UNIVERSITATEA "ALEXANDRU IOAN CUZA", IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

MedPortal

propusă de

Roland Iordache-Țiroiu

Sesiunea: *Iulie, 2019*

Coordonator științific

Drd. Florin Olariu

UNIVERSITATEA "ALEXANDRU IOAN CUZA", IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

MedPortal

propusă de

Roland Iordache-Țiroiu

Sesiunea: *Iulie, 2019*

Coordonator științific

Drd. Florin Olariu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Nume și prenume: Drd. Florin Olariu

Data _____

Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul Iordache-Țiroiu Roland, domiciliul în Bld. Primăverii, nr. 13, orașul Iași, județul Iași, România, născut la data de 10 februarie 1996, identificat prin CNP 1960210226700, absolvent al Universității „Alexandru Ioan Cuza" din Iași, Facultatea de Informatică, specializarea informatică, promoția 2016-2019, declar pe propria răspundere, cunoscând consecințele falsului in declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul *MedPortal*, elaborată sub îndrumarea Drd. Florin Olariu, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop. Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi _____ Semnătură student _____

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul *MedPortal*, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, _____

Absolvent *Roland Iordache-Țiroiu*

_____

(semnătura în original)

Cuprins

Introducere

Avansul tehnologic ne-a ușurat stilul de viață, inovațiile luând amploare în diverse domenii de la descoperirea electricității, procesarea mâncării, eficientizarea comunicării la distanță, oportunitatea pentru noi locuri de muncă în domeniu și multe alte arii diverse, devenind parte integrală din viața fiecăruia. În continuare, tehnologia este într-un proces de evoluție, fiind un domeniu accesibil, oferindu-ne noi metode de a reduce din piedica pe care și-o amprentează birocrația asupra societății. În zilele noastre, întâlnim și ramuri în care procesul tehnologic stagnează și așteaptă îmbunătățiri.

Programarea online la medic este oferită de puține spitale, procesul fiind îngreunat din cauza soluțiilor oferite, fie telefonic, fie la recepția spitalului. De asemenea, prin aceste soluții nu putem alege medicii pe baza unor recenzii.

Istoricul medical nu poate fi vizualizat decât de medicul de familie, atât ceilalți medici, cât și pacientul nu au această posibilitate, reducând șansele unui diagnostic corect din cauza lipsei de informare.

În prezent, nu există o aplicație care să ofere posibilitatea căutării unor eventuali donatori, cât și înscrierea ca voluntar pentru donarea de sânge.

Aceste probleme sunt cauzele nevoii acestui proiect, obiectivele generale ale lucrării fiind: simplificarea și eficientizarea modului în care este păstrat istoricul medical, a modului în care se realizează programările la medici, cu posibilitatea evaluării acestora, ajutând la îmbunătățirea procesului de selecție și adăugării unui modul de donare, unde pacienții nu mai sunt nevoiți să caute donatori pe rețelele de socializare, aplicația oferind șansa de a căuta un donator direct prin intermediul acesteia.

Pentru a realiza această lucrare, am împărțit timpul pe funcționalități principale (modulul de donare, modulul de programări la medic, sistemul de recenzii, istoricul medical, autentificare și autorizare, înregistrare, editare cont) și alte funcționalități secundare, estimând dupa complexitatea fiecăruia cam cât ar trebui sa dureze. Fiecare funcționalitate a fost implementată sincron, atât pentru pacient cât și pentru medic, fiind asignată unui card aferent (neimplementat, implementez acum, testare, implementat). Am început cu cercetarea a ceea ce

există deja pe piață (analiza), proiectarea (cum voi construi aplicația), implementarea și în cele din urmă testarea.

Aplicația web *MedPortal* oferă posiblitatea creării unui cont, fie ca pacient, fie ca medic și își propune să faciliteze relația dintre medici și pacienți. Modulul pentru medici le va pune la dispoziție trei funcționalități principale: crearea unui program flexibil pe zile și ore, vizualizarea programărilor, cât și adăugarea istoricului medical pentru pacienți. Modulul pentru pacienți le va oferi patru funcționalități principale: programarea online la medic, consultarea propriului istoric medical, sistem de recenzii care ajută la facilitarea alegerii unei programări la medic și un modul de donare. Precizez că în această lucrare și în ilustrarea figurilor, numele și datele sunt fictive și nu reprezintă date cu caracter personal.

În cele ce urmează, vor fi prezentate pe larg următoarele trei capitole principale: problema descrisă succint mai sus din cauza căreia a fost necesară această lucrare, abordările anterioare și descrierea soluției.

Contribuții

- Implementarea dosarului electronic de sănătate, ușurând gestionarea relației medic-pacient și făcând posibilă vizualizarea acestuia de către pacient în orice moment
- Modulul pentru donarea sângelui, funcționalitate care ajută la găsirea rapidă a voluntarilor pentru grupa de sânge în cauză
- Sistem de programări online a vizitelor la medic, care aduce numeroase beneficii pentru pacienți
- Sistem de recenzii, pacienții având posibilitatea căutării unui medic bazat pe anumite criterii și experiențe anterioare
- O arhitectură care poate fi ușor extinsă prin adăugarea altor funcționaliăți la cererea pieței și a utilizatorilor
- Design intuitiv și ușor de utilizat
- Utilizarea criptografiei pentru securitatea partajării datelor în rețea, autentificare și autorizare
- Oferirea unei aplicații ca întreg pentru oferirea unor funcționalități diferite, dar din aceeași arie
- Design responsive
- Predicție pentru cancer la sân bazată pe rezultatele în urma unui RMN

1 Descrierea problemei

Ideea de bază a aplicației a pornit de la faptul că pe rețelele de socializare oamenii distribuie cereri, solicitând ajutorul pentru cei care au nevoie de transfuzii de sânge. Extrapolând de la această idee, am întâmpinat alte probleme de ordin medical, la care am reușit să găsesc soluții optime, precum păstrarea și mentenanța istoricului medical, programările la medic și un sistem de recenzii.

Programările la medic se fac astăzi în general fie la recepția spitalului, fie telefonic. În primul caz, minusurile majore sunt atât timpul pierdut pe drumul până la recepția spitalului, cât și obligația de a ajunge într-un interval orar stabilit. În al doilea caz, întâlnim dificultatea consultării clare a programului medicilor, neavând posibilitatea de ne informa despre medicul respectiv. De partea cealaltă, evidența trebuie ținută și actualizată de persoanele avizate, îngreunând procesul și cauzând apariția de erori.

Deseori am fost puși în situația în care a trebuit să actualizăm "baza de date" a medicului de familie, aducându-i hârtiile primite de la alți medici în legătură cu consultările, medicația oferită și problemele constatate. Medicul de familie ține evidența istoricului medical pe baza unui dosar care trebuie să conțină toate aceste informații. De asemenea, medicii nu pot ști istoricul medical al pacientului, fiind dificilă diagnosticarea și administrarea unui tratament. Așadar, se remarcă o centralizare slab organizată, cât și păstrarea unor informații vitale într-un mod nu tocmai bun.

În lume, numărul donatorilor de sânge e preponderent mai mic față de numărul celor care au nevoie urgentă.. De exemplu, în țara noastră se donează 65% din totalul necesar, înregistrându-se un deficit major. Având certitudinea că nu este vorba despre conștiința civică, ci despre campanii greșite de promovare a donării de sânge și despre nevoia de a avea pe voluntari și pe cei care au nevoie de transfuzii de sânge adunați într-un singur loc.

Așadar, încă nu există o aplicație care să ofere rezolvarea acestor probleme, comunicarea dintre părțile implicate realizându-se anevoios.

2 Abordări anterioare

În prezent, nu există o platformă la care te poți înscrie pentru a deveni voluntar și a dona sânge. În unele state din America, dar și în Anglia au fost implementate platforme care încurajează donarea de sânge, dar problema care survine este neputința alegerii momentului oportun donării (din postura de voluntar), cât și selecția, dar și căutarea persoanelor cu o anumită grupă de sânge (din postura de pacient).

Vitalant[1] este o aplicație web care activează în S.U.A., aceasta oferă toate detaliile necesare donării de sânge, cât și cea mai apropiată locație unde poți dona. *MedPortal* vine cu această soluție, oferind posibilitatea selectării, oferirea unor date concrete legate de disponibilitate, grupă de sânge și alte detalii, care simplifică procesul în sine.

Cu privire la sistemul de programări online și recenzii, lucrurile au început să se dezvolte mai rapid. Spitale private din țară și de peste hotare au adoptat deja această idee, de exemplu spitalul privat *Regina Maria*[2]. Deoarece numărul acestora este foarte mic, persistă problema centralizării programărilor, fiind imposibilă vizualizarea și compararea recenziilor de la diferite spitale, cât și alegerea medicilor după anumite criterii.

Istoricul medical păstrează informații vitale printre care alergii și intoleranțe diagnosticate, imunizări, servicii medicale oferite, intervenții și proceduri efectuate, istoricul de boli/diagnostice și așa mai departe. Practic, conține informații vitale și necesare, care ajută la următoarele consultări la medic.

S-a încercat implementarea dosarului electronic de sănătate în România în anul 2014[3], dar fără succes, medicii fiind nemulțumiți de serviciile portalului. Aceștia trebuiau să primească acordul pacienților înainte să modifice istoricul medical, dar întâmpinau și probleme din cauza difciultății de accesare a platformei. În consecință, *MedPortal* vine ca o soluție cu o platformă ușor de accesat, dar și cu eliminarea acestui procedeu dintre medic-pacient care îngreunează situația, oferind altă alternativă pentru securitatea accesării acestuia.

[1] https://vitalant.org/
[2] https://www.reginamaria.ro/
[3] http://www.des-cnas.ro/pub/

În concluzie, lucrarea de față aduce facilități noi, sau parțial noi, atât pentru România, cât și pentru restul țărilor, obiectivul principal fiind centralizarea funcționalităților și facilitarea comunicării dintre pacient și medic, cât și oferirea unui modul pentru donarea de sânge direct pe o platformă.

3 Descrierea soluției

Soluția pe care am propus-o rezolvă problemele descrise în capitolele anterioare. *MedPortal* este o platformă ușor de utilizat, atât de medici cât și de pacienți.

3.1 Tenologii și paradigme utilizate

- Angular 7
- ML.NET
- Bootstrap 4
- .NET Core 2.2
- C#
- SQL Server
- Onion Architecture
- Integration Tests
- Sonar Cloud
- CQS
- Generic Repository Pattern
- MVC
- Entity Framework (ORM)
- EF Code First
- Fluent Validations cu ajutorul Fluent API
- Migrations
- Defensive Coding
- IoC - Dependency Injection
- Async programming
- Swagger
- REST Architecture

3.2 Arhitectura aplicației

În Figura 1 de mai jos, avem nivelul 1 al arhitecturii C4 Model[4], format din sistemul principal (aplicația) și utilizatorii acesteia - pacienții și medicii.

