****

PROIECT

Analiza și proiectarea sistemelor informatice

Aplicație web pentru gestiunea pacienților și programărilor din cadrul unui cabinet medical

|  |  |
| --- | --- |
| **Dobocan Vlad-Ioan** | **C:\Users\Vlad\AppData\Local\Microsoft\Windows\INetCache\Content.Word\vlad.jpg** |

Cuprins

[Partea I – Raport de Analiză 3](#_Toc2087217)

[Introducere 4](#_Toc2087218)

[1. Identificarea și Descrierea Problemei 4](#_Toc2087219)

[1.1. Motivație 4](#_Toc2087220)

[1.2 Context 5](#_Toc2087221)

[2. Cerințe de sistem 5](#_Toc2087222)

[2.1 Surse de cerințe 5](#_Toc2087223)

[2.2 Elicitația cerințelor 5](#_Toc2087224)

[2.2.1 Chestionar 5](#_Toc2087225)

[2.2.2 Workshop 5](#_Toc2087226)

[2.2.3 Modelul use-case 5](#_Toc2087227)

[2.3 Documentarea cerințelor 5](#_Toc2087228)

[2.3.1 Procese și Activități 5](#_Toc2087229)

[3. Model de Dezvoltare 6](#_Toc2087230)

[Partea II – Proiectarea Sistemului Informatic 7](#_Toc2087231)

[4. Proiectarea Logică 8](#_Toc2087232)

[4.1 Arhitectura Sistemului 8](#_Toc2087233)

[4.2 Baza Informaţională 8](#_Toc2087234)

[5. Proiectarea Tehnică 8](#_Toc2087235)

[5.1 Structura fizică a datelor 8](#_Toc2087236)

[5.2 Procese şi Algoritmi 9](#_Toc2087237)

[5.3 Tehnologii specifice 9](#_Toc2087238)

[Glosar 9](#_Toc2087239)

[Bibliografie 9](#_Toc2087240)

# Partea I – Raport de Analiză

# Introducere

Tema selectată este strâns legată de activitatea unui cabinet medical. La începutul semestrului I din anul III, am fost nevoit să cer o adeverință medicală de la medicul meu de familie. Pentru această adeverință, am pierdut mai bine de o jumătate de zi în sala de așteptare din cabinetul acestuia. Timpul pierdut în aceea zi m-a determinat să creez o aplicație în care pacientul nu mai este nevoie să irosească atât de mult timp în sala de așteptare, iar medicul să își poate gestiona mult mai rapid și mai eficient timpul și pacienții săi. Aplicația mea va rezolva această problema, având în vedere faptul că suntem într-o perioadă dificilă, din cauza pandemiei.

Prin urmare, domeniul pe care reușesc să îl ating în cadrul acestei lucrări este domeniul medical. Probabil toți am simțit cum este să pierdem timp la medic și cât de multe am fi putut face în acel timp. Voi face această aplicație împreună cu medicul meu de familie, folosindu-mă de sfaturile și de cunoștiințele sale din domeniu medical, ca mai apoi acesta să o poată folosi.

Grupul țintă a acestei aplicații vor fi atât pacienții, care nu au cunoștințe legate de domeniul IT și nici de cel medical, cât și medicii de familie, persoane care au cunoștințe în domeniul medical. Principalul beneficiar al aplicației va fi medicul de familie, dat fiind faptul că el va deține această aplicație, iar apoi, să fac această aplicație generală, putând fi folosită de mai mulți medici. Sper ca, folosind această aplicație, nici o persoană să nu mai irosească timp la medicul de familie.

Aplicația ce urmează să o dezvolt va fi o aplicație web și se va numi *MedClinic*. Scopul principal al aplicației este de a eficientiza timpul de așteptare și de a oferi o mai bună organizare a pacienților unui medic. De asemenea, *MedClinic* ar fi o alternativă metodei clasice de a face programare la medicul de familie, printr-un apel telefonic, metodă ce pe mine, personal, mă deranjează. Astfel, s-ar elimina timpii morți și conversația inutilă când suni la medicul de familie, programarea făcându-se doar prin câteva click-uri.

Fiind totul mult mai organizat, pacienții care vin fără programare vor intra numai în cazul în care este un loc liber în orar, iar, în caz de urgențe, medicul să poată decala programările cu un anumit interval de timp, restul pacienților știind exact de acest decalaj de timp și când trebuie mai exact să intre.

# 1.Identificarea și Descrierea Problemei

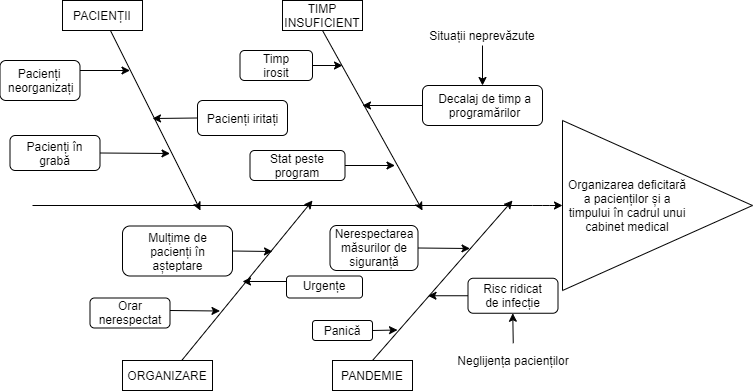
În momentul de față, foarte puține cabinete medicale au un sistem de gestiune a pacienților care să fie capabil să organizeze programările și fluxul de informație care provine de la pacienți. *MedClinic* vine în ajutorul cabinetelor care nu au un astfel de sistem. Prin urmare, asistenții din cadrul cabinetelor medicale nu vor mai fi nevoiți să răspundă la apeluri telefonice pentru ca pacienții să fie programați, ci se vor ocupa de pacienții care au nevoie de ajutor.

Probabil principala problema în cadrul cabinetelor medicale este organizarea, mai ales în cadrul pandemiei, unde este foarte important să reducem pe cât posibil interacțiunea dintre pacienți și pierderea de timp. Astfel, un cabinet medical care nu are un sistem de gestiune stabil nu poate să respecte aceste reguli.

## Motivație

Motivul principal care stă la baza acestei soluției este dorința de a exista un sistem de organizare a programărilor mai eficient în cadrul tuturor cabinetelor medicale, astfel încât timpul de așteptare să fie pe cât posibil diminuat. Iar *MedClinic* va încerca să rezolve acest inconvenient. Însă nu va oferi doar atât. Această aplicație va oferi și un sistem în care medicul își va putea organiza pacienți după cum acesta dorește.

Probabil toți am experimentat cum este să pierdem timp la medicul de familie pentru o simplă adeverință sau o consultație, alături de alți pacienți. Acest timp petrecut alături de ceilalți pacienți este vital datorită faptului că suntem în mijlocul unei pandemii. Un sistem organizat de gestionare a programărilor este necesar în aceste vremuri grele. *MedClinic* va duce la reducerea duratei de timp de așteptare în cabinet și la o mai bună organizare a pacienților și a rezultatelor acestora. De asemenea, va încerca să ofere și o comunicare mai bună între pacient și medic pentru ca, în caz de urgențe, pacientul să știe și să nu fie nevoit să aștepte în cabinet.



#### **Figura 1 – Diagrama Fish-bone**

Un prim obiectiv ar fi “transpunerea” mediului de lucru în format virtual. Fiecare pacient ar trebui să aibe propriul său cont pe aplicație, pentru ca medicul să poată gestiona mult mai ușor pacienții. Pentru aceasta, fiecare pacient va trebui să îndeplinească următoarele:

* Să aibe propriul cont în aplicație
* Să fie aprobat pe medicul său, pentru ca acesta să poată folosi aplicația

Un alt obiectiv ar fi eficientizarea timpului de așteptare, acesta presupunând urmatoarele:

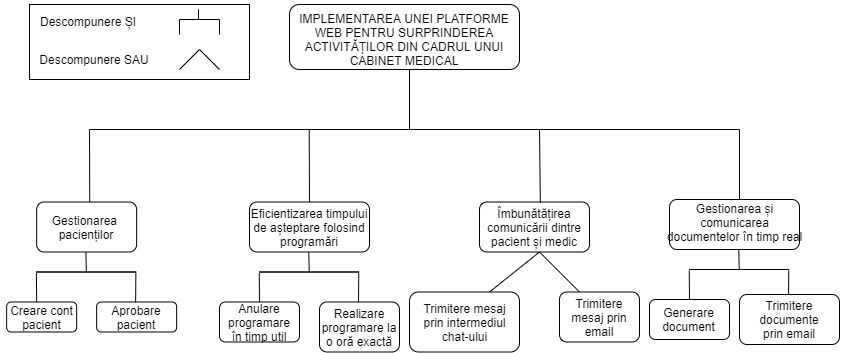
* Realizarea unei programări folosind aplicația web
* Anularea unei programări în timp util, pentru ca medicul să poate folosi acel timp

Urmărind cele de mai sus, putem realiza faptul că este nevoie de o comunicare rapidă între medic și pacient, acest fapt devenind al doilea obiectiv pe lista, fiind realizat prin următorii pași:

* Notificarea medicului în cazul unei urgențe în timp real, folosind chat-ul
* Notificarea folosind metoda clasică, prin email

Nu în ultimul rând, medicul să poată genera anumite documente din aplicație și să le trimită pe e-mail-ul pacientului, pentru ca acesta să poată accesa documentele fără a mai fi necesară deplasarea la cabinetul medical. Acest obietiv poate fi realizat prin următorii pași:

* Generare document
* Trimitere document prin intermediul email-ului



#### **Figura 2 – Diagrama de descompunere a obiectivelor**

## 

## 1.2 Context

# Fațeta subiect

Aplicația web *MedClinic* are ca obiectiv furnizarea unei metode online ușor accesibile tuturor, punând la dispoziție căi eficiente, pentru a surprinde activitatea din cadrul unui cabinet medical. Scopul principal al aplicației este de a oferi o gestiune mai eficientă a activităților ce au loc în cadrul unui cabinet medical și, totodată, de a reduce, pe cât posibil, timpul pierdut de pacient în sala de așteptare.

