ideea de membrii publici și privați. Dacă un obiect sau o funcție încearcă să acceseze un membru privat al altui obiect, atunci compilatorul recunoaște codul ca invalid și erori de construcție vor fi generate. Astfel, se implementează încapsularea, ceea ce înseamnă că previne utilizatorii de obiecte TypeScript să acceseze metode și proprietăți care ar putea fi dăunatoare atunci când sunt manipulate în afara sferei obiectului propriu-zis. Furnizarea de nivele de accesibilitate ajută să limităm domeniul interacțiunilor posibile cu un obiect sau chiar să-l ascundem complet de accesul altor componente sau biblioteci.

Configurarea IDE

Pentru a putea scrie cod în TypeScript, trebuie instalat compilatorul. Compilatorul este disponibil ca un pachet de sine stătător prin intermediul Node.js, un intrument Visual Studio pe care Microsoft îl furnizează, sau prin descărcare directă a codului sursă al compilatorului. Compilatorul TypeScript este dezvoltat în TypeScript, ceea ce înseamnă că va rula în orice gazdă JavaScript. Cu instumentele Microsoft instalate, se poate accesa compilatorul TypeScript direct din linia de comandă și genera cod JavaScript imediat.

3.4 Python

Guido van Rossum, creatorul limbajului de programare Python, a denumit limbajul după emisiunea BBC ”Monty Python’s Circus Flying”. Python este un limbaj de programare ușor de învățat. Are structuri de date eficiente la nivel înalt și o abordare simplă, dar eficientă, a programării obiect orientată. Sintaxa elegantă și tastarea dinamică, împreună cu natura interpretată, fac din Python un limbaj ideal pentru scrierea și dezvoltarea rapidă a aplicațiilor în multe zone de pe majoritatea platformelor. [14]

Caracteristici ale Python:

- Simplitate – Python este un limbaj de programare simplu și minimalist. Această natură pseudo-cod a limbajului Python este una dintre cele mai interesante caracteristici ale sale. Permite concentrarea mai degrabă pe soluția problemei, decât pe limbaj în sine.
- Ușor de învățat – Python are o sintaxă extrem de simplă.
- Sursă gratuită și deschisă – Python este un exemplu de FLOSS (Free / Libré și Open Source Software). În mod simplu, se pot distribui gratuit variante ale acestui software, se poate citi codul sursă, se pot efectua modificări și folosi bucăți din acesta în noile programe gratuite. FLOSS se bazează pe conceptul unei comunități care împărtășește cunoștințele. [14]
- Limbaj de nivel înalt – Scrierea de programe în Python nu este niciodată întreruptă cu detalii de nivel scăzut, cum ar fi gestionarea memoriei utilizate de program.
- Portabilitate – Datorită naturii sale, Python a fost modificat pentru a funcționa pe multe platforme. Toate programele Python pot funcționa pe oricare dintre aceste platforme fără a necesita modificări.
- Interpretat – Python nu are nevoie de compilare la binar. Programul este rulat direct din codul sursă. Pe plan intern, Python convertește codul sursă într-o formă intermediară numită bytecodes și apoi o traduce în limba maternă a calculatorului, urmând executarea.
- Obiect orientat – Python suportă programarea orientată spre procedură, precum și programarea orientată pe obiecte. În limbajele orientate spre proceduri, programul este construit în jurul unor proceduri sau funcții care nu sunt decât piese de program reutilizabile. În limbajele orientate pe obiecte, programul este construit în jurul obiectelor care combină date și funcționalități. Python are o modalitate foarte puternică, dar simplistă de a realiza programarea orientate pe obiect, mai ales atunci când este comparată cu limbaje importante de programare.
- Biblioteci extinse - Librăria standard Python este de proporții crescute. Acesta ajută să se realizeze diverse lucruri care implică expresii regulate, generarea de documente, testarea unităților, filetarea, baze de date, browsere web, email, HTML, criptografie, și alte proceduri dependente de sistem.

Mediul de dezvoltare PyCharm

Pycharm este un mediu de dezvoltare integrat (integrated development environment – IDE), IDE Python cu un set complet de instrumente pentru a se putea realiza programarea în limbajul de dezvolare Python. Este dezvoltat de către compania cehă JetBrains. Oferă analiză de cod, un debugger grafic, un tester integrat de unitate, integrarea cu sisteme de control al versiunilor VCSes și susține dezvoltarea web cu Django, precum și cu Data Science cu Anaconda. [15]

PyCharm este cross-platform, cu versiuni de Windows, MacOs, dar și Linux. Ediția de comunitate (Comunity Edition) este lansată sub licența celor de la Apache, iar ediția profesională ce conține caracteristici suplimentare este lansată sub o licență de proprietate.

Caracteristici principale:

- Navigare codului de proiect – Se poate naviga instantaneu de la un fișier la altul, de la o metodă de declarare la utilizare, și prin ierahia claselor. Există, de asemenea, comenzi rapide de la tastatură pentru o navigare mai rapidă.
- Analiza codului – beneficiul codului on-the-fly, evidențierea erorilor și inspecțiile inteligente.
- Redenumirea Python – Se pot realiza modificări de cod la nivel de proiect fără problema redenumirii.
- Dezvoltare Web cu Django – O dezvoltare Web mai rapidă cu frameworkul Django sprijinit de editoare HTML, CSS și JavaScript.

- Suport pentru Google App Engine – Dezvoltarea de aplicații Google App Engie.
- Integrarea controlului versiunii – check-in, check out, diffe-uri de vizualizare, îmbinare – toate sunt prezente în interfața utilizatorului unificată VCS pentru Mercurial, Seversion, Git, Perforce și altele.
- Depanator grafic – corectarea aplicațiilor Python și testarea unităților utilizând un depanator cu funcții complete cu puncte de întrerupere, trepte, vizualizare cadre, ceasuri și expresii de evaluare.
- Testarea unităților integrate – Rularea unui fișier de testare, o singură clasă de test, o metodă sau toate testele într-un dosar.

4

SISTEME DE GESTIUNE AL BAZELOR DE DATE

Sistemele de gestiune a bazelor de date (în engleză: "database management system"), abreviat SGBD, sunt reprezentate de totalitatea programelor utilizate pentru crearea, întreținerea și interogarea unei baze de date. Există două categorii de module: module comune cu cele ale sistemelor de operare ale calculatoarelor și module cu funcții specifice bazelor de date.

4.1 Baze de date

Baza de date este o colecție de date organizată, în general stocată si accesată electronic dintr-un sistem computerizat/ informatic. Când bazele de date sunt mult mai complexe, acestea se dezvoltă folosind tehnici de proiectare și modelare formale. Sistemul de management al bazelor de date (SGBD) este software-ul ce interacționează cu clienții, aplicațiile și baza de date în sine pentru a captarea și analiza datelor. Software-ul SGBD cuprinde facilitățile de bază furnizate pentru administrarea bazelor de date.

Informațiile sunt organizate în rânduri, coloane și tabele, fiind indexate pentru a facilita găsirea informațiilor relevante. Informația este actualizată, extinsă și ștearsă pe măsură ce se adaugă o nouă informație. Bazele de date procesează încărcările de lucru pentru a le crea și actualiza, interogând datele pe care le conțin și execută aplicații împotriva lor.

De obicei, un manager de baze de date oferă utilizatorilor posibilitatea de a controla accesul la citire / scriere, specifică generarea de rapoarte și analizează utilizarea. Unele baze de date oferă compatibilitatea ACID (atomicitate, consistență, izolare și durabilitate) pentru a garanta că datele sunt coerente și că tranzacțiile sunt complete.

Evoluția bazelor de date

Bazele de date au evoluat încă de la începutul anilor '60, începând cu baze de date ierarhice și de rețea, prin anii 1980 cu baze de date orientate pe obiecte și astăzi cu baze de date SQL, NoSQL și baze de date cloud.

Într-o anumită perspectivă, bazele de date pot fi clasificate în funcție de tipul de conținut: bibliografic, text integral, numeric și imagini. În calcul, bazele de date sunt uneori clasificate în funcție de abordarea lor organizațională. Există numeroase tipuri de baze de date, variind de la cea mai răspândită abordare, baza de date relațională, la o bază de date distribuită, o bază de date cloud sau o bază de date NoSQL.

Baza de date relațională

O bază de date relațională, inventată de E.F. Codd la IBM în 1970, este o bază de date tabulară în care datele sunt definite astfel încât să poată fi reorganizate și accesate în mai multe moduri diferite.

Bazele de date relaționale sunt alcătuite dintr-un set de tabele cu date care se încadrează într-o categorie predefinită. Fiecare tabel are cel puțin o categorie de date într-o coloană și fiecare rând are un anumit exemplu de date pentru categoriile definite în coloane.

Limbajul de interogare structurat (SQL) este interfața standard a utilizatorului și a aplicației pentru o bază de date relațională. Bazele de date relaționale sunt ușor de extins și o nouă categorie de date poate fi adăugată după crearea originală a bazei de date fără a fi necesară modificarea tuturor aplicațiilor existente.

Baza de date distribuită

O bază de date distribuită este o bază de date în care porțiunile bazei de date sunt stocate în mai multe locații fizice și în care prelucrarea este dispersată sau reprodusă între diferite puncte dintr-o rețea.

Bazele de date distribuite pot fi omogene sau eterogene. Toate locațiile fizice dintr-un sistem de baze de date distribuite omogene au același hardware de bază și rulează aceleași sisteme de operare și aplicații de bază de date. Aplicațiile hardware, de sisteme de operare sau de baze de date într-o bază de date eterogenă distribuită pot fi diferite în fiecare locație.

Bază de date NoSQL

Bazele de date NoSQL sunt utile pentru seturile mari de date distribuite. Acestea sunt eficiente pentru problemele de performanță ridicată a datelor pe care bazele de date relaționale nu sunt construite să le rezolve. Ele sunt cele mai eficiente atunci când o organizație trebuie să analizeze bucăți mari de date nestructurate sau date stocate pe mai multe servere virtuale din Cloud.

Date de bază orientate pe obiect

Elementele create folosind limbaje de programare orientate pe obiect sunt adesea stocate în baze de date relaționale, bazele de date orientate pe obiecte sunt potrivite pentru aceste elemente.

O bază de date orientată pe obiecte este organizată mai degrabă în jurul obiectelor decât în jurul acțiunilor, și mai degrabă decât în logică. De exemplu, o înregistrare multimedia într-o bază de date relațională poate fi un obiect de date definibil, spre deosebire de o valoare alfanumerică.

4.2 Sistemul de Management al Bazelor de Date

Sistemul de management al bazelor de date (SGBD) este reprezentat de un sistem software pentru crearea și managerierea bazelor de date. Sistemul de management al bazelor de date oferă utilizatorilor și programatorilor o cale sistematică de a crea, prelua, actualiza și gestiona datele. Un SGBD le oferă posibilitatea utilizatorilor să creeze, citească, actualizeze și șteargă informații dintr-o bază de date. În esență, un SGBD, servește ca o interfață între baza de date și utilizatori sau programele de aplicații, asigurând că datele sunt organizate în mod constant și rămân ușor accesibile.

Un SGBD manageriază trei asecte importante: informația, motorul bazei de date ce permite accesarea, securitatea și modificarea acesteia, schema bazei de date ce definește strctura logică. Aceste elemente funcționale oferă concurență, securitate, integritatea informației și uniformitatea procedurilor administrative. Operațiile tipice de administrare a bazelor de date suportate de SGBD includ managementul schimbării, monitorizarea / reglarea performanțelor, actualizarea și recuperarea. Multe sisteme de gestionare a bazelor de date sunt, de asemenea, responsabile de returnări automate, reporniri și recuperări, precum și de înregistrarea și auditul activității.

Sistemul de gestionare al bazelor de date poate oferi atât independență logică, cât și fizică. Asta înseamnă că poate proteja utilizatorii și aplicațiile de necesitatea de a ști unde sunt stocate datele sau de a fi preocupați de modificările

fizice a datelor (stocare și hardware). Atât timp cât programele utilizează interfața de programare a aplicațiilor (API) pentru baza de date furnizată de SGBD, dezvoltatorii nu vor trebui să modifice programele doar pentru că au fost aduse schimbări în baza de date.

În ceea ce privește SGBD-urile relaționale, interfața de programare a aplicațiilor este SQL, un limbaj standard de programare pentru definirea, protecția și accesarea informațiilor din bazele de date.

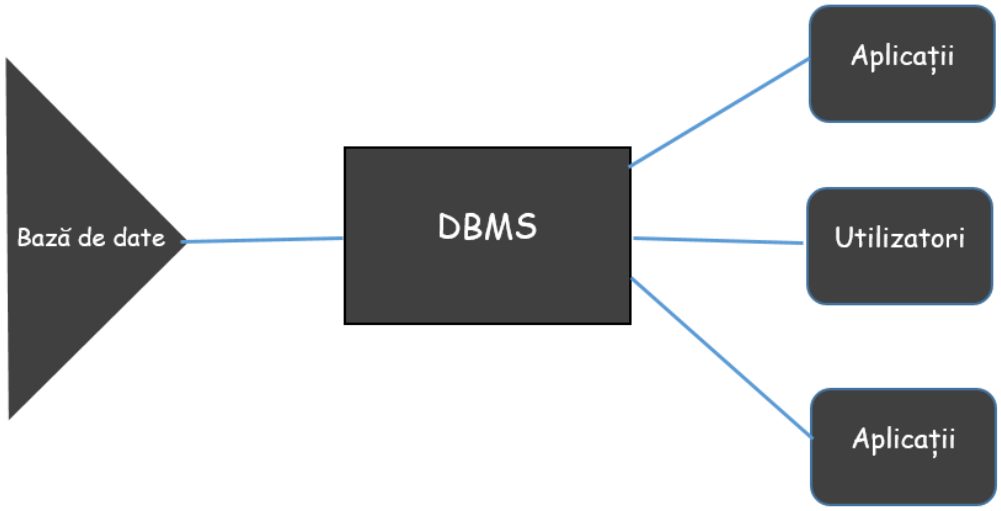


Figura 4.1: Structura unui sistem de management al bazelor de date

Avantajele unui SGBD

Utilizarea unui SGBD pentru stocarea si gestionarea informațiilor are multe avantaje. Unul dintre cele mai mare avantaje a folosirii unui SGBD constă în faptul că le permite utilizatorilor și programelor aplicațiilor să acceseze și să utilizeze aceleași date în timp ce gestionează integritatea datelor. Datele sunt mai bine protejate și menținute atunci cand pot fi partajate urilizând un SGBD în locul creării unor noi iterații ale acelorași date stocate în fișiere noi pentru fiecare aplicație nouă. SGBD oferă un depozit central de date care poate fi accesat de mai mulți utilizatori în mod controlat.

Stocarea centrală și gestionarea datelor în cadrul SGBD oferă:

- Abstractizarea și independența datelor
- Protejarea datelor
- Un mecanism de blocare pentru accesul concomitent
- Manipulare eficientă pentru a echilibra nevoile mai multor aplicații utilizând aceleași date
- Abilitatea de a se recupera rapid de la accidente și erori, inclusiv restabilirea și recuperarea
- Capacități robuste de integritate a datelor
- Înregistrarea și auditul activității
- Acces simplu prin utilizarea unei interfețe standard de programare a aplicațiilor (API)
- Proceduri uniforme de administrare a datelor

4.3 SQL Database

Limbaj de interogare structurată (în engleză: Structured Query Language), abreviat SQL, este un limbaj de programare standard folosit pentru comunicarea și manipularea bazelor de date. Este folosit pentru manipularea structurilor de date unde există relaționări între diferitele entități/ variabile ale informațiilor. SQL prezintă două avantaje față de sistemele anterioare de manipulare a bazelor de date. Prima, introduce conceptul de a accesa mai multe date cu o singură comandă. A doua, elimină nevoia de a specifica cum se poate citi o înregistrare, de exemplu: cu sau fără index. [16]

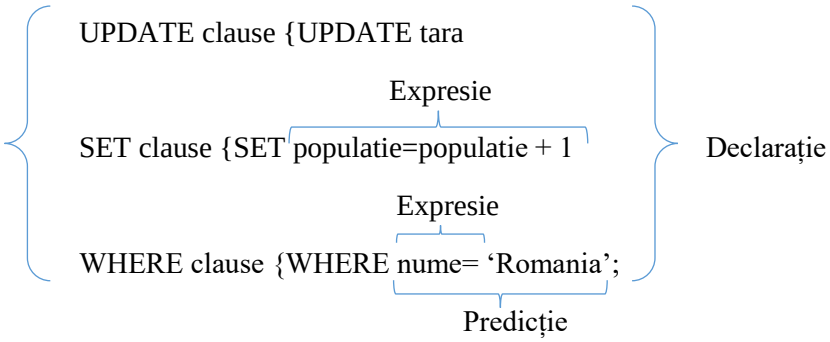
Inițial bazat pe algebra relațională și pe calculul relațional al tuplurilor, SQL constă din mai multe tipuri de instrucțiuni care pot fi clasificate informal ca sub-limbaje: un limbaj de interogare a datelor (DQL), un limbaj de definire a datelor (DDL) sau un limbaj de control al datelor (DML). Domeniul de aplicare al SQL include interogarea datelor, manipularea datelor (inserarea, actualizarea și ștergerea), definirea datelor (crearea schemelor și modificare) și controlul accesului la date. Deși SQL este deseori descris ca un limbaj declarativ, el include și elemente procedurale.

Sintaxă

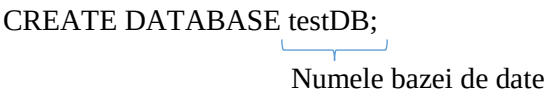
Limbajul SQL este subdivizat în mai multe elemente, printre care:

- Clauzele (în engleză: Clauses) – reprezintă componente ale declarațiilor și interogărilor.
- Expresii (în engleză: Expressions) – pot produce valori scalare sau tabele ce conțin informații dispuse în coloane si rânduri.
- Predicate (în engleză: Predicates) – specifică condițiile ce pot fi evaluate în logica SQL cu trei valori (adevărate/false/necunoscute), sunt utilizate pentru a limita efectele declarațiilor și a interogărilor sau pentru a schimba fluxul de programe.
- Interogări (în engleză: Queries) – recuperează datele pe baza unor criterii specifice.
- Declarații (în engleză: Statements) – pot avea un efect persistent asupra schemelor și datelor sau pot controla tranzacțiile, fluxul de programe, conexiunile sau diagnoza.
- Spațiile albe – sunt ignorate în declarațiile și interogările SQL.

În continuare este prezentată o structură ce introduce câteva din elementele ce compun o declarație:



- Sintaxă pentru a realiza o bază de date SQL:



- Sintaxă pentru ștergerea unei baze de date SQL:

```
DROP DATABASE testDB;
```

- Sintaxa pentru a crea o tablă într-o bază de date existentă:

```
CREATE TABLE nume_tabelă (
   coloană1 tip_date,
   coloană2 tip_date,
   coloană3 tip_date,
);
```

Parametrii *coloană* specifică numele coloanelor din tabelă. Parametrul *tip_date* specifică tipul de date ce se regăsesc în acea coloană (de exemplu: întreg, date, varchar).

- Sintaxă pentru a modifica elemntele unei tabele:

```
ALTER TABLE testDB
ADD nume_coloană tip_date;   ⟶ pentru a adăuga  o nouă coloană
ADD email varchar(255);      ⟶ pentru a adăuga coloana “email”
DROP COLUMN nume_tabelă;     ⟶ pentru a sterge o coloană
MODIFY COLUMN nume_coloană tip_date; ⟶ pentru a modifica
                                       tipul de date al unei coloane
…
);
```

4.4 SQLite

SQLite este o bibliotecă software care oferă un sistem de gestionare a bazelor de date relaționale. Lite-ul în SQLite înseamnă ușurință în ceea ce privește configurarea, administrarea bazei de date și resursele necesare.SQLite are următoarele caracteristici vizibile: autonomie, fără necesitatea unui server, fără configurație, tranzacțională.