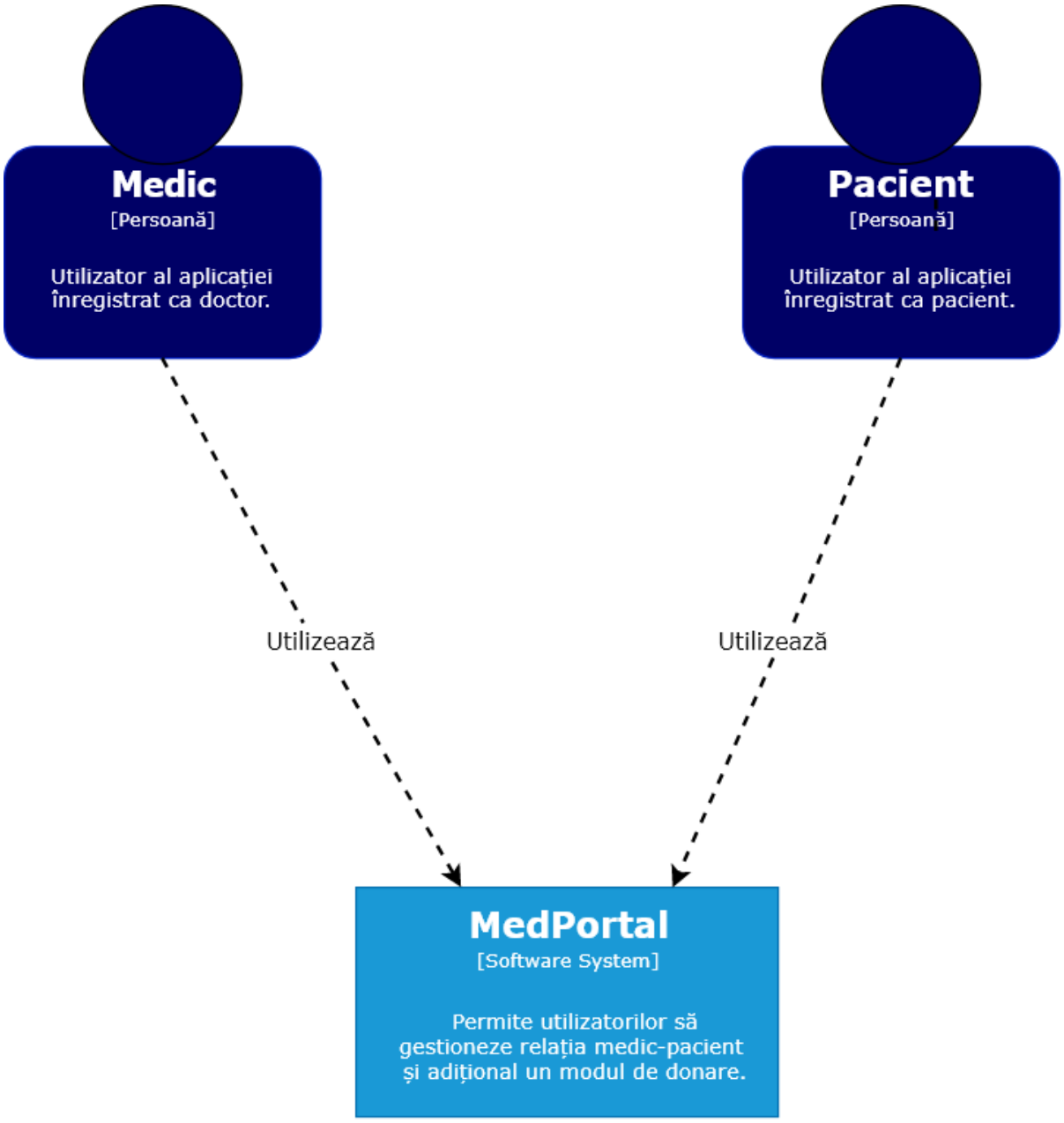


Figura 1: Nivelul 1 – C4 Model

[4] https://www.infoq.com/articles/C4-architecture-model/

În Figura 2 de mai jos, avem nivelul 2 al arhitecturii C4 Model[5], compus din utilizatorii aplicației și sistemul extins. Acesta este format din partea de client creată cu Angular, Web API-ul care primește cererile de la client și baza de date. După autentificare, server-ul trimite JWT către client, astfel încât orice cerere la Web API pe care o va mai face, aceasta trebuie să fie autorizată cu ajutorul token-ului primit de la server la autentificare. Tocmai din acest motiv clientul și serverul folosesc continuu la fiecare cerere, token-ul pentru autorizare.

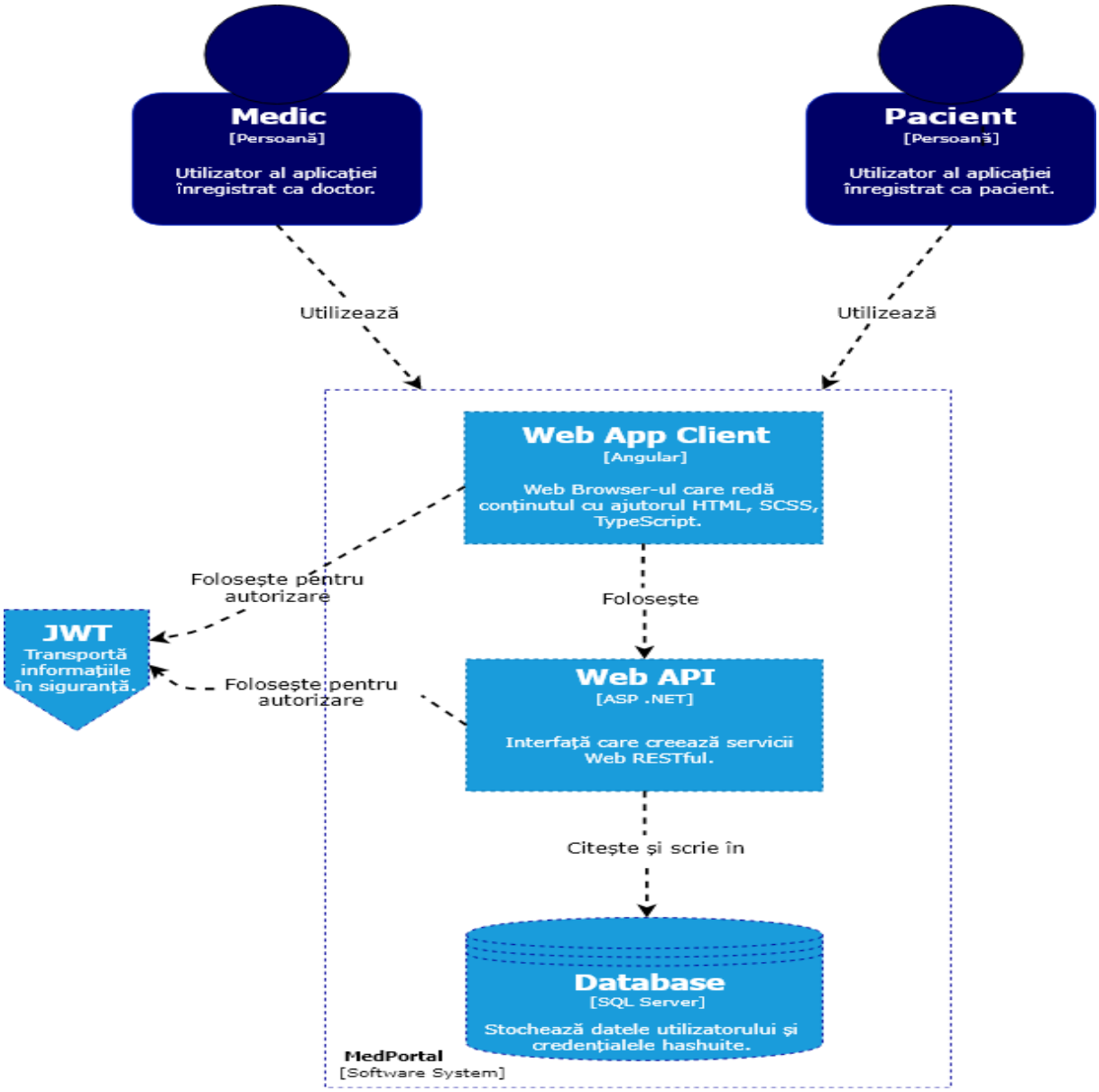
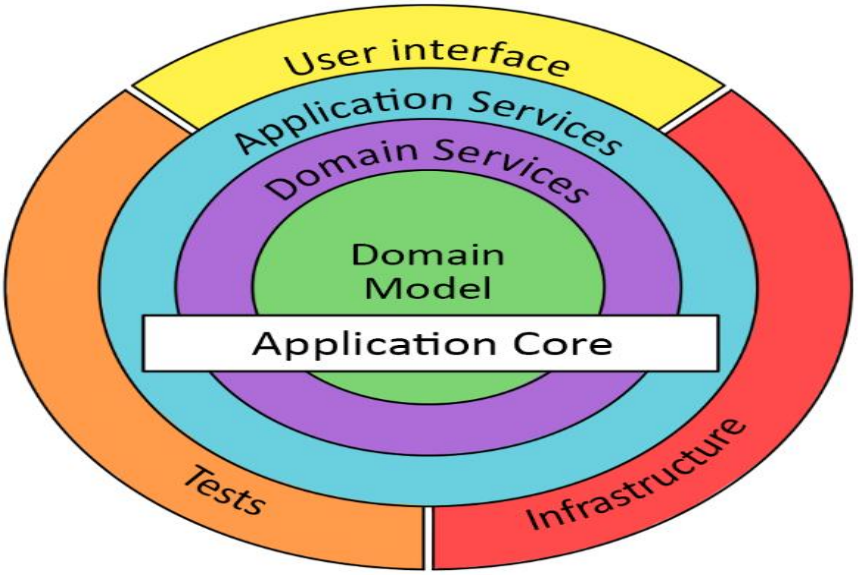


Figura 2: Nivelul 2 – C4 Model

[5] https://www.infoq.com/articles/C4-architecture-model/

În Figura 3 de mai jos, este prezentată arhitectura Onion cu ajutorul căreia am structurat componentele aplicației. În centru se află entitățile care reprezintă tabelele bazei de date. Domain Services sunt reprezentate în arhitectura aplicației de Repository, operațiile CRUD (create, read, update, delete) pentru manipularea datelor din baza de date. Application Services sunt reprezentate de controllere (serviciile apelate de client). Pe ultimul nivel stă interfața Web (UI-ul) și testele. Din miezul arhitecturii către exterior crește cuplajul, acestea din urmă depinzând de cele din interior.



[6]

Figura 3: Arhitectura Onion

În concluzie, am folosit arhitectura client-server cu servicii HTTP RESTful prezentată în Figura 1 și Figura 2, arhitectura Onion, o arhitectură flexibilă, portabilă și ușor de testat.

[6] https://dzone.com/articles/onion-architecture-is-interesting

3.3 Implementarea aplicației

În cele ce urmează, vă voi prezenta pe module funcționalitățile de bază ale aplicației și detalii legate de implementare acestora, atât pe front-end, cât și pe back-end.

Pagina principală a aplicației conține detalii cu referire la folosul utilizării acesteia. De asemenea, după cum se poate vedea în Figura 4 de mai jos, este calea principală către logarea sau înregistrarea ca pacient sau ca medic.

Figura 4: Pagina principală a aplicației

3.3.1 Detalii principale legate de implementare

Prima etapă a fost analiza a ceea ce doresc să implementez. Practic, am cercetat piața, ceea ce există deja sau ce necesită îmbunătățiri, după care am definit cerințele sistemului, neinteresându-mă de modul în care vor fi îndeplinite.

În faza de proiectare, cea de-a doua etapă, am stabilit arhitectura, modul de comportare și am definit mock-up-uri pentru front-end.

Ultimele două etape vor fi descrise în cele ce urmează și anume implementarea propriu-zisă și nu în ultimul rând, testarea.

La baza proiectului stă arhitectura client-server și arhitectura Onion[7], având ca scop decuplarea componentelor, ușurarea testării și respectarea IoC[8]. Nivelele acesteia sunt în această ordine: modelele, repository, controllerele (serviciile) și în final UI[9] alături de Integration Tests[10].

Pentru partea de front-end am folosit Bootstrap[11] 4 și am învățat un framework[12] nou pentru mine – Angular[13] 7. Pentru back-end am folosit .NET Core 2.2 (C#) și ORM-ul[14] SQL Server[15] pentru lucrul cu baza de date. Am folosit Code First[16] pentru a crea tabelele în baza de date, scriind cod C#. În cazul în care apăreau modificări, am folosit migrări[17]. Astfel, mi-a ușurat munca, neavând de-a face cu schimbările și legăturile între tabelele din baza de date.