Chiar dacă aplicația va avea în prim plan eficientizarea gestiunii activităților din cadrul unui cabinet medical, principalii beneficiari ai aplicației nu vor fi doar medici, ci toate persoanele care vor accesa această aplicație, adică atât medicii cât și pacienții acestora.

# Fațeta utilizare

Utilizatorii aplicației se împart în 2 categorii: medici și pacienții acestora. Consider că interfața trebuie să fie sugestivă și să permită un acces propice a celor 2 categorii de utilizatori la principalele funcționalități a aplicației.

* **Personal medical:** vor reprezenta principala categorie de utilizatori din cadrul acestei aplicați. Aceștia vor putea naviga prin aplicație prin diferitele funcționalități puse la dispoziție pentru a eficientiza modul de gestionare a pacienților. Principala activitate a acestora va fi cea de gestionare a programărilor și a pacienților folosind o interfață grafică prietenoasă și sugestivă, fiind ușor de folosit.
* **Pacienții:** categoria de utilizatori ce depinde în mare parte de medici. Principala lor activitate este de a realiza programări în cadrul programului de muncă a medicului. Totuși, nu este singura funcționalitate. Aceștia vor mai avea la dispoziție o metodă eficientă de comunicare cu medicul și un mod de a vizualiza ultimele sale vizite la medic.

# Fațeta IT

Aplicația conține 2 zone mari, bine delimitate între ele: *frontend* și *backend* și o metodă eficientă de a comunica între ele.

*Frontend*-ul aplicației este realizat folosind un framework de TypeScript numit Angular versiunea 10, dezvoltat și întreținut de o echipă din cadrul companiei Google*.* Totuși, acest framework doar mi-a pus la dispoziție o varietate de biblioteci pentru a face mai ușoara dezvoltarea aplicației, însă, stilizarea și elementele ce țin de arhitectură au fost realizate folosind HTML5, CSS și JavaScript.

*Backend*-ul aplicației este realizat folosind ASP.NET Core, versiunea 3.1. Acesta preia datele de pe *frontend* și, în principiu, le stochează într-o bază de date MySQL, unde datele sunt stocate în formă tabelară.

Metoda de comunicare folosită este un serviciu web, de tip RESTful API. Am ales această metodă deoarece consider că aplicația poate fi extinsă și pe mobile folosind acest serviciu web.

# Fațeta dezvoltare

Aplicația s-a realizat folosind o metodă agile de dezvoltare, și anume Scrum.

Metoda Scrum constă într-o echipa formată din persoane specializate pe diferite domenii cu scopul de a realiza diferitele funcționalități a aplicației. Membrii echipei trebuie să lucreze împreuna pentru a realiza produsul final, fiecare membru îndeplinindu-și task-urile primite. Inițial, se stabilește un set de sarcini care descriu ceea ce trebuie facut pentru dezvoltarea aplicației. Apoi, aceste sarcini sunt împărțite echipei și sunt rezolvare pe rând, folosind *sprint*-uri, acestea reprezentând intervaluri scurte de timp destinate îndeplinirii sarcinilor primite. În cadrul acestei aplicații, am ales ca aceste sprint-uri să aibe o durată de două saptamâni.

La finalul sprint-ului, făceam o analiză a ceea ce s-a implementat și, pe baza acestei analize, se planificam următorul sprint. În acest fel, puteam îmbunătăți sau schimba orice pe parcursul procesului de creare a aplicației.

# 2. Cerințe de sistem

## 2.1 Surse de cerințe

Principalele surse de cerințe sunt beneficiari acestei aplicații. Beneficiarii sunt reprezentați de medici și pacienții acestora deoarece aceștia vor fi principali actori în cadrul aplicației web. Având în vedere faptul că beneficiarii sunt două mari categori, cu interese proprii, aplicația trebuie să îndeplinească toate aceste interese.

O altă sursă de cerințe a fost reprezentată de alte aplicații folosite pentru programări și rezervări online. Datorită faptului că folosesc preponderent aplicația mobilă *Booking,* aplicație folosită pentru rezervari la diverse hoteluri,am putut prelua multe din caracteristicile și ideile prezente în aceasta, chiar dacă domeniul de activitate este unul complet diferit (în cazul *Booking*-ului, domeniul este turism). Nu în ultimul rând, propria mea imaginație a reprezentat o sursă de cerințe din care am putut extrage unele funcționalități ale aplicației.

## 2.2 Elicitația cerințelor

Principalele părți implicate în această aplicație sunt beneficiarii aplicației, și anume medici și pacienți.

Medici sunt principali utilizatori ai acestei aplicații. Această aplicație va veni în sprijinul acestora pentru a eficientiza modul de organizare și gestionare a pacienților. Rolul lor principal este de a aproba pacienți și să ofere informație în timp real pacienților.

Deși aplicația are rolul de a eficientiza organizarea din cadrul unui cabinet medical și este destinată, în principal, medicilor, și pacienți joaca un rol important. Obiectivul principal a acestora este de a reduce pe cât posibil timpi morți din cadrul unui cabinet medical. Aceștia vor trebui să folosească interfața grafică sugestivă din aplicația web pentru a se programa la medicul de familie.

|  |  |  |
| --- | --- | --- |
| Actori | Rol | Obiective |
| Personal medical | * Să aprobe pacienți * Să comunice în timp real cu pacienți | * Să eficientizeze modul de organizare și gestionare a pacienților |
| Pacient | * Să realizeze programări online * Să dețină propriul cont valid în aplicație * Să respecte orarul de programări | * Să reducă pe cât posibil timpul de așteptare în cabinet |

### 2.2.1 Chestionar

Prima metoda folosită în cadrul acestui proiect pentru a extrage datele de la viitori utilizatori este chestionarul. Chestionarul a fost completat de diferite categori de utilizatori, fapt ce a dus la identificarea principalelor probleme în sistemul actual de organizare din cadrul unui cabinet medical. Din aceste probleme, am reușit să identific cauzele pentru realizarea acestei aplicații web.

Întrebările din cadrul chestionarului au fost concepute astfel încât să încerce să acopere arii cât mai diverse privind probleme organizaționale din cadrul unui cabinet medical, iar faptul că a fost completat de mai mulți utilizatori din diverse categorii a dus la identificarea problemelor majore.

În realizarea acestui chestionar, am luat în calcul mai mulți factori. Primul factor este vârsta pacienților. Privind acest factor, ne putem da seama că principali pacienți din cadrul unui cabinet medical sunt persoane mai în vârstă, care nu optează pentru programări prin intermediul unei aplicații web, deoarece nu se descurcă foarte bine la folosirea tehnologiei și ar trebui ajutați să realizeze o programare sau să creeze un cont în aplicație. De aici am concluzionat faptul că soluția mea pentru gestionarea pacienților și activităților din cadrul unui cabinet medical nu trebuie să înlocuiască complet metoda clasică de programare.

Un alt factor luat în considerare este organizarea. Marea majoritate a cabinetelor medicale nu funcționează dupa un program strict, astfel că programările sunt realizate pe baza principiului *primul venit, primul servit*. De aici a rezultat principala problemă a sistemelor clasice de gestionar: timpul de așteptare mare în cadrul cabinetelor medicale.

Nu în ultimul rând, un alt factor important luat în considerare este comportamentul personalului medical, respectiv al pacientului. Problemele rezultate până acum pot crea un mediu de lucru stresant și iritant, fapt ce duce la reacții nepotrivite din partea personalului medical sau al pacienților.

Chestionarul a fost distribuit mai mult persoanelor în vârstă, motivul fiind faptul că aceste persoane frecventează mult mai des cabinetele medicale, însă nu numai acestor persoane. Toți am avut nevoie de o adeverință sau o trimitere de la medicul de familie, deci toți putem fi considerați viitori utilizatori ai acestei aplicații web.

### 2.2.2 Workshop

Workshop-ul constă în reunirea tuturor părților interesate în realizarea acestei aplicații. Astfel, am organizat o întâlnire în care au fost invitați persoane din toate categoriile sociale. Întâlnirea avea ca scop extragerea cerințelor din toate părțile implicate, astfel încât aplicația să poată acoperi cât mai multe cerințe în cel mai eficient mod posibil.

În cadrul acestei întâlniri au participat persoane de toate vârstele, deoarece toți am avut nevoie de serviciile din cadrul unui cabinet medical de cel puțin câteva ori până acum, însă am încercat să aduc în prim plan persoanele în vârstă, deoarece aceștia frecventează mai des cabinetele medicale.

Contrar așteptărilor mele, principala lor problemă a fost faptul că pierd prea mult timp într-un cabinet medical, mai ales în situație pandemică din prezent, și nu faptul că tehnologia este prea dificilă de folosit sau faptul că nu au acces la internet.

În urma acestei întruniri, au fost identificate următoarele probleme:

* Organizarea deficitară a programărilor;
* Timpul irosit în sala de așteptare;
* Comunicare ineficientă între medic și pacient;
* Dificultatea folosirii tehnologiei, în special pentru persoanele în vârstă;

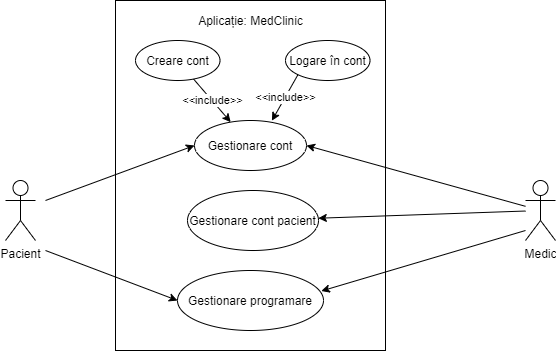
În urma celor discuțiilor și sugestiilor extrase în urma workshop-ului, dar și în urma răspunsurilor primite din cadrul chestionarelor completate, s-a realizat diagrama *Paretto* (figura 3), unde se pot observa principalele probleme ce au dus la realizarea unei aplicații web de gestionare a pacienților și a programărilor acestora. Pe axa verticală apare numărul de apariții a problemelor, iar pe axa orizontală apar probleme propriu-zise. Linia de culoare portocalie reprezintă procentajul cumulativ al problemelor, în funcție de numărul de apariții și raportat la numărul total al acestora.

