Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Programul de studii:
Informatică Aplicată

Lucrare de licență

Aplicație Web pentru gestiunea unui cabinet medical

Autor: **Bogdan-Valentin Alban**

Coordonator ştiinţific: **Lect. dr. Vlad Monescu**

Braşov

Iunie - Iulie 2023

Cuprins

CAPITOLUL 1

INTRODUCERE

- **Scurtă descriere a lucrării**
- **Motivația alegerii temei**
- **Structura lucrării**

1.1 Scurtă descriere a lucrării

Cabinetele medicale reprezintă o componentă esențială a sistemului de sănătate, jucând un rol crucial în îngrijirea și tratarea pacienților. Ele furnizează servicii medicale primare, preventive și curative, asigurând accesul pacienților la îngrijirea medicală necesară.

Ele reprezintă prima linie de contact pentru pacienți, oferindu-le accesul la profesioniști medicali precum medicii de familie sau specialiști din diverse domenii medicale. În aceste cabinete, pacienții pot primi diagnosticul și tratamentul precoce al afecțiunilor, ceea ce joacă un rol esențial în îmbunătățirea prognosticului și calității vieții.

Această aplicație web își propune să îmbunătățească eficiența și calitatea serviciilor oferite prin digitalizarea unor procese. Printre beneficii se numără:

Gestiunea eficientă a pacienților: Aplicația web permite o gestionare centralizată a datelor pacienților, inclusiv programări, istoric medical, medicamente prescrise și alte informații relevante. Astfel, personalul medical poate accesa rapid și ușor informațiile necesare, reducând erorile și creând un flux de lucru mai eficient.

Programări și rezervări online: Pacienții pot programa și gestiona programările lor prin intermediul aplicației web. Acest lucru reduce timpul petrecut la telefon sau în așteptarea fizică și permite pacienților să rezerve rapid un slot convenabil pentru ei.

1.2 Motivația alegerii temei

Ideea conceperii acestei aplicații are o semnificație specială pentru mine, și anume, această idee mi-a apărut în timpul unei vizite la locul de muncă al mamei mele, un cabinet medical de obstetrică-ginecologie. Pe durata acestei vizite, curios fiind, am început să întreb despre infrastructura tehnologică care este folosită în acel loc și să o studiez îndeaproape. După ce m-am "jucat" puțin cu aplicația de gestiune a clienților am rămas cu o stare de neplăcută surprindere văzând că această aplicație are multe lipsuri și e foarte prost optimizată.

Pe lângă partea operațională, acelei aplicații îi lipsește cu desăvârșire o interfață atractivă și intuitivă, multe funcționalități fiind dificil de găsit și utilizat.

În cele din urmă, am decis ca o ambiție personală să încerc să creez eu o aplicație asemănătoare dar cu câteva ajustări și îmbunătățiri pentru un cabinet medical și pe lângă partea de gestiune a pacienților, această aplicație să aibă și rolul de a fi o prezentare a serviciilor cât și de a facilita sistemul de programări.

1.3 Structura lucrării

Lucrarea următoare este structurată în acest mod:

- **Capitolul 1** are în vedere prezentarea temei lucrării, sublinierea importanței acesteia și motivarea alegerii acesteia cititorului.

- În **Capitolul 2** analizăm în detaliu tehnologiile și limbajele de programare utilizate pentru dezvoltarea interfeței aplicației și a funcționalității sale subiacente.

- **Capitolul 3** vine cu misiunea de a expune în detaliu funcționalitățile platformei și de a prezenta interfața grafică a acesteia.

- În **Capitolul 4** vom concluziona și vom aduce idei de posibile îmbunătățiri sau suplimentări cu scopul unei potențiale expansiuni.

- **Capitolul 5** prezintă o listă cu resursele folosite în scopul conceperii acestei lucrări.

CAPITOLUL 2

Tehnologii și limbaje

- **Prezentare introductivă a paginilor web**
- **Aplicațiile web**
- **Limbaje de programare**
- **Tehnologii web și instrumente de dezvoltare**

2.1 Prezentare introductivă a paginilor web

2.1.1 Internetul

Internetul a devenit treptat una dintre cele mai importante și răspândite invenții ale omenirii, conform datelor statistice strânse de cei de la Uniunea Internațională pentru Telecomunicații (ITU), în anul 2022 a fost estimat că 5.3 miliarde de oameni (aproximativ 66 la sută din întreaga populație a lumii) au acces la internet, înregistrând astfel o creștere de 6.1 la sută față de anul 2021. Putem spune că, în zilele noastre, internetul are o mare influență asupra felului în care comunicăm, asupra accesului la informație și asupra multor aspecte legate de interacțiunea noastră cu lumea exterioară.

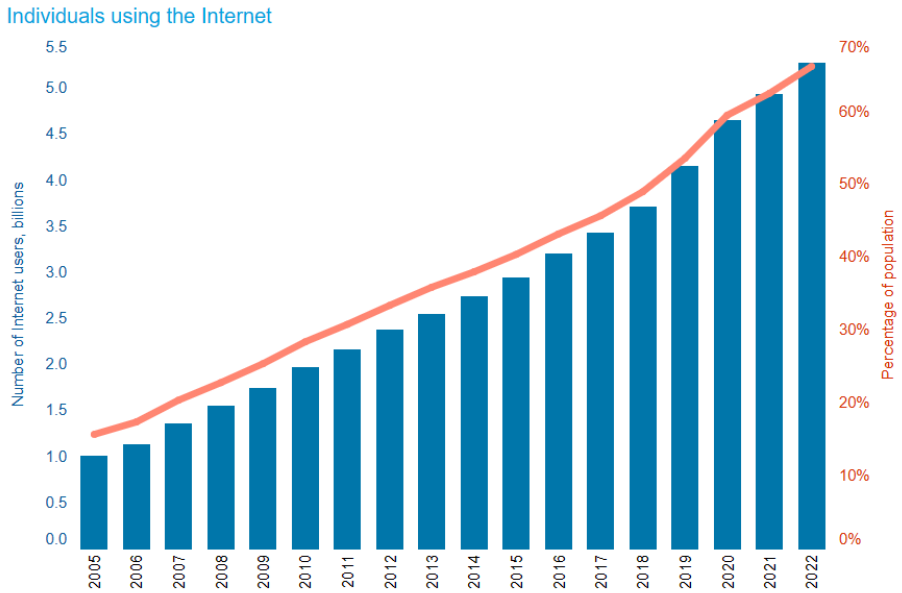

Figura 1. Numărul de utilizatori ai internetului la nivel global

Dar pentru a ajunge la această performanță de a fi atât de răspândit, trebuie să înțelegem cum a luat naștere internetul și stadiile prin care acesta a trecut pe parcursul existenței sale.

Geneza Internetului se leagă de necesitatea schimbului de informații și de evoluția rețelelor de comunicații. Astfel încât, în anii 1960, oamenii de știință alături de cercetători au început să analizeze posibilitățile de a conecta calculatoarele pentru a putea împărți diferite date și resurse. Pe baza acestor demersuri, s-a ajuns la dezvoltarea ARPANET, aceasta fiind prima rețea de calculatoare, premergătoarea a ceea ce cunoaștem noi astăzi ca internet. Această rețea a luat naștere în Statele Unite ale Americii în anul 1969 și a avut ca scop inițial facilitarea schimbului de informații și comunicării între cercetătorii ce erau implicați în proiecte de apărare a țării.

Ca urmare, în deceniile următoare (1970 și 1980), ARPANET s-a extins și s-a dezvoltat pentru a permite conectarea între universități, agenții guvernamentale și institute de cercetare. În anul 1983, ARPANET a început să folosească protocolul TCP/IP (Transmission Control Protocol/Internet Protocol), un protocol de Internet standardizat, astfel ușurând comunicarea și interoperabilitatea între rețele de calculatoare diferite.

În anul 1989, Tim Berners-Lee, om de știință englez, pe durata perioadei în care a lucrat la CERN (Consiliul European pentru Cercetare Nucleară) a dezvoltat un sistem de informații bine-definit cunoscut sub numele de World Wide Web (WWW). Acest sistem a oferit o modalitate simplistică și intuitivă de a accesa informații și de a naviga pe paginile web. Ca urmare a apariției acestui sistem, în anii 1990, Internetul a fost deschis publicului larg și a evoluat într-o platformă globală de acces la informații și comunicare.

Ca urmare a acestei deschideri către public, Internetul a cunoscut o explozie în popularitate la jumătatea deceniului, fapt ce a determinat un efect de "snowball" (bulgăre de zăpadă): conectarea mai multor oameni a determinat o diminuare a costurilor accesării rețelei, apariția mai multor furnizori, creșterea atractivității și fiabilității serviciilor, atrăgând din ce în ce mai mulți utilizatori și creând astfel un circuit ce permitea dezvoltarea continuuă a acestuia.

Internetul s-a afirmat pe parcursul existenței sale ca o invenție revoluționară ce a avut să schimbe viața cotidiană așa cum o știm, în zilele noastre, a devenit o unealtă indispensabilă în toate domeniile și a adus o creștere a calității traiului uman, dar e lesne de înțeles că această creație nu vine doar cu părți pozitive ci și cu numeroase părți neplăcute.

2.1.2 World Wide Web (WWW)

World Wide Web (WWW), foarte des întâlnit sub numele simplist "Web", este un complex și global sistem de informații interconectate. Aceste informații sunt accesibile cu ajutorul infrastructurii Internet. A luat naștere în anul 1989, la Geneva, Elveția, în clădirile CERN, fiind munca savantului de origine engleză Tim Berners-Lee. Web-ul s-a răspândit rapid și a ajuns să fie unul dintre cele mai importante instrumente de comunicare și acces la informații de pe glob.