Am folosit pattern-ul Generic Repository[18] după cum se poate vedea în Figura 5, templetizând modelele pentru a evita codul duplicat.

```
public interface IReadOnlyRepository<T>
{
    Task<List<T>> GetAllAsync(string[] includes);
    Task<T> GetByIdAsync(Guid id);
    Task<PagingResult<T>> GetAllPageAsync(int skip, int take);
    Task<PagingResult<T>> GetByFilter(Expression<Func<T, bool>> predicate, int skip, int take, string[] includes);
    Task<List<T>> GetByConditionsAsync(Expression<Func<T, bool>> predicate, string[] includes);
}
```

Figura 5: Generic Repository Pattern

[7] https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understanding-onion-architecture.html
[8] https://www.tutorialsteacher.com/ioc/inversion-of-control
[9] https://searchmicroservices.techtarget.com/definition/user-interface-UI
[10] https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-2.2
[11] https://getbootstrap.com/docs/4.3/getting-started/introduction/
[12] https://whatis.techtarget.com/definition/framework
[13] https://angular.io/docs
[14] https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a
[15] https://www.microsoft.com/en-us/sql-server/sql-server-2019
[16] https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx
[17] https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/
[18] https://dotnettutorials.net/lesson/generic-repository-pattern-csharp-mvc/

Pentru separarea conceptelor în operații de editare (delete, put, post) și operații de citire (get) am folosit CQS[19], după cum se poate vedea în Figura 6 de mai jos.

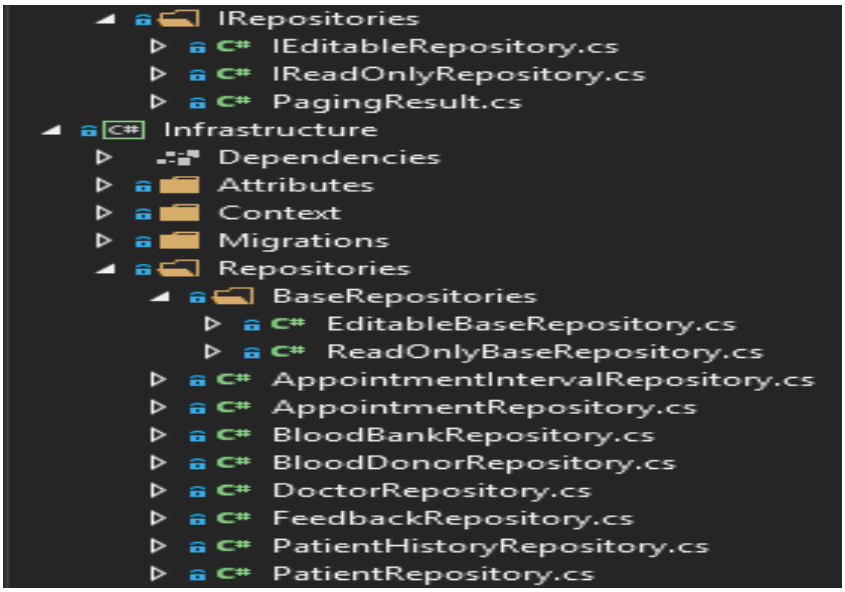


Figura 6: CQS

Pe cât posibil am încercat să pun în practică defensive coding[20], aruncând excepții oricând a fost nevoie, după cum se poate vedea în Figura 7 mai jos.

[19] https://www.dotnetcurry.com/patterns-practices/1461/command-query-separation-cqs
[20] https://medium.com/web-engineering-vox/the-art-of-defensive-programming-6789a9743ed4

```
// GET api/Doctors/page/name/10/10
[HttpPut("page/{skip}/{take}")]
[NoCache]
[ProducesResponseType(typeof(List<Doctor>), 200)]
[ProducesResponseType(typeof(ApiResponse), 400)]
public async Task<ActionResult> DoctorsNamePage([FromBody]DoctorFilterModel filter, int skip, int take, [FromHeader(Name = "Authorization")]string value)
{

    var token = new JwtSecurityTokenHandler().ReadJwtToken(value);
    var issuer = token.Claims.First(claim => claim.Type == "iss").Value;
    var audience = token.Claims.First(claim => claim.Type == "aud").Value;
    if (issuer != "MyIssuer" && audience != "MyAudience")
    {
        return Unauthorized();
    }

    try
    {
        string[] includes = {  };
        var pagingResult = await _repository.GetByFilter(FilterDelegate(filter.Name, filter.Hospital, filter.Speciality, filter.City), skip, take, includes);
        Response.Headers.Add("X-InlineCount", pagingResult.TotalRecords.ToString());
        return Ok(pagingResult.Records);
    }
    catch
    {
        return BadRequest(new ApiResponse { Status = false });
    }
}
```

Figura 7: Defensive coding

În Figura 7 de mai sus observăm și operatorii *async* și *await*, cu ajutorul cărora am făcut apeluri asincrone. Pe partea de front-end, atunci când voi face un apel la Web API-ul[21] meu, am folosit operatorul *subscribe* după cum se poate vedea în Figura 8 de mai jos. Acesta funcționează exact ca în viața reală, atunci când ne abonăm la citirea unui newsletter[22]. Vom fi înștiințați prin e-mail când va apărea un nou articol, atâta timp cât nu ne dezabonăm (operatorul *unsuscribe*).

[21] https://www.tutorialsteacher.com/webapi/what-is-web-api
[22] https://www.bigcommerce.com/ecommerce-answers/what-is-newsletter-marketing/

```
scheduleAppointment() {
    this.isExpired = this.userService.isExpired();
    if(this.isExpired) {
      this.authService.logout();
    }
    this.errors = '';

    this.appointmentDate.setHours(this.savedHour+3, this.savedMinutes);

    this.subscriptions.add(this.userService.appointmentRegister(this.patient.patientId, this.doctor.doctorId, this.appointmentDate, this.appointmentIntervalId)
      .subscribe(
          result => {
            if (result) {
              this.router.navigate(['/patient/account']);
            }
          },
          errors => this.errors = errors));
}
```

Figura 8: Apeluri asincrone, operatorul *subscribe*

Pentru testarea operațiilor CRUD am folosit Swagger[23] prezentat în Figura 9.

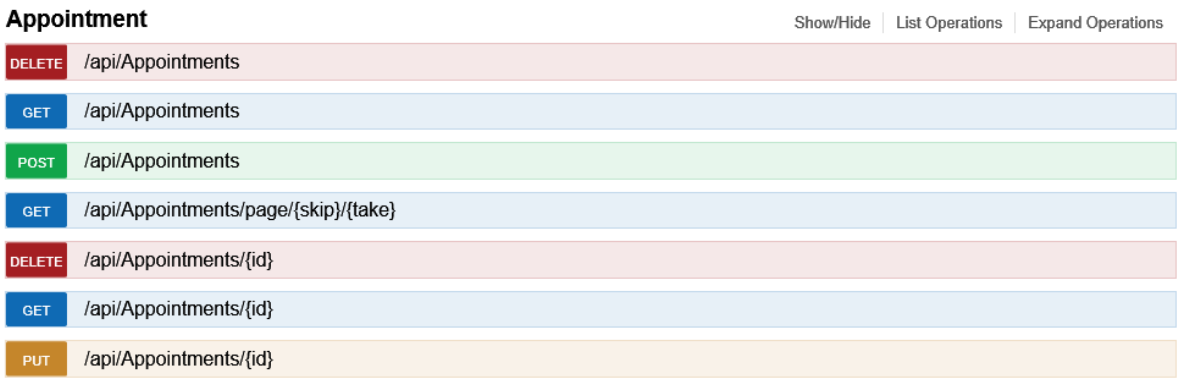


Figura 9: Swagger

[23] https://swagger.io/solutions/api-documentation/

Pentru testarea la nivel de repository am utilizat Integration Tests[24] ca în Figura 10 de mai jos.

```
[TestClass]
public class BloodDonorRepositoryTests : BaseIntegrationTests
{
    [TestMethod]
    public void Given_BloodDonorRepository_When_AddingAnBloodDonor_Then_TheBloodDonorShouldBeProperlySaved()
    {
        RunOnDatabase(async ctx => {
            //Arrange
            var repository = new BloodDonorRepository(ctx);
            var patient = Patient.Create("1234", "Roland", "Iordache", "roland.iordache96@gmail.com", "asfdsdssd", "Iasi", "Romania", new DateTime(1996, 02, 10), "0746524459", null);
            var bloodDonor = BloodDonor.Create("AB4", patient.PatientId, new DateTime());

            //Act
            await repository.AddAsync(bloodDonor);

            //Assert
            string[] includes = { };
            Assert.AreEqual(repository.GetAllAsync(includes).Result.Count, 1);
        });
    }
```

Figura 10: Integration Tests

În Figura 11 de mai jos vedem că testele au acuratețe 100%.

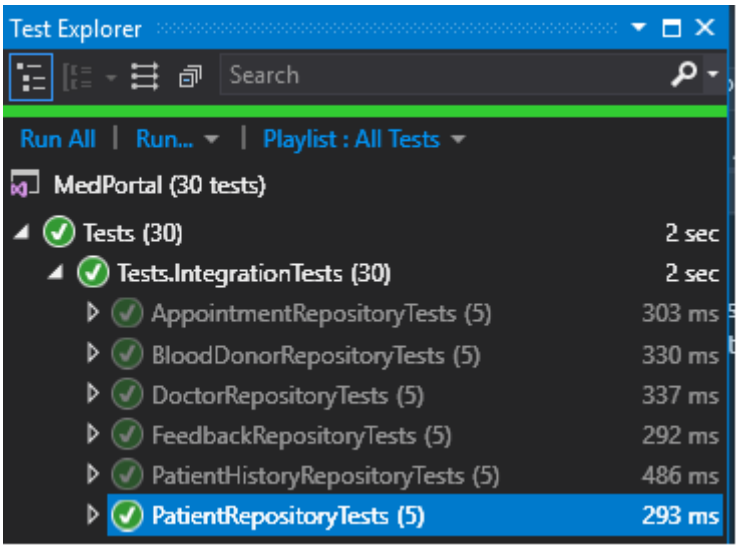


Figura 11: Integration Tests accuracy

[24] https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-2.2

Pentru testarea codului de vulnerabilități, cod duplicat sau erori, am folosit platforma SonarCloud[25], după cum se poate vedea în Figura 12 testul a trecut.

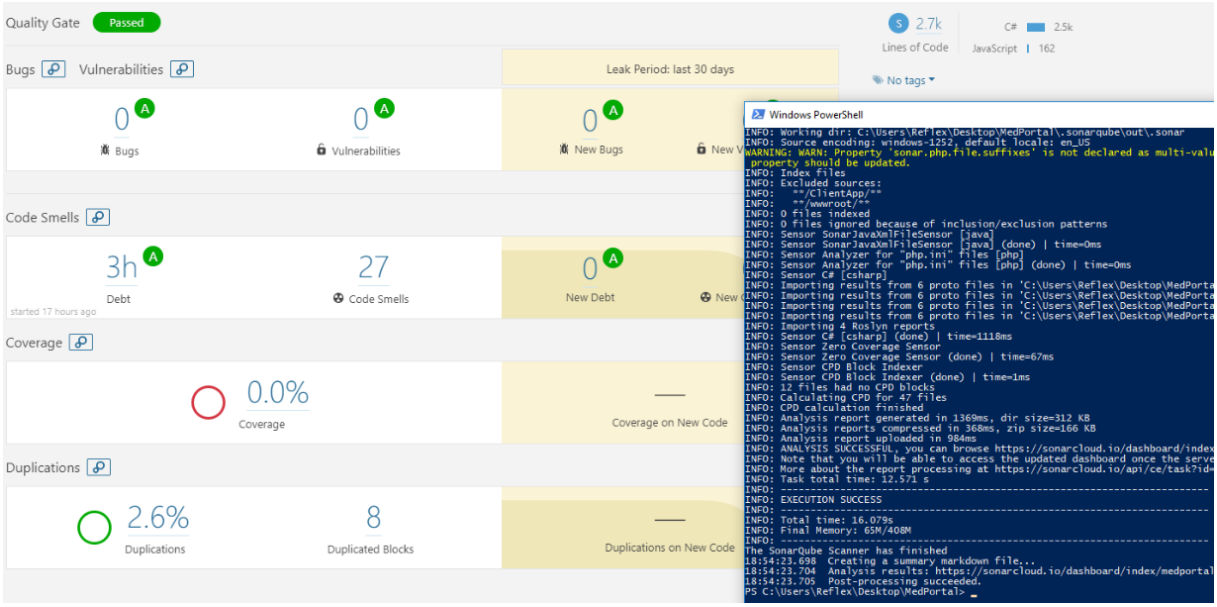


Figura 12: SonarCloud

Pentru funcționalități precum înregistrare, editare cont, schimbare parolă, schimbare e-mail și altele, am folosit validări atât pe partea de server, cât și pe partea de client. Pe partea de server am folosit Fluent Validations cu ajutorul Fluent API[26]. Față de Data Annotation[27] acesta furnizează funcționalități adiționale precum numirea tabelelor în relațiile many-to-many[28] și ușurează modul de gestionare a entităților. Putem observa validările pe server în Figura 13 de mai jos.