#### **Figura 3 – Diagrama Paretto**

Așa cum se poate observa în diagrama Paretto (figura 3), principala problemă a beneficiariilor este timpul pierdut. Această problemă a fost sesizată atât în cadrul work-shop-ului, cât și în cadrul chestionarelor completate de beneficiarii aplicației. Datorită faptului că aproape tot eșantionul a considerat această problemă ca fiind prioritară, am plasat această problemă ca fiind prima în cadrul diagramei de ierarhizare a cerințelor.

Totuși, aceasta nu a fost singura problemă sesizată. Dupa cum se poate observa, o bună parte a eșantionului au considerat o problemă majoră și organizarea deficitară din cadrul unui cabinet medical, cât și folosirea tehnologiei (datorată pacienților cu o vârstă înaintată) și comunicarea deficitară dintre pacient și medic.

## 2.2.3 Modelul use-case



## *Figura 4 – Use-case*

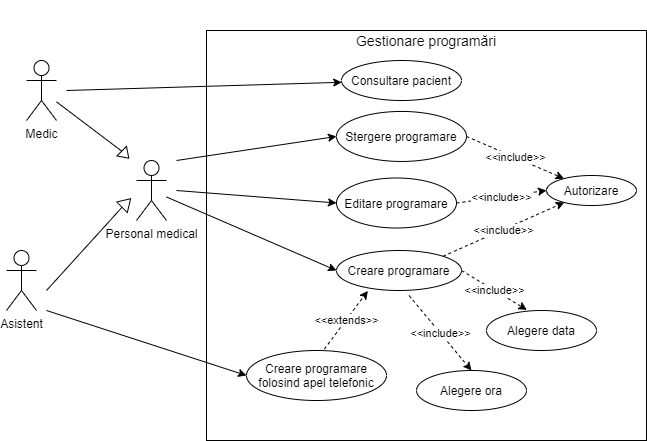
## În cadrul acestui use-case, am încercat să surprind principalele funcționalități din cadrul aplicației. După cum precizasem, vor exista două tipuri de utilizatori în cadrul aplicației: personalul medical, și respectiv pacientul. În figura 4, se poate observa rolurile fiecăruia.

Medicul, respectiv personalul medical, va putea să își gestioneze propriul său cont, introducându-și datele sale proprii și asigurându-se că sunt corect, va putea, de asemenea, să gestioneze datele pacientului și va avea posibilitatea să gestioneze orarul său cu programările (să vizualizeze orarul, să modifice datele din orar și chiar să introducă noi programări).

Pacientul, în schimb, va avea posibilitatea să își creeze programare la cabinet, respectiv să modifice sau să șteargă o anumită programare și va avea posibilitatea de a își gestiona propriul său cont, unde va trebui să își creeze contul, să se autentifice și să fie aprobat de medicul său pentru a reuși să realizeze celelalte funcții din cadrul aplicației.

Pentru o înțelegere mai bună a diagramei și a funcționalităților prezentate, voi realiza în continuare un tabel pentru descrierea primului caz de utilizare a acestei aplicații.

|  |  |  |
| --- | --- | --- |
| Număr | Caz de utilizare | Descriere |
| 1. | Gestionare cont | Funcționalitatea în care, utilizatori, vor avea posibilitatea să își gestioneze propriul lor cont. După cum se poate observa în diagramă, ambele tipuri de utilizatori vor acces la această funcționalitate. Gestionarea contului presupune înregistrarea în cont și apoi autentificarea în cont. |
| 2. | Gestionare cont pacient | Funcționalitate care îi oferă medicului posibilitatea de a edita datele unui pacient. Această funcționalitate este posibilă doar medicului, în caz că este necesar schimbarea din sistem a datelor unui pacient. |
| 3. | Gestionare programare | Probabil principala funcționalitate a aplicației, această funcționalitate va oferi posibilitatea tuturor utilizatorilor de a gestiona eficient programările lor. Această funcționalitate va fi disponibilă ambelor categorii de utilizatori. |



## *Figura 5 – use-case – funcționalitate*

## În figura 5, am încercat să surprind use-case-ul pentru o singură funcționalitate din cadrul aplicației, și anume funcționalitatea de gestionare a unei programări. Aceasta este, probabil, cea mai importantă funcționalitate din cadrul aplicației deoarece îi privește pe ambele categorii de utilizatori (medic și pacient).

Rolul personalului medical, în cadrul acestei funcționalități este asemănator cu cel al pacientului, de gestionare a programărilor sale: editare programare, în caz că orarul de programarări se decalează din cauza urgențelor medicale sau a altor probleme, ștergere programare, sau chiar de creare programare, în caz că programările se realizează pe stilul clasic, adică prin apel telefonic, iar asistentul poate adăuga o programare în orar, în timp ce medicul se poate ocupa de pacientul ce a venit pentru o consultație.

Ca și în cazul use-case-ului general, voi realiza un tabel pentru a înțelege mai bine funcționalitatea de gestionare a programărilor din perspectiva personalului medical.

|  |  |  |
| --- | --- | --- |
| Număr | Caz de utilizare | Descriere |
| 1. | Consultare pacient | Medicul se ocupă de pacientul ce a venit pentru consultație. Deoarece medicul este ocupat de pacient, asistentul medical se poate ocupa de gestionarea programărilor. |
| 2. | Ștergere programare | Opțiunea de ștergere a programării, în cazul în care pacientul nu poate veni și va anunța medicul prin apel telefonic de acest fapt. Această funcționalitate presupune faptul că utilizatorul este autentificat în aplicație. |
| 3. | Editare programare | Opțiunea de a edita o programare, mai exact de a putea să îi schimbe data și ora unei programări, în cazul în care apare o urgență și medicul trebuie să decaleze restul programărilor pentru ca pacienții să nu aștepte. Această funcționalitate presupune faptul că utilizatorul va alege o altă dată și oră validă și că este autentificat în aplicație. |
| 4. | Creare programare | Opțiunea de a crea o programare, am implementat și această opțiune deoarece încă există pacienți care folosesc metoda clasică de programare, și anume un apel telefonic. Astfel, personalul medical poate salva programarea în aplicație pentru ca restul pacienților să fie înștiințați de existența acestei programări. Această funcționalitate presupune faptul că utilizatorul este autentificat în aplicație. |

**Tabel: Descrierea cazurilor de utilizare**

O funcționalitate importantă în cadrul aplicației este reprezentată de înregistrarea/crearea unui cont de pacient în baza de date. Pentru această funcționalitate, voi realiza un scenariu pentru a face cât se poate de clar modul de desfășurare a funcționalități.

|  |  |
| --- | --- |
| **SECȚIUNE** | CONȚINUT |
| **Identificator** | ID1 |
| **Nume** | Creare cont pacient |
| **Autor** | Vlad Dobocan |
| **Versiune** | V 1.0 |
| **Prioritate** | Ridicată |
| **Severitate** | Ridicată |
| **Responsabil** | Vlad Dobocan |
| **Scurtă descriere** | Pentru a putea folosi aplicația, un utilizator trebuie să își creeze cont. |
| **Tip de scenariu** | Scenariu de interacțiune (tipul B) |
| **Obiectiv** | Eficientizarea activităților ce se desfășoară în cadrul unui cabinet medical. |
| **Actori** | Pacientul cabinetului |
| **Precondiție** | Conexiune la internet |
| **Post condiție** | Pacientul va avea acces la principalele funcționalități ale aplicației |
| **Pașii scenariului** | 1. Pacientul accesează aplicația web. 2. Pacientul va introduce datele necesare pentru a crea un cont. 3. Aplicația va trimite un email cu un cod pacientului. 4. Pacientul își va introduce codul de pe email într-un formular. 5. Aplicația va valida contul pacientului. 6. Pacientul va aștepta ca medicul să îi aprobe contul. 7. Medicul va aproba pacientul. 8. Pacientul poate folosi restul funcționalităților din aplicație. |
| **Calitate** | Înregistrarea/crearea unui cont să se realizeze într-un timp scurt. |

**Tabel 1: Documentarea textuală a cazului de creare cont utilizator**

## O altă funcționalitate importantă în cadrul aplicației este realizarea unei programări. Pentru a înțelege și modul de desfășurare a acestei funcționalități, am ales să realizez un scenariu tot sub formă tabelară.

|  |  |
| --- | --- |
| **SECȚIUNE** | CONȚINUT |
| **Identificator** | ID2 |
| **Nume** | Realizare programare |
| **Autor** | Vlad Dobocan |
| **Versiune** | V 1.0 |
| **Prioritate** | Ridicată |
| **Severitate** | Medie |
| **Responsabil** | Vlad Dobocan |
| **Scurtă descriere** | Pacientul va putea folosi aplicația web pentru a realiza o programare la medicul său de familie. |
| **Tip de scenariu** | Scenariu de interacțiune (tipul B) |
| **Obiectiv** | Eficientizarea activităților ce se desfășoară în cadrul unui cabinet medical. |
| **Actori** | Pacientul cabinetului |
| **Precondiție** | Conexiune la internet, cont înregistrat, cont autentificat |
| **Post condiție** | Pacientul va avea stabilită o dată și oră pentru consultația sa. |
| **Pașii scenariului** | 1. Pacientul va trebui să se înregistreze. 2. Pacientul va trebui să se autentifice în aplicație. 3. Aplicația va deschide pagina principală a aplicației. 4. Pacientul va accesa link-ul pentru programări. 5. Pacientul va alege o dată. 6. Aplicația va oferi orele disponibile din aceea dată. 7. Pacientul va alege o oră 8. Aplicația va trimite un email cu data și ora aleasă de pacient. |
| **Calitate** | Realizare programare intr-un timp scurt |
| **Relația cu alte scenarii** | ID1: Creare cont pacient |