World Wide Web are la bază utilizarea hipertextului (eng: hypertext), acesta permite legături interactive între paginile web. Practic, o astfel de pagină web poate fi legată de diverse elemente, fie ele alte pagini web, fie imagini, elemente multimedia sau chiar alte resurse. La un simplu click pe aceste legături (eng: links) se poate face trecerea la elementul respectiv, fie el pagină web sau alt tip de element multimedia, facilitând navigarea între aceste elemente.

Pe lângă folosirea hipertextului, Web-ul folosește protocolul HTTP (Hypertext Transfer Protocol), acest protocol de comunicație are rolul de a facilita transferul de date și accesarea site-urilor web prin impunerea unor reguli și standarde între cele două părți care fac schimb de date (server web – client, clientul fiind de obicei un browser web).

De asemenea, World Wide Web mai are la bază existența unor identificatori unici denumiți Uniform Resource Locators (URL), aceștia reprezentând adrese web care au scopul de a identifica și localiza resursele pe internet. Aceste adrese web au proprietatea de a fi unice și distinctive și facilitează accesul specific la anumite elemente din cadrul internetului: site-uri web, fișiere multimedia sau orice alt tip de conținut online.

O dată cu evoluția tehnologiilor și apariția mai multor limbaje de programare țintite pe partea de web, s-a făcut trecerea de la pagini web statice la pagini web interactive și dinamice, astfel crescând în popularitate și, după sine, crescând și cererea pe piață de existență a paginilor de acest tip.

Pentru accesarea și folosirea acestor pagini web este nevoie de un navigator web, aplicații ce permit afișarea informațiilor conținute de o pagină web, dar și interacționarea cu aceasta. Printre cele mai cunoscute astfel de aplicații se numără: Google Chrome, Mozilla Firefox, Opera sau Microsoft Edge.

2.1.3 HTTP

HTTP (Hypertext transfer Protocol) este denumirea protocolului de comunicare ce este folosit în cadrul World Wide Web (WWW). Acesta facilitează fluxul de date care sunt interschimbate între un client și un server. Astfel, devenind piatra de temelie a comunicării dintre cele două părți care interacționează într-un site web: navigatorul web, folosit de către utilizator și serverul web, unde se găzduiesc site-ul sau aplicația web.

Are rol de reglementare între client și server, stabilind reguli si formate prin care are loc transferul de date de la client și server și vice-versa. Concret, când un utilizator scrie o adresă web în browser, ceea ce se întâmplă este că are loc o înaintare a unei cereri HTTP către serverul care deține datele respective (pagina web). Mai departe, serverul recepționează și analizează această cerere și întoarce către utilizatorul navigatorului web datele solicitate de acesta sau în cazul unei erori îl informează cu privire la aceasta.

HTTP are definită o serie de metode pentru solicitări din partea utilizatorului, printre cele mai cunoscute fiind metodele: GET, POST, PUT și DELETE:

1 HTTP GET: este o metodă ce are ca utilizare principală solicitarea de resurse specifice de la un server web.

Exemple concrete:

Obținerea unei pagini web, atunci când un utilizator introduce în navigatorul său web o adresă web, se trimite o cerere de tip GET către serverul web. Aceasta este analizată și dacă totul este ok, se întoarce către utilizator pagina cerută.

Obținerea unor date, atunci când un utilizator are nevoie de anumite date de la server-ul web, concret, să spunem că utilizatorul are în fața sa o bară de cautare și dorește să caute ceva anume, după ce completează datele cerute, prin apasarea butonului aferent acelei bări de cautare se trimite către server o cerere de tip GET pentru a întoarce acele date. Fluxul final se aseamană cu cel de mai sus dacă după analizarea cererii sunt împlinite criteriile necesare, sunt întoarse către utilizator datele solicitate.

Accesarea API-urilor (Application programming interface), în zilele noastre s-a răspândit conceptul de API, API-ul după cum îi spune și numele este un intermediar între diferite aplicații software, ce permite partajarea de funcționalități și date între aceasta, fără necesitatea de a se cunoaște îndeaproape funcționarea internă a acestuia. Pentru a accesa un astfel de API se folosesc cereri de tip GET trimise către server-ului API-ului.

1 HTTP POST: este o metodă ce are ca utilizare principală trimiterea de date de către un utilizator către un server web cu scopul de a fi procesate. Aceste date sunt incluse în corpul (eng: body) cererii HTTP. De obicei datele sunt trimise într-un format structurat, cele mai răspândit sub formă de JSON (JavaScript Object Notation) sau XML. (Extensible Markup Language).

Exemple concrete:

Trimiterea de date de tip formular, pe un site web un utilizator are un formular cu date de înregistrare (nume utilizator, parolă, adresă e-mail etc.) după completarea acestuia, apăsarea butonului aferent acelui formular va trimite către server datele completate de acesta sub un format structurat, după analizarea datelor și verificarea îndeplinirii criteriilor, serverul poate întoarce fie un mesaj semnificativ din care să reiasă că datele au fost adăugate cu succes, fie o eroare.

1 HTTP PUT: este o metodă ce are ca utilizare principală actualizarea sau înlocuirea unor date deja existente pe serverul web. Clientul indică serverului localizarea resursei ce dorește să o modifice și îi trimite datele noi cu scopul de a fi integrate.

Exemple concrete:

Actualizarea datelor de autentificare ale unui utilizator, când un utilizator dorește de exemplu să își schimbe parola folosită pentru a fi recunoscut de către pagina web acesta va completa un formular în care își va scrie noua parolă, după apăsarea butonului aferent acelui formular, se va trimite către server datele ce se doresc a fi schimbate și locația resursei respective, după analizarea datelor și verificarea îndeplinirii criteriilor, serverul poate întoarce fie un mesaj semnificativ din care să reiasă că datele au fost actualizate cu succes, fie o eroare.

1 HTTP DELETE: este o metodă ce are ca utilizare principală ștergerea unor date deja existente pe serverul web. Clientul indică serverului localizarea resursei ce dorește să fie șterasă și inițează o astfel de cerere.

Exemple concrete:

Ștergerea unui element multimedia, utilizatorul își dorește să șteargă de pe server un element multimedia, presupunem că acesta vede toate aceste elemente multimedia și printre opțiunile care i se oferă se află și un buton roșu inscripționat “Șterge”, la apăsarea acelui buton, se trimite către serverul web o cerere de tip

HTTP DELETE alături de datele aferente localizării acelei resurse, după analizarea mesajului și verificarea îndeplinirii criteriilor, serverul poate întoarce fie un mesaj semnificativ din care să reiasă că datele au fost șterse cu succes, fie o eroare.

Acestea sunt doar câteva dintre tipurile de solicitări care sunt valabile pentru protocolul HTTP, sunt cele mai răspândite deoarece fiecare poate fi asimilat cu o operație de tip CRUD (acronim al operațiilor "Create Read Update Delete"):

- HTTP POST = Create
- HTTP GET = Read
- HTTP PUT = Update
- HTTP DELETE = Delete

Amintim aici și existența unui set de răspunsuri standardizate ale protocolului HTTP, acestea sunt împărțite în categorii:

- Răspunsuri informative (100 – 199)
- Răspunsuri care arată succesul (200 – 299)
- Mesaje de redirecționare (300 – 399)
- Răspunsuri de eroare din partea clientului (400 – 499)
- Răspunsuri de eroare din partea serverului (500 – 599)

Cele mai importante si demne de a fi reținute fiind:

- **200 – OK**

Sugerează că cererea, indiferent de tipul ei, s-a realizat cu succes.

- **201 – Created**

Sugerează că o resursă a fost creată cu succes, întâlnită ca răspuns la cereri de tip HTTP POST.

- **400 - Bad Request**

Antagonic cu codul 200, sugerează că cererea, indiferent de tipul ei, nu s-a realizat cu succes.

- **404 - Not Found**

Sugerează că resursa căutată nu a fost găsită, întâlnită ca răspuns la cereri de tip HTTP GET, când se încearcă acesarea unui URL invalid.

- **500 - Internal Server Error**

Sugerează că există o eroare care impedică conectarea la server, fie că nu este posibilă conectarea la baza de date, fie că sunt erori în cod.

Despre HTTP trebuie să știm că e un protocol fără stare (eng: state-less), însemnând că fiecare dintre cererile trimise de utilizator este tratată ca fiind independentă de restul, deci, neavând posibilitatea de a înregistra detalii despre sesiuni anterioare. Pentru a avea un mai mare control asupra interacțiunilor mai complexe dintre un client și server putem opta pentru folosirea unor tehnologii suplimentare precum cookie-urile.

Pe măsură ce s-a trecut la folosirea standardizată a protocolului HTTP, au început și o serie de probleme, probleme de securitate, HTTP nefiind un protocol care excelează pe partea de securitate în cazul datelor sensibile (informații personale, informații de natură financiară: datele cardului, date despre salarii, etc.). Aceste date puteau, astfel, să fie interceptate de diverse atacuri și interceptări cibernetice, punând în pericol integritatea paginilor web ce folosesc acest tip de protocol și a persoanelor afectate de aceste lovituri.

Ca urmare a acestei probleme, a apărut și soluția: HTTPS (Hypertext transfer Protocol Secure), folosind bazele HTTP, acest protocol a adăugat un strat de securitate suplimentar, folosind elemente de criptografie. HTTPS folosește un protocol criptografic denumit SSL/TLS (Secure Sockets Layer/Transport Layer Security), ce are rolul de a asigura confidențialitatea și integritatea datelor în timpul în care acestea se află în transfer pe internet. HTTPS criptează aceste date, având rolul de a proteja datele în cazul atacurilor și intercepțiilor de către terțe părți.