[25] https://sonarcloud.io/about
[26] https://www.tutorialspoint.com/entity_framework/entity_framework_fluent_api.htm
[27] https://www.codeproject.com/Articles/826304/Basic-Introduction-to-Data-Annotation-in-NET-Frame
[28] https://support.airtable.com/hc/en-us/articles/218734758-A-beginner-s-guide-to-many-to-many-relationships

```
private IEditableRepository<Patient> _repository;

public PatientValidator(IEditableRepository<Patient> _repository)
{
    this._repository = _repository;

    RuleFor(c => c.NIN).NotNull().WithMessage("Trebuie sa specificati un NIN").MustAsync(NinIsUniqueAsync).WithMessage("NIN-ul trebuie sa fie unic!");
    RuleFor(c => c.FirstName).NotNull().WithMessage("Trebuie sa specificati un prenume").Length(3, 30)
        .WithMessage("Trebuie sa aiba intre 3 si 30 caractere");
    RuleFor(c => c.LastName).NotNull().WithMessage("Trebuie sa specificati un nume").Length(3, 30)
        .WithMessage("Trebuie sa aiba intre 3 si 30 caractere")
    .MustAsync(EmailIsUniqueAsync).WithMessage("Email-ul trebuie sa fie unic!");
    RuleFor(c => c.Email).NotNull().WithMessage("Trebuie sa specificati o adresa de email").EmailAddress()
        .WithMessage("Trebuie sa specificati o adresa valida");
    RuleFor(c => c.Password).NotNull().WithMessage("Trebuie sa specificati o parola").Length(6, 35)
        .WithMessage("Parola trebuie sa fie intre 6 si 35 caractere");
    RuleFor(c => c.City).NotNull().WithMessage("Trebuie sa specificati un oras").Length(3, 30)
        .WithMessage("Numele orasului trebuie sa aiba intre 3 si 30 caractere");
    RuleFor(c => c.Country).NotNull().WithMessage("Trebuie sa specificati o tara").Length(3, 30)
        .WithMessage("Numele tarii trebuie sa aiba intre 3 si 30 caractere");
    RuleFor(c => c.Birthdate).NotNull().WithMessage("Trebuie sa specificati data de nastere").Must(BeAValidDate);
    RuleFor(c => c.PhoneNumber).NotNull().WithMessage("Trebuie sa specificati un numar de telefon")
    .Matches(@" ^\(? ([0 - 9]{ 3})\)?[-. ]? ([0 - 9]{3})[-. ]? ([0 - 9]{4})$")
    .WithMessage("Numarul de telefon invalid");
}
```

Figura 13: Fluent Validations

Pe partea de client, utilizez Reactive form validations[29] ca în Figura 14 de mai jos, validări care se actualizează în timp real, în timp ce utilizatorul completează câmpurile formularului respectiv, dupa cum se poate vedea în Figura 15.

```
this.patientRegisterForm = this.formBuilder.group({
  nin:  ['', [Validators.required], this.validatePatientNINNotTaken.bind(this)],
  lastName: ['', [Validators.required, Validators.minLength(3), Validators.maxLength(30), Validators.pattern("[a-zA-Z]+")]],
  firstName: ['', [Validators.required, Validators.minLength(3), Validators.maxLength(30), Validators.pattern("[a-zA-Z]+")]],
  phoneNumber: ['', [Validators.required, Validators.pattern("[0-9]+")]],
  birthdate: ['', [Validators.required, this.validateDOB.bind(this)]],
  city: ['', [Validators.required, Validators.minLength(3), Validators.maxLength(30), Validators.pattern("[a-zA-Z]+")]],
  country: ['', [Validators.required, Validators.minLength(3), Validators.maxLength(30), Validators.pattern("[a-zA-Z]+")]],
  password: ['', [Validators.required, Validators.minLength(8), Validators.maxLength(35)]],
  confirm_password: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(35)]],
  email: ['', [Validators.required, Validators.pattern("[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$")], this.validatePatientEmailNotTaken.bind(this)]
},
  {validator: this.validateConfirmPassword}
);
}
```

Figura 14: Reactive form validations

[29] https://angular.io/guide/form-validation

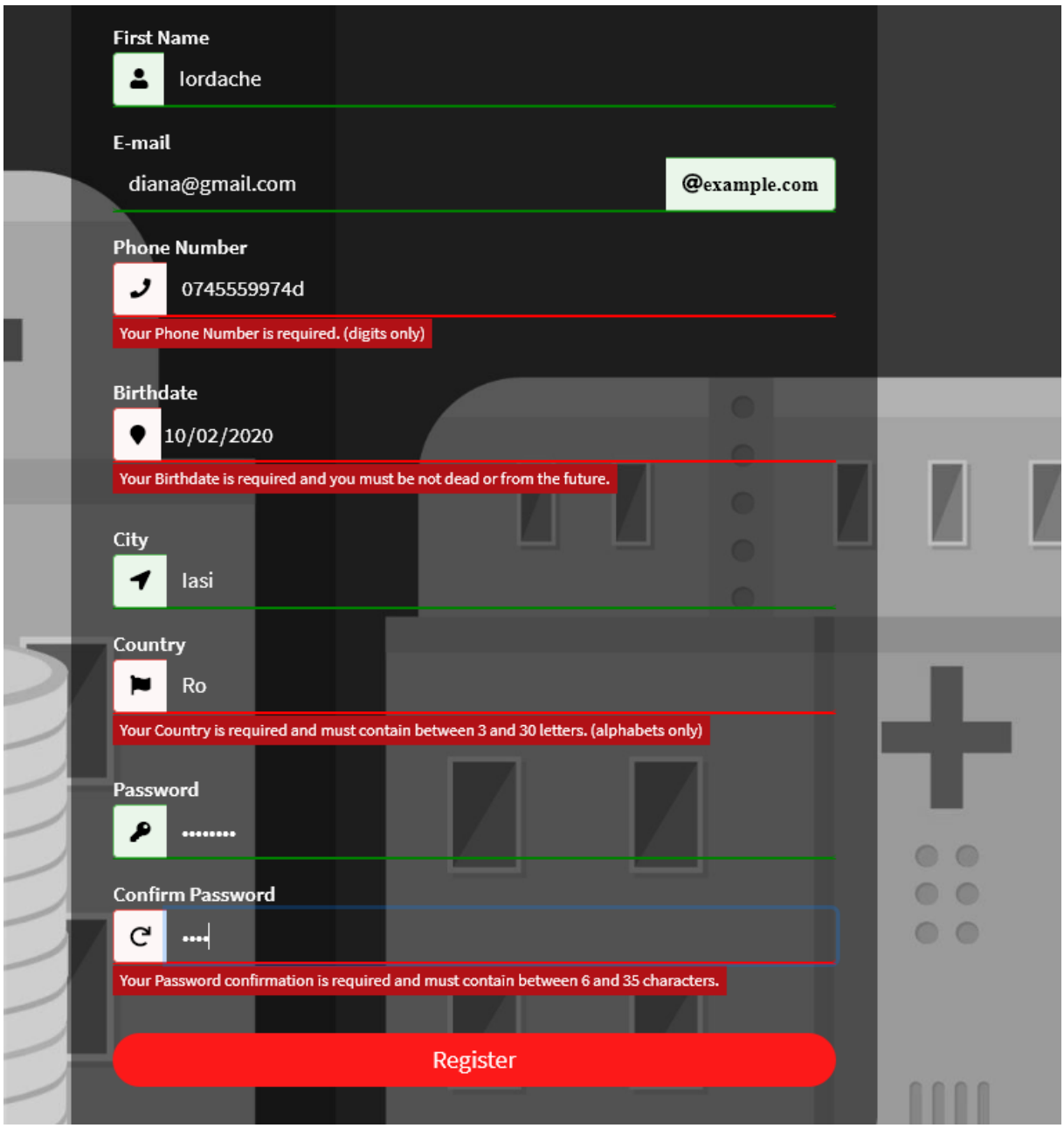


Figura 15:Client validations

Cu ajutorul Bootstrap[30] și Media Queries[31] aplicația este responsive[32] atât pentru desktop, cât și pentru tablete, exemplu în Figura 16 de mai jos sau pentru telefoane mobile, exemplu în Figura 17 de mai jos.

[30] https://getbootstrap.com/docs/4.3/getting-started/introduction/
[31] https://www.w3schools.com/css/css_rwd_mediaqueries.asp
[32] https://developers.google.com/web/fundamentals/design-and-ux/responsive/