În mod normal, un sistem de gestionare al bazelor de date relaționale, cum ar fi MySQL, PostgreSQL, etc., necesită un server separat pentru a funcționa. Aplicațiile care doresc să acceseze serverul bazei de date folosesc protocolul TCP / IP pentru a trimite și a primi solicitări. Așa este definită arhitectura client / server.

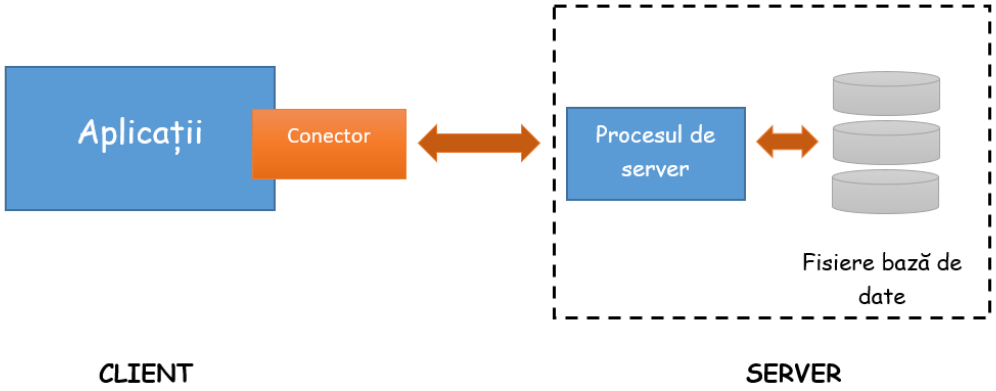


Figura 4.2: Diagrama arhitecturii client/server a unui sistem de gestionare a bazelor de date relaționale[17]

- **Fără server**

Spre deosebire, SQLite nu necesită un server pentru a funcționa. Baza de date SQLite are integrată aplicația care accesează baza de date. Aplicațiile interacționează cu baza de date SQLite, citesc și scriu direct din/în fișierele bazei de date stocate pe disc.

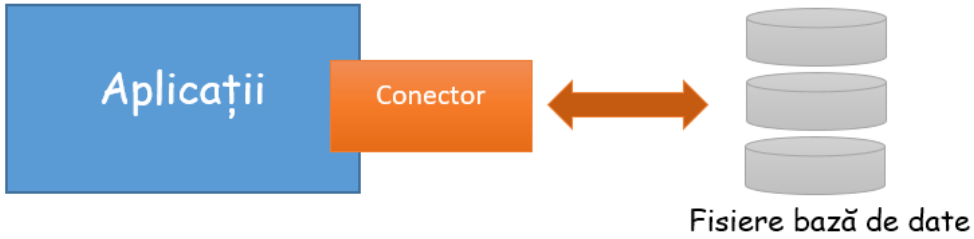


Figura 4.3: Arhitectura SQLite fără server[17]

- **Independență**

SQLite este autonomă, ceea ce înseamnă că necesită un suport minim din partea sistemului de operare sau a bibliotecii externe. Acest lucru face ca SQLite să fie utilizabil în orice mediu, în special în dispozitivele încorporate cum ar fi iPhone, telefoane cu sistemul Android, console de jocuri, playere portabile, sau altele.

SQLite este dezvoltat folosind ANSI-C. ANSI-C se referă la standardele succesive pentru limbajul de programare C publicat de Institutul Național pentru Standarde (“American National Standards Institute” – ANSI) și Organizația Internațională pentru Standardizare (“International Organization for Standardization” – ISO).

- **Configurare-zero**

Datorită arhitecturii fără server, nu este nevoie o pre-instalare a SQLite înainte de a fi utilizat. Nu există niciun proces de server care trebuie să fie configurat, pornit și oprit. În plus, SQLite nu utilizează niciun fișier de configurare.

- **Caracteristici distinctive SQLite**

SQLite utilizează tipuri dinamice pentru tabele. Aceasta înseamnă că se poate stoca orice valoare în orice coloană, indiferent de tipul de date. SQLite permite unei singure conexiuni de baze de date să acceseze simultan mai multe fișiere de baze de date. Aceasta aduce numeroase caracteristici cum ar fi îmbinarea tabelelor în diferite baze de date sau copierea datelor între baze de date într-o singură comandă. SQLite este capabil să creeze baze de date în memorie cu care să lucreze foarte rapid.

5

DEZVOLTAREA PLATFORMEI SOFTWARE INTELIGENTE

5.1 Introducere

Înțelegerea inteligenței umane a reprezentat un interes major în rândul psihologilor de-a lungul secolelor. Inteligența constă, în general, în capacitatea de a învăța prin dobândirea și aplicarea cunoștințelor. Până în prezent, scorurile coeficientului de inteligență (IQ) au devenit un instrument important în selectarea și plasarea persoanelor și grupurilor în arena psihoeducațională. Pentru copiii cu tulburare de specru autist s-a constatat de multă vreme că până la 70% dintre persoanele afectate de această afecțiune au o dizabilitate intelectuală definită printr-un IQ mai mic de 70. Cu toate acestea, deficiențele în inteligență și comportament nu trebuie văzute ca un obstacol pentru ca acești copii să continue viața într-o manieră normală. Pentru a fi diagnosticat cu autism, un copil trebuie să îndeplinească toate cele trei criterii și anume: afectarea calitativă a interacțiunii sociale, comunicarea și restricționarea tiparelor stereotipice de comportament. [18]

Platforma inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist este o aplicație ce oferă posibilitatea utilizatorului, în cazul de față specialistului, de a interacționa cu pacientul în cadrul unei ședințe de terapie prin intermediul robotului umanoid NAO. Această aplicație vine în sprijinul terapeuților, oferindu-le o altfel de metodă de a interacționa și, în esență, de a ajuta copii diagnosticați cu această tulburare.

Platforma realizată pe parcursul acestei teze de diplomă dorește să interfațeze utilizatorului 9 programe terapeutice, gândite de specialiști în acest domeniu, avându-l ca protagonist pe robotul NAO. Programele diferă în funcție de dificultate. Programul 1 conține 6 serii de cuvinte pe care robotul NAO le poate reproduce într-o ordine aleatoare. Programul 2 conține un set de cuvinte aleatoare pe care pacientul trebuie să le repete la indicația robotului NAO, cuvinte precum: “Fata”,”George”, “Mâine”, “Ana” ,”Florin” și altele. În programul 3 sunt prezentate un set de mișcări pe care robotul le execută într-o ordine aleatoare, mișcări precum: mâinile sus, mâinile pe cap, mâinile pe burtă, bate din palme, cucu-bau, bate palma, bea, manâncă cu lingura, tropăie, sari. Programele de la 4 la 8 sunt asemănătoare cu cel descris anterior, singura diferență este reprezentată de tipul indicațiilor ce sunt transmise pacientului. La finalul fiecărei execuții pe care pacientul o realizează, specialistul are posibilitatea de a-i acorda un feedback pacientului.

Programul 9 este diferit de cele prezentate anterior, acesta constă într-o poveste pe care robotul NAO o reproduce. După derularea poveștii sunt puse la dispoziție întrebări bazate pe aceasta. La finalul fiecărei întrebări există, de asemenea, posibilitatea ca specialistul să îi acorde un feedback pacientului. Toate informațiile legate de răspunsurile pacientului sunt înregistrate într-o bază de date pentru o evidență cât mai bună a evoluției pacientului.

În cele ce urmează, va fi descris procesul de realizare practică al proiectului ce constituie obiectul acestei teze. Pentru o descriere facilă, se pornește de la diagrama atașată (Figura 5.1) ce ilustrează cele detaliate anterior referitor la structura celor 9 programe ce sunt incluse în aplicația dezvoltată.

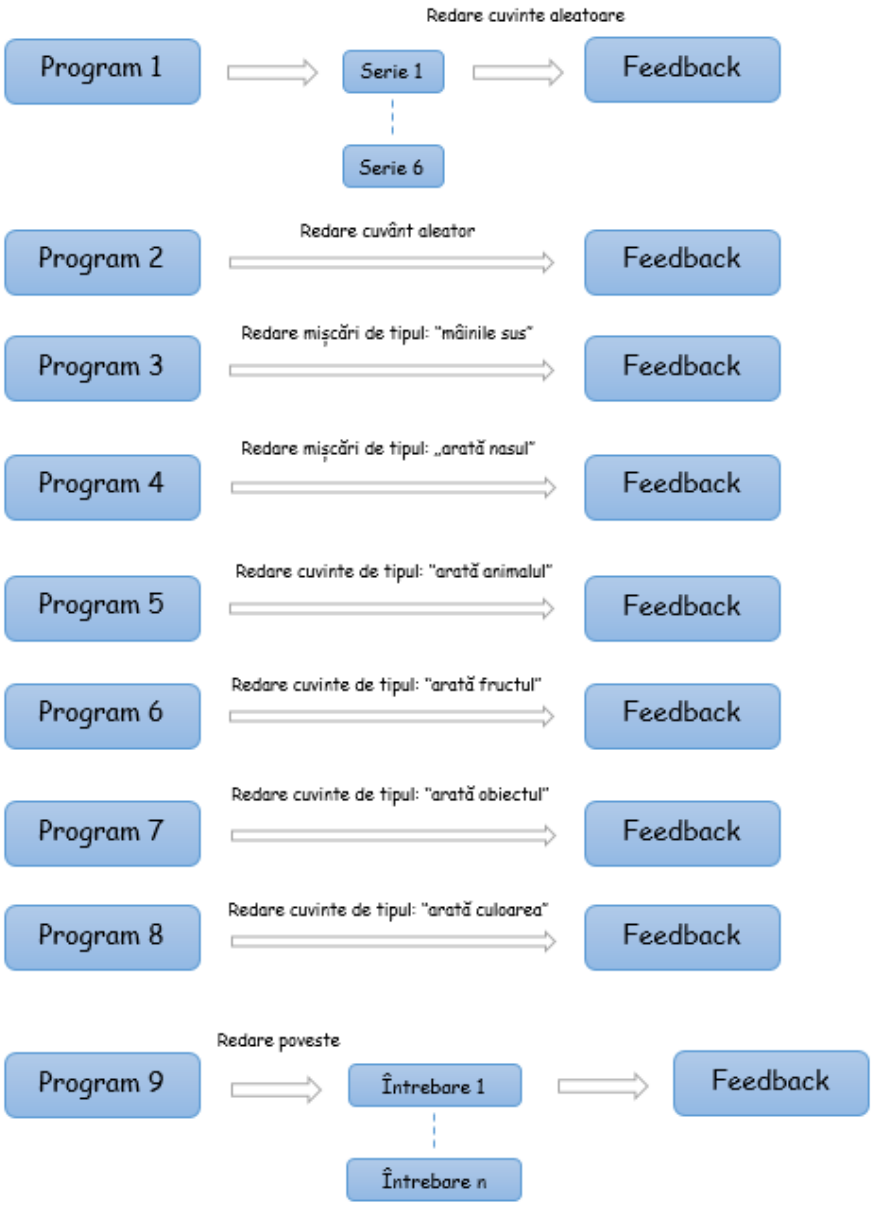


Figura 5.1: Diagramă structură aplicație – Nao Terapia

5.2 Etape de realizare a proiectului

În vederea elaborării soluției propuse pentru dezvoltarea platformei pentru sisteme embedded am decis ca arhitectura acestuia să fie compusă dintr-un ansamblu de module: un server HTTP, o aplicație client și o bază de date. Comunicația dintre aceste componente este realizată prin intermediul mesajelor de tipul “cerere/răspuns” specifice protocolului HTTP (Hyper Text Transfer Protocol) ilustrate în Figura 5.2. Serverul HTTP este implementat prin intermediul limbajului de programare Python.

Partea de client presupune dezvoltarea unei interfețe Web prin intermediul căruia specialistul interacționează cu robotul NAO. Scopul acestui modul este de a permite specialistului/teraputului accesul la o serie de comenzi ce pot fi transmise robotului umanoid NAO. Partea de server presupune implementarea unui serviciu REST cu rolul de a interfața utilizatorului diferite proceduri parametrizabile de care robotul NAO dispune. Comenzile primite de la specialist/terapeut sunt înregistrate într-o bază de date SQL cu scopul de a fi analizate ulterior de specialiști în terapia copiilor cu tulburări de spectru autist.

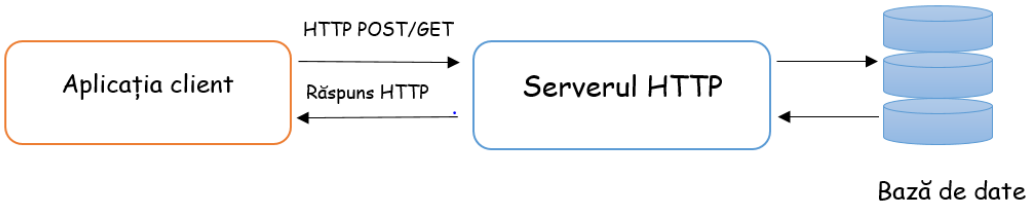


Figura 5.2: Arhitectura aplicației

Diagrama de funcționare

Pentru a putea fi înțeles modul de funcționare a aplicației am realizat o diagramă (Figura 5.3). În cadrul acestei diagrame sunt evidențiate principalele etape care conduc la îndeplinirea funcționalității aplicației. Utilizatorul are acces, în primă fază, la o pagină de start de unde începe ședința de terapie. Acesta își poate selecta profilul, prin alegearea unui terapeut, urmând alegerea pacientului. Abia după ce a trecut prin acești pași, putem spune că a început o ședință de terapie. Terapeutul alege un program de terapie, conceput special pentru genul acesta de ședințe. Acesta poate în continuare să acorde pacientului un feedback, având posibilitatea oricând să se întoarcă la meniul principal.

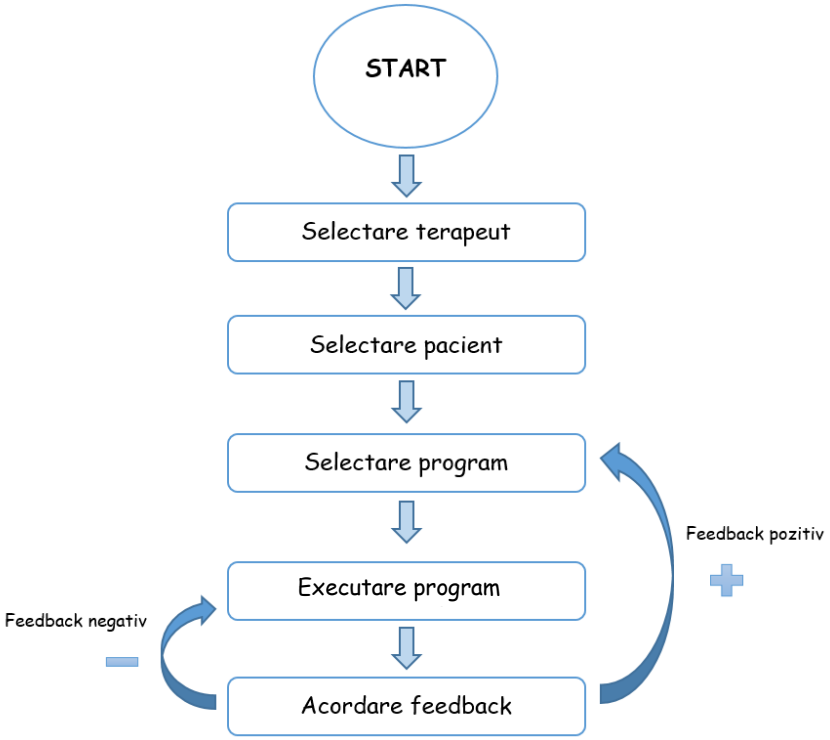


Figura 5.3: Diagrama de funcționare a aplicației Nao Terapie

Comunicarea cu Robotul NAO

Pentru a realiza scopul acestei teze, și anume dezvoltarea unei platforme software inteligente ce poate comunica cu sisteme embedded, în cazul de față robotul umanoid NAO, am dezvoltat o comunicație între aplicațiile dezvoltate și resursele ce se regăsesc pe robotul NAO. Pentru a avea acces la resursele robotului am folosit o conexiune prin SSH descrisă în cele ce urmează.

- Conexiunea prin SSH(Secure Shell)

Pentru a putea accesa amănunțit resursele robotului este necesară conectarea prin protocolul SSH. SSH este un protocol care permite realizarea unor sesiuni de lucru de la distanță, crearea canalelor de comuncații pentru anumite aplicații și transferul de fișiere. Criptarea asigură securitatea si confidențialitatea transmisiei. Autentificarea se realizează prin criptografie asimetrică, serverul dispunând de o cheie secretă, iar clienții dispun de o cheie publică. Ca și autentificarea serverului, autentificarea clientului se realizează prin criptografie asimetrică, dar folosind altă pereche de chei.[17]

În cazul de față, calculatorul utilizatorului reprezintă clientul care inițializează conexiunea prin SSH, iar robotul NAO este serverul. Pentru a suporta acest tip de conexiune, robotul NAO trebuie să aibă instalat un server SSH pe sistemul său de operare. Aproape orice sistem de operare bazat pe Linux are implicit instalat un asemnea server.

Etapele stabilirii unei conexiuni SSH:

- Folosind protocolul Diffie-Heșman, clientul și serverul se înțeleg asupra unei chei de sesiune. Urmând ca o comunicație să fie criptată cu cheia de sesiune.
- Clientul autentifică serverul. Serverul poate trimite rezultatul semnării cheii de sesiune cu cheia secretă.
- Clientul verifică semnătura folosind în acest scop cheia publică a serverului.
- Serverul autentifică clientul. În funcție de cum este configurat serverul, acesta poate accepta autentificare cu criptografie asimetrică, folosind același protocol, sau poate cere clientului o altă parolă.

SSH permite sesiuni de lucru și prin alte aplicații. Odată ce se deschide un canal securizat, pachetele pot fi destinate mai multor aplicații, cum ar fi:

- Sesiunea de lucru în mod text;
- Transferal de fișiere;
- Retransmiterea unor prototipuri TCP;
- Retransmiterea serverului X, clientul SSH acționează ca un server X și retransmite cererile de la clienți către serverul SSH).

În etapa inițială de implementare practică a lucrării, pentru a putea avea acces la resursele robotului și tot odată pentru a realiza un transfer al fișierelor necesare rulării aplicației software am folosit aplicația software FileZilla ce funcționează pe baza protocolului SSH.

Autentificarea clientului SSH se realizează prin parolă sau criptografie. Este lansată o aplicație numită “agent de autentificare” care citește cheia secretă criptată. Urmând să ceară cheia de decriptare pe care o memorează în RAM. În momentul lansării unui client SSH, se încearcă conectarea agentului de autentificare – dacă agentul rulează în acel moment. Comunicația se realizează local prin primitive oferite de sistemul de operare al dispozitivului client. Cheia agentului de sesiune este trimisă de către clientul SSH, iar agentul îi returnează semnătura.

Pentru a emula un terminal ca fiind un client pentru SSH am folosit programul gratuit PuTTY. Pentru a realiza o conexiune trebuie furnizate câteva date necesare, cum ar fi: numele de host sau IP (adresa IP a robotului NAO),

portul utilizat și tipul conexiunii. În figura următoare (Figura 5.4) este ilustrată fereastra principală a utilitarului PuTTY utilizată pe parcursul practic al lucrării.