**Tabel 2: Documentarea textuală a cazului de creare programare**

## 2.3 Documentarea cerințelor

Înainte de documentarea cerințelor, vreau să reamintesc care este interesul principal al persoanlului medical, respectiv al pacientului, în cadrul acestei aplicații și care sunt cele 2 cerințe principale. Interesul principal al medicului este de a avea un sistem de gestiune a programărilor și pacienților, în cadrul căruia, să poată gestiona pacienții în mod eficient. În schimb, interesul pacienților este să își atingă scopurile personale irosind cât mai puțin timp în cadrul unui cabinet medical. De aici se poate extrage principalele cerințe ale aplicației:

* aplicație web care surprinde activitatea unui cabinet medical
* aplicație web pentru realizarea unei programări în cadrul unui cabinet medical, fără cozi de așteptare și fără timpi morți

Prin intermediul metodelor de elicitație, am reușit să găsesc restul cerințelor din cadrul aplicației. Am categorizat aceste cerințe pentru a putea fi cât de clar posibil la ce se referă acestea și unde se aplică. Cerințele sunt împărțite în:

* cerinţele de sistem - funcționalitățile necesare ale sistemul informatic
* cerinţe funcţionale - funcționalitățile pe care sistemul trebuie să le îndeplinească;
* cerințele non-funcționale - constrângeri ale serviciilor și funcțiilor oferite de către sistem;
* cerinţe calitative – cerințele speciale amintite de beneficiar;

**Cerințele de sistem** sunt cerințele pe care trebuie să le îndeplinească dispozitivul pe care va rula aplicația web. În urma datelor obținute, nu a rezultat faptul că sistemul ar trebuie să dețină cerințe speciale pe care trebuie să le îndeplinească, astfel că sistemul va fi construit să funcționeze pe majoritatea motoarelor de căutare care sunt folosite în prezent, precum:

* Google Chrome – versiunea v70+;
* Mozilla Firefox – versiunea v45+;
* Microsoft Edge – versiunea v12+;
* Internet Explorer – versiunea v11+;
* Safari – versiunea v12+;
* Opera - versiunea v48+;

Aceste cerințe de sistem se aplică datorită faptului că în cadrul aplicației este folost cod TypeScript (superclasa a lui JavaScript), care va fi convertit în cod JavaScript ES6 (EcmaScript 2015), iar versiunile de motoare de căutare amintite anterior suportă complet această versiune a codului JavaScript.

În legătură cu sistemul de operare necesar pentru sistemul ce va folosi aplicația, datorită faptului că aplicația ce urmează să fie realizată este o aplicație web, sistemul de operare nu constituie o cerință specială.

**Cerințele funcționale** constituie principalele funcționalități ale aplicației web. În urma datelor obținute din cadrul metodelor de elicitație, au fost concluzionate, în principiu, cerințele funcționale, deoarece acestea au fost extrase de la viitori utilizatori ai aplicației.

Cerințele funcționale din cadrul aplicației sunt:

* Crearea contului de utilizator;
* Validare email – foarte important pentru trimiteri;
* Conectarea în cont;
* Deconectarea;
* Aprobare pacient din partea medicului;
* Adăugare programări – atât din perspectiva pacientului, cât și a medicului;
* Editare programare;
* Ștergere programare;
* Trimiterea de mesaje în timp real între medic și pacient;
* Notificarea pacientului cu privire la programarea sa;
* Generarea de trimiteri pentru pacienți;
* Recepționarea trimiterilor prin intermediul email-ului;
* Gestionarea pacienților într-un mod clar și sugestiv;
* Posibilitatea de vizualizare a istoricului de programări;

Aplicația va fi realizată folosind 2 categorii de utilizatori: personal medical și, respectiv, pacient, prin urmare cerințele funcționale ce au fost prezentate provin de la ambele categorii de utilizatori.

**Cerințele non-funcționale** sunt cerințele strâns legate de constrângeri ale aplicației. Probabil orice beneficiar dorește o aplicație funcțională doar în câteva zile, cu o investiție foarte mică. Însă acest lucru nu este posibil. Pentru această aplicație, am hotărât ca în cadrul cerințelor non-funcționale să intre doar anumite criterii, precum:

* **timpul de dezvoltare a aplicației** – aplicația să fie realizată până la un anumit deadline. Pentru această aplicație, perioada de timp pentru realizarea ei este de 3 luni.
* **limbajele de programare și tehnologii folosite** – aplicația să fie construită folosind unele limbaje de programare, precum C#, TypeScript, JavaScript, etc, și doar anumite programe pentru ajutarea dezvoltării, precum: Figma, Postman.
* **eficiența sistemului** – timpul de răspuns a sistemului să fie scăzut, pentru ca utilizatori să aibe o experiență cât mai plăcută.
* **anumite standarde de programare**

**Cerințele calitative** sunt preferințe speciale impuse de beneficiarii aplicației web. Aceste cerințe au fost culese direct de la beneficiari, prin intermediul workshop-ului, în special.

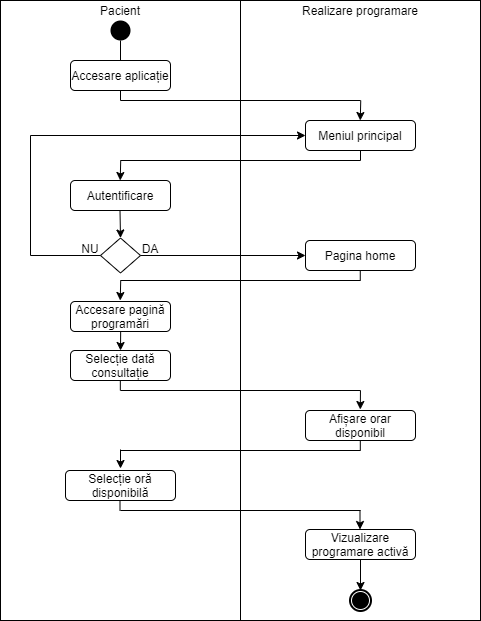
Principalele cerințe calitative ale aplicației web sunt:

* interfață prietenoasă – pentru o interacțiune cât mai bună cu aplicația
* interfață sugestivă
* timp rapid de procesare a cerințelor
* securitate
* confidențialitatea datelor
* o mai bună interacțiune între personalul medical și pacient

Cu ajutorul acestor cerințe pe care le-am extras de la beneficiari prin intermediul chestionarelor și workshop-ului, am realizat această aplicație web care va încerca să rezolve principalele probleme pe care le are sistemul actual de organizare din cadrul unui cabinet medical.

### 2.3.1 Procese și Activități

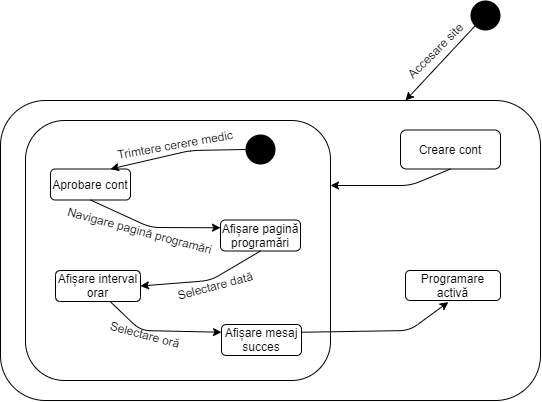
Metoda actuală de realizare a programărilor în majoritatea cabinetelor medicale este prin apel telefonic. Cu toate că această metodă pare a fi una potrivită, de multe ori se dovedește a fi ineficientă, datorită faptului că ține personalul medical ocupat cu conversații care nu au rost. Rezolvarea cu care vine aplicația web este una simplă și ușor de folosit. Astfel, persoanlul medical nu este ocupat cu recepționarea apelurilor telefonice și se poate ocupa mai mult de pacienți. Descriu metoda folosită de aplicația web în următoarea schema:



***Figura 6 – diagrama de activități***

Diagrama de stare ne arată cum și prin ce stări va trece un utilizator când va accesa aplicația web. Inițial, presupunem că utilizatorul nu este încă pe site, așa că acesta trebuie să acceseze link-ul aplicației. Apoi, acesta va trebui să își creeze propriul său cont pentru a putea interacționa cu toate celelalte funcții ale sistemului. După ce acesta va trimite o cerere medicului și va fi aprobat, acesta va putea să selecteze dintr-un mediu sugestiv data și ora când va dori să vină la medic pentru o consultație. În final, acesta se va alege cu o programare activă.

Diagrama de stare a aplicației se poate vedea mai jos:



# *Figura 7 – diagrama de stare*

# 3. Model de Dezvoltare

Metoda de dezvoltare aplicată în cadrul acestei lucrări de licență este metoda Scrum. Folosind această metodă de dezvoltare, am reușit să gestionez dezvoltarea aplicației astfel:

* Am stabilit scopul exact al aplicației și structura sa generală.
* Mi-am propus un set de sarcini pe care am încercat să le îndeplinesc într-o anumită ordine. Pentru fiecare sarcină, am acordat o perioadă de timp, mai exact de 2 săptămâni, astfel încât am reușit să eficientizez modul de dezvoltare a aplicației și să organizez mai ușor proiectul.
* După ce se încheiau cele 2 săptămâni, îmi analizam progresul. Dacă mai rămâneau sarcini de îndeplinit, le adăugam la urmatorul sprint. Astfel, în funcție de progresul fiecărui sprint, cream o listă de sarcini noi ce trebuiau îndeplinite, pentru a dezvolta aplicația până în punctul final.
* Totodată, după fiecare sprint testam și funcționalitățile create în sprint-ul respectiv și că acestea sunt bine integrate în aplicație.
* În cazul în care întâmpinam dificultăți, încercam pe cât posibil să le rezolv în sprint-ul în care acestea apăreau, însă, dacă rezolvarea problemelor dura foarte mult, renunțam la problema respectivă pentru o scurtă perioadă de timp, în cazul în care task-ul care avea problema respectivă nu era o precondiție pentru un alt task.