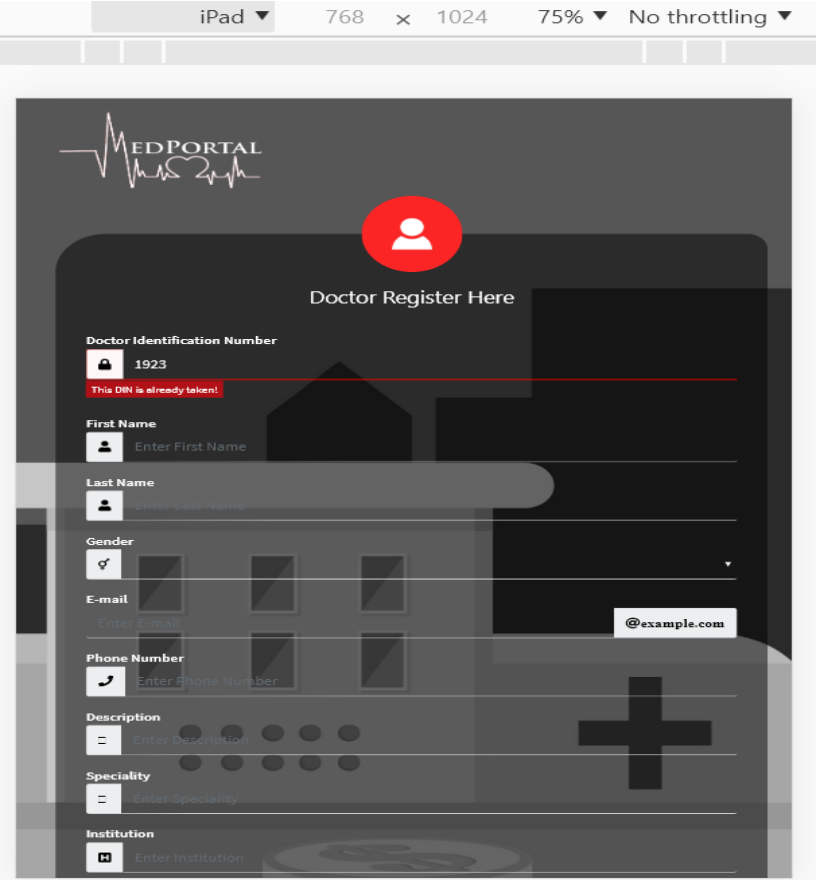


Figura 16:Responsive tabletă iPad

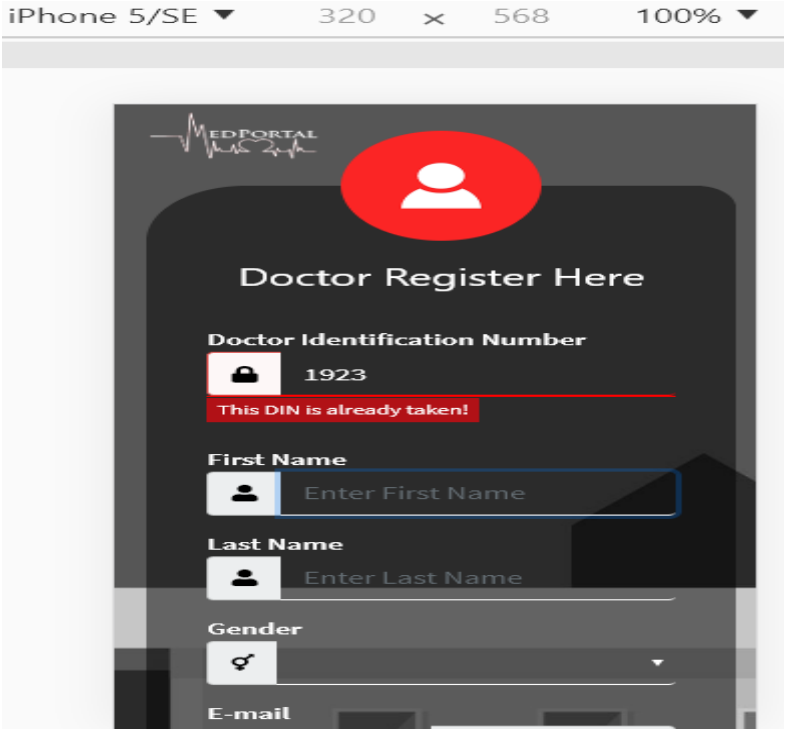


Figura 17:Responsive iPhone 5/SE

3.3.2 Autentificare și autorizare

Pentru partea de autentificare și autorizare am folosit JWT[33] criptat cu RSA[34] care ne ajută să tranferăm datele între client și server într-un mod securizat. Fiecare cerere la Web API, va verifica dacă header-ul conține token-ul[35] cu datele necesare autorizării și dacă nu a expirat sesiunea. Pentru autorizarea bazată pe roluri (medic sau pacient) am implementat funcția CanActivate[36] cu ajutorul căreia am permis accesul medicului sau pacientului doar la funcționalitățile sale, după cum se poate vedea în Figura 18 și Figura 19 de mai jos.

```
RouterModule.forRoot([
  {path:'', redirectTo: '/home', pathMatch: 'full' },
  {path:'home', component:HomeComponent},
  {path:'doctor-login', component:LoginComponent},
  {path:'patient-login', component:LoginComponent},
  {path:'doctor-register', component:RegisterComponent},
  {path:'patient-register', component:RegisterComponent},
  {path:'patient/appointments', component:ScheduleComponent, data:{requiresPatient: true}, canActivate: [ AuthService]},
  {path:'patient/medical-history', component:PatientMedicalHistoryComponent, data:{requiresPatient: true}, canActivate: [ AuthService]},
  {path:'patient/account', component:PatientAccountComponent, data:{requiresPatient: true}, canActivate: [ AuthService]},
  {path:'doctor/account', component:DoctorAccountComponent, data:{requiresDo (property) requiresDoctor: boolean ice]},
  {path:'doctor/appointments', component:DoctorAppointmentsComponent, data:{requiresDoctor: true}, canActivate: [ AuthService]},
  {path:'patient/doctor-reviews', component:ReviewsComponent, data:{requiresPatient: true}, canActivate: [ AuthService]},
  {path:'doctor/medical-history', component:DoctorMedicalHistoryComponent, data:{requiresDoctor: true}, canActivate: [ AuthService]},
  {path:'patient/donor', component:DonorComponent, data:{requiresPatient: true}, canActivate: [ AuthService]},
  {path:'**', component: FourZeroFourComponent}
]),
```

Figura 18: Funcționalitate CanActivate

```
canActivate(route: ActivatedRouteSnapshot): boolean{
  const requiresLogin = route.data.requiresLogin || false;
  if (requiresLogin) {
    // Check that the user is logged in...
    if(!this.checkLogin()) {
      this.router.navigate(['home']);
      return false;
    }
    return true;
  }

  const requiresDoctor = route.data.requiresDoctor || false;
  if (requiresDoctor) {
    // Check that the user is doctor or not
    if(this.checkLogin()) {
      if(!this.checkDoctor()) {
        this.router.navigate(['patient/account']);
        return false;
      }
      else {
        return true;
      }
    }
    this.router.navigate(['home']);;
    return false;
  }
```

Figura 19: Implementare CanActivate

[33] https://jwt.io/introduction/
[34] https://hackernoon.com/how-does-rsa-work-f44918df914b
[35] https://www.techopedia.com/definition/4140/token
[36] https://angular.io/api/router/CanActivate

Comunicarea dintre client și server este securizată de TLS (Transport Layer Security)[37], implementat în HTTPS[38]. Parola, NIN-ul (National Identification Number), DIN-ul (Doctor Identification Number) sunt hashuite de două ori în baza de date cu ajutorul MD5[39] și *salt*[40] (un string de caractere generat aleatoriu și concatenat cu prima hashuire) după cum se poate vedea în Figura 20. Astfel, verificarea parolei, schimbarea acesteia și oriunde este necesară confirmarea ei, se efectuează la nivel de server, cu parola deja hashuită ca mai sus.

```
public class PasswordHashMd5
{
    public static string GetMd5Hash(MD5 md5Hash, string input)
    {
        var data = md5Hash.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sBuilder = new StringBuilder();

        foreach (var t in data)
        {
            sBuilder.Append(t.ToString("x2"));
        }

        return sBuilder.ToString();
    }

    public static bool VerifyMd5Hash(MD5 md5Hash, string input, string hash)
    {
        var hashOfInput = GetMd5Hash(md5Hash, input);
        var comparer = StringComparer.OrdinalIgnoreCase;

        return (0 == comparer.Compare(hashOfInput, hash));
    }
}
```

Figura 20: MD5 – hashuire parolă

În concluzie, am folosit funcții hash[41] cu *salt*, știind că sunt one-way, JWT la autentificare pentru autorizare, funcția CanActivate pentru autorizare bazată pe roluri, pentru securitatea transmiterii datelor, în acord cu normele GDPR[42], va fi folosit TLS (HTTPS).

[37] https://www.networkworld.com/article/2303073/lan-wan-what-is-transport-layer-security-protocol.html
[38] https://www.howtogeek.com/181767/htg-explains-what-is-https-and-why-should-i-care/
[39] https://searchsecurity.techtarget.com/definition/MD5
[40] https://crackstation.net/hashing-security.htm
[41] https://www.invata-online.ro/infosec/data-security/hashing
[42] https://eugdpr.org/

3.3.3 Module pacient

În acest subcapitol, voi prezenta principalele funcționalități ale pacientului prin modulele: programări, recenzii, istoric medical și donare sânge.

Pe pagina contului își va putea modifica informațiile de bază, schimbarea parolei, a e-mail-ului sau anularea programărilor și vizualizarea lor ca în Figura 21 de mai jos.

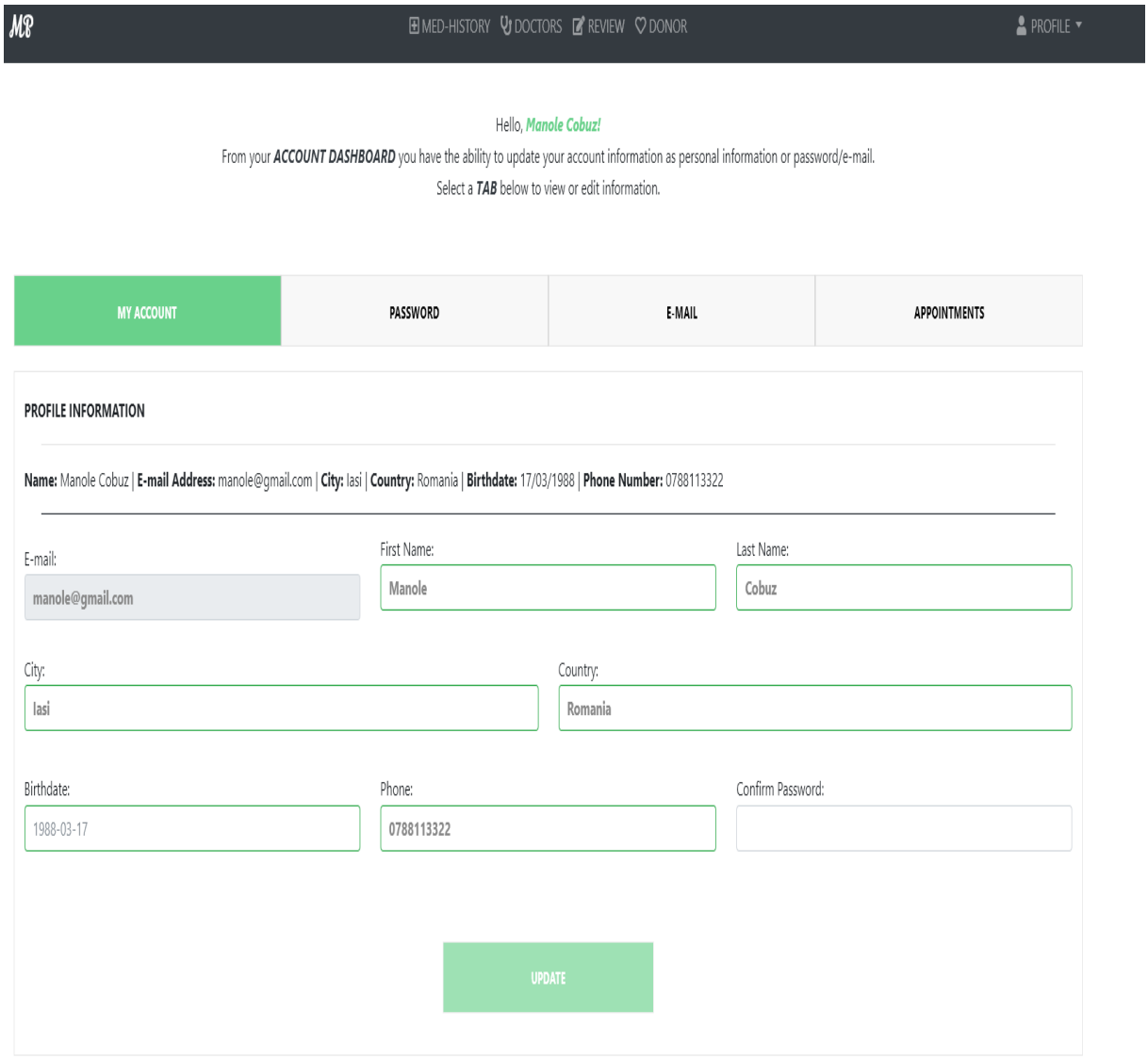


Figura 21: Cont pacient

3.3.3.1 Modulul programări

După înregistrarea și logarea cu succes a pacientului, acesta va avea în meniul disponibil modulul programări. Aici va putea căuta medici după anumite criterii: numele spitalului, orașul în care poate avea loc consultația, specialitate medic sau nume și prenume medic.

Medicii sunt afișați câte zece pe pagină, având afișate datele principale de contact, nota de la 0 la 5 în urma recenziilor și numărul notelor de 1, 2, 3, 4, respectiv 5. Paginarea este făcută cu ajutorul metodelor LINQ[43]: skip și take[44]. Metoda Skip este folosită pentru a preciza câte elemente să sară, *n* spre exemplu, iar metoda Take pentru a specifica câți medici să ia de la *n+1*, în cazul nostru zece. Putem vedea implementarea în Figura 22 și Figura 23 de mai jos.

```
public async Task<PagingResult<TEntity>> GetByFilter(Expression<Func<TEntity, bool>> predicate, int skip, int take, string[] includes)
{
    var query = DatabaseService.Set<TEntity>().AsQueryable();
    if (includes != null)
    {
        foreach (string include in includes)
        {
            query = query.Include(include).AsNoTracking();
        }
    }
    var totalRecords = await query.Where(predicate)
        .CountAsync();
    var records = await query.Where(predicate)
        .Skip(skip)
        .Take(take)
        .ToListAsync();
    return new PagingResult<TEntity>(records, totalRecords);
}
```

Figura 22: Skip și Take - C#

[43] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/
[44] https://www.codingame.com/playgrounds/213/using-c-linq---a-practical-overview/skip-and-take

```
<div id="pagination-div">
  <nav aria-label="pagination" class="row align-items-center justify-content-center" *ngIf="doctorsList.length!=0">
    <ul class="pagination justify-content-center">
      <li class="page-item"><a class="page-link" (click)="getDoctorsByFilter(doctorSearch, 0); activate(1);"> << </a></li>

      <li class="page-item" *ngFor="let page of numbers"><button [ngClass]="active[page] ? 'page-link active' : 'page-link'"
        (click)="getDoctorsByFilter(doctorSearch, page-1); activate(page);">{{page}}</button></li>

      <li class="page-item"><a class="page-link" (click)="getDoctorsByFilter(doctorSearch, lastPage-1); activate(lastPage);"> >> </a></li>
    </ul>
  </nav>
</div>
```

Figura 23: Skip și Take – HTML

După ce am selectat un medic după criteriile căutate, putem face programare selectând o dată și văzând disponibilitatea acestuia din acea zi. ca în Figura 24 de mai jos.