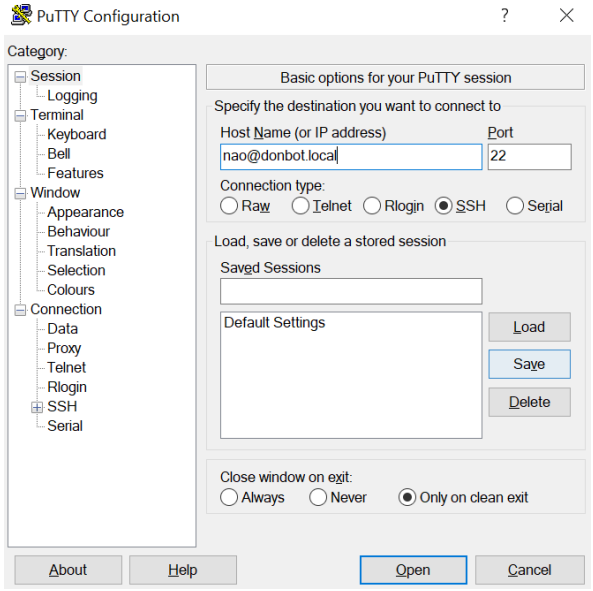


Figura 5.4: Utilitatul PuTTY

Pentru transferul fișierelor de pe calculatorul personal am folosit utilitarul FileZila (Figura 5.5), un program software puternic pentru transmiterea fișierelor. Unul din avantajele principale este faptul că aplicația este user-friendly, ceea ce înseamnă că se poate cu ușurință descărca, încărca sau manageria fișiere și foldere.

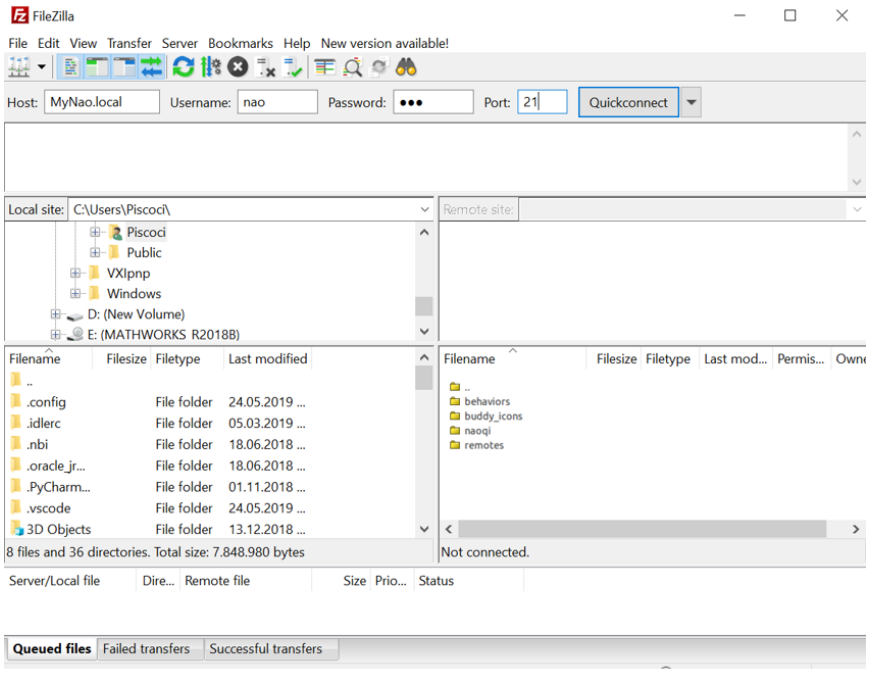


Figura 5.5: Utilitarul FileZila utilizat pentru schimbul de fișiere.

5.3 Arhitectura de comunicație Client-Server

Arhitectura de comunicație Client-Server a fost introdusă pentru prima dată în anii 1960-1970 de către cercetătorii firmei Xerox pentru dezvoltarea design-ului de programare pentru rețelele de calculatoare. Pentru o vreme, acest model a fost folosit pentru a conecta mainframe-urile. (Mainframe-urile în prezent sunt computere cu o putere de calcul foarte mare, însă în contextul actual se referă la primele calculatoare dezvoltate de IBM.) Calculatoarele au evoluat, însă conceptele modelului Client-Server au rămas aceleași. Această arhitectură este reprezentată de două părți, sistemul pentru partea de client și cel pentru partea de server. Aceste două componente interacționează și formează o rețea.

Motivele pentru care este folosit acest model de comunicație:

- Este posibilă stocarea informației într-un singur loc, putând fi redistribuită ușor către clienți;
- Este posibilă dedicarea resurselor calculatorului unor sarcini specifice.

Modelul Client – Server reprezintă o arhitectură sau o structură ce se comportă ca o aplicație distribuită partajând procesarea între furnizorii de servicii, servere și elementele care solicită servicii, clienții. Serverele și clienții comunică prin intermediul unei rețele de calculatoare, utilizând de obicei Internetul, cu suporturi hardware diferite [13].

Arhitectura Client-Server, fiind o arhitectură stratificată, împarte aplicația în trei componente de bază: clientul, infrastructura rețelei, serverul. De asemenea, toată structura Internetului se bazează pe acest tip de arhitectură. Milioane de servere ce rulează continuu sunt conectate la rețeaua globală.

În domeniul calculatoarelor, un server este un program de aplicație sau un dispozitiv care furnizează servicii altor programe sau dispozitive, numite aplicații "client". De obicei, aplicația server așteaptă conexiuni din partea aplicațiilor client. Se mai numește server și calculatorul de mare putere pe care rulează una sau mai multe asemenea aplicații. În multe cazuri soluția pentru aplicații de mărimi mari cu mulți utilizatori se bazează pe arhitectura client-server, care constă în cel puțin 2 aplicații. Serverele sunt deseori folosite simultan și pentru alte scopuri, dar când sunt necesare, pot fi rezervate exclusiv pentru funcția de server. Cuvântul server provine din limba engleză, de la cuvântul to serve – a servi: calculatorul pe post de server poate deservi întreaga rețea de calculatoare pe post de client, pentru a putea oferi accesul la întreaga gamă de forme de conectare și servicii. În multe cazuri, același calculator poate juca atât rol de server, cât și de client în același timp. Host computer – computer gazdă este un alt termen folosit pentru server. [14]

De obicei, un client este o componentă software sau hardware care utilizează un serviciu pus la dispoziție de un server. Dacă clientul accesează serviciul prin intermediul unei rețele, atunci serverul poate fi pe un alt sistem informatic. Principiul a fost aplicat inițial dispozitivelor ce nu reușeau să ruleze propriile programe, dar erau capabile să interacționeze cu camputerele de la distanță prin intermediul unei rețele.

În vederea dezvoltării practice a unei platforme software inteligente ce poate comunica cu sisteme embedded, în cazul de față robotul umanoid NAO, lucrarea se bazează pe o arhitectură de comunicație Client-Server. Robotul umanoid NAO suportă conexiune la Internet. Pentru a putea fi transmise comenzile din partea clientului/ utilizatorului, robotul trebuie să comunice cu aplicația. Astfel, a fost creat un server local și o bază de date ce sunt prezente în sistemul roboului NAO, fiind corelate cu o interfață web cu rolul de a interfața toate cele 9 programe disponibile la momentul prezentării tezei de diplomă.

5.4 Dezvoltarea Serverului HTTP

Introducere

Serverele Web răspund la cereri HTTP (Hypertext Transfer Protocol requests) de la clienți și trimit înapoi răspunsuri ce conțin un cod status și în multe cazuri conținuturi de tipul HTML, XML, sau JSON.

Serverul și Clientul interacționează printr-un limbaj standardizat de World Wide Web. Acest limbaj standard este motivul pentru care un browser vechi Mozilla Netscape poate încă să comunice cu un web server Apache modern, cu toate că nu poate reda design-ul paginii într-un mod corespunzător.

În comunicarea dintre un client și un server, clientul trimite cereri către serverul web. Clientul este de obicei un browser web cum ar fi Chrome sau Firefox. Serverele web procesează cererea de la clienți. Acest proces se finalizează cu un cod de răspuns din partea serverului și un conținut. Într-un caz simplu, clientul va solicita un element static, cum ar fi o imagine sau un fișier JavaScript. Fișierul se află în sistemul de fișiere într-o locație pe care serverul web este autorizată să o acceseze, urmând ca serverul web să trimită fișierul clientului cu un cod de stare de 200. Dacă clientul a solicitat deja fișierul și acesta nu s-a schimbat, serverul web va transmite un răspuns "Nemodificat", cu codul 304, indicând faptul că partea de client are deja cea mai recentă versiune a acelui fișier.

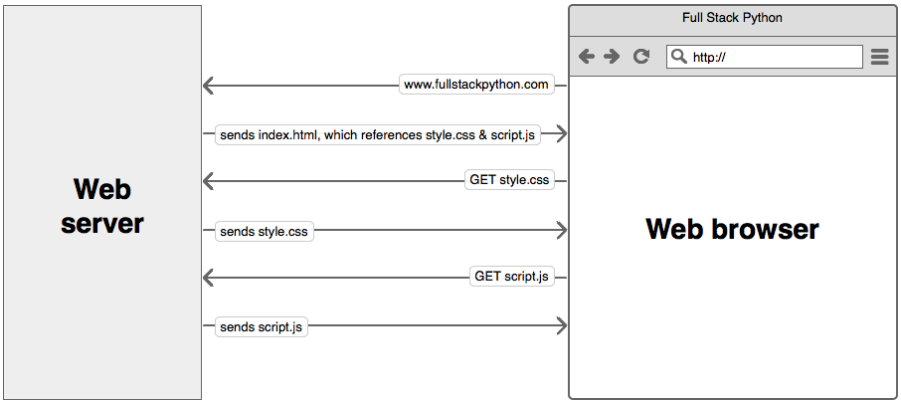


Figura 5.6: Exemplu de comunicație Server Web – Browser Web

În Figura 5.6 este descries modul în care serverul web comunică cu pagina web. Un server web trimite fișiere către un browser web pe baza cererii browser-ului web. În prima solicitare, browser-ul a accesat adresa "www.fullstackpython.com", iar serverul a răspuns cu fișierul index.html formatat în format HTML. Acest fișier HTML conține referințe la alte fișiere, cum ar fi stil.css și script.js pe care browserul le-a solicitat apoi de la server.

Dezvoltare Server web HTTP

Un server web HTTP este un proces care rulează pe un calculator și realizează două lucruri: ascultă cererile HTTP primite pe o anumită adresă de socket TCP (adresă IP și un număr de port) și se ocupă de această solicitare, transmițând un răspuns înapoi utilizatorului.

Distribuția implicită Python are suport încorporat pentru protocolul HTTP pe care îl poate utiliza pentru a crea un server web autonom. Modulul Python care oferă acest suport se numește BaseFTTPServer. Acest modul a fost folosit în dezvoltarea server-ului web prin simpla includere a acestuia în sursele proiectului folosind următoarea sintaxă:

```
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
```

Pentru a difuza fișiere statice cum ar fi index.html, stil.css, cod JavaScript, sau imagini am creat o clasă ce gestionează orice solicitare de la browser – HTTPRequestHandler. Clasa BaseHTTPRequestHandler este utilizată pentru a gestiona cererile HTTP care sosesc la server. Singură, nu poate răspunde la nici o cerere HTTP reală, aceasta trebuie să fie subclasată pentru a gestiona fiecare metodă de solicitare. Am extins cele două metode principale GET și POST ce se ocupă de toate cererile HTTP care sosesc. Operatorul va analiza cererea și anteturile, apoi va apela o metodă specifică tipului de solicitare. Numele metodei este construit din cerere. De exemplu, pentru metoda de cerere GET, metoda do_GET() va fi apelată fără argumente. Toate informațiile relevante sunt stocate în variabilele operatorului (the handler). Subclasele nu suprascriu sau nu extind metoda __init __.

Pachetul *time* scrie pur și simplu timbre pentru momentul în care serverul pornește sau se oprește. Pentru a rula serverul prima linie, *if __name__=='__main__':*, asigură că este rulat acest fișier specific. În continuare este creat un obiect de tip HTTP:

httpd = HTTPServer(('0.0.0.0', 8000), HTTPRequestHandler)

Pasăm un parametru care conține HOST_NAME și PORT_NUMBER, adică numele gazdei și numărul portului, fiind și portul pe care vrem să ruleze serverul. Respectiv: HOST_NAME - '0.0.0.0', PORT_NUMBER - 8000. Ca al doilea argument, este pasată librăria de manipulare al server-ului, HTTPRequestHandler, importată din propriul fișier.

- **do_POST()**

Pentru consecvență, metodele de tipul “request handler” nu primesc nici un argument. Toți parametrii pentru o cerere sunt analizați de către BaseHTTPRequestHandler și stocați ca atribute ale instanței de cerere. Fiecare răspuns are nevoie de un cod de răspuns, setat prin metoda send_response (). Dacă este utilizat un cod de eroare (404, 501 etc.), în antet este inclus un mesaj de eroare implicit sau poate fi transmis un mesaj cu codul de eroare.

```
def do_POST(self):
  global start_therapy
  if None != re.search('/api/v1/program1/', self.path):
    actionsPossible = {'play':pr1.play,'replay':pr2.replay}
    parametersPossible = {'serie1':1,'serie2':2,'serie3':3,'serie4':4,'serie5':5}
    query=urlparse(self.path).query
    query_components = dict(qc.split("=") for qc in query.split("&"))
    action = query_components["action"]
    parameter = query_components["parameters"]
    if start_therapy==0:
      self.wave_hand()
      start_therapy=1
    actionsPossible[action](parametersPossible[parameter])
    self.send_response(200)
    self.send_header('Content-Type', 'application/json')
    self.end_headers()
    return
```

Figura 5.7: Metoda POST – Server web HTTP

Scriptul primei subclase a metodei POST este ilustrată în Figura 5.7. BaseHTTPRequestHandler deține instanța variabilei path – aceasta conține calea de solicitare. Prima instrucțiune if din script, compară string-ul obținut prin path cu un model folosind metoda re.search(). Această metodă scanează string-ul căutând prima locație unde modelul de expresie '/api/v1/program1/' produce o potrivire și returnează un obiect de potrivire corespunzător. În cazul în care nu există nici o poziție în șir care să corespundă modelului, este transmis ‘None’. În situația în care

este găsită o potrivire, rularea continuă cu următoarea linie de cod a scriptului. Au fost declarate două dicționare: actionPossibile conține cele două metode disponibile în script-ul de Python al programului 1; parametersPossibile conține cele 6 serii ale programului 1. Metoda urlprase(), parsează o adresă URL în șase componente. Metoda returnează o tuplă ce conține 6 elemente. Aceasta corespunde structurii unei adrese URL: scheme://netloc/path;parameters?query#fragment. În variabila 'query' este stocat string-ul query al URL-ului. Variabila query_components stochează un dicționar al elementelor de tip key:value din query-ul creat anterior. Action stochează un string de tipul play/replay, iar parameter conține seria.

Toate subclasele funcției do_POST() funcționează într-o manieră asemănătoare. Prima dată se află elementele/ informațiile necesare din query-urile transmise de pe partea de client și se execută comenzile corespunzătoare.

- **do_GET()**

Parametrii unei cereri apar în șiruri de cereri, query sau String, separate prin calea de companentă cu '?'. Calea generală pentru metoda de serviciu este de a extrage informațiile de la obiectul de tip Request primit ca parametru, acces la resurse externe, urmând popularea obiectului Response cu informații. Metoda corectă de a realiza un răspuns este ca prima dată să se obțină un flux de ieșire(output stream) pentru răspuns, să se completeze header-ele răspunsului, apoi să se scrie corpul răspunsului în fluxul de ieșire. Heder-ele trebuie setate întodeauna înainte ca răspunsul să fie trimis.

Obiectivul acestei funcții este de a preleva informații de pe server folosind aplicația client. Scriptul funcției do_GET() începe cu următoarea linie de cod:

```
root=os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),'appTerapie')
```

În scriptul anterior există 3 metode și 2 constante: abspath() – returnează calea absolută a lui path; join() – alătură lui path string-uri; dirname() – returnează directorul unui fișier; __file__ – se referă la numele fișierului scriptului; pardit() – returnează reprezentarea directorului părinte din sistemul de operare. Astfel, expresia returnează numele căii(path-ului) complete a scriptului de execuție într-un mod multiplatform.

```
if self.path == '/api/v1/getTerapeuti':
    jsonData = '{"table_name": "TERAPEUTI", "data": {"NUME": "", "PRENUME":""}}'
    jsonData = self.interfaceDB.json.loads(jsonData)
    self.send_response(200)
    self.send_header('Content-type', 'application/json')
    self.end_headers()
    self.wfile.write(self.interfaceDB.getData(jsonData))
    return
```

Figura 5.8: Metoda GET – Server web HTTP

În biblioteca json, se găsesc metodele load() și loads() pentru a transforma datele codate JSON în obiecte Python. Astfel, variabila jsonData stochează un dicționar ce include datele compatibile. Metodele wfile.write() returnează datele de tip string, jsonData, din baza de date. Metoda send_header() adaugă antetul HTTP într-un buffer intern care va fi scris în fluxul de ieșire atunci când este invocat end_header(). 'Content-type' specifică cuvantul cheie al antetului, a cărui valoare este 'application/json'. Important este ca după terminarea apelului metodei send_header(), să fie apelată metoda end_header() pentru a finaliza operația. Wfile conține fluxul(stream-ul) de ieșire pentru scrierea unui răspuns către client. Toate metodele explicate anterior se regăsesc în componența scriptului din Figura 5.8.

5.5 Dezvoltare aplicație client – Aplicația Nao Terapia

Crearea unui nou proiect

Primul pas în dezvoltarea unei aplicații Angular este de a crea un nou proiect din linia de comandă (Figura 5.9). Această comandă crează un folder denumit "naoterapiaangular-master" și copiază toate dependențele necesare și toate setările de configurație. CLI Angular realizează toate aceste necesități: crează un nou director "naoterapiaangular-master", descarcă și instalează librăriile Angular, instalează și configurează TypeScript, Karma și Protector (librării de testare). Se poate folosi și comanda "ng init".

```
C:\Users\Piscoci\Desktop\NAO>ng new NAOTERAPIAANGULAR-MASTER
```

Figura 5.9: Comandă creare proiect nou Angular

Diferența dintre cele două constă în faptul că *ng new* necesită specificarea numelui de folder, creând acest folder și copiind fișierele necesare, în timp ce "ng init" copiază fișierele în folderul curent. După ce folderul creat este accesat, folosind comanda „ng serve" se poate vizualiza aplicația web într-un browser.

Comanda *ng serve* lansează serverul, urmărește fișierele și reconstruiește aplicația pe măsură ce sunt aduse modificări. Pagina ce este deschisă este ilustrată în următoarea imagine (Figura 5.10).

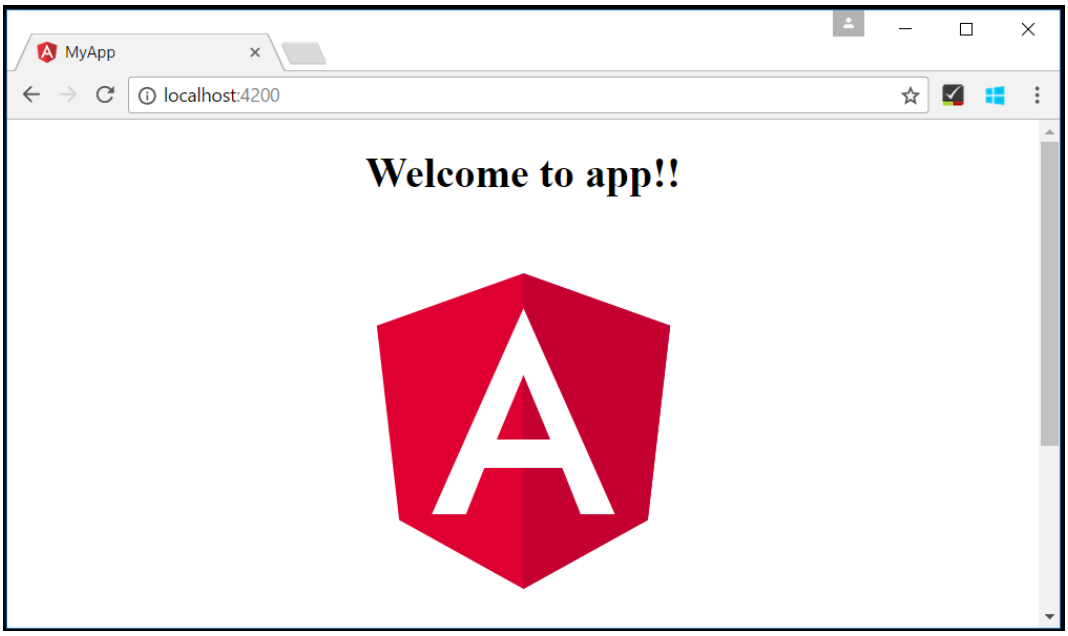


Figura 5.10: Pagină de start în aplicația Angular

Pentru o înțelegere cât mai bună a structurii aplicației și a modificărilor, în continuare vor fi detaliate câteva din fișierele și folderele ce se regăsesc în arhitectura aplicației create – Nao Terapia (Figura 5.11).

