Universitatea POLITEHNICA București
Facultatea Automatică şi Calculatoare
Departamentul Automatică şi Informatică Industrială

LUCRARE DE LICENŢĂ

Aplicație de gestiune a programărilor din cadrul unui cabinet de medicină de familie

Coordonator
Prof. dr. ing. Daniela Saru

Absolvent
Arsene Ramona-Georgiana

2018

Cuprins

Rezumatul proiectului

Lucrarea de față, *Aplicație de gestiune a programărilor din cadrul unui cabinet de medicină de familie*, reprezintă o aplicație Web ce are ca scop automatizarea procesului de programare ce are loc într-un cabinet de medicină de familie.

În **capitolul 1** al acestei lucrări este prezentat motivul alegerii temei.

În **cel de-al doilea capitol** al lucrării sunt descrise pe scurt domeniul din care face parte lucrarea, precum și aplicațiile de acest tip deja existente pe piață.

Capitolul al treilea prezintă problema abordată în cadrul lucrării și modalitățile propuse pentru rezolvarea acesteia.

În cadrul **capitolului patru** sunt prezentate tehnologiile utilizate la dezvoltarea fiecarei componente a soluției software propuse.

Capitolul cinci prezintă modul de implementare a aplicației, precum și rolul fiecărui tip de utilizator în cadrul aplicației.

În **capitolul șase** se regăsesc concluziile și observațiile rezultate în urma realizării proiectului.

1. Introducere

În contextul actual, stabilirea unei programări la un cabinet de medicină de orice tip reprezintă un pas important în cadrul consultațiilor medicale. Procesul de gestiune a acestor programări din orice instituție medicală este, însă, unul dificil, ce presupune implicarea constantă a unui operator uman pentru a prelua apelurile telefonice ale pacienților și a nota programările stabilite în cadrul convorbirilor cu aceștia.

1.1. Justificarea alegeri temei

Tema aleasă a avut drept scop facilitarea activităților de gestionare a programărilor dintr-un cabinet de medicină de familie. Acest lucru este realizat prin implementarea unei soluții software a cărei principală funcție este automatizarea procesului de programare a pacienților. Aplicația rezultată are ca utilizatori țintă atât personalul cabinetului de medicină de familie cât și pacienții acestuia.

Implementarea soluției software a constituit un prilej de aprofundare a cunoștințelor cu privire la tehnologiile și conceptele utilizate în dezvoltarea unei aplicații Web. În cadrul proiectului, am avut în vedere utilizarea unor tehnologii ce oferă utilizatorului posibilitatea de a naviga rapid de la o resursă la alta, precum și o interfață grafică intuitivă și ușor de utilizat. Pentru atingerea aestui scop, am utilizat tehnologiile ASP .NET Web API și platforma Angular, versiunea 5.2, tehnologii de actualitate, ce reduc considerabil durata procesului de dezvoltare, asigurând totodată performanța aplicațiilor dezvoltate.

2. Prezentarea domeniului din care face parte lucrarea

Soluția dezvoltată în cadrul acestui proiect reprezintă o aplicație Web de tipul *e-Appointment* și are ca mediu de aplicabilitate domeniul medical. Acest tip de aplicație este utilizat frecvent și încorporat în aplicații medicale complexe ce aparțin domeniului de e-Health.

2.1. Prezentarea domeniului e-Health

Conceptul de *e-Health* a apărut la începutul anilor 2000 și a avut, de-a lungul timpului, o utilizare variată. În anul 2005, în urma unui studiu, s-au găsit aproximativ 50 de definiții ale acestui termen, fiind considerat fie ca facând referire la procesele electronice sau digitale utilizate în domeniul medical, fie ca având un sens mult mai restrâns, fiind folosit doar cu referire la aplicațiile medicale ce utilizează exclusiv Internetul. Un punct comun al tuturor acestor definiții sugerează faptul că noțiunea de *e-Health* este strâns legată de anumite ramuri ale sănătății ce pot fi tehnologizate, conform informațiilor prezentate în lucrarea [14].

Una din definițiile asociate acestui concept îl prezintă ca fiind un domeniu în plină dezvoltare, aflat la intersecția dintre sănătătea publică sau privată și tehnologia informației și care face referire la serviciile medicale și informațiile din sectorul medical ce sunt puse la dispoziție sau îmbunătățite prin intermediul Internetului, iar compania Intel face referire la *e-Health* ca fiind un efort concentrat al liderilor din dominiul sanitar și din industriile de înaltă tehnologie pentru a valorifica la maximum rezultatele apărute în urma convergenței celor două domenii, conform lucării [2].

Răspândirea rapidă a Internetului a adus noi oportunități și provocări pentru domeniul medical și, mai cu seamă, pentru industria Tehnologiei Informației utilizată în cadrul acestuia. Aceste provocări au constat în creșterea capacității clienților de a interacționa cu sistemele lor în mediul online, respectiv găsirea unor modalități imbunătite de a transmite datele de la o instituție la alta.

În ultimul deceniu, tehnologiile apărute și aplicațiile dezvoltate în cadrul acestui domeniu au avut un impact major în ceea ce privește extinderea accesului la serviciile medicale și îmbunătățirea calității acestora, întrucât au oferit pacienților posibilitatea de a se informa mult mai ușor și mai rapid cu privire la starea lor de sănătate și de a se implica

în imbunătățirea acesteia. Printre aplicațiile ce au făcut posibil acest lucru pot fi amintite dosarele electronice de sănătate, aplicațiile de monitorizare a pacienților care suferă de boli cronice precum diabetul sau platformele și serviciile de e-Mental Health ce au ca scop informarea și educarea cu privire la afecțiunile psihice, după cum este menționat în lucrările [4] și [9].

În domeniul medical local, se observă o creștere de la an la an a tendinței de informatizare a unor servicii medicale și facilitare a comunicării și interacțiunii dintre pacienți și personalul medical. Odată cu această creștere, apare și o necesitate de a informa și instrui atât personalul medical, cât si pacienții pentru a ține pasul cu noile funcționalități și a le integra în viața de zi cu zi.

Conform studiului prezentat în lucrarea [9], se deosebesc patru tipuri principale de aplicații ce țin de Tehnologia Informației și a Cercetării din domeniul sanitar:

1. Inițiative ce se adresează sectorului de sănătate și gestionarii bolilor;
2. Soluții software ce implică activități de sprijin al managementului, administrației, logisticii și al produselor specifice domeniului sanitar;
3. Inițiative și soluții de infrastructură a cunoștințelor din sănătate;
4. Inițative și soluții de infrastructură în IT.

Portalurile de programări online fac parte din ce-a de-a doua categorie, fiind cunoscute sub denumirea de e-Appointment.

2.2. Prezentarea conceptului de e-Appointment

Stabilirea unei programări reprezintă, într-un sens general, o întelegere intre două sau mai mai multe entități (persoane sau instituții) de a se întâlni la un moment de timp stabilit, în vederea unei consultații medicale. Stabilirea unei întâlniri între aceste două entități constă în găsirea unui moment de timp oportun pentru desfășurarea activității respective, moment care să respecte atât intervalele de timp libere din agendele medicilor cât și ale pacienților și, în final, rezervarea acestui interval de timp in scopul mentionat.

2.2.1. Conceptul de e-Appointment în domeniul medical

În domeniul medical, gestiunea programărilor este un proces cheie, cu un impact major asupra performanțelor clinicilor medicale. Acest proces este unul costisitor din punct de vedere al timpului, întrucât personalul ce se ocupă cu gestiunea acestui proces este nevoit să preia constant apeluri telefonice de la pacienții ce solicită programări. De asemenea, pot apărea întârzieri realtiv mari în programul medicilor, datorate erorii umane a operatorilor sau a deciziei pacienților de a anula programările într-un timp prea scurt, sau chiar de a nu se prezenta deloc la programarea respectivă.

Automatizarea acestui proces constituie o solutie ce reduce semnificativ timpul petrecut preluând apeluri telefonice de la pacienți în vederea stabilirii unei programări, premițând astfel maximizarea eficienței personalului cabinetului în rezolvarea sarcinilor din timpul consultațiilor propriu-zise.

Totodată, o aplicație de tipul e-Appointment permite pacienților înregistrarea unei programări la orice oră, fără a mai depinde de programului cabinetului. Programările sunt înregistrate automat, fără a mai fi nevoie de intervenția unui operator uman pentru confirmare.

Din ce în ce mai multe clinici și spitale ce adoptă soluții de e-Health au în vedere și implementarea propriului portal de gestiune a programărilor sau preferă să apeleze la soluții deja existente, pe care le adaptează serviciilor oferite, contra cost.

2.2.2. Soluții deja existente pe piață și modul în care acestea au fost receptate de către utilizatori

Un număr din ce în ce mai mare de clinici medicale au adoptat numeroase soluții pentru a îmbunătăți eficiența gestiunii programărilor și a reduce numărul de ore petrecute peste program, cât și timpul petrecut în cozile de așteptare de către pacienți. Dintre aceste soluții fac parte apelurile telefonice cu rol de memento, programarea în afara orelor de lucru, introducerea unei taxe pentru programările la care pacienții nu s-au prezentat sau instalarea de aplicații de tipul e-Appointment.

Un studiu realizat in Canada în anul 2014, regăsit în lucrarea [5], a avut ca scop principal examinarea modului în care este acceptat de către publicul țintă un sistem de

programări online, intitulat „Doctor Direct”, și frecvența cu care acesta este utilizat pe parcursul a doi ani.

Aplicația respectivă a fost dezvoltată utilizând tehnologia Microsoft .NET și a fost lansată în cadrul studiului cu scopul de a observa impactul produs asupra pacienților și a modului de gestiune a programărilor. Aplicația este constituită dintr-un portal Web securizat ce permite pacienților să adauge cereri de programare pentru oricare din medicii sau secțiile clinicii respective, să modifice aceste cereri sau să le anuleze. De asemenea, soluția software implementată poate trimite automat mesaje pacienților, fie prin email sau telefon, pentru a le reaminti, cu o perioadă de timp prestabilită înainte de data programării, de programarea făcută.

Conform studiului respectiv, din perspectiva personalului clinicilor, aplicația respectivă a permis ca înregistrarea programărilor să fie mai eficientă, mai sigură și mai rapidă, iar monitorizarea modificărilor produse asupra programărilor deja înregistrate în sistem sau a anulării acestora să se producă fără a afecta funcționarea cabinetului. Pentru pacienți, utlizarea acestei aplicație a rezultat în diminuarea timpului necesar stabilirii unei programări, permitând ca programările să fie înregistrate și în afara orelor de program ale clinicii.

Pe plan local, Sistemul Informatic Unic Integrat (SIUI) reprezintă unealta principală de informatizare a activităților din cadrul domeniului sanitar, facând posibilă evidența persoanelor asigurate și generalizarea sistemului de raportare a instituțiilor de sănătate publică către Casa Națională de Asigurări de Sănătate. Aceasta este utilizată în cadrul tuturor instituțiilor medicale de stat din toată țara pentru gestiunea pacienților, a istoricului medical al acestora, al medicamentelor aflate pe piață și a operațiunilor ce au loc în fiecare dintre aceste instituții.

Întrucât acest sistem nu conține și o soluție generală pentru gestionarea programarilor din cadrul instituțiilor de stat, s-au dezvoltat diverse soluții software de programare online a pacienților. Fiecare dintre aceste soluții a încercat sa acopere cât mai multe din nevoile și cererile instituțiilor medicale pentru care au fost proiectate sau să aducă îmbunătățiri funcționalităților deja existente în cadrul SIUI.

Un exemplu de astfel de aplicație o constituie e-plannerMed, o agendă software de programări on-line ce are ca instituții țintă clinicile medicale, cabinetele de medicină individuală, cabinete de stomatologie sau psihologie și care permite totodată gestiunea

stocurilor, evidența pacienților și a istoricului acestora și gestiunea încasărilor și a cheltuielilor interne și generarea de rapoarte, toate acestea prin intermediul Internetului.

Un alt portal de gestiune a programărilor, a evidenței pacienților și a serviciilor oferite, ce poate fi configurat pentru orice cabinet medical individual, este prezentat de site-ul Weblog, iar un scurt demo al aplicației se regăsește la adresa http://cabinet-medical.weblog.info.ro/demo/auth/login.

3. Descrierea problemei abordate şi a metodei de rezolvare propuse

3.1. Etapele procesului de programare

La prima vedere, implementarea unei aplicații de e-Appointment poate părea o doar simplă organizarea a informațiilor cu privire la momentele de timp libere din cadrul agendelor medicilor și potrivirea lor cu cererile utilizatorilor. În esență, însă, este necesară o conceptualizare pentru a rezolva interdependența dintre aspectele tehnice si organizatorice în cadrul instituțiilor medicale. Un prim pas in realizarea acesti lucru este separarea procesului în etapele constituente, aceasta necesitând cunoasterea etapelor necesare stabilirii unei programări.

În tabelul 3.1 se regăsește o descriere a acestor etape impreună cu exemple uzuale de moduri de programare în cele două cazuri – al utilizării unui sistem de programări online și programarea prin contactarea telefonică a personalului clinicii.