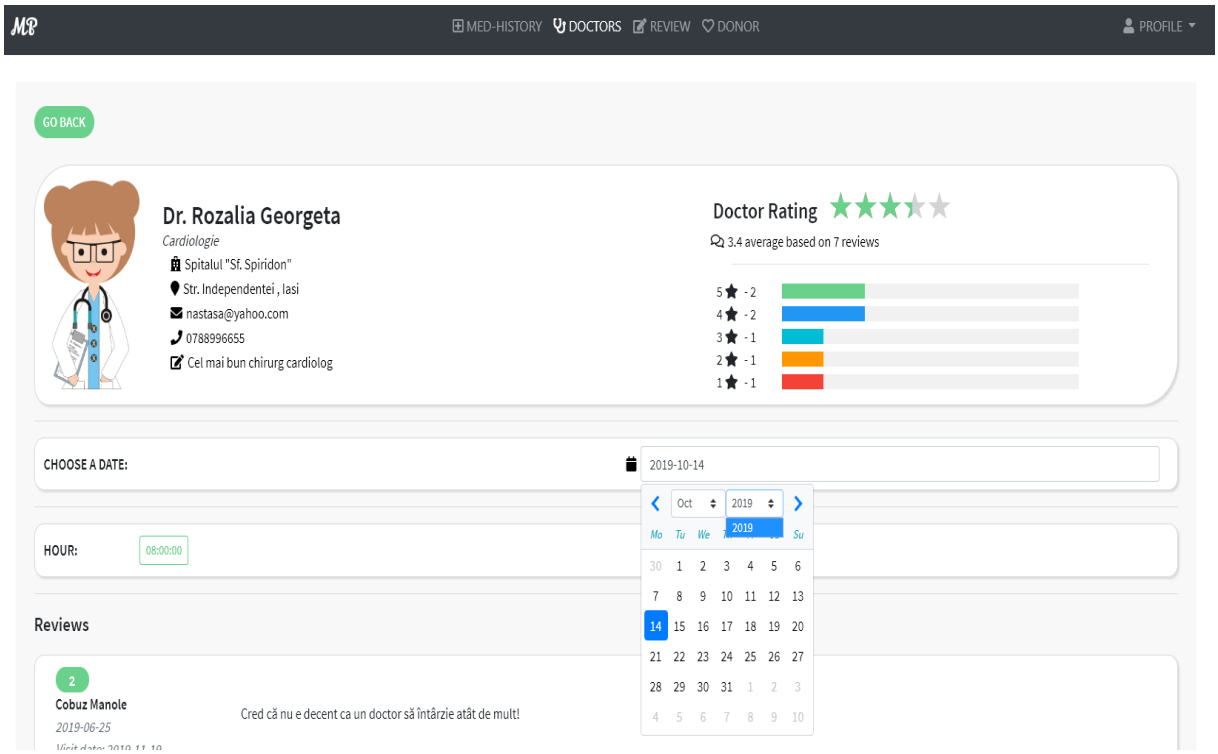


Figura 24: Programare la medic

În concluzie, putem să ne facem programare online ușor și rapid, selectând medicul dorit după anumite criterii și după anumite recenzii pe care le-au împărtășit și alți utilizatori.

3.3.3.2 Modulul recenzii

Fiecare pacient va putea să își evalueze medicul care l-a consultat doar în condițiile în care data programării a depășit data consultării și dacă programarea s-a efectuat. Astfel, eliminăm recenziile date fraudulos. După cum se poate observa în Figura 25 de mai jos, pacientul poate acorda o notă de la 1 la 5 medicului și poate să descrie experiența pe care a avut-o în timpul consultației.

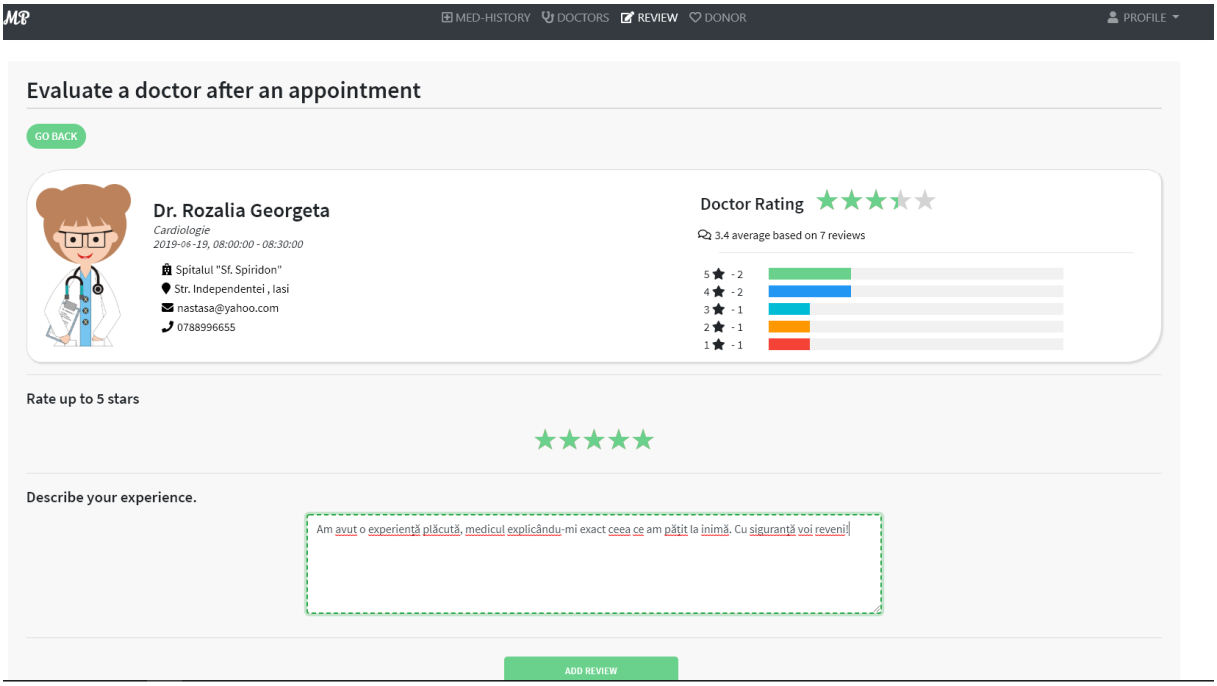


Figura 25: Recenzii

În consecință, cu ajutorul acestui modul, putem să ne orientăm către un medic cât mai bun, bazat pe recenzii pozitive de la pacienții care au fost consultați de către acesta.

3.3.3.3 Modulul istoric medical

Pacientul va putea să își vizualizeze istoricul medical în măsura în care este completat de către medic. Pentru a avea acces la informațiile din trecut, medicii de familie ar trebui să completeze din dosarele stocate fizic, principalele detalii ale pacientului în urma primei consultații pe aplicație. Astfel, pacientul va putea accesa oricând planul de tratament sau vizualiza informații vitale legate de starea sa de sănătate, cât și ce medicamente are voie sau nu.

Observăm datele principale despre pacient care trebuie completate de către medic și anume greutate, înălțime, dacă este fumător și altele. Principalele informații sunt date de listarea consultațiilor avute de acesta până în prezent, data consultației, medicul care l-a consultat, cât și planul de tratament alături de diagnostic. Vor mai fi notate principalele boli, alergii și intoleranțe. Putem observa această structură în Figura 26 de mai jos.

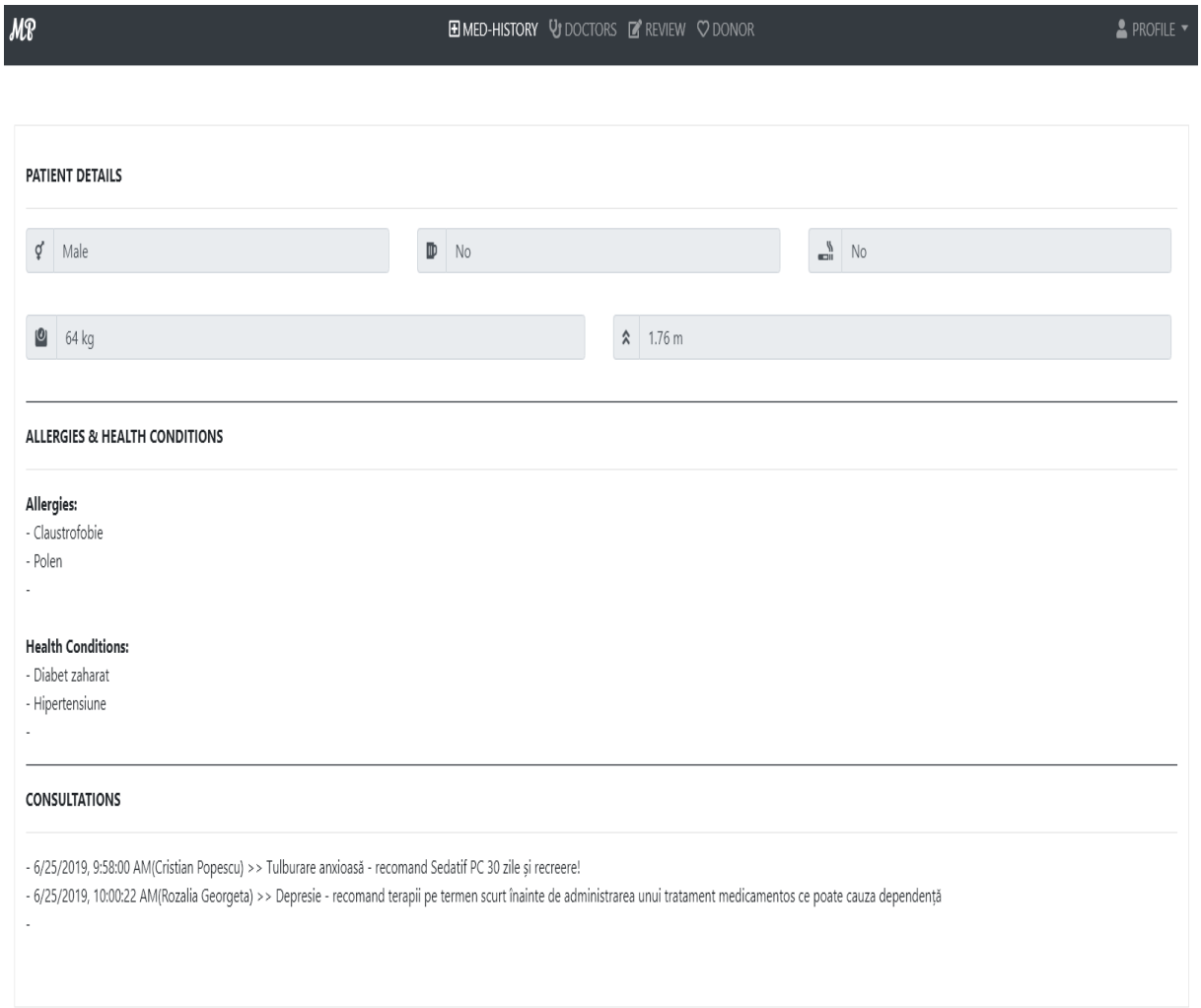


Figura 26: Istoric medical pacient

Prin urmare, istoricul medical oferă următoarele beneficii: centralizarea datelor medicale, vizualizarea în orice moment a planului de tratament și a consultațiilor și facilitarea relației dintre medic și pacient.