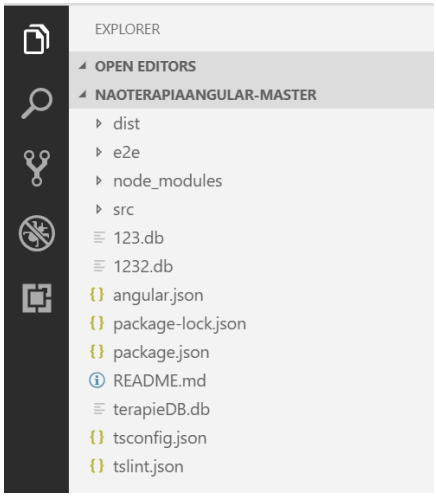


Figura 5.11: Arhitectura aplicației Angular

- e2e – Sau "end-to-end" este parte a programului în care sunt descrise task-uri end to end ale aplicație. În esență, această parte a aplicației conține teste automate ce simulează un utilizator real. Se poate scrie cod pentru a porni browserul, pentru a naviga la pagina de start a aplicației, sau pentru completarea de formulare.
- Node_modules – În acest folder sunt stocate toate librăriile de care depinde aplicația, fiind folosit pentru dezvoltare.
- Src – Reprezintă folderul sursă, aici se alfă codul sursă al aplicației.
- package.json – Ca orice aplicație web modernă, este nevoie de un sistem de pachete și de un manager de pachete pentru a gestiona toate bibliotecile și modulele terțelor părți care utilizează aplicația. În acest fișier se găsesc toate dependențele sau scripturile npm (manager de pachete JavaScript).
- tsconfig.json – Fișier de configurare principal. Trebuie să fie în calea rădăcinii, deoarece acolo este locul unde compilatorul TypeScript îl va căuta.

În interiorul directorului /src se regăsește codul sursă brut, necompilat. Când executăm comanda ng serve, codul din interiorul /src devine pachet (bundled) și este translatat în versiunea corectă Javascript pe care browser-ul o înțelege (în prezent, ES5). Asta înseamnă că se poate lucra la un nivel mai înalt folosind TypeScript, dar putem compila până la forma cea mai veche de Javascript de care browser-ul are nevoie. Directorul /src are structura folderelor principale ilustrată în figura următoare (Figura 5.12).

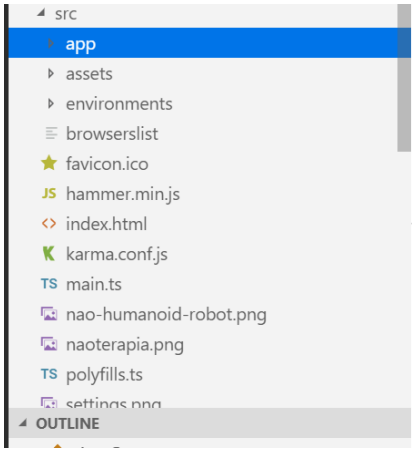


Figura 5.12: Structura fișierului *src* din aplicația Angular

- /app – Conține toate componentele, modulele, paginile ce sunt construite în aplicație.
- /assets – Acest folder conține imagini, exemple de date json și orice alt material de care are nevoie aplicația.
- index.html/ - Este pagina gazdă a aplicației, dar în cazul de față funcționează doar ca substituent. Toate scripturile și stilurile necesare pentru a realiza lucrul cu aplicația vor fi injectate automat de procesul de grupare a pachetelor web.

Structura finală a proiectului este ilustrată în figura următoare (Figura 5.13).

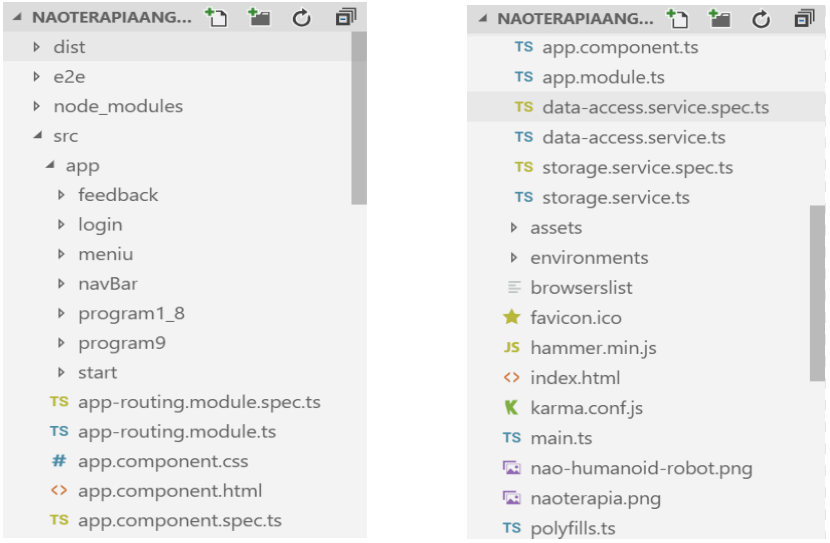


Figura 5.13: Structura finală a aplicației Angular

Cum activează Angular index.html și cum pornește aplicația

Fișierul main.ts conține primul cod ce urmează să fie executat. Ceea ce trebuie să realizeze codul din main.ts este să bootstrapeze/construiască aplicația folosindu-se de resursele disponibile. Acest fișier încarcă și controlează toate resursele necesare pentru ca aplicația să pornească. Cele mai importante linii din codul acestui fișier sunt liniile 4 și 11 (Figura 5.14), în care bootstraps-ul începe aplicația Angular prin folosirea ca parametru a modulului AppModule.

```
1  import { enableProdMode } from '@angular/core';
2  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4  import { AppModule } from './app/app.module';
5  import { environment } from './environments/environment';
6
7  if (environment.production) {
8    enableProdMode();
9  }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.log(err));
```

Figura 5.14: Fișierul main.ts – aplicației Nao Terapia

- platformBrowserDynamic().bootstrapModule(AppModule) are modulul de referință AppModule. Prin urmare, atunci când se execută în browser, fișierul care este accesat este index.html. Acesta din urmă se referă intern la main.ts care apelează modulul părinte, adică AppModule când se execută următorul cod.
- Cand AppModule este apelat, accesează în continuare app.module.ts care urmează să apeleaze AppComponent bazându-se pe comanda boostrap: 'bootstrap: [AppComponent]'.
- În app.components.ts, există selectorul app-root care este folosit în fișierul index.hml. Acesta afișează conținutul prezentat în app.component.html.

Cea mai importantă parte dintr-un fișier, când pornește aplicația Angular, este matricea de bootstrap în decoratorul @NgModule din modulul generat automat app.module.ts (Anexa 1). Decodorul NgModule consolidează componentele, directivele și legăturile între blocurile de funcționalitate, fiecare concentrându-se pe o zonă de caracteristici, domeniu de aplicații, flux de lucru sau o colecție comună de utilități. În principiu, există o listă cu toate componentele și directivele pe care le importă pentru a putea fi cunoscute de aplicația Angular în momentul în care analizează fișierul *index.html*. Dacă o componentă nu este inclusă în matrice, Angular nu știe de existența acesteia, deci nu o caută. În Anexa 1 se regăsește fișierul app.module.ts ce conține matricea de bootstrap din decoratorul @NgModule cu toate componetele declarate și cele importate pentru ca aplicația să funcționeze corespunzător.

În declarații (declarations), se stochează referința componentelor. Fișierul AppComponent este componenta implicită care este creată de fiecare dată când este inițiat un nou proiect. De asemenea, trebuie importate toate modulele/componentele pentru ca aplicația să aibă acces la resursele necesare rulării. De exemplu, BrowserModule exportă CommonModule și oferă aplicației servicii specifice platformelor de tip browser disponibile în @angular/platform-browser.

Sumar:

- Angular rulează pentru prima data fișierul main.ts.
- Bootstrapează o aplicație Angular și oferă ca parametrii app.module.ts. In app.module.ts sunt declarate și importate toate componentele de care are nevoie aplicația să pornească.
- Angular analizează componentele anterior menționate, citind inițializările unde se află și selectorul (SELECTOR) app-root.
- Angluar urmează să manevreze app-root din index.html, cunoscând regulile pentru SELECTOR (acesta trebuie să insereze componentele aplicației și să aibe un cod html/o componentă html).

Pagina de start

Pagina principală/de start a aplicației conține o imagine a robotului NAO (Figura 5.15). Cu un click pe această imagine se realizează accesul la pagina în care se poate selecta terapeutul. După ce a fost selectat un terapeut, există posibilitatea de a selecta copilul pentru care urmează să se realizeze ședința de terapie. Este posibilă, de asemenea, introducerea de noi terapuți sau pacienți în cadrul acestor pagini.

Orice aplicație Angular are nevoie de cel putin o componentă și un modul rădăcină (root module) pentru a funcționa. Componentele sunt în esență clase ce interacționează cu fișierele *.html* ale componentelor, acestea din urmă sunt afișate în browser.

Figura 5.15: Pagina de start a aplicației Nao Terapia

Fișierul app.component a fost creat în mod implicit atunci când aplicația a fost creată din linia de comandă Angular. Structura fișierului app.component este ilustrată în figura următoare (Figura 5.16). app.component.css poate conține o structură de tipul CSS. App.components.html conține codul html, app.component.spec.ts este o componentă generată automat și conține teste pentru componentele sursă. app.component.ts conține clasa pentru componentele definite aici. De asemenea, se poate realiza procesarea structurii .html în fișierele .ts. Procesarea include activități de interacțiune cu alte componente, routare, servicii, sau conectare la data de baze. Așa cum a fost menționat anterior, AppModule este importat din aplicația principalului modul părinte, care la rândul său este importat de către modulul bootstrap ce realizează încărcarea modulului app.module.ts.

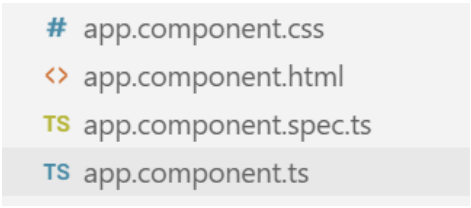


Figura 5.16: Structura fișierului app.component – aplicație Nao Terapia

Pentru a crea o nouă componentă se folosește comanda ‘ng g component start’. Atunci când este apelată această comanda în CLI se va crea automat un fișier ce conține toate componentele necesare. Tot atunci sunt aduse modificări modulului *AppModule* prin adăugarea declarației clasei nou create care se comportă ca o componentă derivată (child component). Fișierul *start.component.ts* (Anexa 1) a fost generat, iar modificările ulterioare sunt explicate în cele ce urmează.

Fisierul *strat.component.ts* crează o nouă clasă numită StartComponent, care moștenește OnInit. OnInit conține o metodă și un constructor numită ngOnInit(). Această metodă este apelată atunci când componenta este executată. Tot aceasta inițializează directiva/componenta după ce aplicația Angular afișează pima dată proprietățiile datelor și stabilește datele de intrare ale directivei /componentei. De asemenea, este declarată variabila *title* ce este folosită în codul html.

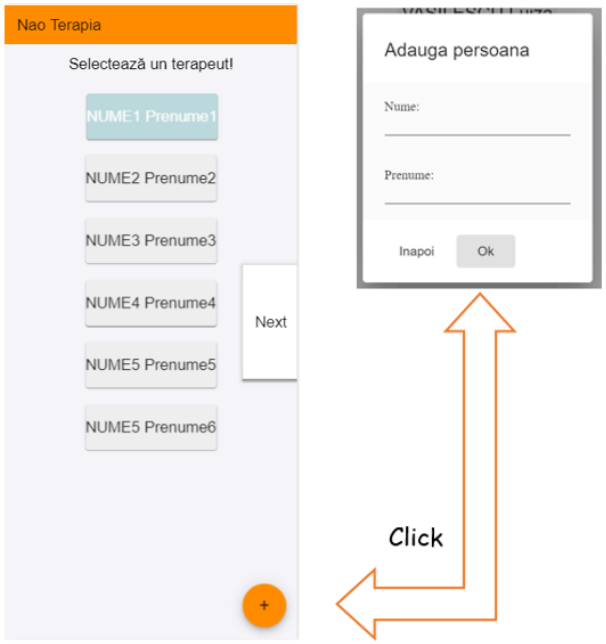
Pentru a putea stoca date pe partea de client local, folosim *LocalStorageServices*. Se vor salva local informații legate de: numele terapeutului, numele pacientului, ID sesiunii de terapie și numele programului pentru a putea fi salvate ulterior în baza de date. Am ales să folosesc această metodă de stocare în locul cookie-urilor pentru că: datele sunt salvate local și nu pot fi citite de server (elimină problemele de sercuritate), se pot salva mai multe date (pana la 10Mb) și este mult mai ușor de folosit datorită sintaxei, fiind suportată de majoritatea browserelor moderne.

În continuare sunt explicate liniile de cod adăugate fișierului de tip html, *start.component.html* din Anexa 2, al paginii de start. Elementul html <a>, sau elementul de ancorare, împreună cu atributul href, creează un hyperlink către alte pagini web, fișiere, locații în aceeași pagină, adrese de e-mail sau orice altă adresă URL. În cazul de față, se realizează rutarea cu ajutorul directivei *routerLink*. Dând click pe imaginea naoterapia.png, care devine o ancoră în interfața utilizatorului, se crează o rută către pagina de selectare a terapeutului. Mai exact: http://localhost:8000/#/ - adresa interfeței paginii de start, devine http://localhost:8000/#/login/terapeut – adresa interfeței paginii de selectare a terapeutului.

Variabilei *title* îi este anterior alocat string-ul de caractere “Nao Terapia”, iar elementul de titlu <h2> definște o descriere a pașilor pe care trebuie sa-i urmeze utilizatorul.

Pagina de selectare a terapeutului

Pentru ca baza de date să conțină informații relevante progresului fiecărui pacient, am ales să dezvolt o pagină de logare, în care fiecare terapeut să își poată accesa profilul prin intermediul numelui și prenumelui. Există posibilitatea de a adăuga noi terapeuți acestei liste cu ajutorul butonului special creat in dreapa ecranului. Acest buton folosește serviciul MatDialog pentru a deschide o fereastră de dialog cu utilizatorul. Design-ul paginii de selectare a terapeutului este prezentat în Figura 5.17(a). Singurele diferențe între această pagină și pagina de selectare a copilului este titlul acesteia și procedurile din codul componentei html.



(a) **(b)**

Figura 5.17: (a) Pagină de logare terapeut – aplicație Nao Terapia. **(b)** Fereastră de dialog, introducere terapeut nou – aplicație Nao Terapia.

Angular conține o bibliotecă dedicată care implementează *Google Material Design* cu componente Angular și materialele Angular. Ideea este de a avea un limbaj de design consecvent în toate aplicațiile, dar și o experiență extraordinară de utilizare și accesibilitate încorporată. Prima parte în crearea unei ferestre de dialog (Figura 5.17(b)) este importarea serviciului MatDialog pentru a putea avea acces la serviciile și directivele acestuia. Pentru a deschide o fereastră de dialog atunci când se face click pe butolul "+", am creat componenta *adaugaterapeut()*. Această componentă este accesată în codul html pentru butonul în cauză.

```
<button [disabled]=”isDisabled” class= ‘mat-fab’ (click)=”adaugaTerapeut()”><h1 class=”mat-h1”>+</h1></button>
```

Componenta corespondentă conține linile de cod detaliate în cele ce urmează. Am folosit, cum am menționat anterior, serviciul MatDialog din Angular Material să creez fereastra de dialog. Folosind metoda *open()* am injectat o instanță a serviciului în componenta *TerapeutComponent*, menționând ce tip de componentă să folosească pentru a crea fereastra de dialog. Metoda open() din cadrul componentei teraput.component.ts se regăsește în Anexa 1.

Prin apelul metodei *open()* o componentă este încărcată, returnând o instanță a MetDialogRef. Pentru a împărți informații cu componenta de dialog, opțiunea *data* pasează informații componentei dialog precum numele și prenumele terapeutului. Atunci cand fereastra de dialog este închisă, fie prin butonul de ieșire sau folosind API-uri, sunt emise valori prin intermediul variabilei *result*. Aceste valori sunt transmise ca rezultat al metodei *afterClosed()* ce aparține componentei terapeut.component.ts (Anexa 1).

După ce utilizatorul introduce numele și prenumele terapeutului, metoda afterClosed() transmite parametrul *result*. Acesta conține datele: NUME și PRENUME. Pentru a transmite informațiile bazei de date am realizat concatenarea URL-ului cu rezultatele obținute anterior cu ajutorul metodei concat(). Astfel, http.post(url,0) transmite noile informații către baza de date.

În pagina de selectare a terapeutului este prezetată lista tuturor terapeuților salvați în baza de date. Aceste informații sunt accesate atunci când componenta este apelată din directiva ngOnInit() cu ajutorul metodei http.get() din codul componentei *terapeut.component.ts*. Codul html, terapeut.component.html din Anexa 2, conține directiva *ngFor care este folosită pentru a dispune lista terapeuților, astfel încât utilizatorul să poată selecta în continuare profilul pe care îl dorește. În această formă, template-ul care trebuie redat pentru fiecare iterație este conținutul unui element de ancorare care conține directiva. *ngFor este varianta prescurtată a directivei ngForOf, această formă se extinde într-o formă care folosește selectorul ngForOf pe un element <ng-template>. Angular extinde automat sintaxa prescurtată pe măsură ce compilează template-ul.

Componenta onSelect() primește ca parametru un obiect de tip Persoana: onSelect(persoana:Persoana). Atunci când utilizatorul dă click pe numele terapeutului, informațiile sunt salvate în LocalStorage al browserului web pentru a putea fi salvate ulterior în baza de date. În LocalStorage sunt salvate perechi de elemente *Key* și *Value*. Comanda storeOnLocalStorage() salvează în LocalStorage un element selectat de tip "terapeut" și obiectul de tip "Persoana".

Componentele de stil sunt realizate prin intemediul componentei terapeut.component.css (Anexa 2). Aici sunt descrise cum toate elementele HTML sunt dispuse în pagină. CSS poate fi adăugat elementelor HTML în 3 moduri: folosind atributele de stil în elementele HTML, utilizând un element <style> în interiorul secțiunii <head>, sau utilizând un fișier extern CSS. Angular folosește ultima variantă. De exemplu, un element CSS folosit în codul html al componentei *terapeutComponent* este „mat-card", iar codul pentru acesta este prezent în Anexa 2.

Dacă utilizatorul selectează un terapeut, își face apariția butonul "Next". Cu ajutorul metodei navigateByUrl() este accesată pagina de selectare a copiilor/pacienților. Această metodă are ca parametru calea (path-ul) care trebuie accesată.

Pagina de selectare a pacientului

Toate principiile gândite și explicate anterior pentru pagina de selectare a terapeutului rămân valabile și pentru această pagină. După selectarea unui copil, apare butonul „Next” care accesează URL-ul paginii de selectare al programelor ce se doresc a fi folosite pe parcursul procesului de terapie de către utilizator/specialist/terapeut. Pagina accesată în browser este foarte asemănătoare cu pagina de selectare a terapeutului, o privire de ansamblu fiind expusă în figura următoare (Figura 5.18 (a)).