# Partea II – Proiectarea Sistemului Informatic

# 4. Proiectarea Logică

Modelarea logică este un pas important pentru crearea oricărei aplicații. Astfel că, la fel ca majoritatea sistemelor, și această aplicație web se folosește de modelarea logică a datelor, ceea ce presupune următoarele etape:

* *Input* de date;
* Prelucrarea *input*-ului de date;
* *Output* de date;

În cele ce urmează, voi folosi diagrama de flux pentru a arăta modul în care datele suferă modificări în cadrul aplicației.

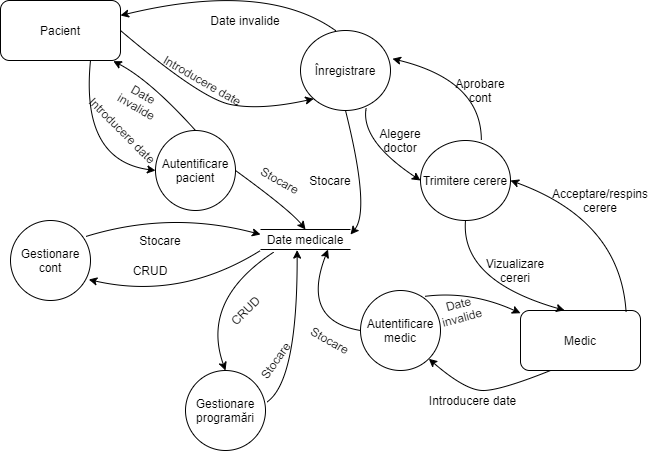


Figura 8 – diagrama de flux de date

Așa cum se poate observa în figura 8 – diagrama de flux de date, funcționalitățile și operațiile care pot avea loc în cadrul acestui sistem informatic sunt strâns legate de tipul de utilizator. Fiecare tip de utilizator poate avea propriile funcționalități, însă pot avea și funcționalități comune, precum gestionarea unui cont sau gestionarea programărilor în cadrul cabinetului medical. Această diagramă este o diagramă de *flow* specifică nivelului întâi.

Totodată, în diagramă se mai poate observa și cum datele circulă prin sistemul informatic. În principiu, în cadrul fiecărui proces intră un set de date, iar în urma procesării *input-*ului primit, procesul fie trimite datele la următorul proces, fie datele sunt stocare în cadrul bazei de date, fie sunt trimise înapoi la utilizator, în cazul în care datele de intrare nu sunt corespunzătoare și corecte. Majoritatea proceselor se află sub protecția autentificări, astfel că utilizatori nu pot face prea multe fără să fie autentificați. Deci, un prim proces ce poate fi atins de ambele categorii de utilizatori este autentificarea (și înregistrarea în cazul pacienților). Ambele categorii de utilizatori trebuie să ofere un set de date de intrare procesului de autentificare, iar acest set de date să fie validat de către server, pentru a putea continua folosirea funcționalităților următoare. Ca *output* în urma acestui proces, utilizatori fie vor avea acces la restul operațiilor și, implicit, la baza de date, fie vor primi un mesaj de eroare în cazul în care datele de intrare nu sunt corecte.

În cele ce urmează, procesele categoriilor de utilizatori sunt asemănătoare deoarece, în principiu, realizează aceleași funcționalități (chiar dacă interfața este diferită). Totuși trebuie menționat faptul că un pacient are și posibilitatea de a crea un cont nou, fapt ce rezultă în salvarea în baza de date a informațiilor introduse de acesta. Odată creat un cont nou, medicul care este ales de către pacient va putea vizualiza lista de cereri pentru pacienți noi și astfel va putea să accepte sau să refuze noul pacient. După ce medicul va accepta cererea pacientului, acesta va avea un cont aprobat și va putea folosi aplicația ori de câte ori este nevoie.

Baza de date relațională este folosită pentru a stoca toate datele medicale ale pacienților și a medicilor. Structura și modul de stocare a datelor vor fi prezentate mai târziu, însă este de reținut faptul că pacienții și personalul medical vor putea accesa datele din baza de date doar după ce sunt autentificați. Fiecare pacient va avea dreptul de a modifica datele create de aceștia și apoi să le salveze, pentru a fi păstrate în baza de date.

## 4.1 Arhitectura Sistemului

## Identificarea arhitecturii sistemului este un pas foarte important în realizarea proiectării oricărui sistem informatic. Arihitectura unui sistem ne arată cum componentele unui sistem lucrează împreună pentru a realiza funcționalitățile prevăzute în cerințe.

Aplicația web prezentată în această lucrare este realizată pe baza arhitecturii client-server. Așa cum sugerează și numele, acest tip de arhitectură are la bază 2 părți importante:

* Clientul – care se ocupă de realizarea cererilor și afișarea răspunsurilor primite de la server – realizat în Angular;
* Serverul – se ocupă de crearea răspunsurilor pe baza cererilor primite de la client și trimiterea acestor răspunsuri înapoi la client – realizat în ASP.NET Core;

Este important de reținut faptul că nu există răspuns fără cerere, însă poate exista cerere care să nu primească niciun răspuns. De asemenea, se pot trimite mai multe cereri de către un client spre server, iar serverul poate să răspundă la mai mulți clienți. Ambele părți din arhitectură lucrează împreună pentru a îndeplini cu succes sarcinile.

În cazul aplicației web *MedClinic,* serverul este format din *endpoint*-uri ce pot realiza operații CRUD (create, read, update, delete) asupra bazei de date, iar clienții sunt reprezentați de utilizatorii finali: pacienți și personalul medical.

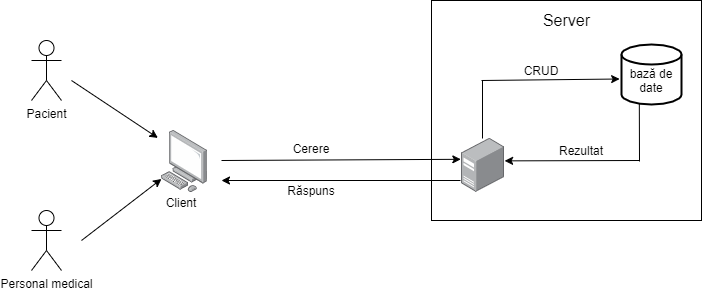
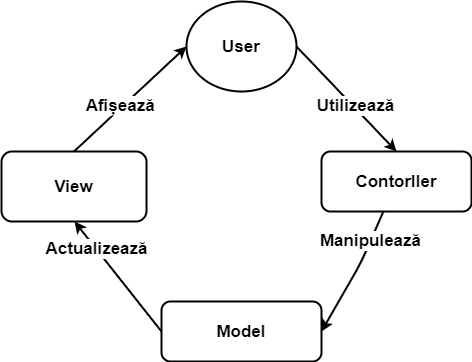


Figura 9 – arhitectura sistemului

În figura 9 – arhitectură sistem - se poate observa mai bine cum componentele sistemul comunică și cooperează una cu cealaltă pentru a îndeplini principalele funcții ale aplicației. Utilizatori interacționează cu clientul (o interfață prietenoasă, ușor de înțeles și simplă) pentru a crea o cerere către server. Această cerere poate conține date noi, pe care utilizatorul dorește să le salveze în baza de date sau poate să fie o cerere prin care utilizatorul dorește să modifice sau să șteargă datele existente în baza de date.

După ce este construită cererea, aceasta este trimisă apoi server-ului unde cererea este procesată. Serverul verifică ce se dorește și va îndeplini ceea ce se află în cerere, iar acesta va construi un răspuns pentru a-l înapoia clientului (de obicei, răspunsul va fi un mesaj că operația a fost îndeplinită cu succes sau un mesaj de eroare în cazul în care ceva nu a mers cum trebuie).

În ceea ce privește structura codului, pe partea de server am folosit un model care se aseamănă foarte mult cu pattern-ul MVC (model, view, controller), numai că view-ul a fost reprezentat la mine de către Angular. Acest pattern ne ajută să împărțim responsabilitățile între diferitele componente ale sistemului și să menținem codul curat, ordonat și ușor de refolosit. Pentru a înțelege mai bine modul în care pattern-ul MVC funcționează, am realizat următoarea diagramă.



***Figura 10 – pattern-ul MVC***

Așa cum se poate observa în diagramă, MVC este alcătuit din 3 componente importante: model, view și controller. În cele ce urmează voi explica fiecare componentă și asemănările și deosebirile între acest pattern și arhitectura ce o folosesc în cadrul aplicației (client-server) și de ce aceste arhitecturi sunt foarte asemănătoare.

Componenta model reprezintă datele cu care aplicația interacționează. Acestea reprezintă fie datele care sunt transferate între celelalte componente sau între celelalte funcții ale sistemului, fie date legate de logica sistemului. O caracteristică importantă a acestei componente este faptul că aceasta nu realizează metode sau modalități de transformare/modificare a informației, ci doar o rețin. În cadrul aplicației de față, modelele sunt strâns corelate cu baza de date, iar baza de date este realizată folosind tehnica CodeFirst (vezi Structura fizică a datelor). Fiecare Model reprezintă o clasă, iar fiecare proprietate din această clasă va reprezenta un atribut în cadrul tabelului din baza de date.

Componenta Controller reprezintă modul în care server-ul interacționează cu diferite cereri, în funcție de ruta pe care o anumită cerere vine. În controller are loc transformarea sau modificarea datelor, criptarea lor și respectiv salvarea acestora în baza de date. Fiecare Controller deține o serie de rute ce permite clientului să trimită/acceseze date. Este important de subliniat faptul că datele ce sunt transmise de la client au forma unei clase-model, fapt ce ne oferă posibilitatea de reutilizare a codului. Totuși, aceste date nu sunt exact clase-model, ci mai degrabă un *DTO* (data transfer object), care reprezintă o imitație a claselor model, însă cu mici diferențe de structură. Aceste obiecte ne ajută să nu fie necesar utilizarea întregilor date reținute de clase model, ci doar porțiuni din aceste date, mai exact putem folosi doar ce este necesar. Spre exemplu, modelul *User* poate conține parole, coduri PIN sau alte date ce nu vrem să le trimitem clientului.