O pagină web care are integrată protocolul HTTPS este identificată printr-un certificat de tip SSL/TLS, acesta confirmându-i statutul de autenticitate și legimitate. Certificatele de tip SSL/TLS sunt emise de o autoritate de certificare de încredere și în cele din urmă este folosit pentru a asigura o conexiune securizată între client și server.

2.1.4 Elementele de bază ale unei pagini web

O dată cu dezvoltarea internetului, a crescut nevoia de a avea pagini web complexe și interconectate, ca urmare, a urmat dezvoltarea unor tehnologii care să modernizeze aspectul și operabilitatea acestor pagini web. În zilele noastre, cele mai răspândite astfel de tehnologii sunt HTML, CSS și JavaScript. Fiecare dintre acestea ocupă un rol bine definit în funcționalitatea pe care o aduce unei pagini web.

2.1.4.1 HTML

HTML (HyperText Markup Language) este cărămida de bază a unei pagini web și aduce diferite utilități, precum specificarea modului de organizare și prezentare a conținutului pe un navigator web.

Figura 2. Logo-ul HTML

HTML ia naștere în anii '90, fiind dezvoltat de același Tim Berners-Lee care a definit și World Wide Web, în cooperare cu colegii săi de la CERN. Prima versiune a "limbajului", cu numele de HTML 1.0, a fost lanasată în 1993, pe parcursul anilor și sub presiunea necesității modernizării au apărut variante noi, cu specificații și funcționalități îmbunătățite. Amintim versiunile HTML 2.0, HTML 3.2, HTML4.01, în prezent folosindu-se HTML5.

Printre elementele pe care le oferă HTML amintim:

- **DOCTYPE**: este o instrucțiune care specifică versiunea de HTML utilizată în acel document, această instrucțiune are loc de prim element într-un astfel de fișier și permite descifrarea coerentă de către navigatorul web al conținutului acelui fișier.

- **Tagul HTML**: are rol de element rădăcină într-un site web și încadrează toți ceilalți copii ai săi.

- **Tagurile head și body**: cele două părți principale ale unui site web. Tagul head deține informații despre pagina web ce nu sunt reprezentate direct în navigatorul web, printre acestea se numară: titlul paginii (care apare în tab-ul din bara browser-ului), meta-date (date despre date, cum ar fi descrierea paginii sau cuvinte cheie) sau legături către alte fișiere, de exemplu de tip CSS sau de tip JS.

- **Taguri de conținut**: elemente ce ajută la definirea diferitelor secțiuni ale unei pagini web, ele include titluri, paragrafe, imagini, legături, liste ș.a.m.d.. Ele se găsesc în interiorul tagului body și sunt structurate în concordanță cu necesitățile paginii.

2.1.4.2 CSS

CSS (Cascading Style Sheets) este denumirea dată limbajului ce are rol de a stiliza paginile web, prin stilizare înțelegem interfața vizuală și aspectul ce îl are o astfel de pagină web.

Figura 3. Logo-ul CSS

El apare ca urmare a evoluției site-urilor web, această evoluție aducând de la sine nevoia de a crea site-uri cu un design din ce în ce mai complex, pentru a atrage utilizatorii. Până la apariția acestei componente, stilizarea se făcea direct în fișierul de HTML prin atribute și stiluri inline (tagul <style>), fapt care determina un cod greu de administrat și întreținut.

Prima versiune apare în decembrie 1996, și este denumită, reprezentativ, CSS1, ea vine ca urmare a propunerii unui grup de experți din domeniul web, condus de Håkon Wium Lie și Bert Bos, de a separa stilizarea efectivă de partea de structurare a paginii.

Ca și HTML, această tehnologie s-a dezvoltat și au apărut versiuni ulterioare care au venit cu îmbunătățiri și adăugiri. Amintim de versiunea CSS2, care a apărut în 1998 și de versiunea CSS3, apărută în 1999. Cea din urmă vine cu o serie impresionantă de noutăți printre care: animații, grafică vectorială, layout-uri flexibile și grile.

CSS-ul s-a impus ca un standard în domeniul dezvoltării paginilor web și în zilele noastre este omniprezent, jucând un rol important în îmbunătățirea calității experienței utilizatorilor prin facilitarea dezvoltatorilor de a le oferi pagini web atractive.

2.1.4.3 JavaScript

JavaScript, cunoscut și de multe ori sub abrevierea JS, este numele limbajului de programare care este cel mai des întâlnit la nivelul dezvoltării de aplicații web. Creat cu scopul de a aduce o latură de interactivitate și funcționalitate dinamică cu paginile web, a devenit un puternic limbaj de programare ce iese în evidență prin versatilitate și capabilitatea de a rula pe o varietate de plaftorme.

Figura 4. Logo-ul JavaScript

El are capabilitatea de a manipula și interacționa cu elemente ale paginii web. Ca și exemplu, JavaScript-ul poate valida formulare, poate crea valoroase efecte vizuale și animații, poate cere și înainta date către server, cu ajutorul tehnologiei AJAX (Asynchronous JavaScript and XML) și nu numai.

Acesta ia naștere în anul 1995 sub denumirea inițială de "LiveScript", creat de către Brendan Eich de la Netscape Communications Corporation. Devine cunoscut sub denumirea de "JavaScript" ceva mai târziu, capitalizând popularitate limbajului Java de la vremea respectivă.

JavaScript este un limbaj interpretat, acest lucru însemnând că bucățile de cod JavaScript se execută direct de navigatorul web, prin intermediul motorului JavaScript al acestuia sau în cazul în care ne aflăm pe o aplicație non-browser, această execuție este preluată de un runtime JavaScript dedicat.

Este un limbaj cu tipare slabă sau dinamică, acest lucru însemnând că nu există necesitatea de a specifica în mod explicit tipurile de date pentru variabilele declarate, permițând astfel o mai mare flexibilitate în gestionarea tipurilor de date din cadrul codului JavaScript.

Această caracteristică a limbajului permite reatribuirea valorilor din variabile cu valori de tipuri diferite pe durata rulării programului. Concret, unei variabile care conține în inițial o valoare numerică i se poate reatribui o valoare care conține un șir de caractere sau chiar un obiect, oferind astfel o mare libertate dezvoltatorilor de a lucra cu datele.

Limbajul de programare JavaScript mai are și capabilitatea de a permite conversia implictă a tipurilor de date, având în vedere modul de folosire a acestor date, limbajul poate face o conversie automată a unei valori de la un tip la altul. De exemplu, având o operație de tip matematic, dacă un operand este un șir de caractere ce reprezintă un număr, JavaScript va facilita această operație, convertind implicit șirul într-un tip numeric pentru a efectua operația matematică respectivă

Deși pare ca fiind o proprietate impresionantă și de mare folos, nu înseamnă că nu vine cu posibile comportamente diferite față de ceea ce intenționează dezvoltatorul. Într-adevăr, ajută din punct de vedere al ușurinței și flexibilității cu care se poate scrie cod, dar în același timp, presupune necesitatea sporirii atenției de către persoana ce scrie codul și o înțegelere avansată a modului de funcționare.

Cu apariția standardului ECMAScript 6 (specificație oficială ce definește limbajul JavaScript) apar cuvinte rezervate ce au rol de a rigidiza tipurile de date: “let” și “const”, oferind o mai mare stabilitate.

2.2 Aplicațiile web

2.2.1 Introducere în aplicațiile web

Aplicațiile Web sunt un tip particular de aplicații software care permit utilizatorilor să interacționeze cu un server printr-o interfață de navigator web. În ultimii ani, acest tip de aplicație a crescut rapid în popularitate, înlocuind aplicațiile desktop și devenind un instrument vital atât pentru întreprinderile mari, cât și pentru cele mici din întreaga lume.

Portabilitatea este un mare avantaj al aplicațiilor web față de aplicațiile desktop. Prin urmare, acest tip de aplicație poate funcționa pe orice dispozitiv care funcționează într-un browser acceptat și care este conectat la internet.

O aplicație este împărțită în două componente principale: clientul și serverul, conform arhitecturii client-server.

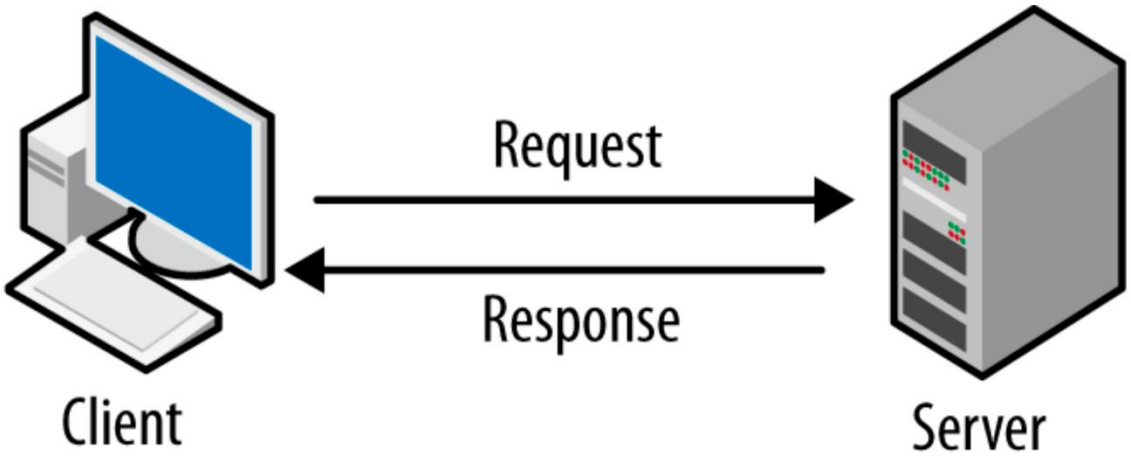

Figura 5. Arhitectura tip client-server

Acest model prezintă modul în care un server poate oferi resurse și servicii unuia sau mai multor clienți. Majoritatea serverelor au relații de unu-la-mai-mulți cu clienții, ceea ce înseamnă că un singur server poate oferi resurse pentru mai mulți clienți simultan.