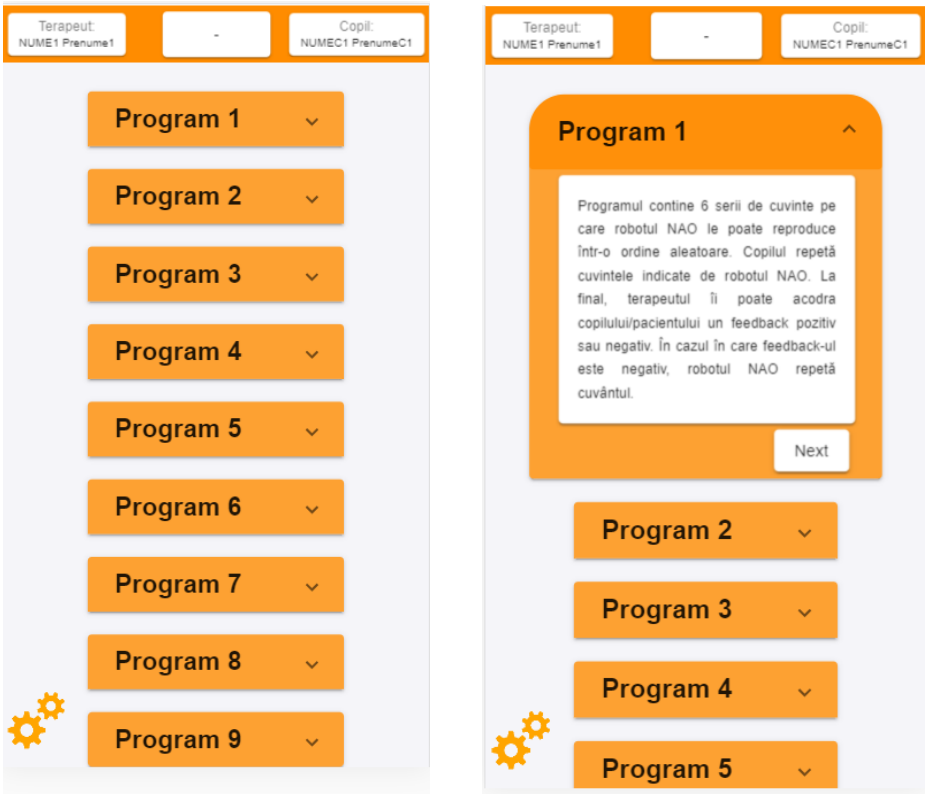
(a) **(b)**

Figura 5.18: (a) Pagină de selectare copil – aplicație Nao Terapia. **(b)** Fereastră de dialog, introducere pacient nou – aplicație Nao Terapia.

Pagina de selectare a programelor

Programarea obiect-orientată permite gruparea datelor și a funcțiilor care operează asupra lor într-o singură structură. Noțiunea centrală în programarea orientată pe obiecte este cea de clasă. O clasă este un domeniu în interiorul căreia se definesc o serie de atribute (metode și variabile). Pentru a defini mai multe instanțe ale programelor ce sunt declarate pe parcursul proiectului am creat clasa Program a cărui variabile de tip string sunt următoarele: *path, nume, descriere*. După ce am creat fișierul meniu a cărui componente construiesc pagina web de selectarea programelor, am început modificarea acestora în modul în care este explicat în cele ce urmează.

Pentru a înțelege dispunerea în pagină a componentelor, Figura 5.19 ilustrează ceea ce construiește codul componentelor. Astfel, pagina de selectare a programelor are următoarea interfață.



(a) **(b)**

Figura 5.19: (a) Meniu accesare programe – aplicație Nao Terapia. **(b)** Meniu accesare programe ce include și descrierea acestora – aplicație Nao Terapia.

În componenta meniu.component.ts (Anexa 1) am instanțiat o listă denumită *programe*, ce conține cele 9 programe ce se regăsesc în aplicația Nao Terapia. Pentru fiecare instanță am specificat cele 3 variabile, adăugând o descriere sugestivă a acestora.

În Angular se pot afișa date prin legarea controalelor într-un șablon(template) HTML la proprietățile unei componente Angular. Pentru a afișa lista numelor de programe și pentru a afișa un mesaj condiționat sub fiecare componentă a listei am folosit directiva Angular ngFor într-un template.

Am realizat un bloc de cod HTML care definește modul în care trebuie afișat un singur element, urmând a fi utilizat ca template pentru redarea fiecărui element din listă – meniu.component.html (Anexa 2). Elementul *ngFor din elementul <*mat-expansion-panel*> este directiva Angular "repeater". Acesta marchează elementul <*mat-expansion-panel*> ca "Template de repetor". Instrucțiunea ngFor "let Program of programe" este o microsintaxă – un limbaj pe care Angular îl interpretează. Acest șir de caractere are următoare semnificație: ia fiecare element de tip Program din lista programe, îl salvează în variabila buclă de element și este pus la dispoziția template-ului HTML la fiecare iterație. Asfel sunt afiște în mod repetitiv, numele și descrierea obiectelor din lista *programe* de tip *Program* create în componenta *meniu.component.ts* menționată anterior.

În Anexa 2 este prezent codul sursă al panoului de expansiune pentru componenta meniu.component.html. <*mat-expansion-panel-header*> prezintă un rezumat al conținutului panoului și acționează ca un control pentru extinderea sau prăbușirea (expending, collapsing) acestuia. Acest antet conține un <*mat-panel-title*> care formatează conținutul antetului pentru alinierea la specificațiile Material Design. În cazul în care panoul este închis/restrâns (panelOpenState=false) este afișată variabila de tip string *nume* a fiecărui program în parte. În cazul

în care panoul este deschis (panelOpenState=true) este afișată variabila de tip string *descriere* împreună cu butonul *Next* ce apelează metoda onSelect(), primind ca parametru programul în cauză.

Constructorul din fișierul meniu.component.ts (Anexa 1) conține metoda onSelect(program: Program) care primește ca parametrul un obiect de tip program. Pentru fiecare program stochează în LocalStorage variabilele de tip path și nume a fiecărui obiect definit pentru a putea fi accesate ulterior. Comanda Swich evaluează o expresie, în cazul de față newTodo în care este salvat path-ul programului, potrivește valoarea expresiei cu o clauză de caz și execută delcarațiile asociate în cazul resprectiv.

De exemplu: dacă în pagina de selectare a programului a fost selectat *Program 9* este apelată metoda *OnSelect()* cu parametrul corespunzător. În interiorul acestei metode se salvează variabilele în LocalStorage, iar variabilei newTodo îi este atribuit stringul "program 9". Valoarea este testată cu toate cazurile, urmând a fi executat blocul de cod corespunzător, adică primul "case". În interiorul fiecărui caz este prezentă o metodă de tip navigateByUrl() ce navighează catre URL-ul pe care îl primește ca parametru.

Metoda *ngOnInit(), present în Anexa 1,* rezolvă orice sarcină suplimentară de inițializare. Această metodă de apel inversă este invocată imediat după ce este detectată o schimbare a proprietăților legate de datele directivei pentru prima data și înainte de orice vizulizare a conținutului să fie verificată. Se invocă o singură dată când directiva este instanțiată.

Această metodă stochează în LocalStorage o cheie (key) 'program_name' cu o valoare nedefinită până în momentul în care este accesat butonul Next a unui program. Următorul pas este verificarea dacă cheia "SESIUNE" din LocalStorage conține o valoare sau este goală. Dacă variabila nu este definită, se folosește comanda JSON.prase() pentru a primi date de la serverul web. În cazul de față se dorește să se identifice copilul și terapeutul pentru care urmează să se realizeze o sesiune de terapie. Când se primesc date de la un server web, datele sunt întodeauna de tip String. JSON.prase() parsează datele astfel încât acestea devin obiecte JavaScript. Urmează să fie adăugate informațiile în baza de date prin intermediul metodei http.post(). Această metodă primește ca parametrii: adresa URL a resurselor de pe serverul http, *„data"* în care sunt stocate toate informațiile necesare. Atunci cand http.post() adaugă datele cu succes, *subscribe* primește noile date și le stochează în LocalStorage. Un exemplu de informații salvate în LocalStorage după accesarea unui program este ilustrat în figura rmătoare(Figura 5.20).

Key	Value
program	"program9"
terapeut	"{\"NUME\":\"Nume1\",\"PRENUME\":\"Prenume1\"}"
copil	"{\"NUME\":\"NumeC1\",\"PRENUME\":\"PrenumeC1\"}"
program_name	"Program 9"
SESIUNE	{"ID":9}

Figura 5.20: Exemplu de informații stocate in LocalStorage – aplicație Nao Terapia

Programul 1

Acest program se diferențiază de celalalte programe prin existența a mai multor serii de fișiere audio pe care robotul le poate reproduce (Figura 5.21). Pentru ca terapeutul să poată accesa aceste serii am creat o pagina de selecție a seriei.

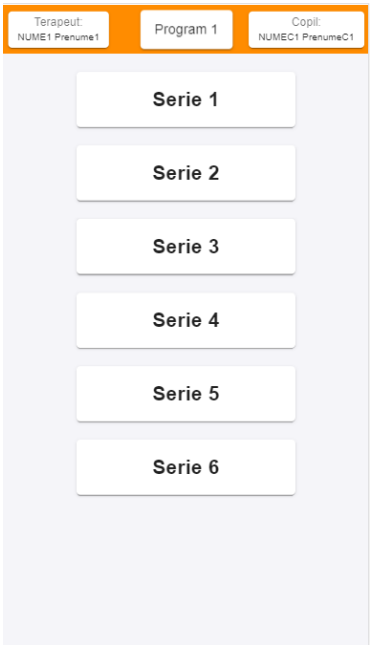


Figura 5.21: Pagină de selectare a seriei, pentru Programul 1 – aplicație Nao Terapia

Componenta *serieProg1.component.ts* (Anexa 1) conține o lista denumită *serii* ce este accesată și afișată într-un mod asemenător cu pagina de selectare a programelor. În constructor, prin intermediul metodei onSelect(), sunt salvate noile informațiile în LocalStorage și navighează utilizând metoda navigateByUrl() către pagina de redare a fișierelor audio ce se regăsesc în fiecare serie. Codul HTML: atunci când se face click pe una dintre seriile ilustrate în figură (Figura 5.23) se apelează funcția onSelect() primind ca parametru seria selectată.

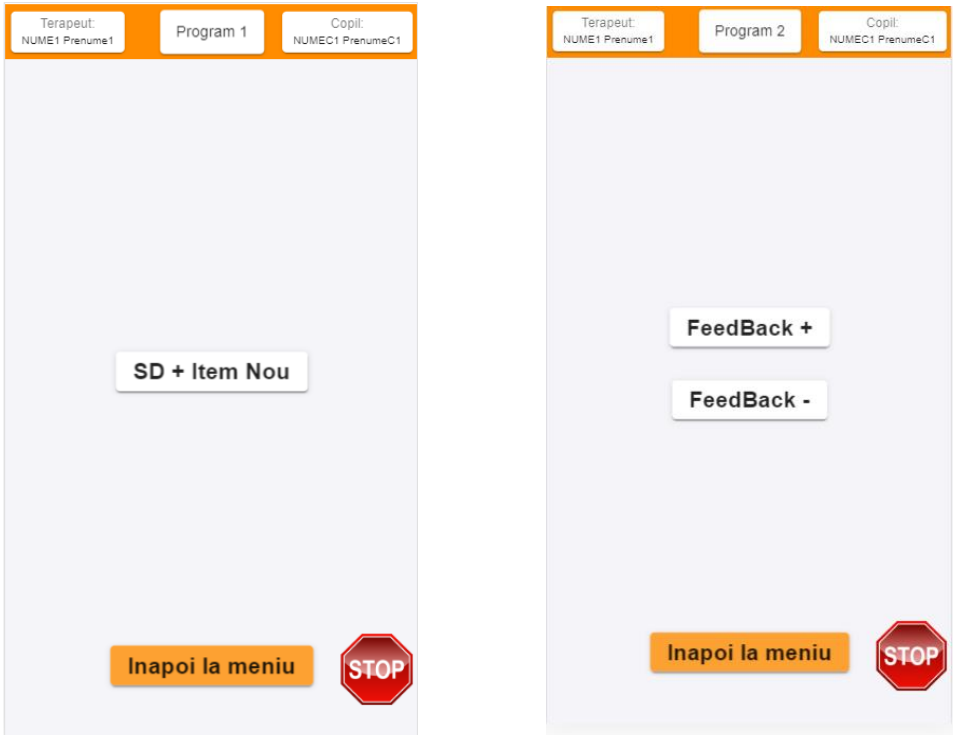
```
<mat-card *ngFor="let serie of serii"
(click)="onSelect(serie)" > <h2> {{serie.nume}} </h2></mat-card>
```

Scriptul realizează o îmbinare a controalelor dintr-un tamplate HTML la proprietățile unei componente Angular. Anterior, în componenta serieProg1.component.ts, a fost creată o listă cu serii. Pentru a afișa lista de serii, am început prin crearea unui dicționar ce conține două key: path și nume. Prima linie a codului utilizează directiva ngFor în tamplate pentru a afișa fiecare item din lista creată. Evenimeniul “Click” conduce la apelarea metodei onSelect() ce primește ca parametru un element al dicționarului. Metoda onSelect stochează path-ul seriei accesate în LocalStorage și accesează următoarea pagina prin metoda navigateByUrl().

Pagina de redare a fișierului audio

Programele 1-8 funcționează asemenător și accesează aceeași pagină de redare a fișierelor audio. După ce sunt selectate programele, în cazul programului 1 după ce este selectată și seria, se accesează o pagină în care se realizează comenzile pentru redarea fișierelor audio ce construiesc o ședință de terapie. Denumirile butoanelor și ale programelor sunt impuse de către specialiștii ce s-au ocupat de dezvoltarea acestor ședințe de terapie. Pagina

de redare a fișierelor audio este cea din Figura 5.22, aceasta conține un buton ce trimite către baza de date stringuri (șiruri de caractere) care parafazează comenzile parametrizabile disponibile pe robotul NAO.



(a) **(b)**

Figura 5.22: (a) Pagină de redare a fișierelor audio – aplicație Nao Terapia **(b)** Pagină de feedback – aplicație Nao Terapia

Atunci cand are loc un eveniment de tipul click pe butonul *SD + Intem Nou* se apelează metoda onSelect(). Această metodă salvează într-o variabilă locală valoarea cheii *program* din LocalStorage, concatenând această valoare cu o altă variabilă locală *base*. În cazul în care programul selectat este *program 1*, variabila *base* este concatenată și cu valoarea cheii *serie* din LocalStorage. Prin http.post() se transmite serverului, iar prin *subscribe* se accesează pagina pentru a acorda un feedback răspunsului pacientului.

Butoul *Înapoi la meniu* crează rutarea acestei pagini cu pagina de selectare a progamelor prin *routerLink*. Butonul de *Stop* oprește redarea fișierelor audio utilizând metoda stop() din componenta *NewSD.component.ts*. Toate metodele și elementele menționate anterior pentru pagina de redare a fișierelor audio se regăsesc în componenta *NewSD.component.ts* din Anexa 1.

Aspectul paginii pentru redarea fișierelor anterioare este identică cu cea pentru redarea fișierelor audio noi, singura diferență fiind reprezentată de numele butonului principal.

Pagina de feedback

Pentru a reține în baza de date progresul fiecărui pacient am creat o pagină de feedback (Figura 5.22(b)). După ce pacientul interacționează cu robotul NAO, răspunde indicațiilor acestuia și repetă ceea ce îi este precizat de robot, utilizatorul are posibilitatea de a-i acorda un feedback.

În momentul în care pacientului îi este acordat un calificativ pozitiv (FeedBack +), utilizatorului îi este prezentată posibilitatea de a reda un nou șir de cuvinte din același program sau aceeași serie. Dacă răspunsul pacientului

este necorespunzător, utilizatorul îi poate acorda un calificativ negativ (FeedBack -), caz în care se accesează pagina în care se poate reda item-ul anterior.

Codul HTML este unul relativ simplu, conține: 2 butoane pentru cele 2 posbilități de răspuns, un buton pentru a ne întoarce la meniul programelor, un buton de stop pentru a opri redarea fișierelor audio. Butoanele de FeedBack apelează metoda onSelect(), având ca parametru un string ce conține query-uri ce vor fi concatenate url-ului ce este transmis serverului. Exemplu query: 'feedback/?action=positive'.

Metoda onSelect() salvează în două variabile locale valorile cheilor de tip *SESIUNE* și *program*. Dacă programul selectat este programul 1 concatenează informațiile legate de serie din LocalStorage cu stringul respectiv. Dacă FeedBack-ul este pozitiv, se transmite serverului prin metoda http.post() URL-ul din care vor fi extrase informațiile în serverul HTTP. Totodată, este accesată prin navigateByUrl() pagina de redare a fișierelor audio noi. Dacă FeedBack-ul este negativ sunt transmise informațiile serverului și este accesată pagina de redare a fișierului audio anterior.

Programul 9

Acest program este diferit de programele anterioare. Robotul NAO redă o poveste, aceasta este urmată de întrebări ce se bazează pe informațiile ce se regăsesc în poveste. Există o pagină în care utilizatorul poate selecta începerea poveștii (Figura 5.23). Copilul trebuie să răspundă la aceste întrebări, iar în urma corectitudinii răspunsurilor acestuia, primește un feedback. Acest feedback este, de asemnea, salvat în baza de date SQLite. În momentul în care terapeutul/specialistul/utilizatorul marchează o întrebare cu feedback-ul pozitiv, aceasta este marcată cu verificat pe tot parcursul ședinței de terapie pentru ca specialistul să poată avea evidența întrebărilor.

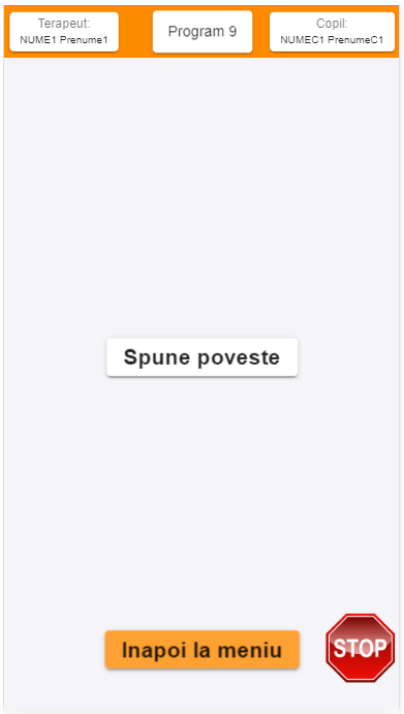


Figura 5.23: Pagină de redare a poveștii, Program 9 – aplicație Nao Terapia

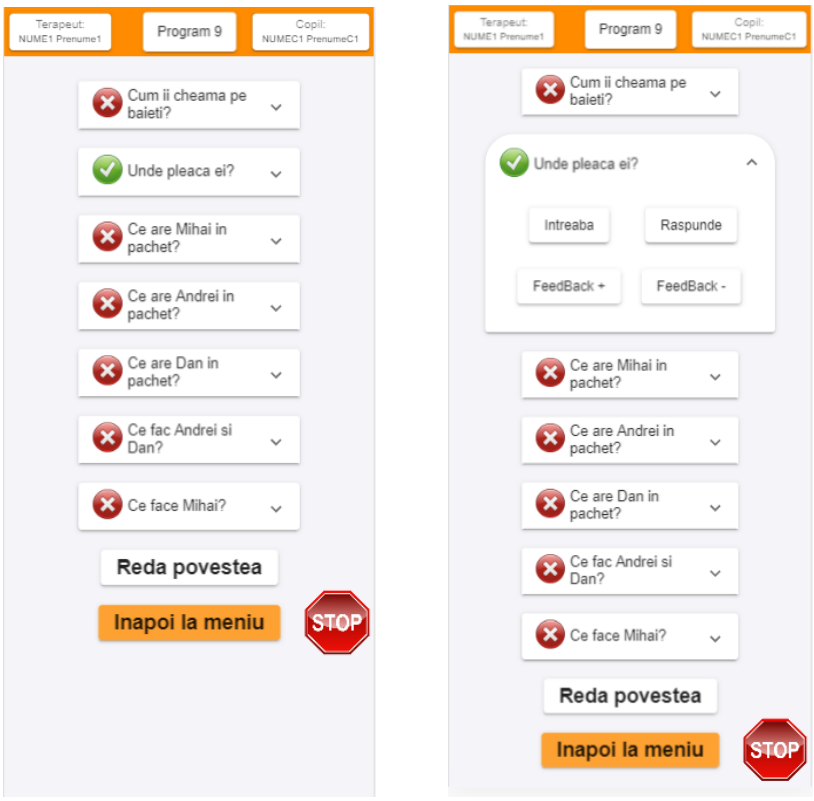
Asemănător claselor Program și Persoana, am creat o clasă Question pentru ca ulterior să creez o listă a întrebărilor ce trebuie afișate în această pagină. După cum se poate observa în figură (Figura 5.24), această clasă are următoarele variabile: id, question_code, question_detail, answer_code, answer_detail, checked.

```
export class Question {
    id: number;
    question_code: string;
    question_detail: string;
    answer_code: string;
    answer_detail: string;
    checked: string;
}
```

Figura 5.24: Clasa Question, Program 9 – aplicație Nao Terapia

Programul 9 – pagină de întrebări

Robotul redă fișierul audio ce conține povestea, urmând ca utilizatorul să poată pune întrebări pacientului pe baza acesteia. După ce întrebarea este selectată prin click, utilizatorului îi sunt interfațate 4 posibilități, ilustrate în figura următoare (Figura 5.25(b)).