Etapă	**Descriere**	**Exemplu – fără utilizarea unui sistem de *e-Appointment***	**Exemplu – cu utilizarea unui sistem de *e-Appointment***
Înaintea programării	Informarea cu privire la serviciul medical necesar și secția care se poate ocupa de acest lucru; identificarea modului de obținere a unei programări la unul din medicii specialiști	Persoana care are nevoie de un serviciu medical va încerca să sune medicul de familie dacă problema de sănătate a apărut în timpul orelor de lucru ale acestuia; în caz contrar, va aștepta până a doua zi. Dacă problelma se agravează, pacientul va apela la serviciul de urgențe	Pacientul va căuta online programul unei clinici și va verifica orele de lucru cu pacienții/se va informa cu privire la problema sa de sănătate; dacă problema de sănătate nu a apărut în timpul orelor de lucru, el va solicita o

			programare pentru a doua zi și primește automat confirmarea pentru data respectivă.
Stabilirea programării	Stabilirea de comun acord a unui moment din zi oportun pentru o consultație	Este posibil ca persoana ce are nevoie de un serviciu medical să nu se poată prezenta la cabinet în timpul orelor de lucru cu pacienții. Personalul responsabil cu stabilirea programărilor va încerca să prezinte pacientului diverse ore, detaliind programul medicului.	Pacientul va putea vedea toate momentele din zi care sunt libere pentru o programare pe parcursul unui număr de zile prestabilit (o lună, o săptămână etc.) și va alege unul din aceste momente. Îi va fi alocată, pe loc, o durată a întâlnirii prestabilită.
După programare	Confirmarea, anularea, modificarea datelor sau a detaliilor cu privire la programrea înregistrată;	În programul pacientului a apărut o urgență și programarea va trebui amânată – va trebui să ia legătura cu personalul clinicii unde are programată consultația și să repete procesul de la prima etapă, justificâdu-și amânarea.	În programul pacientului a apărut o urgență și programarea va trebui amânată – el se poate loga în aplicație și modifica cererea înregistrată, cu cel mult 24 de ore înainte de ziua la

			care este programat.

Tabel 3.1. Fazele din cadrul procesului de stabilire a unei programări

În cadrul etapei de negociere se pot identifica două modalități de stabilire a programării:

— în timpul orelor de consult, caz în care furnizorul de servicii medicale publică datele și intervalele orare pe care le are libere pentru fiecare zi, precum și programul cabinetului, urmând ca utilizatorul să aleagă unul din aceste intervale pentru a înregistra programarea;

— consultatie la cerere, caz în care utilizatorul trmite o cerere de programare către cabinet, în care precizează intervalele orare și zilele în care ar dori să aibă loc consultația, urmând ca medicul să aleagă o dată și o oră, ținând cont de intervalele solicitate de pacient.

3.2. Procesul de programare din cadrul cabinetului de medicină de familie

În cadrul cabinetelor de medicină de familie, cea mai des întâlnită modalitate de programare este cea pe bază pe apeluri telefonice, fiecare medic de familie având în subordinea sa o asistentă care preia aceste apeluri și notează în agendă fiecare programare.

Pacienții care cunosc programul cabinetului respecitv pot propune o zi sau o oră pentru o viitoare programare, iar în cazul în care acest interval nu este liber, să negocieze o nouă oră sau o nouă zi, sau cere asistentei să îi programeze la data și ora cele mai apropiate de ziua curentă, stabilite de comun acord.

Această modalitate limitează intervalul de timp de stabilire a programărilor la orelor de lucru ale personalului cabinetului. Datorită acestui fapt, mulți pacienți preferă să omită această etapă și să se prezinte la cabinetele medicilor fără a fi stabilit în prealabil o programare. Aceasta duce la aglomerarea sălilor de așteptare ale cabinetelor, nerespectarea înregistrărilor deja existente în agendele medicului de familie și decalarea programului acestuia, întrucât acesta va ajunge să țină cont doar parțial de programările deja existenta în agenda sa.

Utilizarea unei aplicații de tipul e-Appointment pentru gestionarea programărilor medicului de familie va face posibilă înregistrarea unei programări și în afara programului cabinetului, întrucât o astfel de aplicație poate fi accesată în orice moment. Acest fapt va încuraja pacienții cabinetului să înregistreze o programare pentru consultațiile ce nu reprezintă o urgență medicală sau pentru eliberarea de documente necesare, ceea ce va conduce la o mai bună administrare a activităților din cadrul cabinetului medicului de familie și la diminuarea timpului petrecut în cozile de așteptare de către pacienții cabinetului.

3.3. Prezentarea cerințelor funcționale ale aplicației și a restricțiilor impuse asupra acestora

Pentru o mai bună întelegere a nevoilor medicului de familie în ceea ce privește întregul proces de gestiune a programărilor la cabinetul său, precum și o definire a funcționalităților pe care acesta ar dori să le regăsească într-o astfel de aplicație, am luat legătura cu un medic de familie în vederea întocmirii listei de cerințe funcționale.

În urma analizei am identificat următoarele funcționalități principale:

1. Medicul trebuie să poată adăuga, vizualiza, edita sau modifica datele personale ale pacienților pe care îi are înregistrați la cabinetul său;
2. Medicul trebuie să poată adăuga și modifica programul de lucru atât al său cât și al angajaților săi (asistente și/sau medici rezidenți);
3. Medicul dorește să poată vizualiza, edita și șterge datele personale ale personalul angajat în cadrul cabinetului său (asistente și/sau medici rezidenți);
4. Medicul/personalul cabinetului trebuie să poată vizualiza toate programările înregistrate în sistem;
5. Medicul/personalul cabinetului trebuie să poată edita și șterge orice programare care nu depășește data curentă;
6. Medicul/personalul cabinetului trebuie să poată adăuga programări ce se pot repeta lunar;
7. Medicul/personalul cabinetului trebuie să poată vizualiza, la momentul logării în sistem, programările pentru ziua respectivă;
8. Intervalul de timp aferent unei consultații la cabinetul medicului de familie este de 15 minute, indiferent de scopul programării, iar numărul maxim de

consultații decontate de Casa Națională de Asigurări de Sănătate este de 20 de consultații pe zi. Astfel, programul medicului de familie este de 6 ore pe zi, lucrate fie dimineața, fie după-amiaza. Aceste ore trebuie să poată fi setate de către medic din cadrul aplicației;

9. Intervalul de timp aferent unei consultații la domiciliu este, în medie, de o oră în fiecare zi. Acestă oră trebuie să poată fi setată de către medic din cadrul aplicației;
10. Interfața grafică a aplicației trebuie să fie intuitivă și ușor de utilizat.

Luând în considerare faptul ca aplicația este dezvoltată cu scopul de a fi utilizată atât de către personalul cabinetului de familie cât și de pacienții înregistrați la respectivul cabinet, am avut în vedere și cerințele pacienților. Astfel, în urma discuțiilor cu pacienți ce ar putea deveni utilizatori ai aplicației, am identificat următoarele funcționalități necesare, ce vin în completarea celor menționate anterior:

1. Pacientul trebuie să poată vizualiza, edita sau anula (șterge) propriile programări înregistrate în sistem;
2. Pacientul trebuie să primească o notificare (*reminder*) cu privire la programarea înregistrată în ziua premergătoare celei pentru care și-a făcut programare;
3. Pacienții trebuie sa poată accesa oricând aplicația pentru a înregistra o programre;
4. Interfața grafică a aplicației să fie intuitivă și ușor de utilizat;

Analizând aceste din punct de vedere tehnic, am adăugat următoarele restricții și specificații la cerințele menționate:

1. Dacă un pacient are înregistrată o programare pentru ziua curentă, el nu o poate modifica sau șterge în ziua în curs;
2. Pacienții nu vor putea adăuga o programare pentru ziua curentă;
3. Pacienții/personalul cabinetului nu vor putea adăuga o programare pentru zile anterioare zilei curente.
4. Pacienții vor putea înregistra o programare pentru următoarele cel mult 60 de zile de la ziua curentă;
5. Vor exista trei roluri în cadrul aplicației – rolul de pacient, rolul de angajat, respectiv rolul de medic. Rolul medicului va fi asemenea celui de

administrator al sistemului și el va avea drepturi asupra tuturor resurselor – va putea adăuga, edita, șterge și vizualiza datele cu privire la programări, pacienți și angajați și va putea crea conturile de utilizatori ale pacienților și angajaților.

6. Conturile de utilizatori vor fi create odată cu introducerea în sistem a pacienților, respectiv a angajaților, din interfața grafică;
7. Parolele utilizatorilor vor fi criptate înainte de a fi introduse în baza de date, acest fapt ajutând la asigurarea securității informațiilor cu privire la conturile utilizatorilor;
8. În momentul înregistrării conturilor, pentru fiecare cont se va genera aleatoriu un token, iar acest token va fi utilizat pentru autorizarea cererilor trimise către server;
9. Aplicația va fi împărțită în două module, iar nicio categorie de utilizatori nu va avea acces la resursele care nu îi sunt permise. Aceste trei module sunt:
    - un modul aferent cabinetului, în cadrul căruia se vor putea vizualiza, adăuga, șterge și edita toate programările, toate intervalele orare referitoare la programul său și al angajaților, toți pacienții și angajații;
    - un modul aferent pacienților, ce vor putea modifica, șterge și adăuga programări;

4. Prezentarea tehnologiilor utilizate la dezvoltarea aplicației

O aplicație Web este definită ca fiind „un sistem software ce rulează într-un browser conform unor standarde definite de consorțiul World Wide Web (W3C)[1]", după cum este menționat în lucrarea [13]. Spre deosebire de aplicațiile software tradiționale, ce pot rula doar pe mașina de calcul pe care sunt instalate, aplicațiile Web rulează pe un server Web și pot fi accesate de mai mulți utilizatori simultan prin intermediul browser-elor. Acest lucru asigură disponibilitatea aplicației la orice moment, propagarea automată către toți utilizatorii a modificărilor aduse codului sursă, centralizarea datelor și faptul ca toți utilizatorii au acces la aceleași informații conform rolurilor atribuite fiecăruia. De asemenea, necesarul de resurse pentru utilizarea unei aplicații Web este minim, prelucrările de date realizându-se pe server-ul pe care aceasta rulează, nu pe dispozitivul de pe care aceasta este accesată.

Am ales ca aplicația realizată în urma acestui proiect să fie una Web, întrucât o astfel de aplicație satisface criteriile de funcționare definite în cadrul proiectului. Aceasta este alcatuită din doua componente – componenta front-end, cu care utilizatorul ia contact și care este executată în browser-ul acestuia, realizând transmiterea de cereri către server și preluarea informațiilor pe care acesta le pune la dispoziție în urma cererilor, respectiv componenta back-end, cu rol de server, care procesează solictările clientului (browser-ului) și returneaza datele solicitate.

Pentru dezvoltarea părții de front-end a aplicației am folosit Angular, versiunea 5.2, o platformă open-source dezvoltată de Google ce are la bază limbajul JavaScript, împreună cu NPM (Node Package Manager) ce contine Node.js, mediul de executie server-side pentru script-urile JavaScript. Modulele aplcației au fost dezvoltate folosind TypeScript, un superset al limbajului JavaScript, ce permite utilizarea conceptelor de programare orientată pe obiecte în dezvoltarea logicii de gestionare a datelor din cadrul componentelor ce alcătuiesc partea de font-end a soluției dezvoltate, așa cum este prezentat în lucrarea [11]. Aceste tehnologii, împreună cu limbajul menționat, oferă o mai mare flexibilitate în ceea ce privește separarea modulelor, în special a logicii de obținere a informațiilor de la server (logica de business) de interfața cu utilizatorul, prin implementarea modelul arhitectural MVVM (Model-View-ViewModel). Totodată, framework-ul Angular facilitează

[1] W3C este o comunitate internațională care se ocupă cu definirea standerdelor conform cărora se realizează dezvoltarea de resurse Web.

manipularea DOM-ului (Document Object Model) prin utilizarea directivelor structurale pentru adăgarea și eliminarea componentelor HTML.

Partea de back-end a aplicației a fost dezvoltată folosind Microsoft ASP .NET, o tehnologie web ce combină ASP (Active Server Pages) destinată creării de aplicații și servicii Web și mediul de programare definit de arhitectura Microsoft .NET, care permite crearea, distribuirea și executarea de aplicații Web. Această tehnologie se bazează pe limbajul de programare C#, un limbaj „managed", compilat și executat sub controlul arhitecturii .NET și care utilizează bibliotecile definite ce aceasta, precum este evidențiat în [8]. Ca server pentru baza de date am utilizat Microsoft SQL Server Express, un sistem de gestiune a bazelor de date relaționale distribuit de Microsoft impreună cu SQL Server Management Studio, utilizat pentru a configura, gestiona și administra bazele de date create cu Microsoft SQL Server prin intermediul unei interfețe grafice.

4.1. Detalierea tehnologiilor folosite pentru dezvoltarea componentei back-end

Componenta server-side este o aplicație ASP .NET Web API care rulează pe server-ul Web IIS[2] Express din Windows, o unealtă software ce permite ca un site Web sa fie vizualizat folosind HTML[3].