Un server are două opțiuni: fie să accepte, fie să respingă cererea de conexiune a clientului. Dacă este acceptată, serverul creează și menține o conexiune cu clientul folosind un anumit protocol. Prin urmare, serverul primește cererile clienților, îndeplinește sarcinile necesare și returnează clienților rezultatele în funcție de situație.

Arhitectura clasică client-server se baza pe 2 straturi: un strat reprezentat de client și un strat reprezentat de server.

Un alt model comun al sistemelor client-server este arhitectura pe 3 niveluri:

- Nivelul de client, cel cu care interacționează utilizatorul
- Nivelul de server, unde se află logica aplicației
- Nivelul de stocare al datelor

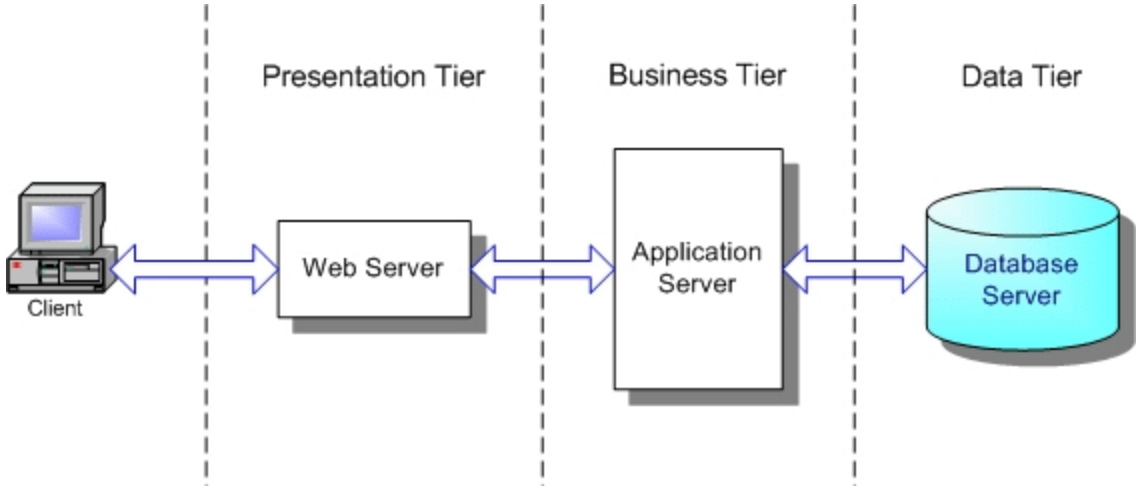

Figura 6. Arhitectură pe 3 niveluri

O aplicație web urmează o serie de pași pentru a funcționa:

- Pasul 1 – Odată ce aplicația web este accesată din navigatorul web, se inițiază o cerere către un depozit care are ca rol stocarea adreselor IP împreună cu numele domenilor, denumit Domain Name Center (DNS) și întoarce adresa IP a server-ului.
- Pasul 2 – Navigatorul web inițiază o solicitare către adresa IP găsită folosind protocolul HTTPS.
- Pasul 3 – Server-ul recepționează solicitarea HTTPS și efectuează acțiunile cerute.
- Pasul 4 – Server-ul inițiază o solicitare către baza de date pentru a accesa informațiile ce sunt stocate în stratul de date (locul unde se află datele).
- Pasul 5 – Stratul de date trimite un răspuns cu datele cerute către server, iar apoi sunt trimise către client printr-un răspuns HTTPS.

Aplicațiile web au multe beneficii, deoarece pot fi folosite în multe domenii. O serie de avantaje comune ale aplicațiilor web includ:

- Permite unui număr de utilizatori să aibă acces la aceeași versiune a aplicației.
- Nu necesită instalare.
- Este disponibil pe o varietate de platforme, inclusiv calculator, telefon mobil și laptop.
- Se pot accesa de pe o serie de navigatoare web, precum Google Chrome, Mozilla Firefox sau Microsoft Edge

2.2.2 Arhitectura aplicațiilor web

Una dintre cele mai folosite tipuri arhitecturi din domeniul web este reprezentată de arhitectura pe mai multe niveluri, aceasta este o arhitectură de tip client-server. Ca și particularități, în acest tip de arhitectură are loc despărțirea și individualizarea funcționalităților: o parte are în vedere logica funcțională, altă parte accesul la date, o alta stocare datelor și una se ocupă de interfața de utilizator. Astfel, fiecare dintre acestea se dezvoltă independent.

Vom discuta în cele ce urmează de arhitectura pe trei niveluri, un tip de arhitectură foarte bine definită, ce vizează împărțirea aplicațiilor în trei straturi sau nivele. Se remarcă această arhitectură prin independența modulelor unul față de altul, astfel fiecare dintre module putându-se dezvolta în paralel, de către dezvoltatori diferiți fără a se afecta între ele. Definim cele trei nivele ale aplicației ca fiind:

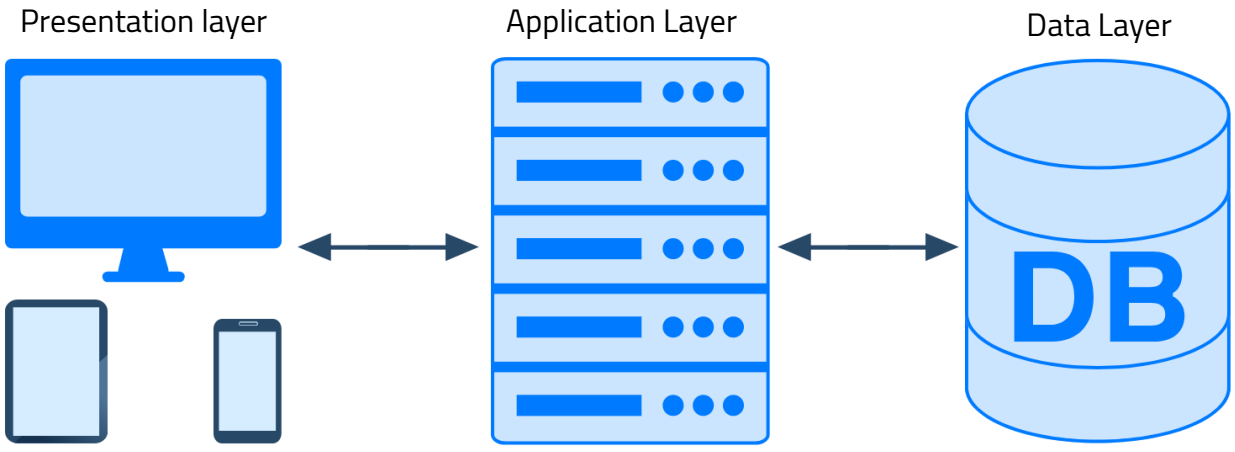

Figura 7. Cele trei nivele ale unei aplicații ce folosește arhitectura pe trei nivele

- **Nivelul de prezentare (Presentation Layer)** – partea cea mai de sus a unei aplicații. Aici, utilizatorul poate interacționa în mod direct cu aplicația, acest nivel fiind reprezentat de interfața grafică. Pe acest nivel sunt afișate către utilizator diverse informații și acesta poate oferi informații prin intermediul elementelor media, informații care ajung ulterior la stratul de aplicație.

- **Nivelul de aplicație (Application Layer)** – denumit frecvent și "Business logic layer", are rol de mediator între stratul de prezentare și stratul de date. La nivelul acesta, are loc procesarea datelor primite din direcția

stratului de prezentare, date de la utilizator, cu scopul de a fi procesate și ulterior trimise mai departe către stratul de date. După aceasta, trimite informațiile necesare înapoi spre nivelul de prezentare.

- **Nivelul de date (Data Layer)** – cunoscut și sub denumirea de "Database layer" este nivelul la care se află stocată și gestionată informația care a fost procesată de aplicație. Se află într-o strânsă legătură cu nivelul de aplicație. La acest nivel găsim metode care vizează conectarea cu o bază de date și avem prezente acțiuni din seria CRUD (Create Read Update Delete) precum adăugarea, citirea, actualizarea și ștergerea datelor.

2.3 Limbaje de programare

2.3.1 C#

C# este denumirea dată limbajului de programare modern, ce permite dezvoltarea de software securizate și robuste ce rulează în ecosistemul .NET. Este un limbaj de programare orientat pe obiecte.

Figura 8. Logo-ul C#

Limbajul C# a luat naștere în anul 2000, sub dezvoltarea inginerului software Anders Hejlsberg de la compania arhicunoscută Microsoft. Se evidențiază ca fiind o adevarată unealtă importantă de creație în lumea software astfel încât de numele ei legându-se apariția multor instrumente și limbaje de programare dintre care amintim de TypeScript și Delphi, un înlocuitor pentru Turbo Pascal.

Inițial limbajul C# a fost denumit "COOL", acest "COOL" fiind de fapt un acronim venit de la sintagma "C-Like Object Oriented Language", de aici observându-se și strânsa legătură

față de limbajele C și C++. Însă, compania Microsoft nu a izbutit să păstreze acest nume din cauza unor probleme de drepturi de autor, așa că în zilele noastre nu ne vom întâlni cu limbajul COOL ci cu limbajul C#.