(a) **(b)**

Figura 5.25: (a) Întrebări poveste, Program 9 – aplicație Nao Terapia. **(b)** Meniul posibilităților de răspuns, Program 9 – aplicație Nao Terapia.

Codul sursă HTML al unui panou de expansiune este prezent în Anexa 2.<*mat-expansion-panel-header*> prezintă un rezumat al conținutului panoului și acționează ca un control pentru extinderea sau prăbușirea (expending, collapsing) acestuia. Acest antet conține un <*mat-panel-title*> care formatează conținutul antetului pentru alinierea la specificațiile Material Design.

Am realizat un bloc de cod HTML care definește modul în care trebuie afișat un singur element, urmând a fi utilizat ca template pentru redarea fiecărui element din lista de întrebări. Elementul *ngFor din tag-ul <*mat-expansion-panel*> este directiva Angular "repeater". Acesta marchează elementul <*mat-expansion-panel*> ca "Template de repetor". Instrucțiunea ngFor "let Question of question" este o microsintaxă – un limbaj pe care Angular îl interpretează. Acest șir de caractere are următoare semnificație: ia fiecare element de tip Question din lista question, îl salvează în variabila buclă de element și este pus la dispoziția template-ului HTML la fiecare iterație. Astfel, sunt afiște în mod repetitiv întrebările din lista *questions* de tip *Question* create în componenta *intrebari.component.ts* (Anexa 1). De asemenea, pentru fiecare element din listă este afișat question_detail, variabilă în care este stocată întrebarea efectivă.

Evenimentul *click* asociat conduce la metoda onSelect() care primește ca parametru un obiect de tip Question. Această metodă salvează într-o variabilă locală instanța de tip Question. Prin selectarea unei întrebări apare un meniu, după cum este ilustrat în Figura 5.28(b), în care utilizatorul are 4 posibilități: să redea întrebarea, să redea răspunsul întrebării, să acorde FeedBack +, sau să acorde FeedBack -. Aceste 4 posibilități sunt ilustrate prin intermediul a 4 butoane ce acționează metode specifice din componenta *intrebari.component.ts*.

mat-grid-list este o afișare bidimensională a unei liste care aranjează celulele pe baza unei grile. Acesta trebuie să specifice numărul de coloane în grilă prin atributul *cols.* Numărul de rânduri este determinat automat bazându-se pe numarul de coloane și numarul de itemi.

```
<mat-grid-list cols="2" rowHeight="2:1">
```

Fiecărei plăci (tile) îi sunt asociate butoane cu evenimente specifice. De exemplu, atunci când utilizatorul apasă pe butonul Intreaba este apelată metoda OnSend() (Figura 5.26), având ca parametru codul întrebării selectate. Metoda OnSend() formează URL-ul transmis serverului HTTP în variabila locală toSend prin concatenarea codului primit ca parametru cu alte date necesare. Dacă a fost selectată prima întrebare și apăsat butonul Intreaba, URL-ul transmis severului este următorul: */api/v1/program9/?action=question1*.

```
onSend(action:string){
  if (this.isDisabled == false){
  this.isDisabled = true;
  let toSend: string = '/api/v1/program9';
  toSend = toSend.concat('/?action=');
  toSend = toSend.concat(action);
  if (action.includes('answer')){
    this.responses = this.responses +1;
  }
  this.http.post(toSend,0)
  .subscribe((data) => {this.isDisabled = false });
  }
}
```

Figura 5.26: Metoda toSend() a componentei intrebari.component.ts– aplicație Nao Terapia

Navigarea și rutarea în aplicație

Angular are un modul specific dedicat pentru navigare și rutare, *RouterModule*. Cu acest modul se pot crea rute, care permit deplasarea dintr-o parte a aplicației într-o altă parte sau dintr-o vizualizare în alta.

Pentru ca rutele să funcționeze, este nevoie de o ancoră sau un element din interfața utilizatorului pentru a mapa acțiunea (de obicei, click-uri pe elemente) pe rute (URL). Se utilizează directiva *routerLink* pentru acest scop. De

exemplu, atunci când utilizatorul face click pe numele unui program din interfața de utilizare, Angular, prin directiva routerLink, va naviga automat către URL-ul următor:

http://localhost:8000/#/program1_8/serieProg1

Pentru a extinde aplicația și a vizualiza pagini separate, cu URL-uri proprii sunt făcute următoarele modificări. Pentru realizarea acestui scop am folosit *router-ul* Angular. Routerul Agular permite utilizatorului să vizualizeze diferite componente și date în funcție de locul în care acesta se poziționează în aplicație. Router-ul permite navigarea de la o vizualizare la alta, pe măsură ce utilizatorii execută sarcini de aplicație: accesarea unui URL într-un browser, click pe un link al unei pagini, sau click pe butoanele înainte și înapoi ale browserului.

O aplicație Angular rutată are o singură instanță a serviciului Router. Când se modifică adresa URL a browserului, routerul caută o rută corespunzătoare de la care poate determina componenta de afișat. Pentru a putea configura ruta Routes în RouterModule trebuie importate simbolurile Routes și RouterModule. Routes indică rutei ce vizualizări trebuie să afișeze atunci când un utilizator dă click pe un link sau scrie un URL în bara de adrese a unui browser.

O rută tipică Angular are două proprietăți:

1. path: un șir care se potrivește cu adresa URL din bara de adrese a browserului.
2. component: componenta pe care ruta ar trebui să o creeze cand navighează către acea rută.

De exemplu, pentru a naviga la componenta *Program9Componet* atunci când URL-ul este: localhost:8000/program9 trebuie importată componenta Program9Component pentru a putea face referință într-o rută. După aceea sunt definite o serie de rute cu un singur traseu, sau mai multe trasee către acea componentă.

```
{
  path: 'program9',
  component: Program9Component,
  children: [
    { path: 'NewPoveste', component: NewPovesteComponent },
    { path: 'Intrebari', component: IntrebariComponent }
  ]
}
];
```

Figura 5.27: Rute definite în app-routing.module.ts pentru Programul 9 – aplicație Nao Terapia

Următorul pas este definirea unui link utilizând directiva Routerlink. *routerLink* definește cum navighează utilizatorul către rută (sau URL) declarată în template-ul componentei. Se dorește ca atunci când utilizatorul dă click pe butonul denumit *Next,* ce apare după selectarea unui program, să acceseze următoarea pagină. În cazul programului 9, următoarea pagină/vizualizare este cea în care se poate apăsa un buton pentru ca robotul să redea o poveste. În figura 5.27 este ilustrată modalitatea în care a fost declarată ruta pentru redarea poveștii din cadrul programului 9.

5.5 Dezvoltare bază de date SQLite

S-a dorit dezvoltarea unei baze de date în care pot fi stocate informații referitoare la evoluția pacienților, informații relevante ce pot fi studiate de către specialiștii în acest domeniu. O bază de date realțională (Relational database) este formată din una sau mai multe tabele ce conțin informații. Rândurile din tabel se numesc înregistrări (records), iar coloanele din tabel sunt numite câmpuri sau atribute(fields/ attributes). O bază de date care conține două sau

mai multe tabele conexe se numește o bază de date relațională, adică date interdependente. SQLite poate fi integrat cu limbajul Python folosind un modul Python numit sqlite3.

Conectarea la baza de date

Pentru a utiliza modulul SQLite3, am adăugat o declarație de import a scriptului python: *import sqlite3*. Atunci când te conectezi la un fișier de bază de date SQLite ce nu există, SQLite crează o nouă bază de date pentru tine. Pentru a crea baza de date am realizat un obiect de conexiune (Connection) care reprezintă fișierul în care vor fi stocate informațiile, utilizând funcția connect() a modulului sqlite3. Primul pas este reprezentat de crearea unei funcții ce se conectează la o bază de date SQLite specificată de către db_path care face referire la terapieDB.db.

```
def __init__(self):
self.db_path = 'terapieDB.db'
conn = self.sqlite3.connect(self.db_path)
table_list = conn.execute("SELECT name FROM sqlite_master")
if table_list.fetchall() == []:
       self.createTables(conn)
 conn.close()
```

Codul Python de mai sus permite conectarea la o bază de date existentă folosind conexiunea obiect *conn*. În cazul în care baza de date nu există, atunci acesta va fi creată și, în final, un obiect de bază de date va fi returnat. Un obiect de tip cursor este interfața cu baza de date, care permite rularea oricărei interogări SQL. Metoda execute() execută un query SQL, la finalul căruia se fomează o listă cu toate elementele de tip name din sqlite_master. Metoda fetchall() captează toate rândurile din setul de interogări și returnează o listă de tuples. Dacă nu mai sunt disponibile rânduri, acesta returnează o listă goală. Ca urmare, scriptul de mai sus, apelează metoda de creare a tabelelor dacă nu există rânduri în baza de date/dacă baza de date este goală. Odată terminate operațiile în fișierul bazei de date, trebuie închisă conexiunea prin metoda .close (). Codul Python sursă se regăsește în Anexa 3.

Crearea tabelelor

Baza de date construită conține 12 tabele (Figura 5.28). Un tablel destinat terapeuților, în care sunt stocate informații despre numele și prenumele acestora. Un tabel destinat pacienților în care, de asemenea, sunt salvate aceleași informații. 9 tabele destinate celor 9 programe disponibile în aplicația Nao Terapia. Ultimul tabel, conține date despre sesiunile de terapie, corelând informații despre terapeut și pacient. Metoda createTables() crează aceste tabele dacă baza de date este goală urmând schematica expusă în cele ce urmează.

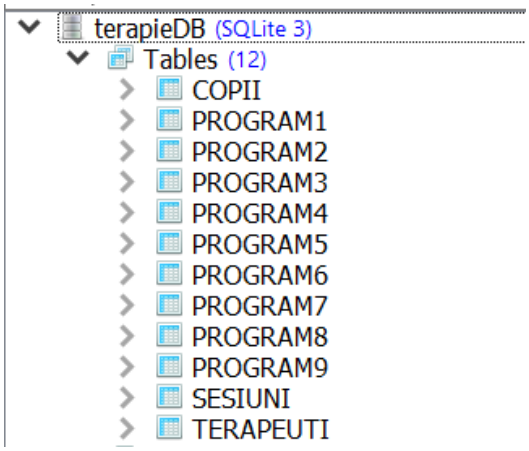


Figura 5.28: Tabele bază de date SQLite – aplicație Nao Terapia

Rutina conn.execute execută instrucțiunile SQL. În figura următoare am creat un tabel numit terapeuți, având următoarele câmpuri/coloane: ID, NUME, PRENUME. De asemenea, este desemnată ca și cheie primară cu incrementare automata câmpul ID, vizibil în Figura 5.29. În mod asemănător sunt realizate și celelalte tabele ce se regăsesc în aplicația Nao Terapia.

```
def createTables(self,conn):
    # Creare tabel pentru terapeuti
    conn.execute('''CREATE TABLE TERAPEUTI
            (ID INTEGER PRIMARY KEY AUTOINCREMENT,
            NUME           TEXT    NOT NULL,
            PRENUME            TEXT     NOT NULL
                );''')
    print "Table TERAPEUTI created successfully";
```

Figura 5.29: Creare tabel pentru terapeuți – aplicație Nao Terapia

JSON este abrevierea pentru "JavaScript Object Notation", fiind o modalitate de a stoca informația într-o formă organizată, având o manieră ușoară de accesat. Pe scurt, oferă o colecție de date ce poate fi citită de om, putând fi accesată printr-o metodă logică. Pentru a înregistra informații în baza de date am creat metoda addData() (Figura 5.30) ce primește ca parametru, jsonData. Aceasta conține informații de tip JSON. Astfel, după ce este realizată conectarea la baza de date se extrag informațiile necesare, precum: numele tabelei în care vor fi adăugate datele și variabila *data* ce conține un dicționar de cuvinte de tipul Dictionary(key: value) transmis de către client. Pentru a accesa informația dintr-un JSON, ne putem referi la numele proprietății de care avem nevoie. De exemplu, pentru a accesa informațiile legate de numele tabelei, am folosit următoarea sintaxă: nume_tabel = jsonData['table_name'].

```
def addData(self,jsonData):
    conn = self.sqlite3.connect(self.db_path)
    nume_tabel = jsonData['table_name']
    data = jsonData['data']
    column_data = str()
    toAdd_data = str()
    for field in data:
        column_data = column_data + str(field) + ","
        toAdd_data = toAdd_data + "'" + str(data[field])+"',"
    conn.execute("INSERT INTO "+nume_tabel+" ("+column_data[:-1]+") \
        VALUES ("+toAdd_data[:-1]+")");
    conn.commit()
    print "Records created successfully";
    conn.close()
```

Figura 5.30: Metodă de adăugare a datelor într-un tabel din baza de date – aplicație Nao Terapia

Metoda str() convertește informațiile dintr-o variabilă într-un string. Variabilele colum_data, toAdd_data sunt variabile de tip string. Pentru fiecare câmp(field) al dicționarului *data* se execută un query ce este creat în mod iterativ. Astfel, sunt introduse toate datele în tabelele bazei de date construită anterior. Metoda commit() trimite o declarație COMMIT către serverul MySQL, efectuând tranzacția curentă. Deoarece Connector/Python nu realizează autoefectuarea (autocmmit), este foarte important ca această metodă să fie apelată după fiecare tranzacție ce modifică tabelele sau orice tranzacție care utilizează motoarele de stocare tranzacționale. La finalul procesului este apelată metoda close() ce închide conexiunea cu serverul MySQL.

Pentru a obține informații din tabelele bazei de date am creat metoda getData() (Anexa 3), având ca parametru o variabilă de tip JSON. Se realizează conectarea la baza de date și extragerea informațiilor necesare pentru identificarea datelor cerute de către client. Pentru fiecare field al dicționarului *data* se crează o variabilă de tip string în care se concatenează toate câmpurile ce se regăsesc în *data*. Urmează să se execute un query de tipul: `SELECT colum_data from nume_tabelă`.

Scopul funcției zip () este de a mapa indecsii similari ai mai multor containere astfel încât acestea să poată fi folosite ca entitate unică. Zip(data.keys(), row) - returnează toate elementele key a dicționarului data și le asociează cu elementele din lista row cu ajutorul metodei zip(), în funcție de indexul acestora. Astfel, se crează un nou dicționar tmp_element, folosind metoda dict(), ce este transmis mai departe. Folosind metoda append() adăugăm elementele la sfârșitul listei dataToSend. Metoda json.dumps() convertește obiectele Python în stringuri JSON. Astfel, metoda returnează un string de tip JSON cu toate informațiile ce s-au dorit a fi primite.

Un exemplu al datelor ce sunt salvate în tabela Program 9 a bazei de date este prezentat în figura următoare (Figura 5.31). ID_SESIUNE este numărul sesiunii de terapie. Tabela SESIUNE (Figura 5.32) conține, de asemenea, ID-ul sesiunii, numele și prenumele terapeutului, dar și al copilului.

Structure | Data | Constraints | Indexes | Triggers

Grid view | Form view

	ID_SESIUNE	INTREBARE	RASPUNS	FEEDBACK
1	5	1	0	1
2	5	1	0	1
3	6	0	0	1
4	7	2	0	1
5	7	1	0	1
6	7	0	0	1
7	7	0	1	1
8	7	3	0	1
9	7	4	0	1
10	7	4	1	1
11	7	5	0	1
12	7	6	0	1
13	9	1	0	1

Figura 5.31: Exemplu de date stocate în tabela Program9 – aplicație Nao Terapia

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

Filter d...

	ID	NUME_TERAPEUT	PRENUME_TERAPEUT	NUME_COPIL	PRENUME_COPIL
1	9	Nume1	Prenume1	NumeC1	PrenumeC1
2	10	Nume3	Prenume3	NumeC2	PrenumeC2
3	11	Nume4	Prenume4	NumeC4	PrenumeC4
4	12	Nume5	Prenume6	NumeC1	PrenumeC1

Figura 5.32: Exemplu de date stocate în tabela Sesiune – aplicație Nao terapia

Concluzii

Înțelegerea inteligenței umane a reprezentat un interes major în rândul psihologilor de-a lungul secolelor. Inteligența este, în general, capacitatea de a învăța prin dobândirea și aplicarea cunoștințelor. Până în prezent, scorurile coeficientului de inteligență (IQ) au devenit un instrument important în selectarea și plasarea persoanelor și grupurilor în arena psihoeducațională. Pentru copiii cu tulburare de specru autist, s-a constatat de multă vreme că până la 70% dintre persoanele afectate de această afecțiune au o dizabilitate intelectuală definită printr-un IQ mai mic de 70. Cu toate acestea, deficientele în inteligență și comportament nu trebuie văzute ca un obstacol pentru ca acești copii să continue viața într-o manieră normală. Pentru a fi diagnosticat cu autism, un copil trebuie să îndeplinească toate cele trei criterii: afectarea calitativă a interacțiunii sociale, comunicarea și restricționarea tiparelor stereotipice de comportament. [18]

Principalul obiectiv al acestei lucrări a fost construirea unei aplicații prin care utilizatorul, specialistul sau terapeutul să poată interacționa cu pacientul prin intermediul robotului umanoid NAO. Această aplicație reușește să interfațeze 9 programe dezvoltate special pentru terapia copiilor cu tulburări de spectru autist și să faciliteze interacțiunea cu pacientul prin diverse mișcări concepute special pentru a realiza o conexiune aparte cu acesta.

Pe parcursul realizării tezei, am înțeles aspectele fundamentale ale unuia dintre cele mai pupulare, utilizate și răspândite framework-uri pe partea clientului (client-side), denumit Angular. Am ales să utilizez această platformă în dezvoltarea web pentru că este un cadru JavaScript robust care a fost dezvoltat și proiectat pentru a simplifica dezvoltarea front-end. Dezvoltarea Angular realizează sincronizarea extrem de eficientă a datelor, în mod automat. Alte motive pentru care am ales să folosesc acest framework în derimentul altuia au fost: arhitectura elegantă, documentația extinsă care permite o căutare rapidă a tuturor informațiilor și îmbunătățirile constante.

Platforma inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist a folosit serviciu REST. Așa cum a fost menționat, serviciul REST prezintă atât avantaje, cât și dezavantaje. Ușurința și simplitatea de implementare se numeră printre cele mai importante avantaje ale acestui serviciu. Un alt avantaj important este acela că nu este dependent de limbajul de programare și de platformă. Astfel un client Python poate apela un server Java sau invers. Un dezavantaj major al serviciului REST este acela ca se folosesc cookie-uri, însă există posibilitatea de a folosi localStorage. LocalStorage este o modalitate de a salva date pe mașina utilizatorului, permițând salvarea de perechi key/value. Cookie-uile sunt păstrate în text clar și pot însemna un risc de securitate din moment ce oricine ar putea deschide și manipula un cookie.