Framework-ul Web API este dedicat realizării de servicii HTTP pentru o varietate de dispozitive (clienți), printre care browsere Web, dispozitive mobile precum tabletele sau smartphone-urile și dispozitive IoT. Acesta are următoarele caracteristici:

— Permite efectuarea operațiunilor CRUD (Create, Retrieve, Update, Delete) prin intermediul metodelor specificate de protocolul HTTP (Hypertext Transfer Protocol) POST, PUT, GET, DELETE;

— Răspunsurile sunt oferite în diverse formate precum XML[4] sau JSON [5]și sunt convertite în aceste formate de către Media Type Formatter[6], ce aparține Web API-ului;

[2] **Internet Information Services Express (IIS Express)** reprezintă un server Web ce aparține de Microsoft și care permite găzdurea și rularea site-urilor și aplicațiilor Web.
[3] **HyperText Markup Language (HTML)** este limbaj utilizat pentru crearea de pagini Web ce pot fi vizualizate într-un browser.
[4] **Extensive Markup Language (XML)** este un limbaj de marcare utilizat pentru transmiterea sau stocarea de informații într-un mod structurat.
[5] **JavaScript Object Notation (JSON)** reprezintă un format de reprezentare a datelor utilizat în special în transmiterea datelor între server și browser.

— Pe lângă formatele menționate anterior, poate accepta sau genera documente PDF, imagini etc.;
— Permite utilizarea caracteristicilor modelului arhitectural MVC (Model-View-Controller), a filtrelor pentru autorizare și a rutelor.

Elementele cele mai importante din cadrul API-ului dezvoltat în urma acestui proiect sunt:

— Resursele (metodele) – fiecare resursă a unui API are asociat un URL unic prin intermediul căruia este adresată;
— Metodele HTTP – acestea specifică acțiunea care se dorește a fi realizată în urma solicitării respective: de a obține o resursă (meotda GET), de a trimite o resursă (metodele POST sau PUT), de a șterge o resursă (metoda DELETE);
— Conținutul solicitării HTTP – în cadrul metodelor POST și PUT va conține resursa trimisă către server;
— Conținutul răspunsului – în funcție de solicitare, răspunsul HTTP poate fi de forma unui document XML, a unui JSON sau o pagina HTML;
— Codurile de stare HTTP – care semnalează dacă solicitarea trimisă către server a fost executată cu succes, sau, în caz contrar, marchează tipul erorii.

Funcțiile principale ale unui Web API sunt acelea de a asigura preluarea cererilor de la client (HTTP Requests), de a realiza prelucrările necesare asupra datelor, dacă este cazul, și de a întoarce un răspund la solicitarea clientului (HTTP Response), conform imaginii din figura 4.1.

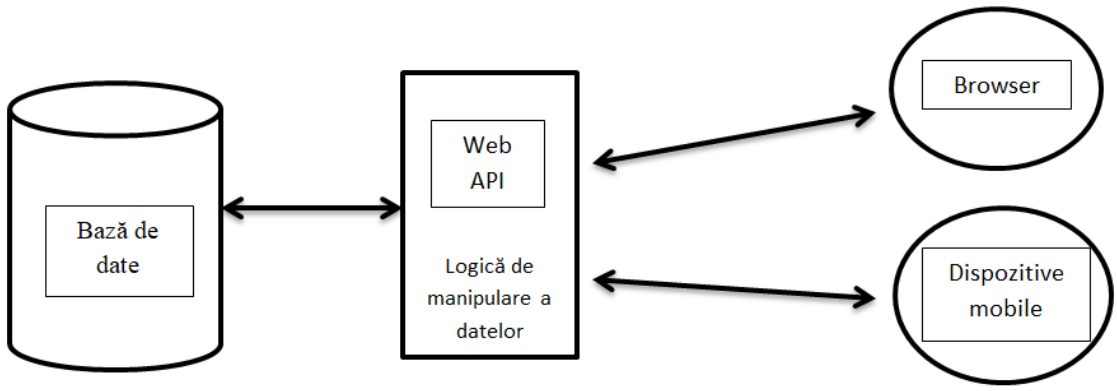

Figura 4.1. Web API în arhitectura unei aplicații Web

[6] **Media Type Formatter** este un instrument ce permite scrierea, respectiv citirea de obiecte specifice C# în și din cadrul unei solicitări HTTP.

4.2. Detalierea uneltelor utilizate pentru crearea, stocarea și gestionarea bazei de date

O bază de date reprezintă o colecție organizată de informații, aflată sub controlul unui porgram sau ansamblu de programe ce permite actualizarea, modificarea și vizualizarea informațiilor stocate, conform [6]. Aceste programe de prelucrare a datelor poartă denumirea de Sistem de Gestiune a Bazelor de Date (SGBD), iar bazele de date pot fi clasificate în funcție de modelul de date în două categorii:

- Baze de date relaționale;
- Baze de date obiect;
- Baze de date realaționale – obiect.

În cadrul proiectului am ales să utilizez o bază de date relațională, întrucât aceasta permite organizarea și stocarea datelor sub formă de înregistrări în cadrul unor tabele, folosind algebra relațională. Structura tabelară a acestui tip de baze de date permite crearea de asocieri între tabele prin intermediul cheilor primare, împreuna cu cheile străine și indecșii. O cheie primară este utilizată în cadrul unui tabel pentru a indentifica în mod unic fiecare înregistrare. În cadrul unei asocieri între două sau mai multe tabele, cheia primară a unei tabele va deveni cheie străină în cadrul unei alte tabele ce face referire la o înregistrare din prima tabelă, identificând în mod unic această înregistrare.

Pentru stocarea și gestiunea datelor pacienților, evidența programărilor și datele de contact ale angajaților, am folosit Microsoft SQL Server Express, sistemul de gestiune a bazelor de date relaționale pus la dispoziție de către Mircrosoft și destinat aplicațiilor ce nu utilizează un volum mare de date.

Pentru interacțiunea cu SQL Server Express am folosit SQL Server Management Studio, o unealtă software ce permite interacțiunea cu infrastructura server-ului SQL prin intermediul unei interfețe cu utilizatorul. Această interfată pune la dispoziția utilizatorului unelte precum Object Explorer, ce permite vizualizarea și modificarea obiectelor dintr-o instanță SQL Server, sau script-uri SQL predefinite pentru fiecare operațiune SQL și pentru fiecare tabelă.

4.3. Detalierea tehnologiilor utilizate pentru dezvoltarea componentei front-end

Componenta front-end a soluției este o aplicație SPA (Single Page Application) dezvoltată folosind Framework-ul Angular, versiunea 5.2.

4.3.1. Prezentarea conceptului de Single Page Application

Principiul de funcționare al aplicațiilot de tipul SPA constă în faptul că tot conținutul aplicației este redat într-o singură pagină HTML, cu scopul de a oferi o navigare mai rapidă între componente, conform [3]. Pagina HTML este încărcată o singură dată în browser-ul în care aplicația este rulată, iar conținutul ei este modificat în mod dinamic la fiecare navigare realizată de către utilizator. Datorită acestui fapt, se condieră că aplicațiile Web de tipul SPA au o performanță mai ridicată în comparație cu aplicațiile Web tradiționale, în cadrul cărora fiecare navigare sau operațiune a utilizatorului (de exemplu, trimiterea unei solicitări către server) necesită reîncărcarea unei noi pagini HTML.

Aplicațiile de tipul SPA pot varia de la simple aplicații de tipul CRUD, ce permit utilizatorului doar realizarea de operații de bază asupra datelor (creare, vizualizare, editare și ștergere), la aplicații complexe, ce conțin un număr mare de componente, script-uri și formulare de preluare și validare a datelor.

Popularitatea acestor aplicații este în creștere, în principal datorită ușurinței cu care pot fi dezvoltate și a fluidității pe care o oferă utilizatorului la navigarea între componente. Tehnologiile Web și limbajele care fac posibile aceste lucruri sunt AJAX, care permite interacțiunea cu server-ul de date si trimiterea asincronă de solicitări HTTP către acesta, șabloanele HTML, framework-urile MVC (Model-View-Controller) sau MVVM (Model-View-View-Model) și nu în ultimul rând, limbajul JavaScript (conform [3]).

AJAX (Asynchronous JavaScript and XML) este o tehnică bazată pe limbajul JavaScript și utilizată în dezvoltarea de aplicații Web, în special în cadrul acelor aplicații ce necesită încărcarea și prelucrarea datelor în mod dinamic, conform lucrării [15]. Această tehnică permite utilizatorului să utilizeze aplicația în timpul execuției solicitarilor HTTP, întrucât acestea sunt trimise către server-ul de date, preluate și prelucrate în mod asincron. Spre deosebire de aplicațiile Web convenționale, în cadrul cărora solicitările HTTP, respectiv răspunsurile primite de la server sunt prelucrate în mod sincron, necesitând redirecționarea către o nouă pagină sau reîncărcarea paginii curente pentru ca utilizatorul să poată vedea informațiile solicitate, prin utilizarea tehnicii AJAX, datele obținute de la server sunt oferite clientului (broswer-ului) prin modificarea paginii curente, fără a necesita redirecționarea sau reîncarcarea acesteia.

Șabloanele HTML sunt utilizate pentru afișarea datelor, oferirea posibilităților de editare a acestora sau înregistrare de noi informații prin intermediul formularelor HTML (HTML forms).

Framework-ul MVC este un model software arhitectural utilizat pentru a separa reprezentarea vizuală a informației de logica utilizată pentru a achiziționa și prelucra informațiile, după cum este prezentat în lucrarea [3]. Acest model arhitectural poate fi folosit atât în cadrul aplicațiilor destinate să ruleze în browser-ul Web, cât și în cadrul aplicațiilor de tip server. În cadrul aplicațiilor de tip client construite conform standardelor MVC se utilizează exclusiv limbajul JavaScript.

În cadrul acestei arhitecturi, cele trei straturi au următoarele funcționalități:

- View (Vedere) – acest strat se află la suprafața arhitecturii MVC și este folosit pentru prezentarea și preluarea datelor de la client. Acesta conține interfața grafică și este singura componentă cu care utilizatorul interacționează în mod direct. Acest strat interacționează în mod direct doar stratul de control (Controller);
- Model (Modelul) – conține modelele de date cu care aplicația interacționează. Aceste modele pot fi organizate sub formă de clase și sunt folosite pentru a modela și păstra informația primită de la server, fără a influența modul în care această informație este prezentată către browser. Acest strat, asemenea celui precedent, interacționează în mod direct doar cu stratul de control, întrucât o expunere directă a modelelor în stratul de vedere ar putea produce probleme legate de securitatea informațiilor prezentare utilizatorului;
- Controller – acesta reprezintă legătura dintre cele două straturi prezentate anterior și conține totodată logica de prelucrare a datelor și de validare a informațiilor introduse de utilizator, precum și definirea rutelor din cadrul aplicațiilor.

Interacțiunea dintre modulele din cadrul arhitecturii MVC este descrisă in figura 4.2:

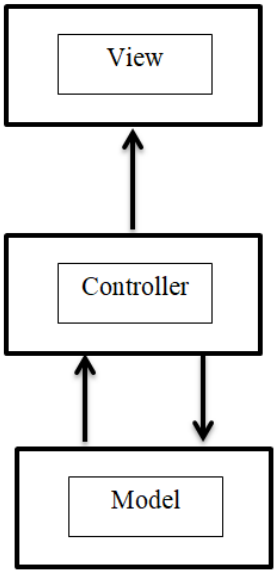

Figura 4.2. Modelul arhitectural MVC

Framework-ul MVVM păstrează două din componentele prezente și în arhitectura MVC, înlociund însă stratul de control cu un nou strat numit *View-Model* (Model-Vedere). Acesta poate fi considerat ca fiind un Controller specializat care prezintă funcționalitățile unui convertor de date, realizând conversia dintre tipurile de date prezente în *Model* în tipurile de informații care se doresc a fi prezente în stratul de vedere, pentru a fi expuse utilizatorului.

Stratul de *Vedere* comunică cu stratul de *Vedere-Model* prin intermediul legăturilor bidirecționare dintre date și prin intermediul evenimentelor. Interacțiunea modulelor din cadrul acestui tip de arhitectură este prezentată în figura 4.3:

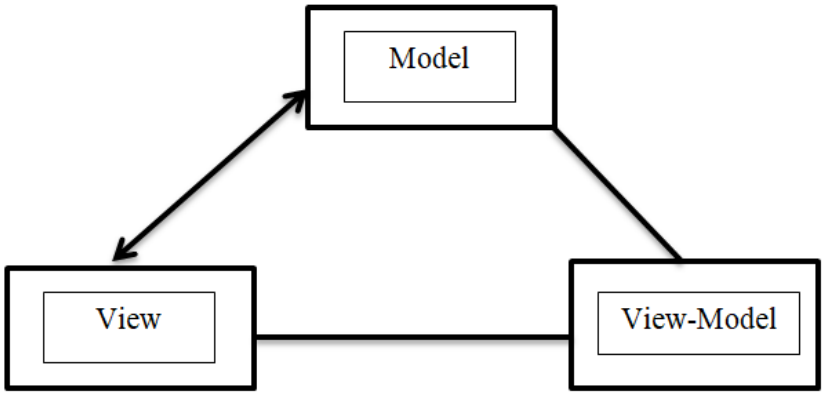

Figura 4.3. Modelul arhitectural MVVM

Legăturile bidirecționale între date reprezintă unul din conceptele ce stau la baza dezvoltării aplicațiilor SPA. Acestea permit propagarea imediată a modificărilor ce au loc asupra datelor din model către stratul de vedere, în cadrul interfeței grafice, conform [3]. Aceste legături bidirecționale reprezintă, de fapt, conexiunea directă între interfața grafică a aplicației și stratul care se ocupă cu obținerea și prelucrarea datelor. Astfel, orice modificare asupra variabilelor ce are loc în cadrul prelucării va fi reflectată imediat în interfața grafică, către utilizator.