Existența limbajului C# se leagă în principal de dorința de a rivaliza limbajul Java, un limbaj foarte popular la acea vreme. Are parte de o primire foarte pozitivă din partea dezvoltatorilor, fie ei experimentați sau începători, și devine popular într-un timp foarte scurt atingându-și astfel scopul. Fiind un limbaj relativ ușor de deprins și înțeles, câștigă admirația dezvoltatorilor. Într-un top al limbajelor de programare realizat în anul 2019, se situează pe poziția a patra, fiind totuși în spatele unor limbaje precum Java și JavaScript.

Limbajul C# s-a remarcat ca fiind o alegere excelentă pentru dezvoltatorii cu experiență moderată până la avansată. Fiind văzut de către experți ca un limbaj de complexitate medie și ușor inteligibil, deci, pe parcursul asimilării informațiilor de bază se poate face rapid tranziția de la nivel de începător la nivel de expert. Acest lucru se datorează și faptului că C# este un limbaj de nivel înalt, devenind o alegere solidă pentru programatorii la început de drum, iar pentru cei mai experimentați fiind o opțiune convenabilă.

Asemănător cu alte limbaje de programare cu scop general, C# joacă un rol important în creația a diferite tipuri de programe și aplicații, cum ar fi: aplicații mobile, aplicații desktop, site-uri web și chiar jocuri.

Amintim trei domenii în care limbajul C# este cel mai des utilizat:

- **Dezvoltarea aplicațiilor web** – C# este adesea folosit pentru a dezvolta site-uri web profesionale și dinamice pe platforma .NET sau folosind alte framework-uri.
- **Dezvoltarea aplicațiilor desktop** – C# a fost creat de compania Microsoft pentru Microsoft, deci este ușor de anticipat de ce este cel mai popular limbaj folosit pentru dezvoltarea aplicațiilor desktop pentru Windows.
- **Dezvoltarea de jocuri** – este printre cele mai bune limbaje de programare pentru crearea jocurilor, iar de departe cel mai popular motor de joc disponibil, Unity, integrează perfect limbajul C#.

2.3.2 TypeScript

TypeScript, cunoscut adesea și sub prescurtarea TS, este un limbaj care a fost creat de către Microsoft și care a fost lansat în anul 2012 ca un program open-source Apache 2.0. Limbajul își focalizează scopul de a crea programe de tip JavaScript ce pot conține până la

zeci de mii de linii de cod. Ca și curiozitate, cei de la Microsoft au folosit acest limbaj pentru a dezvolta portalul de management Azure (în jur de 1.2 milioane de linii de cod TypeScript) și Visual Studio Code (300 mii linii de cod TypeScript). Se evidențiază prin furnizarea de instrumente îmbunătățite pentru procesul de proiectare, verificare pe parcursul compilării și prin încărcarea dinamică a modulelor la timp de execuție, astfel abordând cele mai importante probleme ale progamării JavaScript.

Figura 9. Logo-ul TypeScript

Limbajul TypeScript este un superset cu tipuri al JavaScript, compilat în JavaScript simplu. Acest fapt duce la concluzia că programele scrise în limbajul de programare TypeScript sunt foarte portabile, putând fi rulate de pe aproape orice tip de mașină (navigatoare web, servere web, sau chiar în aplicații native pe diferite sisteme de operare care permit un API JavaScript, ca exemplu WinJS.)

Pe baza relației lor cu JavaScript, caracteristicile limbajului TypeScript pot fi grupate în trei categorii. Primele două categorii sunt legate de versiuni ale standardului ECMA-262 ECMAScript Language Specification, care este specificația oficială pentru JavaScript. ECMAScript 5 reprezintă baza TypeScript fiind responsabilă cu furnizarea celui mai mare număr de caracteristici ale limbajului. Versiunile ulterioare ale specificației ECMAScript fiind incluse în versiunile TypeScript, de multe ori încă de la începutul anului, prin previzualizări timpurii care dispun de compilare la nivel inferior la versiunile mai vechi ale specificației. A treia și ultima categorie de specificații caracteristice ale limbajului TypeScript include elemente care nu sunt planificate pentru a deveni parte a standardului ECMAScript, precum ar fi genericele sau adnotările de tip.

Angular, Ionic, React și Dojo 2 sunt doar câteva dintre framework-urile native TypeScript. Deoarece TypeScript este foarte legat de JavaScript, puteți folosi multe biblioteci și framework-uri JavaScript care sunt deja în programele TypeScript, cum ar fi Aurelia, Backbone, Bootstrap, Durandal, jQuery, Knockout, Modernizr, PhoneGap, Prototype, Raphael, React, Underscore, Vue și altele. Odată ce programul TypeScript este compilat, orice program JavaScript îl poate folosi.

Noile sintaxe, cuvintele cheie și adnotările ce specifică tipul formează limbajul. Deoarece compilatorul și serviciul de limbaj sunt în stadiul cel mai eficient atunci când înțeleg structurile complicate pe care le folosiți în cadrul programului dumneavoastră, cunoașterea modului de furnizare a informațiilor de tip este o bază vitală pentru celelalte componente.

Compilatorul are rolul de a șterge tipurile și de a transforma codul de TypeScript în cod JavaScript. De asemenea, compilatorul va trimite avertismente și erori specifice în cazul în care sunt detectate probleme. Mai are rol în combinarea rezultatului într-un singur fișier, generarea de hărți sursă și nu numai.

E demn de amintit de termenul de "transpiling", termen ce există de mult timp în domeniul dezvoltării software, dar care aduce o mare confuzie ca și semnificație. Mai exact, multă lume confundă procesele de compilare și transpilare. Compilarea definește procesul de a prelua codul sursă scris într-un limbaj și a-l transforma într-un cod dintr-un limbaj diferit, în timp ce, transpilarea este un subset al compilației și definește procesul prin care se preia un cod sursă al unui limbaj de programare și se convertește într-un alt limbaj de programare, care nu este foarte diferit, având nivel similar de abstractizare. Deci compilarea se face deseori din direcția unor limbaje mai ușor de înțeles de către dezvoltator către limbaje greu inteligibile de dezvoltator, dar foarte inteligibile de mașină (precum limbajul de asamblare) iar transpilarea se face între limbaje asemănătoare cu același nivel de abstractizare. Astfel de transpilări se întâmpla de exemplu între limbajele TypeScript și JavaScript, între C++ și C, între CoffeeScript și JavaScript sau între PHP și C++.

Printre problemele pe care JavaScript le aduce prin modul său de operare amintim:

- **Moștenirea prototipală**
- **Egalitate și jonglerie de tip (conversii)**
- **Gestionarea modulelor**
- **Lipsa tipurilor**

TypeScript deși rămâne fidel JavaScript vine cu soluții sau facilitări în cazul problemelor amintite mai sus. Specificația TypeScript adaugă multe astfel de caracteristici ce previn problemele dar nu încearcă să schimbe în mod elementar stilul și comportamentul limbajului JavaScript. Scopul limbajului TypeScript este de a facilita gestionarea și întreținerea limbajelor JavaScript de scară largă.

2.4 Tehnologii web și instrumente de dezvoltare

2.4.1 Mediul de dezvoltare Microsoft Visual Studio

Visual Studio este denumirea dată mediului de dezvoltare integrat avansat (IDE), ce a fost dezvoltat de compania Microsoft în anul 2000. Are o vastă gamă de domenii de utilizare printre care: dezvoltarea aplicațiilor web, dezvoltarea aplicațiilor desktop, dezvoltarea aplicațiilor pentru telefonul mobil și chiar a jocurilor video. Visual Studio oferă suport pentru un număr de peste 36 de limbaje de programare, permițând compilarea și rularea aproape oricărui limbaj de programare. Printre aceste limbaje de programare se numără și opțiunile populare precum Visual Basic, C, C++, C#, Java sau Python.

Figura 10. Logo-ul Visual Studio

Visual Studio furnizează cele mai avansate caracteristici pe care orice dezvoltator software le caută. Printre cele mai importante caracteristici se regăsesc:

- **Editorul de cod** - Cel mai bun editor de cod la nivel de clasă este disponibil în acest mediu de dezvoltare. Poate îndeplini o varietate de operațiuni, cum ar fi restructurarea codului, redenumirea variabilelor și metodelor, reordonarea parametrilor și extragerea interfețelor. În plus, Visual Studio are o caracteristică de completare inteligentă a codului care ajută dezvoltatorii software să reducă greșelile de scriere și alte greșeli comune. Această caracteristică de completare a codului se numește IntelliSense în Visual Studio.

- **Debugger-ul** - Debugger-ul din Visual Studio ne ajută să navigăm prin cod pentru a vedea starea unei aplicații și oferă o varietate de modalități de a vedea cum funcționează codul în timpul rulării. Este posibil să folosim comenzi rapide, puncte de întrerupere (eng: breakpoints) și comenzi de debug de la tastatură pentru a găsi codul dorit. Este un instrument vital pentru identificarea și remedierea defectelor aplicației. Procesul de debug implică executarea codului pas cu pas cu un debugger, cum ar fi cel din Visual Studio, pentru a identifica greșelile de programare. Deci, știm ce modificări trebuiesc pentru a face codul funcțional. Execuția aplicației este oprită în

modul de „pauză”, ceea ce înseamnă că variabilele, obiectele și metodele rămân în memorie. Putem rula codul odată ce debugger-ul este în modul de „pauză”.

Cel mai frecvent mod de a intra în acest fel este fie folosirea de comenzi rapide de la tastatură și anume F10 sau F11, care ne permit să găsim rapid punctul de intrare al aplicației, iar apoi putem continua să folosim aceste comenzi pentru a parcurge codul, fie putem seta unul sau mai multe puncte de întrerupere pentru a rula într-o locație specifică. Astfel, odată ce pornim aplicația și ajunge la punctul de întrerupere, aceasta va intra în modul de “pauză”.