Componenta *View* în pattern-ul MVC are un rol foarte important deoarece prin intermediul acestei componente, utilizatorii interacționează cu sistemul și cu informațiile ce sunt stocare în baza de date. Totuși, în cadrul acestei aplicații, view-ul este reprezentat de client, ce poate fi o aplicație web, ca în cazul aplicației de față, sau o aplicație mobile/desktop. Clientul este realizat folosind framework-ul Angular 10, un framework de TypeScript ce facilitează crearea aplicațiilor de tip Single Page Applications (SPA). Câteva diferențe între componenta view clasică și cea abordată în această aplicație web ar fi:

* Gestionare datelor – în cadrul aplicațiilor de tip MVC, gestionarea datelor este realizată de server și pe partea de Client, însă în cadrul aplicațiilor de tip client-server, clientul este nevoie să își gestioneze singur datele deoarece server-ul nu este implicat în partea de front-end al aplicației; acesta doar trimite/recepționează datele.
* Viteza de răspuns – în cadrul aplicațiilor MVC, view primește datele mai repede și poate duce la o experiență mai plăcuta pentru utilizator; pentru aplicațiile de tip client-server, datele pot avea un delay destul de mare – acest delay depinde de mai mulți factori, însă e important de realizat faptul că un client se poate folosi de rutele unui controller de oriunde de pe glob.
* Securitatea – pattern-ul MVC oferă o securitate a datelor puțin mai bună decât aplicațiile realizate folosind o structură client-server.

Totuși, aplicațiile SPA sunt aplicațiile ce interacționează cu utilizatorul într-un mod dinamic și nu este necesară reîncărcarea întregii pagini în timpul utilizării. De aici rezultă faptul că timpul de așteptare ar trebui să fie cât mai mic, existând o singură pagină pe care o poți vizita și ce apoi încarcă alt conținut.

## 4.2 Baza Informaţională

Baza informațională este una dintre principalele componente ale unui sistem informatic. Aceasta cuprinde ansamblu colecțiilor de date cu care aplicația va lucra pentru a realiza scopul pentru care aplicația a fost proiectată.

Sistemul de față este construit cu scopul de a surprinde activitatea din cadrul unui cabinet medical. Pentru aceasta, este nevoie să identificăm principalele tipurile de date de care avem nevoie pentru ca sistemul să funcționeze și să își îndeplinească scopul. După cum am preziat în capitolul anterior (capitolul 1.2.1, Fațeta Utilizare), avem 2 tipuri de utilizatori în cadrul aplicației: personal medical/medic și, respectiv, pacienți. Este nevoie să identificăm propriile interese a fiecărui tip de utilizator pentru a reuși să identificăm cu succes tipurile de date de care avem nevoie. Aceste interese au fost identificate folosind metode de elicitație a cerințelor, astfel că interesele stackholderilor au devenit cerințe funcționale pentru aplicație. Cerințele funcționale reprezintă principala sursă de unde putem afla ce tipuri de date avem nevoie.

Fiecare categorie de utilizator prezintă interese diferite, ceea ce înseamnă că este nevoie să stocăm date despre fiecare categorie în parte pentru a putea să oferim utilizatorilor ce au nevoie. De exemplu, pacientul dorește să realizeze o programare la medic fără să mai folosească un apel telefonic, iar data și ora să fie clar stabilite, în timp ce medicul dorește să gestioneze programările sale și pacienții într-un mod eficient. Pentru a fi posibil acest fapt, este nevoie să reușim să diferențiem tipurile de utilizator între ei și să reținem infomații diferite în baza de date în funcție de tipul de utilizator. De asemenea, există funcționalități specifice numai unei categorii de utilizatori. De exemplu, pacientul poate vizualiza un istoric medical, să vadă când acesta a fost ultima oară la medic, iar medicul să poată vizualiza programările întregilor pacienți și când acestea sunt programate.

Putem conluziona faptul că baza de cunoștințe reprezintă o componentă principală în cadrul oricărui sistem informatic, componentă fără de care un sistem nu ar reuși să îndeplinească principalele sale funcționalități și nu ar putea diferenția între diferitele tipuri de utilizatori. Pentru a îndeplini toate acestea, am creat o bază de date relațională care va stoca toate informațiile de care are nevoie un sistem de acest fel pentru a reuși să își atingă cu succes toate scopurile sale.

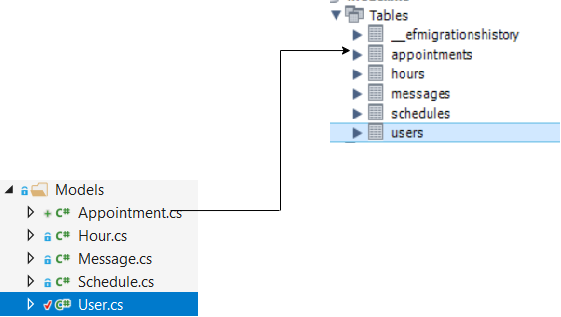
# 5. Proiectarea Tehnică

## 5.1 Structura fizică a datelor

Un pas foarte important în crearea aplicației a fost proiectarea bazei de date deoarece acolo stocăm datele provenite de la utilizatori. O bază de date este o modalitate de a stoca informații și date provenite de la stackholderi, cu facilități de găsire a datelor într-un mod rapid și eficient. Proiectarea bazei de date din cadrul aplicației a fost realizată astfel încât să acopere principalele cerințe ale stackholderilor. Ne amintim că, în cadrul aplicației web, avem 2 categorii de utilizatori: personalul medical și pacienți.

Scopul bazei de date este de a surprinde întreaga activitate a unui cabinet și de a stoca datele atât al pacienților, cât și al medicilor. Este important ca baza de date să stocheze datele fiecărui tip de utilizator, pentru a nu se produce erori în cadrul aplicației. Logica de business a aplicației va trebui să identifice tipurile de utilizator, pentru a reuși să ofere fiecărei categorii de utilizatori de ce are nevoie.

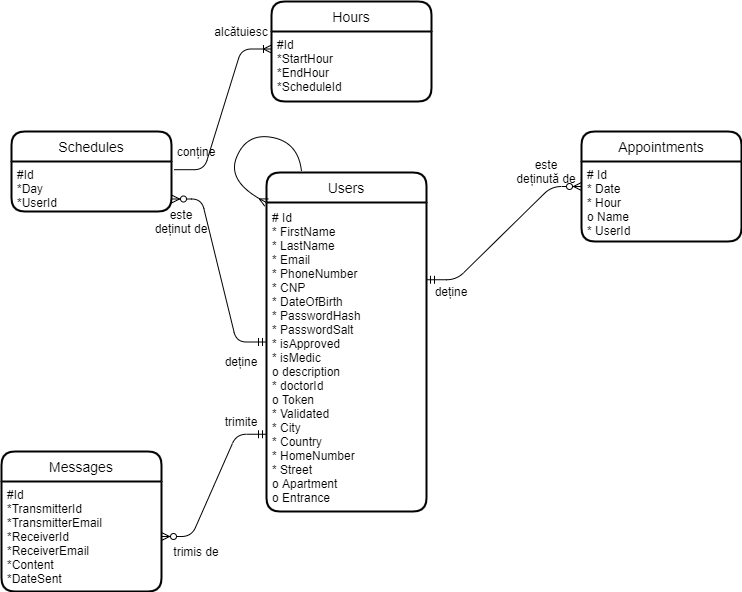
O dată stabilit scopul bazei de date, am realizat tabelele. SGBD-ul ales pentru baza de date este MySQL, iar tabelele din baza de date au fost generate de către *EntityFramework*, folosind pattern-ul *CodeFirst*. Primul pas a fost să creez clasele model în cadrul proiectului. Aceste clase-model conțin toate informațiile considerate necesare pentru a îndeplinite toate cerințele stackholderilor. Apoi, am folosind comenzile specifice EntityFramework-ului pentru a genera tabelele în baza de date cu relațiile dintre acestea prin intermediul migrărilor. Relațiile dintre tabele se realizează prin intermediul constrângerilor referențiale. Acestea sunt denumite și chei străine. Pe lângă cheile străine, fiecare tabel include și cheia primară, un câmp ce poate identifica unic fiecare rând.



***Figura 11 – relație model-entitate***

Așa cum se poate vedea în imagine, fiecare clasă model a generat un tabel în baza de date, însă numele tabelului nu a preluat întocmai numele clasei. Se poate observa că tabelul a luat numele clasei, l-a transformat în lower-case și a pus la plural.

Diagrama bazei de date generată de EntityFramework se poate vedea în figura următoare. Se poate vedea că baza de date se află în forma normală a III-a și că relațiile dintre tabele sunt de one-to-many. De asemenea, se poate observa și care atribut este opțional în baza de date și care este obligatoriu (cele marcate cu „o” sunt opționale, iar cele marcate cu „\*” și „#” sunt obligatorii).



***Figura 12 – Schema bazei de date***

În continuare voi lua fiecare tabel și voi explica ce rol are fiecare atribut din cadrul fiecărui tabel și relațiile dintre tabele.