4.3.2. Framework-ul Angular

AngularJS este un framework JavaScript open-source dezvoltat de Google în anul 2010 și destinat dezvoltării aplicațiilor Web de tipul SPA, conform lucrării [1]. Acesta urmărește standerdele modelul arhitectural MVC în vederea separării componentelor de prezentare a informațiilor către utilizator, de obținere și prelucrare a acestor informații și de definire a modelelor de date. Dezvoltarea acestui framework a făcut posibile separarea logicii aplicației de manipularea elementelor ce țin de interfața cu utilizatorul și izolarea componentei de server de componenta client, pentru a permite dezvoltara în paralel a celor două. De asemenea, acest framework pune la dispoziție o structura complexă ce permite construirea interfeței destinate utilizatorului, definirea logicii interne de prelucrare a datelor și dezvoltarea de module de testare a aplicației dezvoltate.

Cu toate acestea, codul JavaScript rezultat în urma dezvoltării folosind acest framework atingea, în cadrul aplicațiilor cu un număr mare de module, o complexitate ce încetinea procesul de încărcare a paginii de pe server-ul Web în browser-ul utilizatorului.

Angular 2, apărut în anul 2016 și cunoscut și sub denumirea de Angular, reprezintă o nouă versiune a framework-ului AngularJS, complet rescrisă și aduce cu sine o performanță îmbunătățită a aplicațiilor. Această versiune a fost dezvoltată folosind limbajul TypeScript, un limbaj ce aparține de Microsoft și care permite utilizarea conceptelor de programare regăsite în cadrul limbajelor de nivel înalt precum Java sau C#, conform informațiilor prezentate în lucrarea [10].

Angular 2, precum și versiunile de Angular apărute în urma acesteia, folosesc directive și componente pentru a realiza legăturile dintre stratul de prezentare a aplicației și logica de manipulare a datelor, conform [10], [11]. Acestea permit o structurare mai bună a modulelor în cadrul aplicației, fapt ce conduce la diminuarea timpului de încărcare a paginii chiar și în cazul aplicațiilor Web complexe, rezolvând astfel problemele apărute în cadrul aplicațiilor dezvoltate folosind AngularJS. Totodată, Angular 2, precum și

versiunile de Angular apărute în urma acestuia, facilitează dezvoltarea de aplicații rapide și scalabile ce pot avea drept dispozitive țintă, pe lângă browserele Web, și sisteme mobile sau chiar desktop.

În cadrul proiectului am ales utilziarea versiunii de Angular 5.2, ultima versiune de Angular lansată în anul 2017. Pentru instalarea dependențelor și generarea fișierelor de configurare necesare rulării proiectului dezvoltat folosind acest framework, am utilizat Angular CLI, un instrument pus la dispoziție de dezvoltatorii Angular. Acesta oferă posibilitatea de a crea și gestiona fișierele și modulele din cadrul proiectului direct din linia de comandă, oferind interfața necesară și o comandă specifică, numită *„ng”*, alături de un set de opțiuni ce facilitează lucrul cu fișierele din cadrul aplicației.

Pentru instalarea uneltei Angular CLI și gestionarea dependențelor din cadrul proiectului am utilizat Node Package Manager (NPM), respectiv Node.js. Acestea oferă și uneltele necesare transpunerii din limbajul TypeScript în limbajul JavaScript, ce va fi mai apoi interpretat de către browser-ul Web, realizând și legăturile între componentele aplicației și gruparea acestora într-o singură pagină HTML, ce va fi executată în browser.

Elementele specifice framework-ului Angular utilizate în cadrul aplicației sunt componentele, serviciile, modulele și router-ul.

Componentele (*Components*), conform lucrării [11], definesc logica de manipulare a datelor introduse de către utilizatori în interfață, respectiv deservire către interfata grafică a datelor primite de la server (clase componente), precum și interfața grafică (view-uri html). Prin convenție, acestea se denumesc conform convenției generale „nume-componenta.component.ts” pentru fișierele TypeScript ce se ocupă cu prelucrarea datelor, respectiv „nume-componenta.component.html”, pentru template-urile HTML aferente fiecărei componente.

Serviciile (*Services*), încorporează metodele de trimitere de date către server, respectiv solicitare de date de la server, iar funcționalitatea lor nu afectează în mod direct elementele din componenta HTML. Acestea sunt utilizate în cadrul componentelor pentru trimiterea sau preluarea datelor de la server.

Modulele (*NgModules*) permit gruparea componentelor, serviciilor si a șabloanelor HTML conform unei logici interne de acces la resurse, premitând totodată încărcarea de module doar atunci când acestea sunt solicitate prin navigarea la adresa URL asociată acestora.

În vederea dezvoltării interfeței utilizator am utilizat PrimeNG, o bibliotecă de componente grafice dezvoltate pentru a fi utilizate în cadrul proiectelor ce au la bază framework-ul Angular (conform [17]). Aceasta pune la dispoziție o suită de teme CSS[7] și componente grafice ce au funcționalități multiple. Dintre acestea, cele utilizate frecvent în cadrul proiectului sunt:

- *DataTable*, un element de tip tabel pentru prezentarea datelor, ce oferă posibilitatea de sortare, filtrare sau paginare a înregistrărilor;
- *Dropdown*, ce permite filtrarea opțiunilor;
- *Calendare* pentru lucrul cu date calendaristice;
- Elemente de tip *Input* pentru text specific parolelor;
- Elemente de tip *Input* pentru text cu restricții asupra tipurilor de date
- Elemente de tipul *Panel* utilizate pentru gruparea elementelor de tip input;
- Elemente de tipul *Growl* utilizate pentru a oferi mesaje de notificare utilizatorului;
- Elemente de tipul *ConfirmDialog* utilizate pentru a solicita confirmarea utilizatorului înainte de executarea unei acțiuni (de exemplu, ștergerea unei înregistrări);

[7] Cascading Style Sheets, un standard folosit pentru formatarea elementelor HTML.

5. Prezentarea modului de implementare a aplicației

5.1. Prezentarea modului de implementare a componentei back-end

Pentru un grad de abstractizare care să faciliteze dezvoltarea ulterioară în cadrul aceleiași aplicații și un bun control asupra resurselor, îndeosebi asupra informațiilor la care clientul (browser-ul) va avea acces, am ales să împart aplicația în 5 module (straturi). Soluția în sine conține un proiect ASP .NET Web API cu suport MVC denumit **MedApplication.Backend**, ce reprezintă aplicația cu rol de Web API care va rula efectiv pe serverul IIS Express, un proiect Web API gol denumit **MedApplication.Frontend**, ce conține aplicația client (front-end) care rulează pe serverul Node.js și alte trei proiecte de tipul Class Library ce sunt utilizate pentru gestiunea datelor server-side, fiecare având un rol specific. Acestea pot fi observate în figura 5.1.

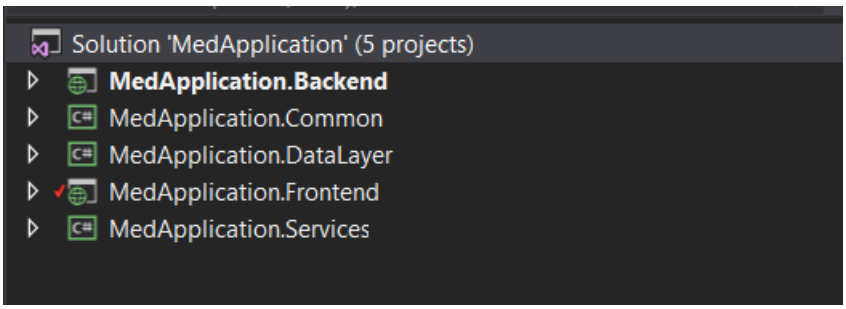

Figura 5.1. Modulele din cadrul proiectului

5.1.1. Prezentarea modulelor ce intră în alcătuirea aplicației

Proiectul **MedApplication.Common** conține clasele ce vor modela entitățile utilizate în cadrul aplicației. Legătura dintre aceste clase și entitățile bazei de date se realizează prin intermediul unui framework ORM (Object Relational Mapping).

Object Relational Mapping este o tehnică de programare care realizează conversia între tipuri de date ce aparțin de medii diferite, prin intermediul limbajelor de programare orientate pe obiecte, conform lucrării [16]. Această tehnică este utilizată frecvent în aplicațiile ce necesită lucrul cu baze de date deoarece permite construirea, în cadrul aplicației dezvoltate, a unui model a entităților acestora. Modelul va permite manipularea înregistrărilor din tabelele bazei de date, întrucât acestea vor fi considerate ca fiind colecții de obiecte de tipul claselor ce definesc fiecare din aceste tabele.

În cadrul aplicației, pentru realizarea acestei corespondențe am folosit framework-ul EntityFramework, versiunea 6.2.0. Acest framework facilitează gestionarea resurselor din baza de date, precum și adăugarea, modificarea sau eliminarea de tabele folosind operațiile specifice SQL.

Am ales sa utilizez o abordare Code First, care constă în crearea, mai întâi, a claselor care vor modela entitățile utilizate în cadrul aplicației. Spre deosebire de abordările clasice, care presun construirea mai întâi a bazei de date și apoi a claselor aferente tabelelor create și modelarea legaturilor dintre acestea, Code First se ocupă de gestionarea tabelelor, de generarea bazei de date și de realizarea legăturilor între tabele conform claselor definite și configurărilor specificate în cod (conform [16]). Modelul poate fi modificat pe parcursul dezvoltării aplicației tot prin intermediul instrucțiunilor din cod (modificarea membrilor claselor și reconfigurarea legăturilor, dacă este cazul) și prin aplicarea de propagări automate sau explicite (*automatic/manual migrations*) ale acestor modificări către baza de date.

Cu alte cuvinte, prin utilizarea Entity Framework împreună cu o abordare Code First se realizează transpunerea obiectelor din .NET într-o bază de date relațională, de unde se pot accesa sau modifica ori de câte ori este nevoie prin intermediul interogărilor LINQ[8]. Modul de funcționare al acestei arhitecturi este ilustrat în Figura 5.2.

MedApplication.Common conține doar clasele corespunzătoare entităților ce vor fi păstrate în baza de date, instrumentul de gestiune fiind instalat propriu-zis în cadrul modulului **MedApplication.DataLayer**.

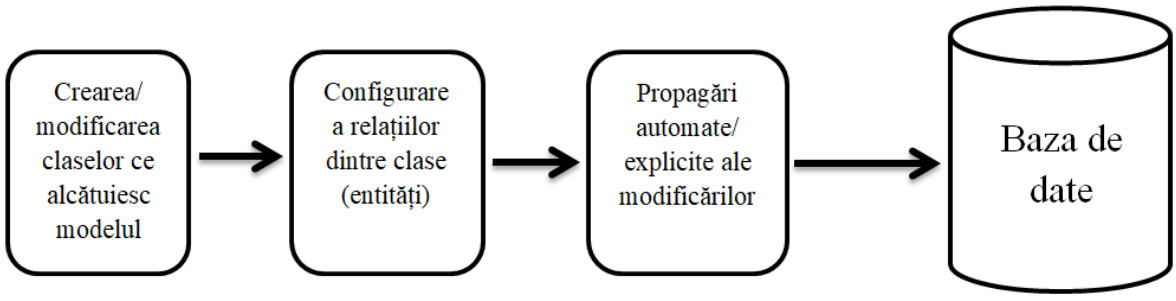

Figura 5.2. Modul de funcționare al EntityFramework Code First

[8] **Language-Integrated Query (LINQ)** denumește un set de tehnologii ce integrază limbajul interogărilor SQL în limbajul de programare C#.

MedApplication.DataLayer este proiectul care se ocupă exclusiv cu accesul la entitățile din baza de date, cu definirea legăturilor și cu operații asupra tabelelor folosind DML[9] și DDL[10]. Entity Framework a fost instalat în cadrul acestui proiect utilizând NuGet Package Manager, o unealtă în cadrul mediului de dezvoltare Visual Studio utilizată pentru gestiunea pachetelor externe.

EF ține cont de anumite convenții pentru asocierea proprităților obiectelor cu coloanele și tabelele protrivite, dintre care fac parte următoarele:

- Pentru o proprietate de tipul „NumeClasăId”, sau pur și simplu „Id” a unei clase ce intră în componența modelului pentru baza de date, EF va considera automat acea proprietate ca fiind identificatorul unic al tabelei modelate;
- Numele tabelelor create în baza de date sunt considerate ca fiind forma de plural a denumirilor claselor definite în model (în limba engleză);
- Legăturile de tipul *One-to-Many* sunt create automat, pe baza proprietătilor definite în interiorul claselor;

Totodată, framework-ul permite configurarea de noi convenții pentru a satisface cerințele de proiectare a bazei de date.