- **Designer -** Visual Studio include un număr semnificativ de designeri vizuali care ajută la dezvoltarea aplicațiilor. Printre acestea se numără:

    - **Windows Presentation Foundation (WPF)** – permite crearea unor aplicații desktop pentru Windows care oferă o experiență vizuală impresionantă.
    - **Windows Forms Designer (WFD)** – este utilizat pentru crearea aplicațiilor de tipul interfețe grafice pentru utilizator (GUI) folosind Windows Forms care reprezintă o arhitectură de tipul interfață utilizator pentru crearea aplicațiilor desktop pentru Windows.
    - **Class Designer** – este folosit in principal pentru design-ul, vizualizarea și restructurarea claselor și a altor tipuri din cod folosind modelarea UML care utilizează diagrame de clasă pentru crearea și editarea claselor.
    - **Web Designer** – este un editor care permite crearea site-urilor web ce afișează o interfață pentru utilizator, procesează datele și oferă multe dintre comenzile și caracteristicile pe care le-ar putea oferi o aplicație standard pentru Windows.
    - **Data Designer** – permite vizualizarea și proiectarea bazei de date la care ești conectat. Acesta ne permite creerea, modificarea sau să ștergea de tabele, coloane, chei, relații, constrângeri. De altfel, putem vizualiza baza de date, creând una sau mai multe diagrame care să ilustreze tabele, coloane, chei și relații.

- **Alte instrumente -** Visual Studio oferă instrumente care ne permit să vizualizăm diferite părți dintr-un proiect precum:

Server Explorer – Permite să gestionăm conexiunile cu baza de date dintr-un computer.

Solution Explorer – Permite să gestionăm fișierele dintr-un proiect.

Team Explorer – Integreaza Azure DevOps în mediul de dezvoltare Visual Studio.

- **Extensibilitatea -** Dezvoltatorii de software au ocazia de a crea noi extensii pentru Visual Studio folosind mediul de dezvoltare al Visual Studio. Mediul de dezvoltare este o altă opțiune disponibilă dezvoltatorilor pentru a construi extensii suplimentare.

    - **Macro-uri** – oferă cea mai simplă modalitate de an extinde Visual Studio. Acest mediu de dezvoltare poate înregistra macro-uri, ceea ce permite automatizarea rapidă a sarcinilor repetitive.
    - **Add-ins** - Una dintre cele mai bune opțiuni pentru dezvoltarea extensiilor este add-ins, care vă oferă capacitatea de an accesa modelul obiect din Visual Studio și de an adăuga noi elemente pentru interfețele de utilizator la mediul de dezvoltare.
    - **Packages** - O modalitate excelentă de a adăuga noi caracteristici este fără îndoială dezvoltarea pachetelor pentru Visual Studio. Acest lucru este demonstrat de faptul că toate funcțiile mediului Visual Studio sunt construite din pachete integrate. Este un pachet cu toate limbajele de programare, debugger-ul și multe alte componente.

2.4.2 Mediul de dezvoltare Microsoft Visual Studio Code

Visual Studio Code este un editor de cod sursă care a fost dezvoltat de Microsoft și este gratuit și open-source. A fost lansat pentru prima dată în aprilie 2015 și, datorită caracteristicilor sale puternice și extensibilității, a devenit rapid popular printre dezvoltatorii de software.

Figura 11. Logo-ul Visual Studio Code

Nu este doar un alt Notepad îmbunătățit care include indentare automată și colorare a sintaxei. Pe de altă parte, este un instrument de dezvoltare extrem de puternic, axat pe cod, conceput pentru a facilita crearea de aplicații pe platforme web, mobile și cloud. Acesta funcționează cu mai multe limbaje de programare și are caracteristici integrate pentru a sprijini ciclul de viață al dezvoltării aplicațiilor, cum ar fi un depanator încorporat și un sistem integrat de suport pentru popularul motor de control Git.

Se poate lucra cu fișiere de cod separate sau cu foldere care conțin proiecte sau fișiere libere cu Visual Studio Code. A fost primul instrument de dezvoltare multiplatformă din familia Microsoft Visual Studio. Funcționează pe sistemele de operare Windows, Linux și macOS. Este un instrument centrat pe cod, este gratuit și open-source, și care facilitează editarea fișierelor de cod și a sistemelor de proiect bazate pe dosare, precum și crearea de aplicații web, mobile și cloud pe platforme populare, cum ar fi Node.js și .NET 5, precum și versiunile .NET Core anterioare, având un suport integrat pentru o gamă largă de limbaje și funcții de editare sofisticate.

Bazată pe Electron, un cadru pentru dezvoltarea de aplicații multiplatformă cu tehnologii native, Visual Studio Code combină ușurința unui editor de cod puternic cu instrumentele de care au nevoie dezvoltatorii pentru a sprijini dezvoltarea ciclului de viață al aplicației, cum ar fi depanarea și integrarea controlului versiunilor bazată pe Git. Prin urmare, Visual Studio Code nu este doar un editor de cod; este un instrument de dezvoltare complet.

În schimb, Visual Studio Code poate funcționa pe o varietate de sisteme de operare și poate gestiona o varietate de tipuri de proiecte, precum și limbaje populare, chiar dacă cu siguranță nu este un înlocuitor potrivit pentru medii mai puternice. Pentru a realiza acest lucru, Visual Studio Code oferă funcții precum:

- Suport incorporat pentru programarea în mai multe limbaje, chiar și cele ce sunt folosite de obicei pentru a dezvolta aplicații multiplatformă, precum C# sau JavaScript, cu avansate funcții de editare și suport suplimentar pentru aceste limbaje prin extensibilitate.

- Control al versiunii, ce se bazează pe Git, un puternic și popular astfel de sistem de control al versiunii, oferind astfel o experiență integrată pentru colaborare, și venind în sprijinul dezvoltatorului.

2.4.3 Sistem de control al versiunilor – Git & Github

Controlul versiunilor (Version Control) este un sistem ce vizează înregistrarea modificărilor prin care trece un fișier sau un set de fișiere pe o perioadă de timp, facilitând astfel tranziția între versiuni anterioare ale acelor fișiere.

Un mare avantaj pe care îl reprezintă folosirea unui Sistem de Control al Versiunilor (VCS) este faptul că în cazul în care ștergem din greșeală un fișier sau se întâmplă ceva cu fișierele noastre, avem șansa de a le recupera cu ușurință prin revenirea la o versiune anterioară. Avem 3 tipuri de sisteme de control al versiunilor:

- **Sisteme locale de control al versiunilor**

Este o bază de date locală pe computerul personal. Are marele dezavantaj că în cazul în care se întâmplă ceva cu baza de date locală, nu mai avem niciun mod de a recupera acestea fișiere.

- **Sisteme centralizate de control al versiunilor**

Aceste sisteme au fost create pentru a veni în ajutorul dezvoltatorii software să lucreze împreună la un proiect. Acestea au un singur server pe care sunt stocate toate fișierele versionate, precum și un număr de clienți care verifică fișierele din locația de bază.

Unul dintre numeroasele avantaje ale acestui sistem este că fiecare individ știe până la un anumit punct ce fac alții în același proiect. Administrarea unui sistem centralizat este, de asemenea, mai ușoară decât lucrul cu o bază de date locală pentru fiecare individ.

Principalul dezavantaj îl reprezintă dependența de un server central, dacă acel server central se oprește pentru oarecare motiv, nimeni nu mai are acces la salvarea schimbărilor făcute pe serverul principal.

- **Sisteme distribuite de control al versiunilor**

În acest tip de sisteme, proiectul în întregime, incluzând istoricul acestuia, poate fi copiat pe computerul fiecărui membru al echipei ce lucrează la proiect. În cazul în care apare vreo problemă la nivelul unui server, oricare dintre copiile aflate în computer-ul fiecărui dezvoltator poate fi readăugată înapoi pe server pentru a-l aduce în starea inițială.

Figura 12. Logo-ul GIT

În prezent, cel mai utilizat astfel de sistem de control al versiunilor este **GIT**, fiind un sistem de tip distribuit și are trei stări principale în care se pot afla fișierele:

- **Committed** – stare ce înseamnă că fișierul a fost stocat în siguranță în baza de date locală
- **Modified** – stare ce înseamnă că fișierul a fost modificat, dar nu a fost stocat în baza de date locală
- **Staged** – stare ce înseamnă că fișierul modificat în versiunea curentă a fost marcat și urmează să fie salvat în baza de date locală

GitHub este denumirea dată platformei de gestionare a codului sursă, este și un serviciu de găzduire. A luat naștere în luna aprilie a anului 2008 și este strâns legat de sistemul de control al versiunilor GIT. A crescut în popularitate rapid și a devenit o unealtă arhifolosită în lumea dezvoltării software. Apare ca o nevoie a Git-ului de a avea o platformă de găzduire pentru proiecte, unde dezvoltatorii își pot partaja codul sursă și pot colabora.

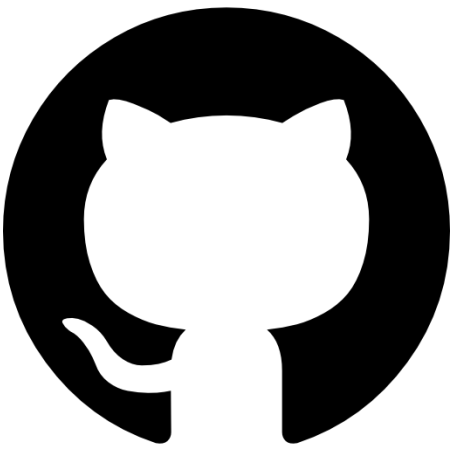

Figura 13. Logo-ul GitHub

O importantă componentă din cadrul platformei GitHub o reprezintă sistemul de gestionare a problemelor și a solicitărilor de extragere (pull requests), fapt ce permite dezvoltatorilor să aducă modificări proiectelor existente și să propună adăugarea de noi funcționalități. Ceea ce a dus la o mai mare colaborare între dezvoltare și a încurajat contribuțiile open-source.