În ceea ce privește stocarea fișierelor web și transmiterea acestora local am dezvoltat un server web în Python. Distribuția implicită Python are suport încorporat pentru protocolul HTTP pe care l-am putut utiliza pentru a crea un server web autonom. Am început dezvoltarea prin exemple simple de tipul “Hello World”, continuând să descopăr fundamente noi ce m-au ajutat să construiesc serverul din aplicația Nao Terapia. În ceea ce privește baza de date, consider absolut necesar ca versiunile viitoare ale platformei să conțină informații cu o structură mai relevantă pentru a servi mai bine specialiștilor ce vor analiza rezultate obținute în urma ședințelor de terapie. Această parte a platformei, a aplicației Nao Terapia, necesită îmbunătățiri ce vor fi aduse la îndrumarea specialiștilor.

Vechea versiune a platformei inteligente pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist, sau aplicația Nao Terapia, este prezentă pe roboții umanoizi NAO ce se regăsesc în unele spitale din București, datorită echipei de la Laboratorul de cercetare Speed (Speech and Dialogue). Colaboratori: Clinică privată de terapie din București, Școala gimnazială specială „Constantin Păunescu” și Spitalul Clinic de Psihiatrie „Prof. Dr. Al. Obregia”. Terapeuții/Specialiștii utilizează acestă aplicație în programele de terapie a copiilor cu

deficiențe de spectru autist, urmând ca rezulatele obținute să fie recunoscute. Următorul pas în dezvoltarea acestui proces este încărcarea și utilizarea noii aplicații ce promite a fi scalabilă și fiabilă în comparație cu cea veche. Noua Aplicația Nao Terapia reușește să integreze în structura sa diferite module deja existente pe robotul umanoid NAO cu cele noi aduse în structura acestuia.

Cercetările anterioare au presupus că aspectul uman al lui NAO, capacitatea lui de a clipi, de a vorbi și de a dansa, împreună cu simplitatea programelor vor putea atrage interesul copiilor de a se angaja în comunicare. Copiii cu deficiențe de spectru autist răspund mult mai bine mașinilor (cum ar fi computerele sau roboții) decât oamenilor obișnuiți. Roboții respectă necesitatea unor acțiuni repetate și a unui mediu stabil în timpul ședințelor de terapie a copiilor.

Contribuțiile personale:

- Conceperea aplicației software pentru interfața grafică;
- Implementarea unui serviciu REST cu rolul de a interfața utilizatorul;
- Dezvoltarea unui server HTTP;
- Crearea unei baze de date SQLite.

Consider că lucrarea de față și-a atins scopul propus, acela de a realiza o platformă inteligentă pentru sisteme embedded cu aplicații în terapia copiilor cu tulburări de spectru autist. De asemenea, aplicația dezvoltată reprezintă un instrument util în terapia copiilor, primele rezultate fiind promițătoare (conferința ARPCAPA 2018), iar datorită modificărilor aduse aplicației, aceste rezultate vor fi din ce in ce mai satisfăcătoare. Structura framework-ului Angular a condus la o platformă flexibilă, ce răspunde la cerințele specialiștilor. De asemenea, pentru un management al evoluției progamului de terapie dezvoltat cu ajutorul robotului NAO, informațiile relevante pentru terapeuti sunt salvate.

Soluția propusă de autor este realizabilă practic, după cum se poate dovedi în aplicația practică, ce a fost realizată în scopul înțelegerii modului în care aplicația interacționează cu robotul umanoid NAO. Pentru realizarea unor concepte software mai complexe, sistemul descris în această teză poate veni ca un punct de plecare. Pe viitor, atât specialiștii cât și dezvoltatorii acestui program și-au propus ridicarea gradului de autonomie al robotului prin introducerea inteligenței artificiale. Se dorește ca robotul să fie capabil să recunoască pacientul și terapeutul, să rescunoască cuvintele rostite de copil pentru a-i putea acorda într-o manieră autonomă un feedback. De asemenea, următorul pas ar fi dezvoltarea unei interfețe intuitive prin intermediul căreia terapeutul să-și poată construi programele de terapie.

Bibliografie

[1] *NAOqi Documentation*, http://doc.aldebaran.com/2-1/, accesat la data: 17.05.2019

[2] *Softbank Robotics Domentation NAO– Developer Guide*, http://doc.aldebaran.com/2-5/family/index.html, accesat la data: 19.05.2019

[3] *Web Development*, https://www.fullstackpython.com/web-development.html, accesat la data: 19.05.2019

[4] *About JavaScript*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, accesat la data: 19.05.2019

[5] *About HTML*, http://www.math.md/studlib/informatica/editii_html/introducere.pdf, accesat la data: 19.05.2019

[6] *HTML Introduction*, https://www.w3schools.com/html/html_intro.asp, accesat la data: 19.05.2019

[7] *CSS (Cascading Style Sheets)*, https://www.theserverside.com/definition/cascading-style-sheet-CSS, accesat la data: 19.05.2019

[8] *Introducing JSON*, https://www.json.org/, accesat la data: 20.05.2019

[9] *Hypertext, URL, Domain, Name*, https://www.vskills.in/certification/tutorial/category/dtp/page/4/, accesat la data: 20.05.2019

[10] *URL address: Definition, Structure and Examples*, https://sitechecker.pro/what-is-url/, accesat la data: 20.05.2019

[11] *RESTful Web Services*, https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html, accesat la data: 21.05.2019

[12] *Introduction to the Angular Docs*, https://angular.io/docs, accesat la data: 21.05.2019

[13] *TypeScript Documentation*, https://www.typescriptlang.org/docs/home.html, accesat la data: 21.05.2019

[14] *A Byte of Python*, https://python.swaroopch.com/about_python.html, accesat la data: 24.05.2019

[15] *About PyCharm*, https://www.componentsource.com/product/pycharm/about, accesat la data: 25.05.2019

[16] *SQL Database (DB)*, https://searchsqlserver.techtarget.com/definition/database, accesat la data: 28.05.2019

[17] *What is SQLite*, http://www.sqlitetutorial.net/what-is-sqlite/, accesat la data: 28.05.2019

[18] T. Charman, A. Pickles, E. Simonoff, S. Chandler, T. Loucas, and G. Baird, *"IQ in children with autism spectrum disorders: data from the Special Needs and Autism Project (SNAP),"* Psychological Medicine, vol. 41(3), pp. 619-627

Anexa 1

Fișierul app.module.ts – Aplicație Nao Terapia

```
import 'hammerjs';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http'
import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { LoginModule } from './login/login.module';
import { StartComponent } from './start/start.component';
import { FormsModule } from '@angular/forms';
import {LocalStorageService} from './storage.service'
import {DataAccessService} from './data-access.service'
import { Program1_8Component } from './program1_8/program1_8.component';
import { NewSDComponent } from './program1_8/NewSD/NewSD.component';
import { SerieProg1Component } from './program1_8/serieProg1/serieProg1.component';
import { FeedbackComponent } from './feedback/feedback.component';
import { OldSDComponent } from './program1_8/OldSD/OldSD.component';
import { Program9Component } from './program9/program9.component';
import { NewPovesteComponent } from './program9/newpoveste/newpoveste.component';
import { IntrebariComponent } from './program9/intrebari/intrebari.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import {MatCardModule} from '@angular/material/card';
import { NavBarComponent } from './navBar/navBar.component';
import {MatToolbarModule} from '@angular/material/toolbar';
import { MeniuComponent } from './meniu/meniu.component';
import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';
import {MatExpansionModule} from '@angular/material/expansion';
import {MatButtonModule} from '@angular/material/button';
import {MatGridListModule} from '@angular/material/grid-list';
import { SettingsComponent } from './meniu/settings/settings.component';
import {MatSliderModule} from '@angular/material/slider';
@NgModule({
   declarations: [
      AppComponent,
      StartComponent,
      Program1_8Component,
      NewSDComponent,
      OldSDComponent,
      SerieProg1Component,
      FeedbackComponent,
      Program9Component,
      NewPovesteComponent,
      IntrebariComponent,
```

```
      NavBarComponent,
      MeniuComponent,
      SettingsComponent
   ],
   imports: [
      BrowserModule,
      AppRoutingModule,
      LoginModule,
      HttpClientModule,
      FormsModule,
      BrowserAnimationsModule,
      MatCardModule,
      MatToolbarModule,
      MatProgressSpinnerModule,
      MatExpansionModule,
      MatButtonModule,
      MatGridListModule,
      MatSliderModule
   ],
   exports: [
      NavBarComponent
   ],
   entryComponents: [
      SettingsComponent
  ],
   providers: [
      LocalStorageService,
      DataAccessService
   ],
   bootstrap: [
      AppComponent
   ]
})
export class AppModule { }
```

Fișierul start.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import { LocalStorageService } from './../storage.service';
@Component({
  selector: 'app-start',
  templateUrl: './start.component.html',
  styleUrls: ['./start.component.css']
})
export class StartComponent implements OnInit {
  title = 'Nao Terapia';
```

```
  constructor(private localStorageService: LocalStorageService) { }
  ngOnInit() {
  }
}
```

Fișierul terapeut.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import { Persoana } from '../persoana';
import { LocalStorageService } from '../../storage.service';
import { DataAccessService } from '../../data-access.service';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import {Router} from '@angular/router';
import {MatDialog, MatDialogRef, MAT_DIALOG_DATA} from '@angular/material';
import {NewTerapeutComponent} from './newTerapeut/newTerapeut.component';
@Component({
  selector: 'app-terapeut',
  templateUrl: './terapeut.component.html',
  styleUrls: ['./terapeut.component.css']
})
export class TerapeutComponent implements OnInit {
  isDisabled = false
  waiting = false;
  terapeuti: any
  adauga : any
  selectedPersoana : Persoana
  normal: any
// tslint:disable-next-line: max-line-length
  constructor(public dialog: MatDialog,private router: Router, private dataAccessService:
DataAccessService, private localStorageService: LocalStorageService,private http:HttpClient)
{
  }
  adaugaTerapeut(){
    this.isDisabled = true
    const dialogRef = this.dialog.open(NewTerapeutComponent, {
      width: '250px',
      data: {NUME: '', PRENUME : ''}
    });
    dialogRef.afterClosed().subscribe(result => {
      if (result != null){
        if (result.NUME != '' || result.PRENUME != ''){
      this.waiting = true;
      console.log(result);
      let url = '/api/v1/adugaTerapeut/?NUME=';
      url = url.concat(result.NUME);
      url = url.concat('&PRENUME=');
```

```
      url = url.concat(result.PRENUME);
      this.http.post(url,0).subscribe(res => {this.waiting = false; ; this.ngOnInit();});}
    }
    this.isDisabled = false;
    });
  }
  onSelect(persoana:Persoana){
    this.selectedPersoana=persoana;
this.localStorageService.storeOnLocalStorage('terapeut',JSON.stringify(this.selectedPersoana
))
  }
  ngOnInit() {
    this.isDisabled = true;
    this.waiting = true;
    this.http.get<Persoana[]>('/api/v1/getTerapeuti')
    .subscribe((data) => {this.terapeuti = data;this.isDisabled = false; this.waiting =
false;});
  }
}
```

Fișierul meniu.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import {Program} from './Program';
import {Router} from '@angular/router';
import { LocalStorageService } from '../storage.service';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Persoana } from '../login/persoana';
import {MatDialog, MatDialogRef, MAT_DIALOG_DATA} from '@angular/material';
import { SettingsComponent } from './settings/settings.component';
@Component({
  selector: 'app-meniu',
  templateUrl: './meniu.component.html',
  styleUrls: ['./meniu.component.css']
})
export class MeniuComponent implements OnInit {
  normal : any;
  programe=[
// tslint:disable-next-line: max-line-length
    {path : 'program1', nume : 'Program 1',descriere: 'Programul contine 6 serii de cuvinte
pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul repetă cuvintele
indicate de robotul NAO. La final, terapeutul îi poate acodra copilului/pacientului un
feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă
cuvântul.'},
// tslint:disable-next-line: max-line-length
```

```
    {path : 'program2', nume : 'Program 2',descriere: 'Programul contine o serie de cuvinte pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul repetă cuvintele indicate de robotul NAO. . La final, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă cuvântul.'},
    {path : 'program3', nume : 'Program 3',descriere: 'Programul contine un set de miscări pe care robotul NAO le poate executa într-o ordine aleatoare. Tipuri de mișcări: mâinile sus, mâinile pe cap, mâinile pe burtă, bate din palme, cucu-bau, bate palma, bea, manancă cu lingura, tropăie, sări. Copilul reproduce mișcările indicate de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă mișcarea.'},
    {path : 'program4', nume : 'Program 4',descriere: 'Programul contine un set de miscări pe care robotul NAO le poate executa într-o ordine aleatoare. Tipuri de mișcări: arată capul, burta, ochii, nasul, gura, piciorul urechea. Copilul reproduce mișcările indicate de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă mișcarea.'},
    {path : 'program5', nume : 'Program 5',descriere: 'Programul conține cerințe de tipul: "arată animalul" pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul arată animalul indicat de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă cerința.'},
    {path : 'program6', nume : 'Program 6',descriere: 'Programul conține cerințe de tipul: "arată fructul/leguma" pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul arată fructul/leguma indicat/ă de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă cerința.'},
    {path : 'program7', nume : 'Program 7',descriere: 'Programul conține cerințe de tipul: "arată obiectul" pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul arată obiectul indicat de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă cerința.'},
    {path : 'program8', nume : 'Program 8',descriere: 'Programul conține cerințe de tipul: "arată culoarea/forma geometrică" pe care robotul NAO le poate reproduce într-o ordine aleatoare. Copilul arată culoarea/forma geometrică indicată de robotul NAO. La finalul execuției, terapeutul îi poate acodra copilului/pacientului un feedback pozitiv sau negativ. În cazul în care feedback-ul este negativ, robotul NAO repetă cerința.'},
    {path : 'program9', nume : 'Program 9',descriere: 'Programul conține o poveste pe care robotul NAO o reproduce. După derularea poveștii se pot selecta întrebări la care copilul trebuie să răspundă. Pentru fiecare răspuns primit din partea copilului, terapeutul poate acorda un feedback pozitiv sau negativ.  În cazul în care feedback-ul este negativ, robotul NAO repetă întrebarea.'},
  ];
```

Constructor al componentei meniu.component.ts – aplicație Nao Terapia

```
 constructor(public dialog: MatDialog,private router: Router,private localStorageService:
LocalStorageService,private http:HttpClient) {}
  waiting: boolean;
  onSelect(program: Program){
    const newTodo = program.path;
    this.localStorageService.storeOnLocalStorage('program',newTodo);
    this.localStorageService.storeOnLocalStorage('program_name',program.nume);
    switch (newTodo) {
      case 'program9':{
        this.router.navigateByUrl('program9/NewPoveste');
        break;
      }
      case 'program1':{
        this.router.navigateByUrl('program1_8/serieProg1');
        break;
      }
      default:{
        this.router.navigateByUrl('program1_8/NewSD');
        break;
      }
    }
  }
  settingsFunction(){
    const dialogRef = this.dialog.open(SettingsComponent, {
      width: '250px',
    });
    dialogRef.afterClosed().subscribe(result => {})
  }
```

Metoda ngOnInit() al componentei meniu.component.ts – aplicație Nao Terapia

```
 ngOnInit() {
    this.localStorageService.storeOnLocalStorage('program_name','-');
    if (this.localStorageService.getFromLocalStorage('SESIUNE')==null){
    this.waiting=true;
    let copil:Persoana = JSON.parse(this.localStorageService.getFromLocalStorage('copil'));
    let terapeut:Persoana =
JSON.parse(this.localStorageService.getFromLocalStorage('terapeut'));
// tslint:disable-next-line: max-line-length
    let data = {'table_name': 'SESIUNI', 'data':
{'NUME_TERAPEUT':terapeut.NUME,'PRENUME_TERAPEUT':terapeut.PRENUME,
'NUME_COPIL':copil.NUME,'PRENUME_COPIL':copil.PRENUME}};
// tslint:disable-next-line: max-line-length
    this.http.post('/api/v1/adugaSesiune',data).subscribe((data) =>
{this.localStorageService.storeOnLocalStorage('SESIUNE',data); this.waiting=false;});
```

```
    }
  }
}
```

Fișierul serieProg1.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import { LocalStorageService } from '../../storage.service';
import {Router} from '@angular/router';
@Component({
  selector: 'app-serieProg1',
  templateUrl: './serieProg1.component.html',
  styleUrls: ['./serieProg1.component.css']
})
export class SerieProg1Component implements OnInit {
  serii = [
    {path : 'serie1', nume : 'Serie 1'},
    {path : 'serie2', nume : 'Serie 2'},
    {path : 'serie3', nume : 'Serie 3'},
    {path : 'serie4', nume : 'Serie 4'},
    {path : 'serie5', nume : 'Serie 5'},
    {path : 'serie6', nume : 'Serie 6'},
  ]
  constructor(private router: Router,private localStorageService: LocalStorageService) { }
  onSelect(serie: any){
    const newTodo = serie.path;
    this.localStorageService.storeOnLocalStorage('serie',newTodo);
    this.router.navigateByUrl('program1_8/NewSD');
  }
  ngOnInit() {
  }
}
```

Fișierul NewSD.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import {Router} from '@angular/router';
import { DataAccessService } from '../../data-access.service';
import { LocalStorageService } from '../../storage.service';
import { HttpClient, HttpHeaders } from '@angular/common/http';
@Component({
  selector: 'app-NewSD',
  templateUrl: './NewSD.component.html',
  styleUrls: ['./NewSD.component.css']
})
export class NewSDComponent implements OnInit {
  isDisabled = false
```

```
  constructor(private router: Router,private localStorageService:
LocalStorageService,private dataAccessService: DataAccessService,private http:HttpClient) {
}
  onSelect(){
    this.isDisabled = true;
    let toSend: string = this.localStorageService.getFromLocalStorage('program');
    let base: string = '/api/v1/'
    base = base.concat(toSend);
    if (base == '/api/v1/program1'){
      base =
base.concat('/?action=play&parameters=',this.localStorageService.getFromLocalStorage('serie'
));
    }else{
      base = base.concat('/?action=play');
    }
    this.http.post(base,0).subscribe(res => {this.isDisabled = false ;
this.router.navigateByUrl('program1_8/feedback')});
  }
  stop(){
    this.isDisabled = true;
    this.http.post('/api/v1/stop',0).subscribe(res => {this.isDisabled = false })
  }
  ngOnInit() {
  }
}
```

Fișierul intrebari.component.ts – Aplicație Nao Terapia

```
import { Component, OnInit } from '@angular/core';
import { DataAccessService } from '../../data-access.service';
import { LocalStorageService } from '../../storage.service';
import {Question} from './question'
import { HttpClient, HttpHeaders } from '@angular/common/http';
@Component({
  selector: 'app-intrebari',
  templateUrl: './intrebari.component.html',
  styleUrls: ['./intrebari.component.css']
})
export class IntrebariComponent implements OnInit {
  image_source: string = 'X.ico';
  isDisabled = false;
  nonselected : any;
  selectedQuestion : Question;
  responses : number;
```

// Lista de iterații a elementelor de tip Question – aplicație Nao Terapia

```
  questions: Question[]=[
    {id: 0, question_code : 'question1', question_detail : 'Cum ii cheama pe baieti?',
```