În cadrul abordării Code First, accesul către baza de date se realizează prin intermediul unei clase ce derivează din DbContext, o clasă existentă în framework-ul EF și implementează anumite funcționalități ale acesteia. În proiectul **MedApplication.DataLayer**, am denumit această clasă *MedApplicationContext*. Aceasta conține proprietățile de tipul DbSet<T>, ce poate fi considerat ca fiind un set generic de obiecte de tipul „T” (unde „T” va fi înlocui de fiecare din clasele definite în modulul **MedApplication.Common**) care se ocupă cu operațiile de scriere și citire în și din baza de date, precum și cu operațiile de ștergere, așa cum este prezentat în lucrarea [16].

Aceste operații sunt definite în interiorul unei clase, denumită în mod generic Repository, care conține metodele necesare interogărilor principale asupra înregistrărilor stocate în baza de date (*Insert*, *Update*, *Delete*, *GetById*), conform indicațiilor din lucrarea [18]. Aceste operații vor fi realizate asupra fiecăreia din entități, așadar fiecare clasă asociată unei entități va avea definit un repository propriu ce va deriva din Repository-ul

[9] **Data Manipulation Language (DML)** este un limbaj utilizat în cadrul lucrului cu baze de date și definește operațiunile de inserare, modificare, ștergere sau vizualizare a înregistrărilor din tabelele acesteia.
[10] **Data Definition Language (DDL)** este un limbaj utilizat în cadrul lucrului cu baze de date și definește operațiunile de modificare, ștergere sau creare de tabele în cadrul acesteia.

generic și va conține, deci, toate metodele definite în interiorul acestuia. Ulterior vor putea fi definite și alte metode specifice pentru fiecare entitate în parte, dacă este cazul.

MedApplication.Services este modulul ce conține metodele prin intermediul cărora se realizează operații asupra elementelor din baza de date. Această tehnologie permite ca operațiile asupra înregistrărilor din tabelele bazei de date, (instrucțiuni SQL precum *SELECT, INSERT, UPDATE, DELETE*, clauze precum FROM, WHERE, ORDER BY, sau operații agregat cum ar fi COUNT, MIN, MAX), scrise de obicei sub forma unui șir de caractere și trimis mai apoi spre execuție către baza de date, să fie scrise direct în limbajul de programare C#, sub formă de metode ce utilizează expresii Lambda sau folosind întocmai sintaxa interogărilor SQL.

Pe lângă definirea logicii de acces la date, acest proiect implementează și modelul de design Data Transfer Object (DTO). Noțiunea de Data Transfer Object, sau obiecte destinate transferului de date, face referire la un tip de date utilizat doar pentru a transmite informații și care nu prezintă nicio metodă de prelucrare asupra câmpurilor pe care le conține.

Acest model facilitează transferul de informații de la server către client, asigurând transmiterea integrală a datelor necesare prin regruparea proprietăților claselor ce modelează entitățile. Spre exemplu, se poate defini o clasă de tip DTO ce conține câmpurile referitoare la programări, la care se adaugă un nou câmp, de tipul Patient, ce conține informațiile referitoare la pacientul pentru care a fost înregistrată programarea. Astfel, clientul va avea acces la informațiile complete în urma unei singure solicitări către server.

Totodată, în urma regrupării proprietăților claselor ce definesc entitățile se pot elimina anumite proprietăți ce nu sunt relevante pentru solicitarea primită de la client. Spre exemplu, în urma unei solictări HTTP de tipul GET pentru a obține informațiile cu privire la toți pacienții înregistrați în aplicație, nu este necesar și nici de dorit să se returneze și datele cu privire la conturile utilizatorilor, deși acest câmp există în clasa Patient. Pentru a evita acest lucru, se va utiliza o clasa de tipul DTO ce va conține doar câmpurile ce fac strict referire la informațiile relevante despre pacient.

Așadar, utilizarea claselor de tipul Data Transfer Object aduce următoarele beneficii aplicației (conform lcurării [18]):

— Eliminarea referințelor circulare și, implicit, eliminarea problemelor ce pot apărea odată cu serializarea obiectelor în momentul în care datele sunt oferite către client;

— Ascunderea anumitor proprietăți ale entităților la care este de dorit ca browser-ul să nu aibă acces (ex: proprietatea User ce aparține clasei Patient și care conține datele referitoare la contul de utilizator pentru fiecare înregistrare din tabela Patients);
— Eliminarea anumitor proprietăți ce nu sunt necesare pentru a reduce cantitatea de informație care va fi preluată de la server (nu este necesar ca fiecare obiect de tipul Patient să conțină mereu informațiile referitoare la programările pe care le-a înregistrat în aplicație, deci aceasta proprietate poate fi eliminată din obiectul transferat);

Pentru realizarea transformărilor asupra obiectelor în noile tipuri definite de clasele DTO, am creat o clasa numită Transformations în care am definit exclusiv metodele de conversie de la tipurile de date ce definesc entitățile la tipurile dată destinat transferului către browser. Aceste transformări vor fi utilizate strict în cadrul serviciilor, în fiecare metodă și pentru fiecare operație cu baza de date, realizându-se astfel separarea dintre câmpurile destinate transferului către partea de client și cele utilizate doar în logica internă a aplicației.

Proiectul **MedApplication.Backend** conține aplicația ASP .NET Web API ce va rula pe serverul IIS Express. Această aplicație se va ocupa de preluarea cererilor de la client, a datelor primite de la acesta, dacă este cazul, și punerea la dispoziție a datelor solicitate de acesta sau returnarea unui mesaj de eroare dacă solicitarea nu s-a putut efectua cu succes.

Acest proiect va rula local și va putea fi accesat la adresa http://localhost:62405/, comunicarea realizându-se prin intermediul portului 62405. În cadrul arhitecturii MVC, elementele care se ocupă cu generarea de răspunsuri la solicitările primite de la browser sunt Controllere (Controllers), conform lucării [7]. Fiecare solicitare are ca destinație un controller, respectiv o metodă (acțiune) din interiorul acestui controller, conform URL-ului către care este trimisă. Astfel, dacă dorim să accesăm metoda GetPatientById, ce returnează datele cu privire la pacientul cu id-ul specificat, URL-ul la care se va regăsi metoda respectivă va fi: http://localhost:62405/api/CabinetPatient/GetPatientById/{id}, unde {id} va fi înlocuit de un număr ce va reprezenta id-ul pacientului.

Adresele URL ale resurselor controllerelor (metodelor din API) sunt construite cu scopul de a fi simple și intuitive, conform modelului definit în fișierul de configurare *WebApiConfig.cs* prezentat în figura 5.3. Parametrul „name” reprezintă denumirea dată rutei respective, „routeTemplate” definește modul în care se construiește ruta respectivă (în

cadrul proiectului, ruta către fiecare resursă din API va fi construită urmărind modelul „api/nume_controller/nume_metoda/parametru_opțional”), iar „defaults” marchează parametrul „id” ca fiind opțional.

```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{action}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

Figura 5.3. Configurarea rutelor pentru resursele (metodele) din API

Pentru ca resursele puse la dispoziție de către API să nu fie accesate de orice utilizator indiferent de rolul atribuit, este necesară realizarea unei filtrări a solicitărilor primite. ASP .NET Web API facilitează implementarea filtrării solicitărilor HTTP, punând la dispoziție filtrele de autentificare și autorizare, așa cum este enunțat în [10]. Aceste filtre sunt reprezentate prin intermediul unor clase ce implementează anumite interfețe ce aparțin framework-ului .NET, dintre care cea mai des utilizată este *AuthorizationFilterAttribute*. Aceste clase sunt mai apoi folosite ca atribute pentru fiecare metodă din cadrul Controller-elor. În momentul primirii unei solicitări de la browser, cursul de execuție al Web API-ului va ține cont de filtrul de autorizare, executând mai întâi instrucțiunile din cadrul acestuia pentru a verifica daca utilizatorul care a emis solicitarea are acces la resursa respectivă. În caz afirmativ, se va executa în final resursa din API care a fost solicitată.

Metoda de autorizare aleasă este bazată pe token-uri și constă în atribuirea unui token de autorizare, în momentul creării contului, pentru fiecăre utilizator. Acest token va fi salvat în baza de date alături de informațiile referitoare la contul utilizatorului (tabela „Users”). La momentul autentificării unui utilizator în aplicație, credențialele introduse în interfața grafică, în pagina de Login, sunt trimise mai apoi către server, în cadrul unei solicitări POST la URL-ul http://localhost:62405/api/Login/Login, unde metoda *Login* din cadrul API-ului va verifica existența acestor credențiale în baza de date. Această metodă este este singura care va putea fi accesată în mod anonim.

Dacă aceste credențiale au fost regăsite în sistem, server-ul va răspunde clientului printr-un răspuns HTTP ce va conține codul de stare 200 (OK), care marchează faptul că solicitarea a fost executată cu succes și utilizatorul a fost autentificat, precum și o nouă înregistrare în secțiunea Headers ce va conține token-ul de autorizare alături de un cod ce marchează rolul asociat utilizatorului respectiv. Dacă credențialele nu au fost regăsite în baza de date, server-ul va răspunde clientului printr-un răspuns HTTP ce va avea un header

denumit „Authentication” cu mesajul „Credentiale utilizator necunoscute”. În acest caz, se consideră identitatea utilizatorului ca fiind necunoscută și acesta va fi rugat să reintroducă credențialele de autentificare sau să ia legătura cu medicul de familie în caz ca nu se poate înregistra în aplicație.

După stabilirea identității utilizatorului, token-ul de autorizare preluat de la server va fi stocat în memoria browser-ului, în secțiunea *Local Storage*, sub forma unei structuri de tipul „cheie – valoare”.

Pașii descriși anterior sunt prezentați sub forma unor scheme în figurile 5.4 și 5.5, pentru ambele cazuri de autentificare – autentificare reușită, respectiv autentificare nereușită.

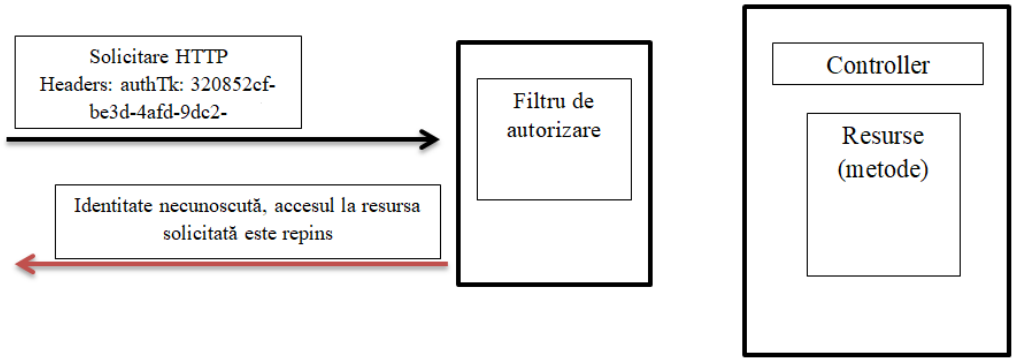

Figura 5.4. Stabilirea identității utilizatorului pe baza credențialelor în momentul autentificării – autentificare reușiă, conform [10]

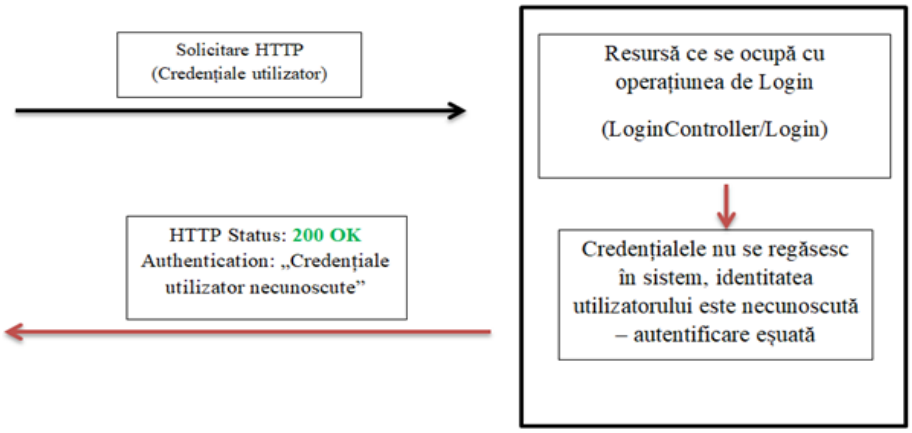

Figura 5.5. Stabilirea identității utilizatorului pe baza credențialelor în momentul autentificării – autentificare nereușită, conform [10]

După autentificarea utilizatorului, orice solicitare trimisă către server va avea asociată un header ce va conține acest token. Pe baza acestui token, filtrele de autorizare

lansate în execuție de către framework-ul Web API vor stabili identitatea utilizatorului care a lansat solicitarea și vor decide, pe baza rolului asociat acestuia, dacă acesta poate avea acces la resursele solicitate sau nu, conform figurilor 5.6 și 5.7:

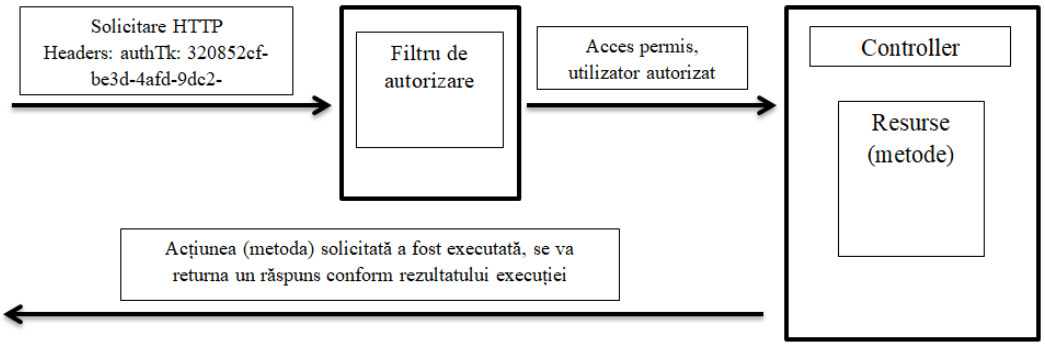

Figura 5.6. Verificarea identității utilizatorului pe baza token-ului de autorizare – identitate recunoscută, conform [10]

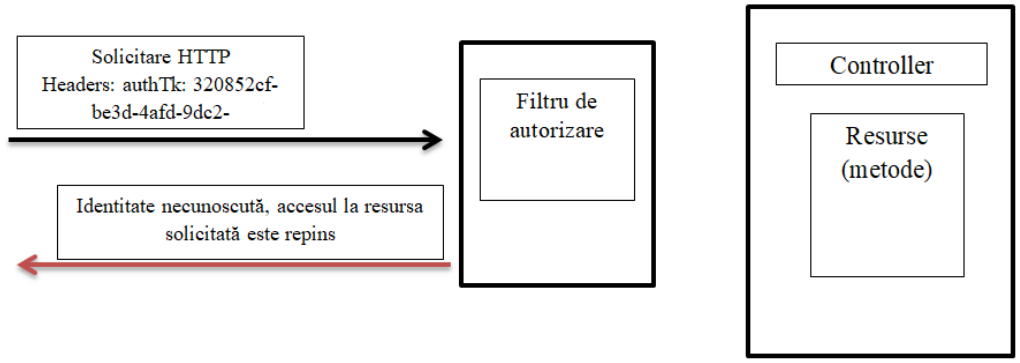

Figura 5.7. Verificarea identității utilizatorului pe baza token-ului de autorizare – identitate necunoscută, conform [10]

Tot în cadrul componentei back-end sunt implementate și metodele prin intermediul cărora pacienții vor primi un mesaj de notificare, sub forma unui SMS, cu privire la programările înregistrate. Acest mesaj va fi trimis o singură dată, cu o zi înainte de data la care pacientul va trebui să se prezinte la cabinet.

Pentru implementarea acestei funcționalități este necesară verificarea periodică a programărilor existente în baza de date. Așadar, va fi necesară existența unei unelte ce va verifica periodic dacă în baza de date există programări pentru ziua următoare și care, în caz afirmativ, va trimite o notificare fiecărui pacient în parte cu data și ora la care este programat. Este necesar, însă, ca execuția acestor funcții să nu îngreuneze funcționarea

aplicației. Acest rezultat poate fi obținut prin utilizarea unui instrument ce se va ocupa cu lansarea în execuție și gestionarea de task-uri care vor rula în background.

Am ales utilizarea instrumentului Hangfire, o bibliotecă open-source ce permite execuția de metode în exteriorul pipeline-ului destinat procesării solicitărilor HTTP din cadrul IIS. Aceste metode sunt invocate în cadrul unui thread ce rulează în background în cadrul aplicației. Acest thread este lansat în execuție în momentul pornirii aplicației, din cadrul funcției *Application_Start()*. Această funcție este executată de fiecare dată când este creată o nouă instanță a clasei *HttpApplication* de către arhitectura framework-ului ASP .NET, clasă utilizată pentru procesarea de solicitări HTTP.

Am configurat Hangfire pentru a utiliza aceeași bază de date în care sunt stocate și informațiile cu privire la entitățile utilizate în cadrul proiectelor. *Hangfire* va crea, în cadrul bazei de date, tabelele necesare gestionării de fire de execuție. Informațiile cu privire la funcțiile executate și la procesarea acestora sunt păstrate în tabelele create automat în urma configurării instrumentului, fapt ce le asigură persistența.

Asocierea task-ului dorit unui fir de execuție și lansarea sa în execuție se va realiza prin intermediul instrucțiunilor din figura 5.8.