2.4.4 Angular

Angular este o platformă și un cadru de dezvoltare (eng: framework), ce are ca întrebuințare construirea de aplicații web de tip SPA (Single Page Application), se bazează pe HTML, CSS și TypeScript. Acest gen de aplicație este o aplicație web ce oferă utilizatorilor o experiență de o rapiditate și fluiditate crescută, fiind asemănătoare în acest aspect cu aplicațiile de tip desktop.

Figura 14. Logo-ul Angular

În aplicațiile web de tip SPA, aplicația se încarcă o singură dată, și pe măsură ce se interacționează cu aceasta se actualizează dinamic, pe partea de client, folosind de obicei JavaScript. Astfel, utilizatorul are o experiență mai lină, spre deosebire de alternativa MPA (Multi-Page Application). În acest tip de aplicații se folosește o abordare tradițională de a tranziționa prin mai multe pagini, fiecare pagină fiind practic o cerere de tip HTML trimisă către server pentru a aduce documente și date noi. Deși este o practică mai simplu de întreținut, duce la o experiență mai neplăcută pe partea de utilizator din cauza așteptării datelor și a reîncărcării paginilor.

Angular a fost dezvoltat de o echipă de ingineri software din cadrul companiei Google și a fost îmbunătățit cu ajutorul comunității de utilizatori individuali.

Arhitectura unei astfel de aplicații este bazată pe anumite concepte fundamentale. Elementele de bază ale acestui framework sunt componentele, care se organizează în module, denumite NgModules.

Astfel, avem principalele componente ale aplicației Angular:

- **Module** – toate componentele sunt organizate în aceste module numite NgModules, mai pot conține servicii, directive sau alte fișiere de cod al căror scop este definit în modulul din care face parte. Această organizare a codului, pe module cu diferite funcționalități are ca scop dezvoltarea unor aplicații complexe și permite reutilizarea codului.

- **Componente –** fiecare aplicație de tip Angular are cel puțin o astfel de componentă și anume componenta de bază care conectează celelalte componente într-o ierarhie.

- **Servicii –** acestea sunt clase ce definesc funcții de tip JavaScript care sunt necesare pentru componente atunci când apar sarcini, cum ar fi preluarea datelor de pe server, validarea de credențiale sau adăugarea de date pe server.

De asemenea, în cadrul Angular mai avem și procese precum încărcarea leneșă (eng: lazy loading), ce permite o încarcare eficientă de resurse pe baza cererii, procese de rutare (eng: routing), ce permit navigarea prin aplicație fără a fi necesară reîncărcarea de pagini. Pe lângă acestea, Angular vine în ajutorul dezvoltatorilor cu sisteme de legare a datelor (eng: data-binding), injecții de dependență (eng: dependency injection) și o gamă largă de biblioteci ce vin în scopul facilitării dezvoltării aplicațiilor web.

2.4.5 Baza de date – Microsoft SQL Server

O bază de date este o colecție organizată de date, stocate în mod electronic pe un dispozitiv de stocare, aceste date fiind în cele din urmă ușor de accesat și de actualizat. În mod normal, pentru prelucrarea acestor date, se folosesc sisteme de gestionare a bazelor de date (SGBD).

În momentul de față, cel mai răspândit model de organizare a datelor dintr-o bază de date este modelul realțional. În cadrul acestui model, datele sunt distribuite, în mod obșinuit, pe linii și coloane, în diferite tabele, devenind o modalitate intuitivă, eficinetă și

flexibilă de stocare și accesare a datelor. Deci într-o bază de date vom întâlni mai multe tipuri de relații între aceste tabele:

Relația unu la unu (one-to-one) – în acest tip de relație, unei înregistrări dintr-un tabel îi este asociată o altă singură înregistrare dintr-un alt tabel și viceversa. Acest tip de relație va fi creată folosind constrângerea de tip "Cheie primară – Cheie străină" (Primary key – Foreign key). Astfel, în exemplul următor folosind proprietatea de Student_ID din tabela Student ca fiind o cheie primară și proprietatea Student_ID din tabela InfoStudent reprezentată ca o cheie străină vom forma o astfel de relație unu-la-unu.

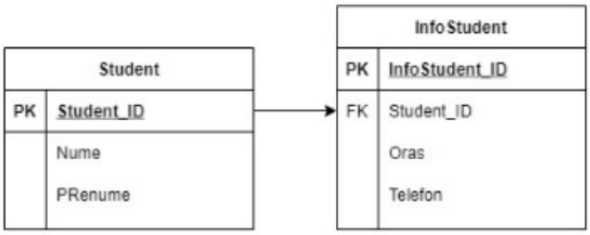

Figura 15. Relație unu la unu

Relația unu la mai mulți (one-to-many) – în acest tip de relație, unei înregistrări dintr-un tabel i se pot asocia mai multe înregistrări dintr-un alt tabel. Acest tip de relație va fi creată folosind constrângerea de tip "Cheie primară – Cheie străină" (Primary key – Foreign key). Așa încât, folosind proprietatea Masina_ID din tabela Masina ca fiind cheie primară și Masina_ID din tabela Mecanic ca fiind cheie străină, putem implementa o relație de unu-la-mai-mulți. Logica fiind că o mașină poate avea mai mulți mecanici

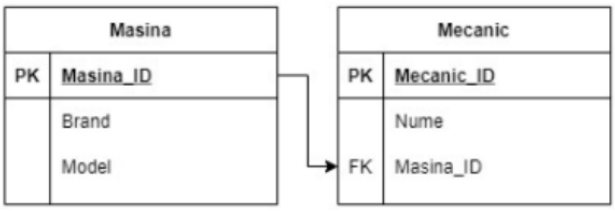

Figura 16. Relație unu la mai mulți

Relația mai mulți la mai mulți (many-to-many) – în acest tip de relație, o înregistrare din primul tabel corespunde mai multor înregistrări dintr-un alt tabel și o înregistrare din al doilea tabel poate corespunde mai multor înregistrări din primul tabel. Această relatie va fi creată

folosind un tabel de legătură între cele două. Acest nou tabel va reține cheile primare ale celor două tabele ca și chei străine. Vom folosi tabelul StudentCurs în care putem găsi două chei străine Student_ID și Curs_ID, care corespund cheilor primare Student_ID din tabela Student și respectiv Curs_ID din tabela Curs, astfel implementând o relație de mai mulți la mai mulți.

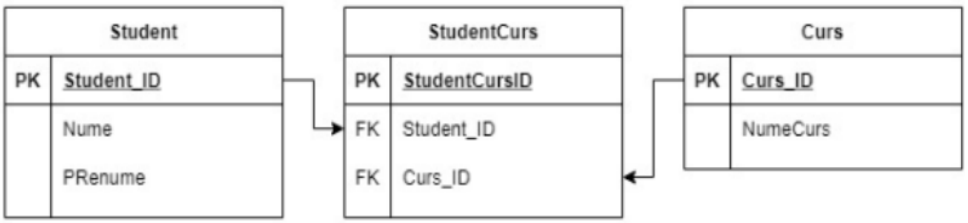

Figura 17. Relație mai mulți la mai mulți

Microsoft SQL Server este un astfel de sistem de gestionare a bazelor de date relaționale (SGBD). Principala funcție a acestui sistem de gestionare îl constituie abilitatea de a stoca și accesa date de către alte aplicații de tip software. Similar cu alte sisteme de gestionare a bazelor de date, are la bază limbajul structurat de interogare SQL (Structured Query Language) care este folosit pentru scrierea și interogarea datelor dintr-o bază de date. Microsoft SQL Server extinde limbajul SQL prin folosirea a Transact-SQL (T-SQL) care oferă ceva mai multe funcționalități decât oferă SQL.

Figura 18. Logo-ul Microsoft SQL Server

Microsoft SQL Server s-a remarcat ca o soluție facilă si ușor inteligibilă, fiind folosită în zilele noastre la o scară largă. Popularitatea pe care această unealtă a cunoscut-o se datorează și faptului că aduce multe instrumente ce facilitează dezvoltarea bazei de date, oferind un proces rapid și fluent. A devenit o opțiune serioasă când vine vorba de un sistem de gestionare a bazelor de date sigur, performant și ușor de gestionat.

2.4.6 Entity Framework

Entity Framework este un cadru de lucru puternic și versatil de cartografiere obiect-relațională (ORM) dezvoltat de Microsoft. Acesta oferă o punte de legătură perfectă între paradigma de programare orientată pe obiecte și bazele de date relaționale, simplificând accesul și gestionarea bazelor de date pentru dezvoltatori. Cu Entity Framework, dezvoltatorii pot lucra cu bazele de date utilizând concepte și tehnici familiare orientate pe obiecte, reducând necesitatea de a scrie interogări SQL de nivel scăzut și permițând un cod mai eficient și mai ușor de întreținut.

Unul dintre beneficiile cheie ale Entity Framework este capacitatea sa de a abstractiza schema de bază de date subiacentă și de a oferi un model conceptual care seamănă foarte mult cu modelul de domeniu al aplicației. Acest lucru le permite dezvoltatorilor să se concentreze pe logica de afaceri a aplicațiilor lor, mai degrabă decât să se împotmolească în complexitatea operațiilor bazei de date. Entity Framework realizează acest lucru prin maparea tabelelor și coloanelor din baza de date în clase și proprietăți, iar relațiile dintre tabele în asociații între obiecte.