```
    answer_code: 'answer1',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 1,question_code : 'question2', question_detail : 'Unde pleaca ei?',
    answer_code: 'answer2',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 2,question_code : 'question3', question_detail : 'Ce are Mihai in pachet?',
    answer_code: 'answer3',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 3,question_code : 'question4', question_detail : 'Ce are Andrei in pachet?',
    answer_code: 'answer4',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 4,question_code : 'question5', question_detail : 'Ce are Dan in pachet?',
    answer_code: 'answer5',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 5,question_code : 'question6', question_detail : 'Ce fac Andrei si Dan?',
    answer_code: 'answer6',answer_detail: 'Raspuns', checked: 'X.ico'},
    {id: 6,question_code : 'question7', question_detail : 'Ce face Mihai?',
    answer_code: 'answer7',answer_detail: 'Raspuns', checked: 'X.ico'}
  ]
  constructor(private localStorageService: LocalStorageService,private dataAccessService:
DataAccessService,private http:HttpClient) { }
  onSelect(question : Question){
    this.selectedQuestion = question;
  }
  onFeedback(action:string,value:string){
    this.isDisabled = true;
    this.questions[this.selectedQuestion.id].checked=value;
    this.dataAccessService.postData(action);
    let Sesiune = this.localStorageService.getFromLocalStorage('SESIUNE');
    let data = {'table_name': 'program9', 'data':
{'ID_SESIUNE':Sesiune['ID'],'INTREBARE':this.selectedQuestion.id
,'RASPUNS':this.responses,'FEEDBACK':null}};
    if (value){
      data['data']['FEEDBACK']='1';
      this.http.post('/api/v1/adugaFeedbackP9',data)
        .subscribe((data) => {this.isDisabled = false});
    }else{
      data['data']['FEEDBACK']='0';
      this.http.post('/api/v1/adugaFeedbackP9',data)
        .subscribe((data) => {this.isDisabled = false });
    }
    this.responses = 0;
  }
  onSend(action:string){
    if (this.isDisabled == false){
    this.isDisabled = true;
    let toSend: string = '/api/v1/program9';
    toSend = toSend.concat('/?action=');
    toSend = toSend.concat(action);
    if (action.includes('answer')){
      this.responses = this.responses +1;
```

```
    }
    this.http.post(toSend,0)
    .subscribe((data) => {this.isDisabled = false });
    }
  }
  stop() {
    this.isDisabled = true;
    this.http.post('/api/v1/stop',0).subscribe(res => {this.isDisabled = false })
  }
  ngOnInit() {
    this.responses = 0;
  }
}
```

Anexa 2

Fișierul start.component.html – Aplicație Nao Terapia

```
 <a  style="display: flex;align-items: center;justify-content: center;height: 100%;width:
100%;">
    Bine ati venit pe {{ title }}!
  </a>
<a routerLink="login/terapeut" (click)="localStorageService.clearAllLocalStorage()"
 style="display: flex;align-items: center;justify-content: center;height: 100%;width:
100%;">
  <img height="300" src="naoterapia.png">
</a>
<h2 style="align-items: center"> Click pentru a incepe terapia </h2>
```

Fișierul terapeut.component.html – Aplicație Nao Terapia

```
<div>
<h2 class="mat-h2">Selectează un terapeut!</h2>
</div>
<div (click)="selectedPersoana=null">
<ng-template #normal>
    <span class="spinCont">
    <mat-spinner></mat-spinner>
    </span>
</ng-template>
<div *ngIf ="!waiting; else normal" >
  <ul class=terapeuti >

  <mat-card *ngFor="let Persoana of terapeuti"
    [class.selected]="Persoana === selectedPersoana"
    (click)="onSelect(Persoana)" (click)="$event.stopPropagation()">
```

```
    <span class="persoana" >
        <h2 class="mat-h2">{{Persoana.NUME | uppercase}} {{Persoana.PRENUME}}</h2>
    </span>
    </mat-card>
  </ul>
<button [disabled]="isDisabled" class = 'mat-fab' (click)="adaugaTerapeut()"><h1 class="mat-
h1">+</h1></button>
  <div *ngIf="selectedPersoana">
      <span class="contRis">
      <button class="mat-raised-button" (click)="router.navigateByUrl('login/copil')"><h2
class="mat-h2">Next</h2></button>
    </span>
    </div>
</div>
</div>
```

Fișierul meniu.component.html – Aplicație Nao Terapia

```
<app-navBar></app-navBar>
<ng-template #normal>
    <span class="spinCont">
        <mat-spinner></mat-spinner>
    </span>
</ng-template>
<div *ngIf ="!waiting; else normal" >
<mat-accordion>
        <mat-expansion-panel (opened)="panelOpenState = true"
                             (closed)="panelOpenState = false"
                             *ngFor="let Program of programe">
          <mat-expansion-panel-header>
            <mat-panel-title>
                <h2>{{Program.nume}}</h2>
            </mat-panel-title>
          </mat-expansion-panel-header>
          <mat-card>
                <mat-card-content>
                        <p>{{Program.descriere}}</p>
                </mat-card-content>
            </mat-card>
            <button mat-raised-button (click)="onSelect(Program)">Next</button>
        </mat-expansion-panel>
</mat-accordion>
</div>
<img  (click)="settingsFunction()" style='position: fixed;width: 50px;height:
50px;transform: translate3d(0,-22.5%,0);left: 5px;bottom: 5px;' src='settings.png'>
```

Fișierul intrebari.component.html – Aplicație Nao Terapia

```
<div>
<mat-accordion>
    <mat-expansion-panel (opened)="panelOpenState = true"
                         (closed)="panelOpenState = false"
                         *ngFor="let Question of questions" (click)="onSelect(Question)">
      <mat-expansion-panel-header>
        <mat-panel-title style="overflow: visible;">
          <span class="imgClass">
            <img style='max-width: 30px;' src={{Question.checked}}>
          </span>
          <span style="width: fit-content;height: fit-content;top: 50%;position:
relative;transform: translate3d(0,-50%,0);">
            {{Question.question_detail}}
          </span>
        </mat-panel-title>
      </mat-expansion-panel-header>
      <div *ngIf ="isDisabled ; else normal">
      <span class="spinCont">
          <mat-spinner></mat-spinner>
          </span>
      </div>
      <ng-template #normal>
      <mat-grid-list cols="2" rowHeight="2:1">
          <mat-grid-tile>
              <button mat-raised-button (click)="onSend(selectedQuestion.question_code)"
[disabled]="isDisabled">Intreaba</button>
          </mat-grid-tile>
          <mat-grid-tile>
              <button mat-raised-button (click)="onSend(selectedQuestion.answer_code)"
[disabled]="isDisabled">Raspunde</button>
          </mat-grid-tile>
          <mat-grid-tile>
              <button mat-raised-button
(click)="onFeedback('feedback/?action=positive','V.ico')" [disabled]="isDisabled" >FeedBack
+</button>
          </mat-grid-tile>
          <mat-grid-tile>
            <button mat-raised-button
(click)="onFeedback('feedback/?action=negative','X.ico')" [disabled]="isDisabled" >FeedBack
-</button>
          </mat-grid-tile>
        </mat-grid-list>
      </ng-template>
    </mat-expansion-panel>
```

```
    <p>
    <button mat-raised-button class="aliBlk" (click)="onSend('replay')" ><h2>Reda
povestea</h2></button>
    </p>
    <p>
    <button mat-raised-button class="aliBlk2" routerLink="/meniu"><h2>Inapoi la
meniu</h2></button>
    <img style='position: absolute;width: 65px;height: 65px;transform: translate3d(0,-
22.5%,0);right: 5px;' src='stop.png' (click)="stop();">
  </p>
</mat-accordion>
</div>
```

Fișierul terapeut.component.css – Aplicație Nao Terapia

```
  .terapeuti {
    margin: 0 auto;
    list-style-type: none;
    padding: 0;
    width: 15em;
  }
  .persoana {
    width: fit-content;
    vertical-align: middle;
    position: absolute;
    top: 50%;
    transform: translate3d(-50%,-50%,0);
  }
  .terapeuti mat-card {
    cursor: pointer;
    position: relative;
    left: 0;
    background-color: #EEE;
    margin: 1.5em;
    padding: .3em 0;
    height: 3.6em;
    border-radius: 4px;
    text-align: center;
  }
```

Anexa 3

Baza de date

```
class serverDB:
    import json
    import sqlite3
    db_path = str()
    def __init__(self):
        self.db_path = 'terapieDB.db'
        conn = self.sqlite3.connect(self.db_path)
        table_list = conn.execute("SELECT name FROM sqlite_master")
        if table_list.fetchall() == []:
            self.createTables(conn)
        conn.close()

    def createTables(self,conn):
        print "Opened database successfully";

        # Creare tabel pentru terapeuti
        conn.execute('''CREATE TABLE TERAPEUTI
                (ID INTEGER PRIMARY KEY AUTOINCREMENT,
                NUME           TEXT    NOT NULL,
                PRENUME            TEXT     NOT NULL
                    );''')
        print "Table TERAPEUTI created successfully";

        # Creare tabel pentru copii
        conn.execute('''CREATE TABLE COPII
                (ID INTEGER PRIMARY KEY AUTOINCREMENT,
                NUME           TEXT    NOT NULL,
                PRENUME            TEXT     NOT NULL
                    );''')
        print "Table COPII created successfully";

        # Creare tabel pentru sesiuni de terapie
        conn.execute('''CREATE TABLE SESIUNI
                (ID INTEGER PRIMARY KEY AUTOINCREMENT,
                NUME_TERAPEUT           TEXT    NOT NULL,
                PRENUME_TERAPEUT            TEXT     NOT NULL,
                NUME_COPIL          TEXT    NOT NULL,
                PRENUME_COPIL            TEXT     NOT NULL
                    );''')
        print "Table SESIUNI created successfully";

        # Creare tabele pentru program 1
```

```
    conn.execute('''CREATE TABLE PROGRAM1
            (ID_SESIUNE    INTEGER     NOT NULL,
            SERIE           INTEGER    NOT NULL,
            FEEDBACK           INTEGER    NOT NULL
                );''')
    print "Table PROGRAM1 created successfully";
    # Creare tabele pentru programe 2 - 8
    for index_program in range(2,9):
        conn.execute('''CREATE TABLE PROGRAM'''+str(index_program)+'''
                (ID_SESIUNE    INTEGER     NOT NULL,
                FEEDBACK           INTEGER    NOT NULL
                    );''')
        print "Table PROGRAM"+str(index_program)+" created successfully";

    # Creare tabele pentru program 9
    conn.execute('''CREATE TABLE PROGRAM9
            (ID_SESIUNE    INTEGER     NOT NULL,
            INTREBARE           INTEGER    NOT NULL,
            RASPUNS           INTEGER    NOT NULL,
            FEEDBACK           INTEGER    NOT NULL
                );''')
    print "Table PROGRAM9 created successfully";

def addData(self,jsonData):
    conn = self.sqlite3.connect(self.db_path)
    nume_tabel = jsonData['table_name']
    data = jsonData['data']
    column_data = str()
    toAdd_data = str()
    for field in data:
        column_data = column_data + str(field) + ","
        toAdd_data = toAdd_data + "'" + str(data[field])+"',"
        conn.execute("INSERT INTO "+nume_tabel+" ("+column_data[:-1]+") \
        VALUES ("+toAdd_data[:-1]+")");
    conn.commit()
    print "Records created successfully";
    conn.close()

def getData(self,jsonData):
    conn = self.sqlite3.connect(self.db_path)
    nume_tabel = jsonData['table_name']
    data = jsonData['data']
    column_data = str()
    for field in data:
        column_data = column_data + field + ","
    cursor = conn.execute("SELECT "+column_data[:-1]+" from "+nume_tabel)
```

```
        dataToSend = []
        for row in cursor:
            tmp_element = dict(zip(data.keys(), row))
            dataToSend.append(tmp_element)
        conn.close()
        return self.json.dumps(dataToSend)

    def getDataFiltered(self,jsonData):
        conn = self.sqlite3.connect(self.db_path)
        nume_tabel = jsonData['table_name']
        cursor = conn.execute("SELECT * FROM "+nume_tabel+" ORDER BY id DESC LIMIT 1")
        result = cursor.fetchone()
        dataToSend = result[0];
        conn.close()
        return self.json.dumps(dataToSend)
```

Anexa 4

Serverul HTTP

```
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from SocketServer import ThreadingMixIn
from serverDB_test import serverDB_test
import os
import threading
import argparse
import re
import cgi
import sys
import time
import random
import socket
import wave
import contextlib
import threading
from urlparse import urlparse
from program1 import program1
from MyClass import program2
from MyClass import program3
from MyClass import program4
from MyClass import program5
from MyClass import program6
from MyClass import program7
from MyClass import program8
from program9 import program9
```

```
from MyClass import feedback
fbk = feedback()
pr1 = program1()
pr2 = program2()
pr3 = program3()
pr4 = program4()
pr5 = program5()
pr6 = program6()
pr7 = program7()
pr8 = program8()
pr9 = program9()
class HTTPRequestHandler(BaseHTTPRequestHandler):
  interfaceDB = serverDB_test()
  def do_POST(self):
    import time
    if None != re.search('/api/v1/program1/', self.path):
      actionsPossible = {'play':pr1.play,'replay':pr2.replay}
      parametersPossible = {'serie1':1,'serie2':2,'serie3':3,'serie4':4,'serie5':5}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      parameter = query_components["parameters"]
      actionsPossible[action](parametersPossible[parameter])
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/program2/', self.path):
      actionsPossible = {'play':pr2.play,'replay':pr2.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/stop', self.path):
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/program3/', self.path):
      actionsPossible = {'play':pr3.play,'replay':pr3.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
```

```
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/program4/', self.path):
      actionsPossible = {'play':pr4.play,'replay':pr4.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return

    elif None != re.search('/api/v1/program5/', self.path):
      actionsPossible = {'play':pr5.play,'replay':pr5.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/program6/', self.path):
      actionsPossible = {'play':pr6.play,'replay':pr6.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/program7/', self.path):
      actionsPossible = {'play':pr7.play,'replay':pr7.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
```

```
      return
    elif None != re.search('/api/v1/program8/', self.path):
      actionsPossible = {'play':pr8.play,'replay':pr8.replay}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    if None != re.search('/api/v1/program9/', self.path):
      actionsPossible =
{'story':pr9.play,'replay':pr9.replay,'answer1':pr9.playAnswer,'answer2':pr9.playAnswer,'ans
wer3':pr9.playAnswer,'answer4':pr9.playAnswer,'answer5':pr9.playAnswer,'answer6':pr9.playAns
wer,'answer7':pr9.playAnswer,'question1':pr9.playQuestion,'question2':pr9.playQuestion,'ques
tion3':pr9.playQuestion,'question4':pr9.playQuestion,'question5':pr9.playQuestion,'question6
':pr9.playQuestion,'question7':pr9.playQuestion}
      paramsPossible =
{'story':[],'replay':[],'answer1':0,'answer2':1,'answer3':2,'answer4':3,'answer5':4,'answer6
':5,'answer7':6,'question1':0,'question2':1,'question3':2,'question4':3,'question5':4,'quest
ion6':5,'question7':6}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action](paramsPossible[action])
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/feedback/', self.path):
      actionsPossible = {'positive':fbk.positive,'negative':fbk.negative}
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      action = query_components["action"]
      actionsPossible[action]()
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('api/v1/adugaTerapeut/', self.path):
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      NUME = query_components["NUME"]
      PRENUME = query_components["PRENUME"]
```

```
      jsonData = '{"table_name": "TERAPEUTI", "data": {"NUME": "'+NUME+'",
"PRENUME":"'+PRENUME+'"}}'
      jsonData = self.interfaceDB.json.loads(jsonData)
      self.interfaceDB.addData(jsonData)
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('api/v1/adugaCopil/', self.path):
      query=urlparse(self.path).query
      query_components = dict(qc.split("=") for qc in query.split("&"))
      NUME = query_components["NUME"]
      PRENUME = query_components["PRENUME"]
      jsonData = '{"table_name": "COPII", "data": {"NUME": "'+NUME+'",
"PRENUME":"'+PRENUME+'"}}'
      jsonData = self.interfaceDB.json.loads(jsonData)
      self.interfaceDB.addData(jsonData)
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/adugaSesiune', self.path):
      content_len = int(self.headers.getheader('content-length', 0))
      post_body = self.rfile.read(content_len)
      jsonData = self.interfaceDB.json.loads(post_body)
      self.interfaceDB.addData(jsonData)
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      self.wfile.write('{"ID": '+self.interfaceDB.getDataFiltered(jsonData)+'}')
      return
    elif None != re.search('/api/v1/adugaFeedback', self.path):
      content_len = int(self.headers.getheader('content-length', 0))
      post_body = self.rfile.read(content_len)
      jsonData = self.interfaceDB.json.loads(post_body)
      self.interfaceDB.addData(jsonData)
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/adugaFeedbackP9', self.path):
      content_len = int(self.headers.getheader('content-length', 0))
      post_body = self.rfile.read(content_len)
      jsonData = self.interfaceDB.json.loads(post_body)
      self.interfaceDB.addData(jsonData)
      self.send_response(200)
```

```
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return
    elif None != re.search('/api/v1/setVolume', self.path):
      content_len = int(self.headers.getheader('content-length', 0))
      post_body = self.rfile.read(content_len)
      print(post_body)
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      return

  def do_GET(self):
    root = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'appTerapie')
    urlPath = self.path.split('?')

    if self.path == '/api/v1/getTerapeuti'
      jsonData = '{"table_name": "TERAPEUTI", "data": {"NUME": "", "PRENUME":""}}'
      jsonData = self.interfaceDB.json.loads(jsonData)
      self.send_response(200)
      self.send_header('Content-type', 'application/json')
      self.end_headers()
      self.wfile.write(self.interfaceDB.getData(jsonData))
      return
    elif self.path == '/api/v1/getCopii':
      jsonData = '{"table_name": "COPII", "data": {"NUME": "", "PRENUME":""}}'
      jsonData = self.interfaceDB.json.loads(jsonData)
      self.send_response(200)
      self.send_header('Content-type', 'application/json')
      self.end_headers()
      self.wfile.write(self.interfaceDB.getData(jsonData))
      return
    elif None != re.search('/api/v1/getVolume', self.path):
      self.send_response(200)
      self.send_header('Content-Type', 'application/json')
      self.end_headers()
      self.wfile.write('20')
      return
    elif self.path[0] == '#':
        self.send_response(200)
        self.send_header('Content-Type', 'text/html')
        self.end_headers()
        self.wfile.write('<!doctype html><html lang="en"><head>  <meta charset="utf-8">
<title>AppTerapie</title>  <base href="./">  <meta name="viewport" content="width=device-
```

```
width, initial-scale=1">  <link rel="icon" type="image/x-icon"
href="favicon.ico"></head><body>  <app-root></app-root></body></html>')
        #print "inGEt"
        return
    else:
      if self.path == '/':
        filename = root + '/index.html'
      else:
        filename = root + urlPath[0]
      self.send_response(200)
      if filename[-4:] == '.css':
        self.send_header('Content-type', 'text/css')
      elif filename[-5:] == '.json':
        self.send_header('Content-type', 'application/javascript')
      elif filename[-3:] == '.js':
        self.send_header('Content-type', 'application/javascript')
      elif filename[-4:] == '.ico':
        self.send_header('Content-type', 'image/x-icon')
      elif filename[-4:] == '.png':
        self.send_header('Content-type', 'image/png')
      else:
        self.send_header('Content-type', 'text/html')
      self.end_headers()
      with open(filename, 'rb') as fh:
        html = fh.read()
        #html = bytes(html, 'utf8')
        self.wfile.write(html)
      #print "inGEt"
    return
class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
  allow_reuse_address = True
  def shutdown(self):
    self.socket.close()
    HTTPServer.shutdown(self)
class SimpleHttpServer():
  def __init__(self, ip, port):
    self.server = ThreadedHTTPServer((ip,port), HTTPRequestHandler)
  def start(self):
    self.server_thread = threading.Thread(target=self.server.serve_forever)
    self.server_thread.daemon = True
    self.server_thread.start()
  def waitForThread(self):
    self.server_thread.join()
  def stop(self):
    self.server.shutdown()
    self.waitForThread()
```

```
if __name__=='__main__':
#  parser = argparse.ArgumentParser(description='HTTP Server')
#  parser.add_argument('port', type=int, help='Listening port for HTTP Server')
#  parser.add_argument('ip', help='HTTP Server IP')
#  args = parser.parse_args()
  httpd = HTTPServer(('0.0.0.0', 8000), HTTPRequestHandler)
  httpd.serve_forever()
  print('HTTP Server Running...........')
```