```
var connectionString = "Data Source=.\\SQLEXPRESS;" +
    "Database=MedApplication.DataLayer.MedApplicationContext;" +
    "Trusted_Connection=True;";
Hangfire.GlobalConfiguration.Configuration.UseSqlServerStorage(connectionString);
_backgroundJobServer = new BackgroundJobServer();
RecurringJob.AddOrUpdate<SendNotificationsJob>(job => job.Execute(), Cron.Hourly);
```

Figura 5.8. Configurarea și lansarea în execuție a instrumentului Hangfire

Legătura cu server-ul Hangfire se realizează prin instanțierea unui obiect de tipul clasei *BackgroundJobServer*, pusă la dispoziție de către acest instrument. Ultima instrucțiune are ca rezultat înregistrarea metodei ce se ocupă de trimiterea de mesate text către pacienți ca task recurent ce trebuie executat în fiecare oră. Execuția acestuia oră de oră asigură faptul ca și pacienții pentru care s-au înregistrat programări în ziua curentă pentru ziua imediat următoare vor primi un mesaj de notificare.

Pentru trimiterea mesajelor de tip SMS am utilizat un API dezvoltat de Nexmo ce permite expedierea automată de mesaje text către numere de telefon din diverse părți ale lumii. Fiind utilizat, într-un mediu de testare acesta permite trimiterea unui număr limitat de mesaje.

Atât *Hangfire* cât și *Nexmo* au fost adăugate ca referințe în cadrul proiectului prin intermediul intrumentului NuGet Packages Manager din cadrul mediului de dezvoltare Visual Studio.

5.1.2. Prezentarea modelelor bazei de date

Datele cu privire la pacienții înregistrați în aplicație sunt păstrate în tabelul „Patients”, modelat de către clasa Patients. Se vor păstra în baza de date următoarele date personale ale pacientului: numele, prenumele, și CNP-ul pentru identificare, numărul de telefon și adresa de email ca date de contact, data nașterii, grupa sanguină și legăturile cu tabela ce conține tipurile de asigurări, cu tabela de adrese, respectiv cu tabela de conturi ale utilizatorilor. Aceste legături sunt marcate prin cheile străine InsuranceTypeId, AddressId și UserId, între fiecare din aceste tabele și tabela Patient existând o relație de tipul *One-to-One*. Legătura către tabela de programări este marcată prin poprietatea Appointments, de tipul ICollection<Appointment>, iar între cele două tabele exista o relație de tipul *One-to-Many*.

Tipurile standard de asigurare se găsesc înregistrate în tabela InsuranceTypes, iar clasa aferentă conține proprietățile referitoate la codurile si numele tipurilor de asigurări.

Adresele pacienților sunt stocate în tabela Adresses, având proprietățile: StreetName, Street No, Block, Entrance și Apartment. Aceasta se află într-o relație *Many-to-One* cu tabela pacienți, mai mulți pacienți putând avea același domiciliu.

Informațiile ce țin de conturile utilizatorilor sunt păstrate în tabela Users. Această tabelă conține câmpurile Username și EncryptedPassword, pentru stocarea numelui de utilizator si a parolei, precum și elementele necesare criptării parolei (Hash, Length, Iterations) și token-ul pe baza căruia se va verifica identitatea utilizatorului curent la fiecare cerere trimisă către server. Token-ul de autorizare va fi generat aleatoriu la crearea conturilor de utilizator, în momentul înregistrării unui nou pacient în baza de date.

Fiecărui utilizator al aplicației îi este asociat un rol, aceste roluri fiind stocate în tabela Roles. Pe baza acestor roluri se va realiza restricționarea accesului asupra metodelor din API în asa fel încât aparținătorul fiecărui rol să nu aibă acces decât la resursele care îi sunt alocate.

Datele cu privire la programările din sistem sunt stocate în tabela Appointments, ce conține câmpurile Date, ce reprezintă data pentru care este înregistrată programarea,

DateIssued, ce păstrează data la care a fost înregistrată programarea, un câmp care marchează faptul că pacientul a primit o notificare cu privire la programrea respectivă și un câmp ce semnalează dacă programarea se va reactiva automat în fiecare lună sau nu. Acesta din urmă spre deosebire de celelalte, acesta putând fi modificat doar de personalul cabinetului.

Duratele standard ale programărilor sunt păstrate în tabela AppointmentTimeSlots. Durata unei programări la cabinetul medicului de familie este de 15 minute, conform programului standard al acestuia – un program de 5 ore pe zi, 5 zile pe săptămână, cu o oră adițională pentru deplasări la domiciliu în fiecare zi, conform contractului cadru. De asemenea, este impunsă un număr maxim de 20 de consultații pe zi decontate de către Casa Națională de Asigurări de Sănătate.

Datele cu privire la programul medicului sunt păstrate în tabela EmployeeTimeSlot, ce conține drept informații numărul de ore lucrate pe zi, denumit Duration, având valoarea implicită 5, respectiv orele de începere a programului cabinetului din fiecare zi a saptămânii (MondayHours, TuesdayHours, WednesdayHours, ThursdayHours, FridayHours).

O schemă a tabelelor bazei de date, ce include și legăturile dintre acestea, este prezentată în figura 5.9.

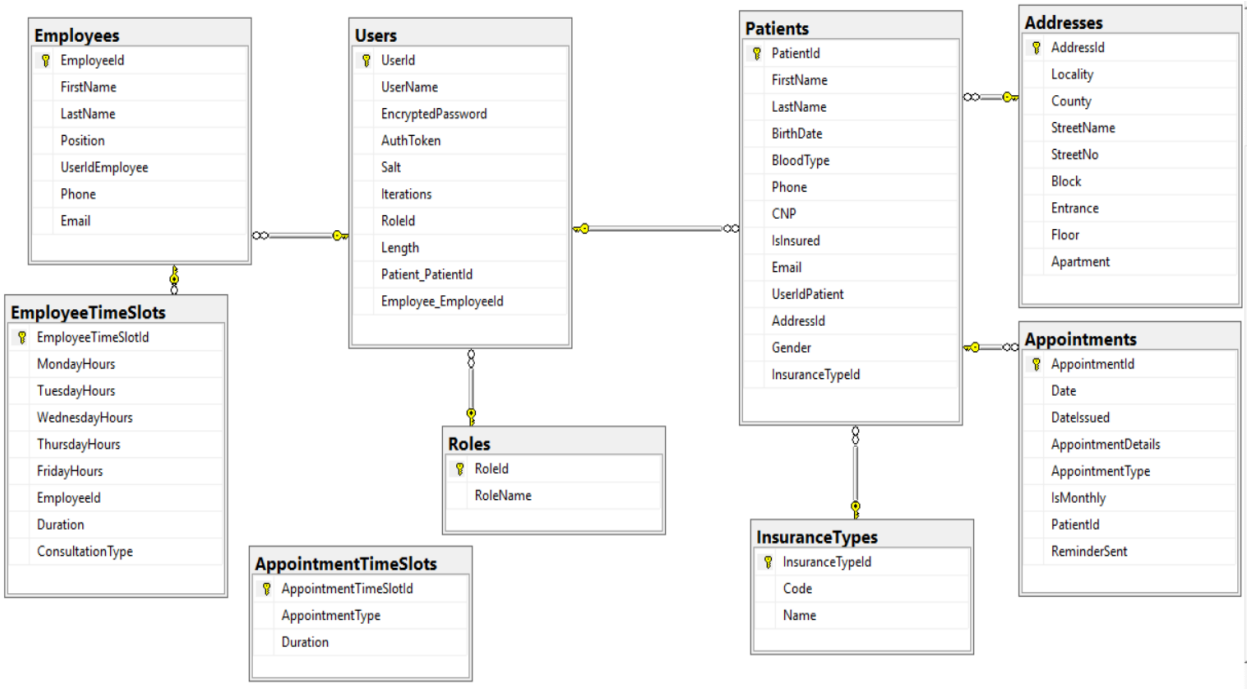

Figura 5.9. Tabelele bazei de date și legăturile dintre acestea

5.2. Prezentarea componentei front-end

Componenta front-end se regăsește în modulul **MedApplication.Frontend**, dezvoltat utilizând framework-ul Angular, versiunea 5.2 și limbajul TypeScript.

Pentru instalarea instrumentului Angular CLI și generarea fișierelor necesare cu ajutorul acestuia, am utilizat următoarele comenzi introduse în linia de comandă din cadrul Node.js:

- *npm install -g @angular/cli* – ceastă instrucțiune are ca efect instalarea uneltei Angular CLI pentru utilizarea ulterioară a comnezii *ng*;
- *ng new MedApplication.Frontend* – această instrucțiune va genera fișierele necesare proiectului în directorul specificat, anume MedApplication.Frontend;
- *ng serve* sau *npm start*– în urma executării oricărei din aceste instrucțiuni, prin intermediul instrumentului Node.js, aplicația Angular este lansată în execuție în cadrul unui server Web minimal, ce rulează local. În mod implicit, portul prin intermediul căruie se realizează comunicarea cu acest server este 4200, iar aplicația poate fi accesată la adresa *http://localhost:4200*

Unul din fișierele de configurare generate poartă denumirea de *package.json* și conține metadatele referitoare la proiectul generat și anume numele proiectului și versiuena acestuia, respectiv informații cu privire la dependițele din cadrul proiectului, printre care se numără versiunea de Angular și versiunea limbajului TypeScript utilizate, precum și versiunea bibliotecii PrimeNG utilizată. Acestea pot fi observate în figura 5.10.