3.3.3.4 Modulul de donare sânge

Pentru a te putea înscrie ca voluntar în cadrul platformei *MedPortal* în privința donării sângelui, trebuie întâi să fii eligibil din punct de vedere a vârstei (trebuie să ai minim 16 ani) și să nu mai fi donat în ultimele 8 săptămâni. Dacă poți fi donator sau nu, se va stabili ulterior la o clinică specializată, pe baza altor criterii care reies doar din rezultatele analizelor de sânge.

Odată înscris, cei care au nevoie de acea grupă vor avea o secțiune de căutare după oraș sau grupă de sânge. Va trebui să ia contact cu persoana care se oferă voluntar, trimițând o cerere pentru a nu mai figura în secțiunea rezultate și pentru a înștiința voluntarul. Persoana care s-a oferit să doneze sânge va avea, în cele din urmă, o secțiune în care va confirma dacă a donat sau nu. Dacă a confirmat, voluntarul va trebui să aștepte 8 săptămâni până va putea să se înscrie și până va putea dona din nou. În caz contrar, aceasta va reveni în lista de căutare a voluntarilor eligibili într-o primă selecție.

Observăm cele patru opțiuni principale în Figura 27 de mai jos, printre care cerințele principale pentru donarea de sânge, înscrierea ca voluntar, căutarea după grupe de sânge și nume a acestora și afișarea lor paginată, cât și confirmarea dacă a donat sau nu pentru a se reînscrie automat sau a finaliza procesul.

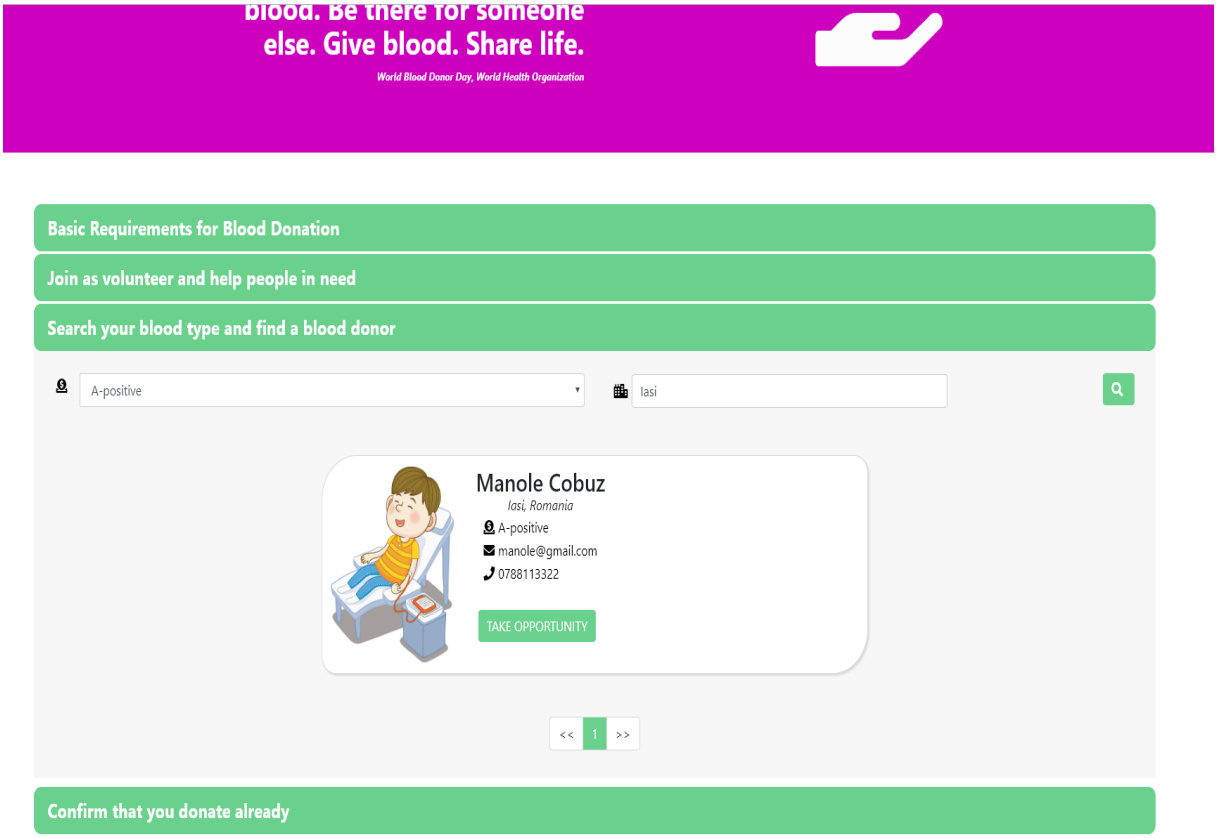


Figura 27: Modul donare

În concluzie, soluția oferă posibilitatea utilizării unui portal unde pacienții care au nevoie de sânge să poată efectua căutări în vederea găsirii unor voluntari, ușurând procesul de selecție și căutare.

3.3.4 Module medic

În acest subcapitol, voi prezenta principalele funcționalități ale medicului prin modulele: istoric medical și situația programărilor..

Pe pagina contului, medicul își va putea modifica informațiile de bază, va putea vizualiza recenziile și gestiona programul de lucru. Programul de lucru este împărțit pe zile a săptămânii de luni până sâmbătă inclusiv, putând adăuga sau șterge intervale orare într-un mod flexibil, astfel încât să-și poată crea propriul program. Vom vedea un caz ipotetic prezentat în Figura 28, când medicul și-a adăugat deja intervalul fixat și un mesaj de atenționare că intervalul

există deja, validările fiind și pentru cazurile în care ora de început depășește ora de sfârșit sau dacă două intervale se suprapun.

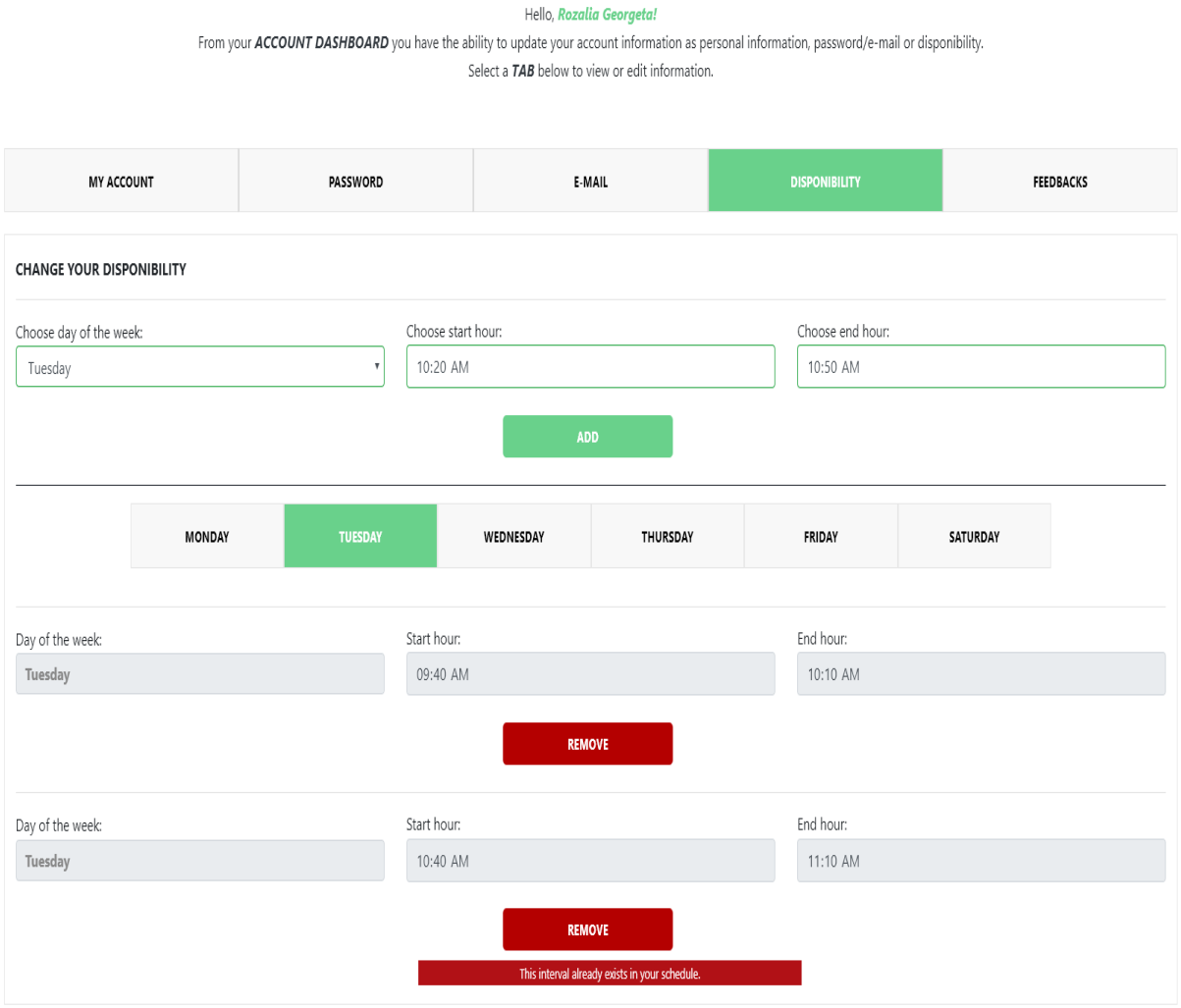


Figura 28: Gestionarea disponibilităților

3.3.4.1 Modulul istoric medical

În acest modul, medicul va putea, la fel ca în cazul pacientului, să nu poată completa istoricul medical decât pacienților cu care are programare și nu mai devreme de ziua consultării acestora. Poate actualiza date cu caracter personal (greutate, înălțime ș.a.m.d.), dar va trebui să completeze date referitoare la consultație, diagnosticul, perioada de tratament și tratamentul în sine. În Figura 29 de mai jos, observăm funcția care generează acei pacienți cu condițiile de mai sus satisfăcute.

```
generatePatients(doctorId: string) {
  this.isExpired = this.userService.isExpired();
  if(this.isExpired) {
    this.authService.logout();
  }
  this.errors = '';
  this.titleArray = [];

  this.subscriptions.add(this.userService.getAppointments()
    .subscribe(
        result => {
            for(let appointment of result.json()){
                let date = new Date(appointment.appointmentDate);
                date.setHours(5,30,0);
                if(this.now.getTime()>=date.getTime() && appointment.doctor.doctorId==doctorId && appointment.haveMedicalHistory==false)
                {
                    this.titleArray.push(appointment);
                }
            }


        },
        errors => this.errors = errors));
}
```

Figura 29: Generarea pacienților pentru istoric medical

În consecință, medicul gestionează mai ușor relația cu pacientul și poate accesa istoricul medical, nefiind nevoie de interogarea acestuia pentru stabilirea unor noi diagnostice.

3.3.4.2 Modulul programări

Medicul poate verifica online situația programărilor pe zile și pe intervale orare, eliminând munca repetitivă și solicitantă a recepționerelor și impiedicând apariția de erori în programările efectuate de acestea. Putem vedea în Figura 30 de mai jos un exemplu pentru o dată oarecare selectată de către medic.

Medicul nu poate anula programarea, în cel mai rău caz, dacă nu se poate prezenta din anumite motive, va trebui să ia contact cu pacientul și să-l anunțe, ceea ce este deontologic și moral și în viața reală.