1. Tabelul ***Users***

Tabelul Users este principalul tabel din cadrul bazei de date. Acest tabel cuprinde toate informațiile necesare despre un anumit utilizator. În continuare voi explica fiecare câmp din acest tabelul, ce rol are acesta și unde este folosit.

|  |  |  |  |
| --- | --- | --- | --- |
| CP | \* | Id | Identificatorul unic al utilizatorului |
|  | \* | FirstName | Prenumele |
|  | \* | LastName | Numele de familie |
| CU | \* | Email | E-mailul utilizatorului |
| CU | \* | PhoneNumber | Numărul de telefon al utilizatorului |
| CU | \* | CNP | CNP-ul utilizatorului |
|  | \* | DateOfBirth | Data nașterii utilizatorului |
|  | \* | PasswordHash | Parola criptată a utilizatorului |
|  | \* | PasswordSalt | Cheie de criptare pentru parolă |
|  | \* | isApproved | Câmp ce arată starea contului:  1 – cont aprobat de medic  0 – cont încă neaprobat de medic |
|  | \* | isMedic | Câmp ce arată rolul unui utilizator  0 – Pacient  1 – Medic |
| CS | \* | doctorId | Cheia străină pentru tabelul virtual pentru medici |
|  | o | Token | Token trimis pe e-mail pentru validare |
|  | \* | Validated | Câmp ce arată starea adresei de e-mail:  1 – adresa de e-mail validată  0 – adresă de e-mail nevalidată |
|  | \* | City | Orașul de proveniență a utilizatorului |
|  | \* | Country | Țara de proveniență a utilizatorului |
|  | \* | HomeNumber | Numărul de la locuința utilizatorului |
|  | \* | Street | Numele străzi utilizatorului |
|  | o | Apartment | Apartamentul locuinței utilizatorului |
|  | o | Entrance | Scara utilizatorului |

1. Tabelul ***Appointments***

Tabelul Appointments stochează date cu privire la programările unui utilizator: a cui este programarea, la ce oră este programată și data acesteia. De asemeenea, această tabelă are și un câmp special: *Name*, câmp ce este folosit pentru posibilitatea de a realiza programări de către medic (în cazul în care un pacient va apela medicul pentru o programare, medicul va putea salva în baza de date această programare cu un nume pentru a știi a cui este programarea).

|  |  |  |  |
| --- | --- | --- | --- |
| CP | \* | Id | Identificatorul unic pentru programare |
|  | \* | Date | Data când va avea loc programarea |
|  | \* | Hour | Ora programării |
| CU | o | Name | Nume pentru programare |
| CS | \* | UserId | Cheie străină ce realizează legatura cu tabelul Users |

1. Tabelul ***Messages***

Acest tabel stochează mesajele trimise de la un utilizator la altul. Am ales să stochez și mesajele pentru a putea vizualiza mai târziu data când acestea au fost trimise și textul din interiorul mesajelor.

|  |  |  |  |
| --- | --- | --- | --- |
| CP | \* | Id | Identificatorul unic al mesajului |
|  | \* | TransmitterId | ID-ul utilizatorului care a trimis mesajul |
|  | \* | TramsmitterEmail | E-mailul utilizatorului ce a trimis mesajul |
|  | \* | ReceiverId | ID-ul utilizatorului ce primește mesajul |
|  | \* | ReceiverEmail | Email utilizatorului ce primește mesajul |
|  | \* | Content | Conținutul mesajului |
|  | \* | DataSent | Data când mesajul a fost trimis |

1. Tabelul ***Schedules***

Acest tabel conține programul de funcționare a unui cabinet medical. Este în strânsă legătură cu tabelul ***Hours***. Un medic poate lucra de luni până vineri, iar introducerea unui orar în baza de date este realizată de către administratorul bazei de date (se va implementa și partea de administrator, însă acest lucru este prevăzut în direcții de dezvolvări viitoare).

|  |  |  |  |
| --- | --- | --- | --- |
| CP | \* | Id | Identificatorul unic al unei zile |
|  | \* | Day | Numele zilei respective |
| CS | \* | UserId | Cheie străină ce realizează legatura cu tabelul Users |

1. Tabelul ***Hours***

Acest tabel stochează orele când un medic este disponibil. În cadrul acestei aplicații vom presupune că orice programare durează 30 de minute, astfel că EndHour va fi StartHour la care adăugăm cele 30 de minute. Însă acest lucru se va schima pe viitor (vezi direcții de dezvoltări viitoare) .

|  |  |  |  |
| --- | --- | --- | --- |
| CP | \* | Id | Identificatorul unic al unei zile |
|  | \* | StartHour | Ora la care este programat un pacient |
|  | \* | EndHour | Ora la care este încheiată programare |
| CS | \* | ScheduleId | Cheie străină ce realizează legătura cu tabelul Schedules |

## 

## Procese şi Algoritmi

Pentru a reuși să își îndeplinească scopul și funcționalitățile principale, această aplicație se folosește de procese și algoritmi bine definiți. În cele ce urmează, voi descrie principalele procese și algoritmi și modul în care aceștia cooperează pentru a realiza ceea ce este propus.

1. **Procesul de înregistrare**

Pentru a putea folosi această aplicație web, este nevoie ca utilizatori să își creeze propiul cont. Procesul de creeare a unui cont este cel utilizat în majoritarea aplicațiilor, și acesta constă în:

**Procedura** regiter()

*User-ul introduce date in formularul de register;*

*Date sunt trimite la server folosind cerere HTTP POST;*

*Se verifică dacă email-ul există în baza de date:;*

***IF*** *email ∃ în BDE*

***Return*** *“Email deja existent”;*

*Se cripteaza parola folosind algoritmul HMACSHA512;*

*Se salvează PasswordSalt rezultat;*

*Se salveaza utilizatorul în BDE;*

*Trimire un email spre adresa de email salvată anterior;*

*Adaugă user-ul în lista de așteptare medic;*

***Return*** *“Cont creat”;*

***Redirect*** *user la Home;*

1. **Procesul de autentificare**

Este foarte important să nu permitem oricui să realizeze modificări în baza de date. Pentru asta va fi nevoie să autentificăm utilizator înainte de a putea orice altceva în cadrul aplicației. Procedura *Login()* de mai jos prezintă implementarea procesului de autentificare pas cu pas.

**Precondiție: contul user-ului să fie deja în baza de date**

**Procedura** Login()

*User-ul introduce datele in formularul de login;*

*Datele sunt trimise la server folosind cerere HTTP POST;*

*Se caută în baza de date email-ul introdus de user;*

*Se creeaza un obiect de tipul HMACSHA512 folosind PasswordSalt de la user-ul găsit anterior;*

*Se criptează parola introdusă de utilizator folosind obiectul creat anterior;*

***For*** *fiecare bit din parola criptată*

***If*** *bit din parola criptată != bit din PasswordHash al user-ului din baza de date*

***Return*** *"Invalid credentials";*

*Se creează un token folosind* ***Procedura*** *createToken();*

***Return*** *token;*

*Salvăm token în local storage* ***;***

1. **Procesul de autorizare**

Amintim că, în cadrul aplicației, există 2 tipuri de utilizatori. Este nevoie să diferențiem aceste tipuri de utilizatori pentru a furniza funcționalități diferite acestora. Autorizarea este realizată pe baza unui token, ce este necesar să fie atașat la cererile de pe front-end. Modul prin care este creat un token este prezentat în continuare:

**Precondiție: contul user-ului să fie deja în baza de date**

**Procedura** CreateToken()

*Se creează* ***claims****, obiect ce conține: emailul user-ului, rolul, approved și validated;*

*Se salvează* ***claims*** *în tokenDescriptor;*

*Se salvează timpul de expirare a token-ului în tokenDescriptor;*

*Se creează token-ul folosind acest tokenDescriptor;*

*Se returneaza token-ul serializat;*

1. **Procesul de creare a unei programări:**

Probabil una dintre cele mai importante funcționalități în cadrul aplicației este funcționalitatea de a crea o programare doar prin câteva click-uri. Procesul pentru această funcționaitate este descris în cele ce urmează:

**Precondiție: contul user-ului să fie deja în baza de date**

**Procedura** CreateAppointment()

*Procedura Login();*

*Navigare la meniul de creare programare;*

*Afișarea datei și orelor disponibile;*

*Selectare data și ora;*

*Trimiterea datelor pe server, folosind cerere HTTP POST;*

*Verificarea disponibilități datelor selectate: Procedura AppointmentDateExist();*

*Salvarea datelor selectate în baza de date;*

*Notificare pe email cu data și ora;*

***Return*** *“Programare creată”;*

**Procedure** AppointmentDateExist()

*Se extrage din baza de date programarea ce conține data și ora selectată de user;*

***If*** *data și parola din baza de date == data și ora aleasă de user*

***Return*** *"This date is already used!";*

1. **Procesul de aprobare pacient**

După ce un pacient își creează cont în aplicație, acesta trebuie să fie aprobat de medicul ales de acesta pentru ca acesta să poate să realizeze programări în cadrul aplicației sau să utilizeze restul funcționalităților. Procedura de aprobare a unui cont de către medic este descrisă în cele ce urmează:

**Precondiție: contul medicului să fie deja în baza de date**

**Procedură** approveUser()

*Procedura Login();*

*Navigare la pagina Pacients;*

*Vizualizare utilizator nou;*

*Apăsare buton Approve User;*

*Trimitere cerere HTTP GET cu* ***ID****-ul user-ului ales;*

*Căutare user după ID în baza de date;*

*Modificarea câmpului* ***isApproved*** *în 1;*

*Salvare modificare în baza de date;*

***Return*** *“User approved”;*

1. **Procedura de modificare dată și oră a unei programări - medic**

În unele cazuri, medicul va trebui să modifice ora (în special) și data când are loc o programare. Astfel, această funcționalitate joacă un rol important în această aplicație.

**Procedură updateAppointment()**

*Procedura Login();*

*Navigare la pagina Appointments;*

*Selectare data din cadrul unui calendar;*

*Selectare programare;*

*Deschidere meniu de update programare;*

*Selectare oră și data nouă;*

*Trimitere date la server;*

*Procedura AppointmentDateExist();*

*Salvare data și ora în baza de date;*

*Notificare prin email pe utilzatorul ce deține programarea;*

***Return*** *“Succes”;*

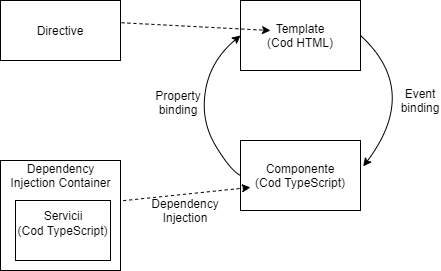
## Tehnologii specifice

În fațeta IT am realizat o descriere succintă a tehnologiilor folosite în cadrul acestei aplicații web. În acea secțiune, am specificat faptul că aplicație are un *backend* și un *frontend* realizat în două limbaje de programare diferite.