```
"name": "med-application.frontend",
"version": "0.0.0",
"license": "MIT",
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build --prod",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
,,
"devDependencies": {
  "@angular/cli": "~1.7.3",
  "@angular/compiler-cli": "^5.2.0",
  "@angular/language-service": "^5.2.0",
  "typescript": "~2.5.3"
  "primeng": "^5.2.7"
```

Figura 5.10 – Fișierul de configurare package.json

În fișierul de configurare *angular-cli.json* se regăsesc informații cu privire la componentele principale ale aplicației. Fișierul HTML ce va conține pagina principală se numește *index.html*, iar fișierele CSS specifice componentelog grafice din cadrul PrimeNG utilizate în cadrul proiectului sunt specificate în secțiunea *styles*, după cum se poate vedea în figura 5.11.

```
"index": "index.html",
"main": "main.ts",
"styles": [
  "styles.css",
  "../node_modules/font-awesome/css/font-awesome.min.css",
  "../node_modules/primeng/resources/themes/kasper/theme.css",
  "../node_modules/primeng/resources/primeng.min.css"
```

Figura 5.11 – Fișierul de configurare angular-cli.json

6. Prezentarea cazurilor de utilizare

Modulul **MedApplication.Frontend** este alcătuit, la rândul său, din alte trei module: un modul principal, ce încorporează funcția de login în aplicație și logica de acces la resurse pe baza rolului utilizatorilor autentificați, respectiv alte două module, fiecare corespunzând celor două cazuri de utilizare. Primul caz de utilizare constă în accesarea aplicației de către pacienți, caz în care se vor încarca în browser doar fișierele ce definesc interfața grafică și funcțiile de solicitare de resurse corespunzătoare, iar cel de-al doilea caz constă în utilizarea de către personalul cabinetului, unde fiecare utilizator autentificat va avea acces doar la componentele și funcțiile ce sunt rezervate rolului asociat.

Separarea cazurilor de utilizare în cadrul logicii apliației s-a realizat prin definirea a trei module:

- un modul principal, ce conține componenta, serviciul și șablonul HTML utilizate pentru funcția de autentificare în aplicație care, la accesarea fiecărui modul, va verifica identitatea și rolul asociat utilizatorului autentificat;
- un modul ce va fi accesat doar de către pacienți și va conține funcționalitățile asociate acestora;
- un modul ce va putea fi accesat doar de angajați, cu funcționalitățile aferente acestui grup de utilizatori;

Router-ul este un element de tipul NgModule ce permite navigarea de la un șablon HTML la altul, conform adreselor de navigare configurate în cadrul acestuia.

În cadrul aplicației dezvoltate există trei rute principale:

- http://localhost:4200/cabinet - în urma navigării la această adresă va fi încărcat în browser modulul aferent funcționalităților puse la dispoziție personalului cabinetului;
- http://localhost:4200/pacient - prin accesarea acestei adrese vor fi aduse în browser fișierele din cadrul modulululului ce conține funcționalităților puse la dispoziție pacienților;
- http://localhost:4200/login - constituie pagina de pornire a aplicației, și se regăsește în modului principal al aplicației (*app.module*).

Rutele din cadrul proiectului au fost configurate în fișierul *app-rounting.module.ts,* conform figurii 6.1 din anexă, ce conține fragmentul de cod corespunzător.

6.1. Operațiunea de Login:

Fișierele care conțin template-ul și logica necesară operației de autentificare în cadrul aplicației sunt *login.component.ts, login.component.html,* respectiv *login.service.ts*.

În fișierul HTML sunt definite câmpurile în care utilizatorul își va introduce numele de utilizator și parola generate în momentul în care acesta a fost înregistrat în aplicație de către medicul de familie. La apăsarea butonului *Autentificare* din interfața utilizator, se va executa funcția *Login* definită în componenta *login.component.ts,* ce va apela metoda din serviciu ce va trimite credențialele introduse de către utilizator către API prin intermediul unei solicitări de tipul POST. În cadrul API-ului se va verifica dacă credențialele introduse se regăsesc în baza de date, iar în caz afirmativ, solicitarea lansată va primi drept răspuns token-ul de autorizare asociat contului de utilizator identificat prin aceste credențiale. În cadrul funcției *Login* se va păstrea în spațiul local de stocare al browser-ului (Local Storage) token-ul respectiv, precum și codul asociat rolului pe care utilizatorul îl are în cadrul aplicației. Totodată, conform acestui cod, se va realiza redirecționarea către mdulul aferent fiecărui rol. În caz contrar, serverul va întoarce un răspuns în care se va menționa că numele de utilizator și parola nu au fost regăsite în baza de date.

Pentru restricționarea accesului asupra resurselor la care pacienții, asistenții cabinetului sau medicii rezidenți nu ar trebui să aibă acces, am utilizat un serviciu denumit AuthGuard, ce implementează interfețele puse la dispoziție de către framework-ul Angular în acest scop, anume CanActivate, CanActivateChild și CanLoad. Prin implementarea funcțiilor din cadrul acestor interfețe se va lansa în execuție, la fiecare navigare către un nou URL, o metodă ce va verifica de fiecare dată dacă utilizatorul care realizează navigarea către acel URL este autorizat să acceseze resursa respectivă sau nu. Această verificare se realizează pe baza rolului asociat utilizatorului curent. Dacă utilizatorului nu îi este permis accesul la resursa respectivă, el va fi redirecționat către o pagină ce va conține mesajul „Pagina solicitată nu a fost găsită.”.

6.2. Utilizarea aplicației de către medic (admin)

Dacă utilizatorul este înregistrat în aplicație ca admin (medic), după operațiunea de login, acesta va fi redirecționat către URL-ul http://localhost:4200/cabinet, iar în browser va fi încărcat modul aferent cabinetului, intitulat *CabinetModule*, care se regăsește în directorul *cabinet*. Utilizatorul cu rol de admin va avea acces la toate resursele, putând vizualiza toate datele cu privire la pacienți/angajați/programări înregistrați/înregistrate în aplicație, va putea modifica aceste date sau elimina din sistem fiecare din pacienți/angajați/programări.

Acest modul, asemenea celui principal, va conține un nou router, numit CabinetRoutingModule ce se regăsește în fișierul *cabinet-routing.module.ts*, în cadrul căruia sunt definite adresele de navigare aferente cabinetului, anume:

- http://localhost:4200/cabinet/pacienti - în urma navigării, se va aduce în browser componenta *CabinetPatientsComponent*, ce se regăsește în fișiserul *cabinet-patients.component.ts*, respectiv șablonul HTML de care aceasta este conectată, anume *cabinet-patients.component.html* și serviciul aferent, *cabinet-patients.service.ts*. Conținutul HTML din pagina principală va fi înlocuit de elementele definite în acest template;
- http://localhost:4200/cabinet/pacienti/*id*, unde *id* va fi înlocuit, în momentul navigării, de id-ul pacientului selectat, reprezentat printr-un număr întreg și se vor încărca componentele și fișierele necesare, sau de cuvântul „nou”, dacă se dorește adăugarea unui nou pacient;
- http://localhost:4200/cabinet/angajati - în urma navigării la acest URL se vor încărca în browser componentele *cabinet-employees.component.ts* și *cabinet-employees.component.html*, ce definește template-ul HTML ce va fi redat în pagina principala;
- http://localhost:4200/cabinet/angajati/*id*, *id* fiind și în acest caz înlocuit de identificatorul unic al angajatului sau de cuvântul „nou”, dacă se dorește adăugarea unui nou angajat, iar template-ul necesar va fi adus în browser;
- http://localhost:4200/cabinet/programări - va aduce în pagina HTML principală template-ul *cabinet-appointments.component.html*, respectiv serviciul utilizat pentru trimiterea cererilor HTTP către API, *cabinet-appointments.service.ts*.

— http://localhost:4200/cabinet/programări/*id*, câmpurile componentei HTML aferente (*cabinet-appointment-add.component.html*) fiind completate cu datele programării, sau fiind goale, în cazul în care link-ul nu conține niciun id ci doar cuvântul „nou”;

— http://localhost:4200/cabinet/program – va încărca în browser componentele utilizate pentru vizualizarea și editarea programului cabinetului, anume *cabinet-dashboard.component.html*, respectiv *cabinet-dashboard.component.ts*;

Modul de configurare a rutelor este prezentat în figura 6.2 din anexă, ce conține fragmentul de cod aferent.

Componenta **CabinetPatientsComponent**, alături de template-ul HTML asociat**,** permite vizualizarea tuturor pacienților înregistrați în aplicație, organizați sub forma unui tabel. Aceste informații sunt obținute de la server în urma unei solicitări HTTP de tipul GET către adresa http://localhost:62405/api/CabinetPatients/GetAllPatients. Ultima coloană a acestui tabel va conține două butoane pentru operațiunile de ștergere și editare a înregistrărilor. Prin apăsarea butonului pentru editare, utilizatorul va fi redirecționat către o nouă pagină ce va conține câmpurile aferente informațiilor cu privire la pacientul respectiv. Această componentă utilizează în cadrul său serviciul **CabinetPatientsService,** definit în fișierul *cabinet-patients.service.ts* , ce conține metodele prin intermediul cărora se vor trimite solicitările HTTP către server-ul de date. Totodată, în cadrul acestui serviciu se va realiza și maparea conținutului răspunsului primit de la server pe tipurile de date definite în directorul *models*. Aceste tipuri de date sunt clase ce conțin întocmai câmpurile definite în cadrul claselor DTO.

În urma navigării către componenta **CabinetPatientAddComponent**, regăsită în fișierul *cabinet-appointments-add.component.ts*, utilizatorul va putea edita sau adăuga un pacient. Dacă parametrul *id* din cadrul URL-ului de navigare conține id-ul unui pacient, câmpurile din pagina de editare vor fi populate cu datele ce există deja în baza de date cu privire la pacientul respectiv. Acest lucru se realizează printr-o solicitare HTTP, de tipul GET, trimisă către server la adresa http://localhost:62405/api/CabinetPatients/GetPatientById/*id*, în momentul în care pagina a fost încărcată. Datele primite în cadrul răspunsului de la server sunt păstrate într-un obiect de tipul PatientDto ce va conține întocmai câmpurile definite în clasa DTO corespunzătoare entității Patient din cadrul componentei back-end.

Componenta **CabinetEmployeesComponent** se ocupă cu obținerea datelor cu privire la angajații înregistrați în aplicație folosind serviciul CabinetEmployeesService din în fișierul *cabinet-employees.service.ts*, precum și cu prezentarea acestora către utilizator prin intermediul template-ul HTML asociat. La fel ca în cazul precendent, aceste date pot fi șterse dau editate.

În cadrul componentei **CabinetEmployeesAddComponent**, dacă în adresa de navigare se regăsește un id, atunci se va apela metoda GetEmployeeById definită în serviciul CabinetEmployeesService și se vor popula câmpurile din șablonul HTML corespunzător cu datele respective. În caz contrar, câmpurile vor fi goale, în vederea adăugării unui nou angajat.

Componenta **CabinetAppointmentsComponent** prezintă, sub formă tabelară, datele cu privire la programările înregistrate în aplicație. Utilizatorului îi sunt puse la dispoziție 3 *tab*-uri, unul pentru vizualizarea programărilor din ziua curentă, unul pentru vizualizarea, ștergerea sau editarea programărilor viitoare și unul pentru vizualizarea programărilor anterioare. Datele cu privire la programări sunt obținute utilizând metodele GetCurrentAppointments, GetNextAppointments și GetPastAppointments, definite în serviciul CabinetAppointmentsService din fișierul *cabinet-appointments.service.ts*. Fiecare din aceste metode va trimite o solicitare HTTP către API în momentul în care pagina a fost încărcată, iar tabelele vor fi populate cu datele primite de la server.

Componenta **CabinetAppointmentAddComponent** se ocupă cu editarea și adăugarea programărilor din cadrul cabinetului. Aceasta utilizează același serviciu ca și componentele CabinetAppointmentsComponent și CabinetCurrentAppointmentsComponent. În momentul încărcării paginii asociate URL-ului de navigare, se vor trimite solicitări HTTP către server-ul de date pentru a obține zilele în care nu se mai pot înregistra programări. Aceste zile vor fi setate ca fiind nevalide în cadrul calendarului utilizat pentru selectarea datei, alături de zilele de sâmbătă și duminică din fiecare săptămână, întrucât acestea sunt zile nelucrătoare. Totodată, am restricționat intervalul de alegere a datei astfel încât să nu se poată înregistra o programare decât pe o perioadă de cel mult două luni de la data curentă.

Pentru a putea alege o dată în vederea înregistrării unei programări, utilizatorul va trebui mai întâi să selecteze tipul programării. Selectarea unui tip de programare va activa calendarul, prin intermediul căruia se va alege data dorită. În urma selectării unei date

valide, utilizatorului îi va fi generat un element de tip *dropdown* ce va conține toate orele disponibile, la care se poate înregistra o programare pentru ziua selectată. Aceste ore sunt calculate în funcție de înregistrările deja existente în baza de date pentru ziua respectivă. Pentru obținerea acestora, se trimite, în momentul selectării datei, o solicitare HTTP POST ce conține ziua aleasă de către utilizator, iar server-ul va returna drept răspuns toate orele la care există programări deja înregistrate în sistem pentru data respectivă. Vor fi populate opțiunile elementului de tip *dropdown* cu momentele de timp valide, ținând cont de lista returnată de către server.

De asemenea, personalul cabientului poate decide daca o programare se repetă lunar sau nu, marcând acest fapt prin selectarea unui element de tip *checkbox*. Dacă această opțiune este selectată, programarea va fi considerată ca fiind actuală în fiecare lună.

Componenta **CabinetDashboardComponent** permite medicului să modifice informațiile cu privire la programul cabinetului. Medicul poate seta orele de la care începe programul cabinetului pentru fiecare din zilele de lucru, precum și orele de la care este disponibil pentru vizite la domiciliu. Pentru trimiterea soliciărilor HTTP către server este utilizat serviciul CabinetEmployeesService, iar solicitările de obținere a informațiilor referitoare la program și de a modifica aceste informații sunt trimise către adresele din API http://localhost:62405/api/CabinetEmployees/GetAllTimeSlots, pentru obținerea datelor, respectiv http://localhost:62405/api/CabinetEmployees/SaveTimeSlots pentru salvarea modificărilor.

6.3. Utilizarea de către asistenți/medici rezidenți

Conform cerințelor funcționale impuse aplicației, asistenții din cadrul cabinetului nu pot vizualiza sau edita informațiile cu privire la angajați. Restricționarea accesului la resurse se realizează prin intermediului serviciul AuthGuard, ce verifică identitatea utilizatorului logat în aplicație în cadrul sesiunii curente și resursele la care are acces. Asistenții sau medicii rezidenți vor putea accesa accesa doar resursele ce se ocupă cu gestionarea programărilor și a pacienților, prezentate anterior.

6.4. Utilizarea aplicației de către un pacient

Dacă utilizatorul este înregistrat în aplicație ca pacient, din pagina de login, acesta va fi redirecționat către adresa http://localhost:4200/pacient, iar în broswer va fi încărcat modulul corespunzător acestui caz de utilizare, anume PatientModule, din directorul *patient*. Și acest modul, la fel ca cel corespunzător cabinetului, conține un router numit PatientRoutingModule, regăsit în fișierul *patient-routing.module.ts*, modul ce va permite navigarea între componentele aferente. Pacientul va putea vedea doar programul cabinetului, respectiv datele de contact ale personalului cabinetului - numele, prenumele, numărul de telefon și adresa de email – la care acesta poate apela în cazul unei urgențe, precum și programările înregistrate de el în aplicație, ordonate descrescător în funcție de data programărilor.

Rutele definite în cadrul fișierului sunt următoarele:

— http://localhost:4200/pacient/program - navigarea la acest URL va duce la înlocuirea componentelot HTML din pagina principală cu elementele definite în cadrul șablonului HTML din fișierul *patient-dashboard.component.html* către care face referire componenta PatientDashboardComponent, asociată acestei rute;

— http://localhost:4200/pacient/programari - navigarea către acest URL, având asociată componenta PatientAppointmentsComponent și șablonul regăsit în fișierul *patient-appointments.component.html* va permite urilizatorului să vizualizeze programările înregistrate în aplicație;

— http://localhost:4200/pacient/programari/editeaza/*id* – în urma navigării la acest URL, pacientul va accesa componenta PatientAppointmentAddComponent, ce permite utilizatorului sa adauge o programare nouă sau să editeze una deja existentă, prin intermediul șablonului HTML din fișierul *patient-appointment-add.component.html*.

Definirea acestor rute este prezentată în figura 6.3 din anexă, ce conține fragmentul de cod aferent.

Componenta **PatientDashboardComponent** este utilizată pentru a prezenta utilizatorului informațiile cu privire la programul cabinetului și la datele de contact ale angajaților din cadrul cabinetului, la care pacientul poate apela în cazul unei urgențe medicale. Aceste informații sunt puse la dispoziția utilizatorului în urma a două solicitări

HTTP de tipul GET trimise către serverul de date, una pentru a obține informațiile cu privire la angajați, iar cea de-a doua, pentru a obține informațiile referitoare la programul cabinetului. Ambele solictări sunt trimise către server folosind serviciul PatientAppointmentsService ce se regăsește în fișierul *patient-appointments.service.ts*. Prima solicitare este trimisă către adresa http://localhost:62405/api/PatientAppointments/GetAllTimeSlots, iar cea de-a doua este trimisă către adresa http://localhost:62405/api/CabinetEmployees/GetAll Employees.

Componenta **PatientAppointmentsComponent** permite utilizatorului curent să vizualizeze programările pe care le-a înregistrat în aplicație, sub forma unui tabel, în ordine descrescătoare în funcție de data fiecăreia. Ultima coloană a tabelului conține butoanele prin intermediul cărora se poate edita sau șterge o programare. În cazul editării, pacientul va fi redirecționat către pagina http://localhost:4200//pacienti/programari/editeaza/*id*, unde câmpurile existente vor fi populate cu informațiile aduse din baza de date în urma unei solicitări GET conform id-ului specificat în URL, la adresa din API http://localhost:62405/api/CabinetAppointments/GetAppointmentById/*id*.

În cadrul componentei **PatientAppointmentsAddComponent**, dacă se dorește adăugarea unei noi programări, pacientul va completa câmpurile puse la dispoziție. La fel ca în cazul înregistrării unei programări din poziția unui angajat, pacientul va trebui să selecteze mai întâi tipul de programare din elementul *dropdown* pus la dispoziție, iar în urma selecției, va putea alege o dată pentru care dorește să înregistreze programarea. În momentul selectării datei, prin intermendiul unui eveniment marcat de schimbarea conținutului câmpului, se va lansa o solicitare HTTP de tipul POST către server-ul de date la adresa http://localhost:62405/api/PatientAppointments/GetUnavailableTimes, solicitare ce va conține data selectată de pacient. Serverul va oferi drept răspuns toate orele pentru care au fost înregistrate programări în ziua respectivă, urmând ca, în cadrul componentei, să se elimine aceste ore din lista celor disponibile. Programarea va fi înregistrată prin apăsarea butonului *Adaugă programare*, ce va apela metoda Save. Aceasta va avea ca efect trimiterea unei solicitări POST către API, la adresa http://localhost:62405/api/PatientAppointments/SaveAppointment, ce va conține obiectul de tip *AppointmentInsertPatient* cu informațiile introduse de utilizator. Butonul *Resetează câmpuri* va șterge datele introduse de pacient, aducând câmpurile la starea inițială.

7. Concluzii

7.1. Rezultatele proiectului

Acest proiect are drept rezultat final o soluție software ce vine în ajutorul medicului de familie, rezolvând problema gestionării programărilor din cadrul cabinetului prin automatizarea acestui proces. Totodată, aplicația dezvoltată aduce drept beneficiu pacienților un sistem de notificare prin SMS cu privire la programările înregistrate, cu o zi înainte de data la care sunt stabilite consultațiile.

Procesul de dezvoltare a acestei aplicații a reprezentat o oportunitate de studiu asupra tehnologiilor utilizate și aprofundare a conceptelor de programare Web cu care am luat contact în timpul facultății.

7.2. Îmbunătățiri și dezvoltări ulterioare ale aplicației

O îmbunătățire ce poate fi adusă aplicației constă în înlocuirea sistemului de mesagerie cu unul mai performant, contra cost. O posibilitate în vederea realizării acestei schimbări o constituie Twilio, un API ce permite expedierea automată de mesaje text către numerele de telefon ale pacienților fără nicio restricție asupra demensiunii mesajelor sau a numărului de mesaje trimise din cadrul aplicației.

Apliația Web a fost în așa fel construită încât poate permite dezvoltarea ulterioară de module în cadrul API-ului sau aplicații de sine stătătoare care să utilizeze resursele din cadrul API-ului. O posibilitate ar fi dezvoltarea unei aplicații de tip mobile folosind uneltele puse la dispoziție de mediul de programare Visual Studio și platforma Xamarin ce permite dezvoltarea de aplicații Android, iOS sau Windows folosind limbajul C#.

8. Bibliografie

1. Brad Dayley (2014). *Node.js, MongoDB, and AngularJS Web Development*. Addison-Wesley Professional, pag. 396 – 398.
2. Eysenbach, G. (2001). *What Is E-Health?* Journal of Medical Internet Research 3.2. Articol Internet - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1761894/. Accesat: 2018
3. Fernando Monteiro (2014). *Learning Single-page Web Application Development*. Packt Publishing. pag. 10 – 17.
4. *From Innovation to Implementation - eHealth in the WHO European Region.* Documentație Internet - http://www.euro.who.int/en/health-topics/Health-systems/e-health/publications. Accesat: 2018.
5. Guy Paré, Marie-Claude Trudel, Pascal Forget, *Adoption, Use and Effects of an e-appointment System: Results of a Quebec Showcase Project*, Canada Research Chair in Information Technology in Health Care, pag. 5 – 6.
6. Jan L. Harrington (2016). *Relational Database Design and Implementation*. Elsevier Inc., pag. 15 – 28.
7. Jon Galloway, Brad Wilson, K. Scott, Allen David Matson (2014). *Professional ASP.NET MVC 5*. John Wiley & Sons, Inc., pag. 159 – 170.
8. Joseph Albahari, Ben Albahari (2016). *C# 6.0 in a Nutshell*. O'Reilly Media, Inc., pag. 335 – 344.
9. Maria Gheorghe-Moisii, Eugenia Tîrziu (2016). *Abordarea domeniului eHealth privind evaluarea soluțiilor de eHealth*. Revista Română de Informatică şi Automatică, vol. 26, nr. 3, 2016, pag. 53.
10. Muhammad Imran Hussain and Naveed Dilber (2014). *Restful Web Services Security by Using ASP.NET Web API MVC Based.* Journal of Independent Studies and Research – Computing, vol. 12, nr. 1, pag. 2-7.
11. Nate Murray, Ari Lerner, Felipe Coury, Carlos Taborda (2017). *Ang-book 2: The Complete Book on Angular 2*. Fullstack / Gistia, pag. 78 – 80.
12. Nate Murray, Felipe Coury, Ari Lerner, Carlos Taboda (2017). *ng-book. The Complete Guide to Angular 5.* Fullstack.io, pag. 4 – 10, 216 – 222.

13. Nușa Dumitriu-Lupan, Rodica Pintea, Adrian Niță, Mioara Niță, Cristina Sichim, Nicolae Olăroiu, Mihai Tătăran, Petru Jucovschi Tudor-Ioan Salomie (2007). *Introducere în Programarea .Net Framework*. BYBLOS SRL, pag. 40-60.
14. Oh, Hans & Rizo, Carlos & Enkin, Murray & Jadad, Alejandro. (2005). *What is eHealth? A systematic review of published definitions*. World hospitals and health services: the official journal of the International Hospital Federation, pag. 32-40.
15. Ryan Asleson, Nathaniel T. Schutta (2005). *Foundations of Ajax*. Apress, pag. 10 – 20.
16. Sergey Barskiy (2015). *Code-First Development with Entity Framework*. Pakt Publishing, pag. 35 – 55.
17. Sudheer Jonna, Oleg Varaksin (2017). *Angular UI Development with PrimeNG*. Packt Publishing. pag. 43 – 46.
18. Bipin Joshi (2016). *Beginning SOLID Principles and Design Patterns for ASP.NET Developer*. Apress, pag. 309 – 320.

9. Anexă

Lista tabelelor

Lista figurilor

Cod sursă:

```
const routes: Routes = [
  {
    path: 'cabinet',
    loadChildren: 'app/cabinet/cabinet.module#CabinetModule',
    canLoad: [AuthGuard]
  },
  {
    path: 'pacient',
    loadChildren: 'app/patient/patient.module#PatientModule',
    canLoad: [AuthGuard]
  },
  {
    path: '**',
    component: PageNotFoundComponent
  }

];
```

Figura 6.1. Rutele din cadrul *AppRoutingModule*

```
const cabinetRoutes: Routes = [
  {
    path: '',
    component: CabinetComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        canActivateChild: [AuthGuard],
        children: [
          { path: 'pacienti', component: CabinetPatientsComponent },
          { path: 'angajati', component: CabinetEmployeesComponent },
          { path: 'programari', component: CabinetAppointmentsComponent },
          { path: 'pacienti/editeaza/:id', component: CabinetPatientAddComponent },
          { path: 'angajati/editeaza/:id', component: CabinetEmployeeAddComponent },
          { path: 'program', component: CabinetDashboardComponent },
          { path: 'programari/editeaza/:id', component: CabinetAppointmentAddComponent}
        ]
      }
    ]
  }
];
```

Figura 6.2. Rutele din cadrul *CabinetRoutingModule*

```
const patientRoutes: Routes = [
  {
    path: '',
    component: PatientComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        canActivateChild: [AuthGuard],
        children: [
          { path: 'programari', component: PatientAppointmentsComponent },
          { path: 'programari/editeaza/nou', component: PatientAppointmentAddComponent },
          { path: 'program', component: PatientDashboardComponent },
          { path: '', redirectTo: 'program', pathMatch: 'full' }
        ]
      }
    ]
  }
];
```

Figura 6.3. Rutele din cadrul *PatientRoutingModule*