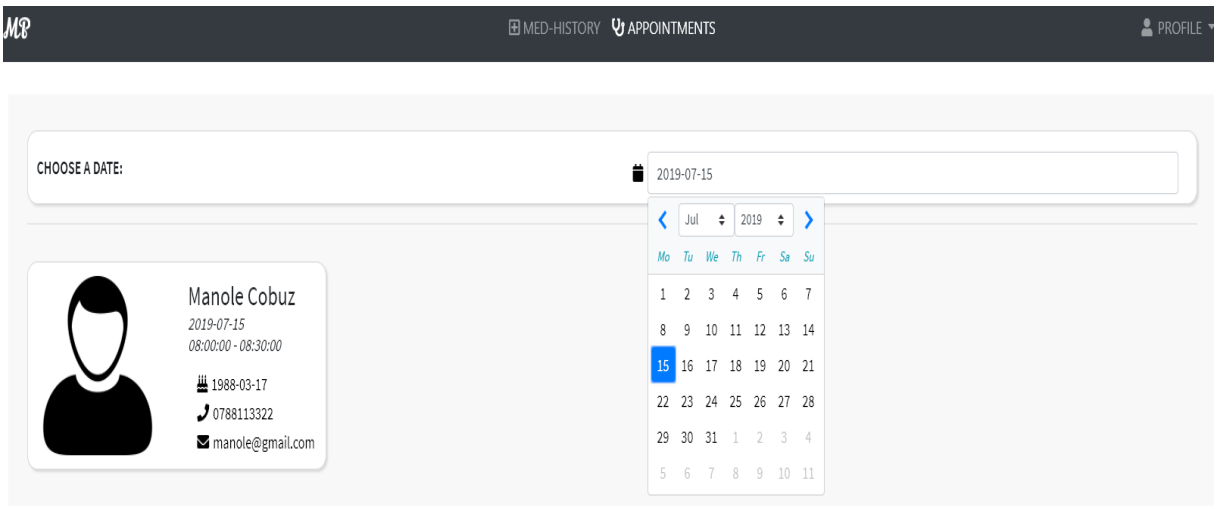


Figura 30: Listare programări către medic

Așadar, transparența programului și ușurarea accesării lui în orice moment, susțin soluția prezentată ca o alternativă a modului în care se realizează programarea.

Concluzii

Obiectivul principal a fost să ofer o soluție problemelor descrise în capitolele anterioare, care să fie integră, să cuprindă într-o singură aplicație, funcționalități diferite dar din același domeniu, medicină.

Dorința a fost să ofer un portal ușor de utilizat, ușor de întreținut de către ambele părți implicate și cu un design simplu și responsive.

Inițial, am fost oarecum neîncrezător, deoarece nu am avut încredere că pot dezvolta o aplicație pe partea de front-end design. Tocmai ca să văd cum m-aș descurca, mi-am propus singur o provocare: să învăț de la zero tehnologii noi, precum Angular, TypeScript și să încerc să utilizez ultimele versiuni ale acestora, deși a fost greu să găsesc documentații și răspunsuri la problemele pe care le-am întâmpinat.

Cea mai grea parte a fost să mă acomodez cu tehnologiile noi, pe parcursul lucrării, am obținut suficiente cunoștințe încât să îmi dau seama ce am făcut greșit și s-ar putea face mai bine sau ce bucăți de cod ar putea fi refactorizate. De asemenea, deși am folosit Code First, fiind destul de ușor să fac ulterior modificări la baza de date prin migrări, a trebuit să revin și să tot revin asupra tabelelor, adăugând sau ștergând proprietăți pentru entități, sau chiar creând noi tabele. Astfel, mi-am dat seama cât de importantă este înainte de implementarea propriu-zisă, analiza cerințelor și plănuirea dinainte a arhitecturii pe care urmează să o folosesc. Ca un exemplu, am pornit de la repository normale, după care am templetizat totul, aplicând Generic Repository Pattern, după care am trecut la varianta CQS. De asemenea, deși componentele nu au legături strânse între ele, codul fiind decuplat, m-am lovit de refactorizare și de erori din cauza acesteia. Ca o învățătură, pot spune că partea cu abordarea noilor tehnologii a fost o idee bună, dar trebuie să aloc mai mult timpul pentru plănuirea propriu-zisă a schemelor, a mock-up-urilor, a relațiilor dintre tabelele bazei de date.

Prin urmare, am deprins cunoștințe noi, atât de tehnologii, cât și de arhitecturi, design patternuri, best practices și ideea conform căreia teoria, chiar și în informatică, legată de rezolvarea de probleme, necesită un studiu mai aprofundat decât credeam. Am prins încredere în mine și am dobândit plăcerea de a lucra la proiecte.

Pe viitor, ar putea fi implementate următoarele funcționalități:

- doctorul să-și poată gestiona programul în funcție de o zi liberă, de concedii, adică adăugarea unei tabele noi care să corespundă unui doctor în relație one-to-many, acesta setându-și concediile și zilele libere, cu un interval orar aferent
- date și statistici referitoare la donarea de sânge bazate pe grupe de sânge, intervale de vârstă, donări reușite, donări nereușite ș.a.m.d.
- chat între medic și pacient
- robot interactiv care să te poată îndruma în cazul în care ești pe platformă inactiv de câteva minute
- rezultate analize de sânge, scanări și rezultate raze X, CT, radiografii, ecografii ș.a.m.d.
- adăugarea unui modul ascuns pentru administrator, în care să poată vedea mesajele legate de îmbunătățirea aplicației, probleme, sugestii, să poată controla sistemul de recenzii împotriva limbajului indecent, să poată trimite mesaje de avertizare acestora
- repartizarea voluntarilor pentru donarea de sânge bazată pe activitatea lor (numărul de donări într-un anumit timp), afișându-i, cu acordul lor, pe cei mai activi pe platformă

Bibliografie și Webografie

1. Alexandra Jeles (20 iunie 2012). „Cum poți să salvezi 10 vieți într-un an”. România liberă.
2. https://bootstrapmade.com/demo/BizPage/
3. https://www.kisspng.com/png-avatar-computer-icons-patient-clip-art-1612359/download-png.html
4. http://www.ibpf.org/blog/10-tips-your-next-doctor%E2%80%99s-appointment
5. http://chittagongit.com/icon/doctor-icon-5.html
6. https://www.komando.com/tips/344610/3-apps-to-track-your-medical-history/all
7. https://pixabay.com/vectors/hospital-bless-you-professional-1706646/
8. https://wtop.com/health-fitness/2018/07/need-for-blood-donations-in-dc-area-is-at-high-critical-level/
9. https://pixabay.com/vectors/doctor-medicine-man-medical-2025725/
10. https://www.who.int/news-room/fact-sheets/detail/blood-safety-and-availability
11. http://www.pngmart.com/image/116459
12. http://www.pngmart.com/image/117293
13. https://security.stackexchange.com/questions/5126/whats-the-difference-between-ssl-tls-and-https
14. https://medium.com/angular-japan-user-group/why-developers-and-companies-choose-angular-4c9ba6098e1c
15. https://angular.io/docs
16. https://getbootstrap.com/docs/4.0/layout/grid/
17. https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/new-db?tabs=visual-studio
18. https://www.tutorialspoint.com/typescript/
19. https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/
20. https://jwt.io/introduction/
21. https://hackernoon.com/how-does-rsa-work-f44918df914b
22. https://www.learnrxjs.io/
23. https://coderwall.com/p/app3ya/read-excel-file-in-c

24. https://www.mayoclinic.org/diseases-conditions/male-breast-cancer/symptoms-causes/syc-20374740
25. https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet
26. https://www.kaggle.com/sarahvch/breast-cancer-wisconsin-prognostic-data-set/downloads/breast-cancer-wisconsin-prognostic-data-set.zip/1
27. https://towardsdatascience.com/building-a-simple-machine-learning-model-on-breast-cancer-data-eca4b3b99fa3
28. https://crackstation.net/hashing-security.htm#salt

Anexă – Tehnologii

- **Angular –** Angular este un framework care ajută la dezvoltarea produsului software pe partea de client. În primul rând, l-am ales deoarece se potrivește cu C# și .NET, și având deja minime cunoștințe cu privire la aceste limbaje, am decis să îl studiez și să îl folosesc. În al doilea rând, are librării integrate pentru aproape orice, directive folositoare pentru autorizare ca Guards, RxJS pentru task-urile asincrone (subscribe). Este a doua cea mai căutată tehologie după Node.js, având o documentație detaliată chiar și pentru ultimele versiuni. De asemenea, are suport de la Google, ceea ce denotă faptul că este un limbaj de încredere. Dispune de CLI, cu ajutorul căreia poți crea componente noi, genera un proiect nou, șterge componente, instala librării. Componentele diverse respectă dependency injection ca în Figura 31 de mai jos. Developerii adaugă versiuni noi foarte des, aceasta fiind dovada că va mai rămâne stabil pe piață mult timp

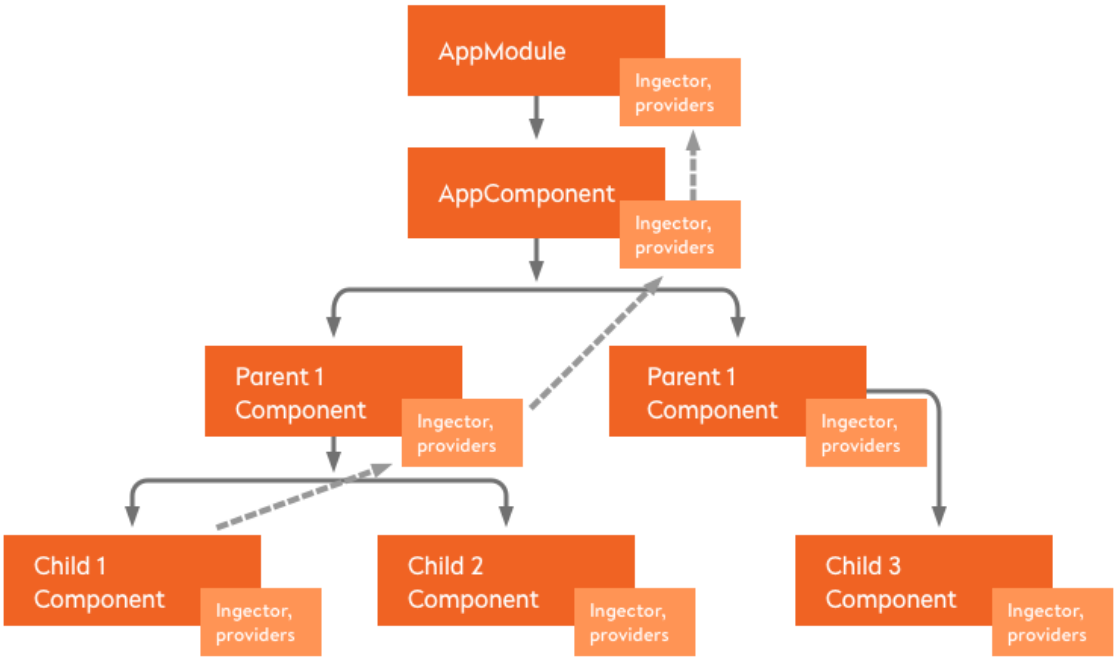


Figura 31: Angular structură preluată de pe **https://yalantis.com/blog/when-to-use-angular/**

- **Bootstrap –** Bootstrap este cel mai utilizat framework pentru HTML, SCSS și TYPESCRIPT. Salvează timp, având clase și template-uri deja predefinite pentru design. E ușor de customizat, compatibil, documentație amplă și bine structurată, e open source, design responsive. L-am folosit în special pentru structurarea paginilor pe coloane și rânduri

- **.NET Core –** .NET Core este un framework pe partea de server open-source și este ușor de utilizat. Totodată, este foarte popular și este succesorul .NET Framework, având o documentație ușor de parcurs

- **C# –** Este un limbaj foarte popular de programare. Este denumit limbajul Cool (C-like Object Oriented Language). Poate fi folosit pentru aproape orice, aplicații pe mobil, servicii cloud, aplicații software și jocuri. Este ușor de utilizat și fiind dezvoltat de către Microsoft, este într-o continuă evoluție

- **SQL Server –** SQL Server este a doua cea mai utilizată tehnologie pentru lucrul cu baze de date. Este un suport bun pentru Visual Studio și pentru .NET