* **Frontend-ul aplicației**

*Frontend*-ul aplicației este realizat folosind *Angular* v10, un framework scris în limbajul de programare TypeScript. Acest framework este dezvoltat de o echipa Angular de la google și programatori individuali și diverse companii. Totuși, trebuie subliniat faptul că Angular v10 (sau *Angular* *2+*) este diferit de predecesorul său, *AngularJS*. Echipa de dezvoltare a rescris complet framework-ul *AngularJS* pentru a crea o versiune mai bună și mai eficientă a acetuia, denumind această nouă versiune *Angular 2+*.

În cele ce urmează, voi prezenta arhitectura acestui framework și cum reușește acesta să coopeze în cadrul sistemului. Pentru aceasta, voi folosi următoarea diagramă (figura 12).



***Figura 13 – arhitectura Angular***

Așa cum se poate observa în figura 13 – arhitectura Angular – la baza acestui framework stau componentele. **Componentele** sunt *directive* ce sunt reutilizabile, iar aceste componente pot conține diverse proprietăți și diverse metode ce sunt legate de codul HTML. Fiecare componentă prezintă câte o funcționalitate a sistemului întreg, iar această componentă poate fi utilizată ori de câte ori vrem și oriunde vrem. Acesta este marele avantaj a acestor componente, reutilizabilitatea lor. Angular este capabil să injecteze aceste componente în paginile de frontend a aplicației.

**Directivele**Angular sunt clase de TypeScript ce sunt utilizate pentru a extinde puterea codului HTML, oferindu-i o nouă sintaxă. Fiecare directivă are un nume, fie unul definit de Angular, fie unul personalizat de utilizator. Acestea se pot folosi în diferite locuri, în funcție de cum sunt definite: poate fi un tag în cod HTML (componentele), poate fi un atribut pentru un cod HTML, o clasă sau chiar un comentariu.

Directivele sunt de 2 feluri:

* Directive structurale – aceste directive sunt menite să modifice conținutul paginii HTML, nu doar stilul acestuia (ca exemplu: \*ngFor, \*ngIf, etc);
* Directive funcționale – aceste directive sunt menite să modifice modifice anumite elemente în pagina HTML, însă nu modifică conținutul în sine (nu adaugă conținut nou) (ca exemplu: ngStyle, ngClass, etc);

**Serviciile** sunt clase care au ca principal obiectiv organizarea și partajarea modele, date sau funcții cu diferite componente ale unei aplicații Angular. Acestea sunt instanțiate o singură dată pe durata de viață a unei aplicații. Și aceste servicii pot fi de mai multe tipuri, în funcție de rolul pe care acesta îl au în cadrul sistemului:

* Servicii de date – în cadrul acestor servicii putem defini conexiunea cu baza de date (sau cu backend-ul aplicației). Putem specifica rutele de unde vor veni datele și putem forma diverse cereri pentru aceste rute;
* Servicii de comunicare – în cadrul acestor servicii putem defini funcții necesare în mai multe componente sau putem stoca date, iar componentele vor putea prelua aceste date și se vor putea folosi de ele.

După cum specificasem anterior, componentele se folosesc de servicii pentru a prelua date sau pentru a trimite date către server. Pentru a se folosi de aceste servicii, angular apelează la mecanismului de dependency injection**. Dependency Injection** este un design pattern prin care o clasă solicită dependențe dint-o sursă externă, mai degrabă decât să le insanțieze. Angular se folosește de acest mecanism pentru a injecta toate dependențele unei componente la instanțierea acesteia, iar folosind acest pattern, se sporește flexibilitatea și modularitatea aplicației. Serviciile din angular sunt salvate în containerul de dependency injection, iar când o componentă sau alt serviciu are nevoie de o instanță din acel container, angular se va ocupa de aceasta.

Un ultim aspect de discutat în cadrul acestui framework este modalitatea de comunicare dintre pagina HTML și o componentă (sau codul de TypeScript). Așa cum specificasem la început, componentele conțin diverse proprietăți și metode ce pot fi utilizate de către codul HTML. Pentru a realiza comunicarea dintre proprietățile și metodele unei componente și codul HTML, Angular apelezează la **property binding** și **event binding.**

**Property binding** este modalitatea prin care Angular oferă elementelor/tag-urilor HTML conținut. Cu property binding, putem face lucruri cum ar fi comutarea funcționalității unui butonului, setarea căilor programate și partajarea valorilor între componente.

**Event binding** ne permite să reacționăm la acțiuniile unui utilizator, precum apăsarea unui buton, apăsarea tastelor, mișcările mouse-ului, etc. Cu event binding, putem lega aceste evenimente de diverse funcții, iar când acestea au loc să putem răspunde utilizatorului cu o acțiune (trimiterea datelor pe server, navigarea pe altă pagină, etc).

Tot pe partea de frontend s-a folosit și un software ce a ajutat la crearea design-ului paginilor web, software denumir **Figma**. Acest software este un editor de grafică și o unealtă de prototipare care se bazează în principal pe design-ul aplicațiilor web. În cadrul acestuia, am realizat diferite imagini ce au fost translatate în limbajul **SVG** și apoi au fost folosite pe frontend. Totodată, tot folosind acest software am realizat și imaginea de pe pagina Home, o imagine caracteristică unui cabinet medical și care să sugereze seriozitate și punctualitate.

* **Backend-ul aplicației**

Backend-ul aplicației a fost realizat folosind framework-ul ASP.NET Core versiunea 3.1 și limbajul de programare C#. Acest framework este dezvoltat și întreținut de o echipă de la Microsoft. Spre deosebire de succesorul său, ASP.NET, acesta este un framework gratuit și open-source și rulează pe toate sistemele de operare, nu doar pe Windows. Acest framework se folosește de NuGet pentru a-și organiza pachetele și dependențele sale. **NuGet** este un manager de pachete conceput pentru a permite utilizatorilor să partajeze cod reutilizabil.

Framework-ul se folosește de mai multe **servicii web (Web API)** pentru a reuși să comunice cu componenta de frontend. Am ales acest tip de aplicației deoarece se dorește implementarea și unei aplicații mobile pe viitor, pentru ca utilizatori să poată accesa mult mai ușor informația din cadrul aplicației.

O altă componentă foarte importantă în cadrul ASP.NET Core ce am folosit este Entity Framework. **Entity framework** este un framework de mapare obiect-relațional ce oferă un mecanism automat pentru dezvoltatori să stocheze și acceseze datele din cadrul unei baze de date. Acest framework a introdus pattern-ul **Code-First** pentru crearea și gestionearea ulterioară a bazei de date. În acest pattern, dezvoltatori se concentrează pe aplicație și pe implementarea claselor model ce urmează ca mai apoi acestea să reprezinte entități în cadrul bazei de date. Tot procesul are loc prin intermediul migrărilor, care reprezintă o cale de a face actualizarea bazei de date pentru ca aceasta să țină pasul cu aplicația și logica din spatele acesteia.

Pentru testarea serviciilor web, s-a folosit **Postman**. Postman este o platformă de colaborare pentru dezvoltarea și testarea API-urilor. Funcțiile Postman simplifică fiecare pas al construirii unui API și eficientizează colaborarea, astfel încât să putem crea API-uri mai bune și mai rapid. În cadrul acestuia am creat o colecție denumită **MedicProjectCollection** unde se va realiza testarea fiecărui API în parte.

Pentru dezvoltarea aplicației, am folosit ca și IDE **visual studio code** (VS Code). VS Code este un editor de cod realizat de Microsoft, iar funcțiile acetui editor ajută foarte mult dezvoltatorul la scrierea de cod. Chiar dacă pentru ASP.NET Core este recomandat folosirea editorului Visual Studio (Visual Studio), eu am preferat VS Code deoarece acesta facilitează foarte mult scrierea de cod atât în TypeScript, cât și în CSS și HTML, iar adăugând o simplă extensie pentru C#, acesta poate fi folosit și pentru scrierea de cod pe backend, cât și pentru rularea comenzilor de migrare.

*Descrieţi şi justificaţi tehnologiilesoftware/hardware folosite, limbaje de programare, sisteme de operare, alte dependenţe tehnice etc.*

*Conținutul din această secțiune trebuie să fie corelat cu fațeta IT din Partea I.*

*Descrieți framework-uri, API-uri folosite și asigurați-vă în special de coerența cu secţiunile 4.1 și 5.1 ale proiectului.*

# 

# Glosar

Svg, api, IDE

*Aceasta este o secțiune foarte importantă a raportului de analiză, aici fiind prezentate definiții, explicații, descrieri detaliate ale conceptelor importante care caracterizează spațiul problemei abordate.*

*Conceptele sunt prezentate sub formă de listă ordonată alfabetic. În glosar pot fi folosite și scheme sau diagrame pentru o cât mai clară explicare a termenilor. Totodată, pot fi descrise relații între concepte.*

***Obligatoriu, începeți elaborarea glosarului odată cu specificarea spațiului problemei.***

***Minimum 10 de termeni. Introducerea termenilor banali pentru atingerea numărului minim de termeni este penalizată cu 0.1 puncte/termen din nota raportului de analiză.***

# Bibliografie

# Anexe

Chestionar – model de chestionar completat de un utilizator

1. În ce mediu trăiți? \*

* Urban
* Rural

1. Ce vârstă aveți? \*

* Sub 25 ani
* 25-40 ani
* 40-60 ani
* Peste 60 ani

1. Pe o scară de la 1 la 10, cât de mult folosiți internetul? \*

Răspuns: 6

1. Cât de des mergeți la medicul de familie? \*

* O dată pe lună
* O dată la câteva luni
* dată pe an
* Numai când am nevoie

1. Cât timp pierdeți, în medie, la medicul de familie? \*

* 0-30 minute
* 30-60 minute
* Peste 60 de minute

1. Medicul vă oferă activități pe care să le faceți cât timp sunteți în așteptare? \*

* Da
* Nu
* Nu am nevoie.

1. Cum se comportă personalul medical când mergeți la consultație? \*

* Amabil, respectuos
* Nerespectuos
* Indiferent
* Other:

1. Ce metodă de comunicare folosiți pentru a comunica cu medicul dumneavoastră? \*

* Telefon
* Email
* Mesaje
* Other:

1. Ați mai folosit alte aplicații pentru a realiza programări online? \*

* Da
* Nu