Entity Framework acceptă mai multe abordări pentru dezvoltarea bazelor de date, inclusiv "code-first", "database-first" și "model-first". În cadrul abordării code-first, dezvoltatorii își definesc modelul de domeniu folosind clase și atribute în limbajul de programare preferat, cum ar fi C# sau Visual Basic. Entity Framework generează apoi schema corespunzătoare a bazei de date pe baza acestor definiții. În abordarea "database-first", schema bazei de date este deja definită, iar Entity Framework generează modelul de obiecte corespunzător pe baza schemei. Abordarea "model-first" permite dezvoltatorilor să proiecteze modelul conceptual cu ajutorul unui designer vizual, iar Entity Framework generează schema bazei de date și modelul de obiecte pe baza acestei proiectări.

Odată ce modelul este definit, Entity Framework oferă un set bogat de caracteristici pentru interacțiunea cu baza de date. Acesta include un limbaj de interogare puternic, numit Language-Integrated Query (LINQ), care permite dezvoltatorilor să scrie interogări sigure și puternic tipizate pentru modelul de obiecte. Interogările LINQ pot fi compuse și executate împotriva bazei de date, iar rezultatele sunt materializate automat în obiecte.

Entity Framework suportă, de asemenea, operațiile CRUD (Create, Read, Update, Delete), oferind un API simplu și intuitiv pentru inserarea, recuperarea, actualizarea și ștergerea datelor. Modificările aduse obiectelor sunt urmărite automat de Entity Framework, iar aceste modificări pot fi păstrate în baza de date cu un singur apel. Acest mecanism de urmărire a modificărilor simplifică foarte mult procesul de gestionare a modificărilor de date și asigură coerența datelor.

O altă caracteristică notabilă a Entity Framework este suportul său pentru relațiile complexe dintre entități. Acesta poate gestiona relații de tip unu-la-unu, unu-la-mulți și mulți-la-mulți, precum și ierarhii de moștenire. Entity Framework se ocupă de gestionarea integrității referențiale a acestor relații, asigurându-se că datele aferente sunt sincronizate și menținute corect.

Entity Framework acceptă o gamă largă de sisteme de baze de date, inclusiv Microsoft SQL Server, Oracle, MySQL, PostgreSQL și SQLite, printre altele. Acesta oferă o interfață de programare coerentă, indiferent de baza de date de bază, permițând dezvoltatorilor să treacă de la o platformă de baze de date la alta fără a fi nevoie să își rescrie codul.

Pe lângă funcțiile sale de bază, Entity Framework oferă diverse puncte de extensibilitate care permit dezvoltatorilor să îi personalizeze comportamentul pentru a se potrivi cerințelor lor specifice. Printre acestea se numără capacitatea de a defini corespondențe personalizate între tabelele bazei de date și proprietățile obiectelor, de a intercepta și modifica comenzile bazei de date și de a implementa strategii personalizate de validare a datelor și de stocare în memoria cache.

În plus, Entity Framework este bine integrat cu alte tehnologii și cadre Microsoft, cum ar fi ASP.NET, Windows Presentation Foundation (WPF) și Windows Communication Foundation (WCF). Se integrează perfect cu aceste cadre, oferind o experiență de dezvoltare coerentă și permițând dezvoltatorilor să creeze aplicații robuste și scalabile.

Entity Framework a evoluat de-a lungul anilor, fiecare versiune nouă introducând îmbunătățiri și perfecționări. Cea mai recentă versiune, Entity Framework Core, este o versiune ușoară și multi-platformă a Entity Framework care a fost reconstruită de la zero. Entity Framework Core păstrează conceptele și caracteristicile de bază ale predecesorului său, oferind în același timp performanțe îmbunătățite, o amprentă mai mică și suport pentru baze de date non-relaționale.

Rădăcinile Entity Framework

Entity Framework de la Microsoft a evoluat de la o metodologie cunoscută sub numele de Entity Relationship Modeling (ERM), care a fost prinsă pe tablă timp de peste 30 de ani. Un ERM definește o schemă de entități și relațiile dintre ele. Entități nu sunt același lucru cu obiectele. Entitățile definesc schema unui obiect, dar nu și comportamentul acestuia. Așadar, o entitate este ceva asemănător cu schema unui tabel din baza de date, cu excepția faptului că aceasta descrie schema obiectelor dvs. de afaceri. Dezvoltatorii au desenat ERM-uri de ani de zile pentru a ne ajuta să ne dăm seama cum să transpunem datele tabelare structurate ale unei baze de date în obiecte de afaceri pe care le putem programa.

Având în rândurile sale o mulțime de guru ai bazelor de date, Microsoft Research a început să conceapă o modalitate de automatizare a procesului de conectare a unui model conceptual cu schemele bazelor de date. Și trebuia să fie o stradă cu două sensuri, astfel încât dezvoltatorii să poată prelua date din baza de date,

să populeze entitățile și să persiste modificările înapoi în baza de date.

În iunie 2006, Microsoft Research a publicat prima sa lucrare despre EDM, răspunsul său la ERM. Printre autorii lucrării se numără și Jim Gray, o legendă a bazelor de date, care a dispărut în mod tragic în timp ce naviga în largul coastei Golfului San Francisco în 2007.

Modelul de relații între entități

Un beneficiu central al Entity Framework este că vă scutește de grija de a vă ocupa de de structura bazei de date. Toate accesarea și stocarea datelor se face în raport cu o structură model conceptual de date care reflectă propriile obiecte de afaceri.

Cu ADO.NET DataReaders și cu multe alte tehnologii de acces la date, cheltuiți mult timp pentru mult timp să scrieți cod pentru a obține date dintr-o bază de date, să citiți rezultatele, să extrageți fragmente de date pe care le doriți și le introduceți în clasele dvs. de afaceri. Cu Entity Framework, puteți nu mai interoghezi în funcție de schema unei baze de date, ci mai degrabă în funcție de o schemă care reflectă propriul dvs. model de afaceri. Pe măsură ce datele sunt recuperate, nu sunteți obligat să raționați coloane și rânduri și să le introduceți în obiecte, deoarece acestea sunt returnate ca obiecte. Când este momentul să salvați modificările în baza de date, trebuie să salvați doar acele obiecte. Entity Framework face munca necesară pentru a transforma obiectele dvs. înapoi în rândurile și coloanele din stocul relațional. Entity Framework face această parte a treabă pentru dumneavoastră, similar cu modul în care funcționează un instrument de Mapare relațională a obiectelor (ORM).

Entity Framework utilizează un model numit Entity Data Model (EDM), care a evoluat de la Entity Relationship Modeling (ERM), un concept care a fost utilizat în bazele de date. dezvoltare a bazelor de date de mulți ani.

Modelul Entity Data

Un model de date de entitate (EDM) este un model de date pe partea clientului și reprezintă nucleul entității. Framework. Nu este același lucru cu modelul bazei de date, care aparține bazei de date. Modelul de date din aplicație descrie structura obiectelor dvs. de afaceri. Acesta este

ca și cum vi s-ar da permisiunea de a restructura tabelele și vizualizările bazei de date în baza de date a întreprinderii dvs. astfel încât tabelele și relațiile să arate mai mult ca și cum ar fi domeniul de afaceri, mai degrabă decât schema normalizată care este proiectată de baza de date administratorii de baze de date.

Atunci când lucrați cu Entity Framework, veți implementa un EDM care este specific Entity Framework. În Entity Framework, un EDM este reprezentat prin un singur fișier XML în momentul proiectării, care este împărțit într-un set de trei fișiere XML în momentul execuției, dintre care numai unul reprezintă un model conceptual. Celelalte două furnizează metadate care permit Entity Framework să interacționeze cu o bază de date.

Bazele de date bine concepute pot reprezenta o problemă pentru dezvoltatori.

În lumea datelor, o bază de date este proiectată pentru mentenabilitate, securitate, eficiență și scalabilitate. Datele sale sunt organizate într-un mod care să satisfacă cerințele unei baze de date bune proiectare a unei baze de date, dar oferă provocări pentru dezvoltatorul care trebuie să acceseze aceste date.

EDM urmează conceptul de modelare a relațiilor între entități, dar în Entity Framework, acesta mută modelul în fișiere XML care sunt utilizate de către execuția Entity Framework. Cu un EDM în vigoare, dezvoltatorii se pot concentra asupra obiectelor de afaceri chiar și atunci când recuperează date din baza de date sau le păstrează în baza de date. Dumneavoastră, dezvoltatorul, nu va trebui să vă faceți griji cu privire la structura bazei de date, numele tabelelor sau vizualizări, sau numele procedurilor stocate sau al parametrilor necesari acestora. De asemenea, nu veți va trebui să creați obiectele necesare pentru realizarea conexiunilor la baza de date, sau să fiți preocupați de schema datelor returnate și apoi să transformați rezultatele în obiecte pe care să le folosiți în codul dumneavoastră.

Se va lucra pur și simplu pe baza modelului conceptual și a claselor care reprezintă entitățile modelului conceptual. Iar atunci când faceți acest lucru în cadrul Entity Framework, se va aplica Entity Framework se va ocupa de conexiunile la baza de date, de generarea comenzilor de bază de date, de executarea interogărilor, de materializarea obiectelor și de detaliile persistenței modificărilor înapoi în baza de date.

Baza de date backend

Modelul nu are nicio cunoștință despre stocul de date - ceea ce tipul de bază de date este, cu atât mai puțin care este schema. Și nici nu are nevoie să știe. Baza de date pe care o alegeți ca backend nu va avea niciun impact asupra modelului sau asupra codul.

Entity Framework comunică cu aceiași furnizori de date ADO.NET cu care ADO.NET folosește deja, dar cu o precizare. Furnizorul trebuie să fie actualizat pentru a susține Entity Framework. Furnizorul participă la remodelarea interogărilor și comenzilor Entity Framework în interogări și comenzi native. Tot ce trebuie să faceți este să identificați furnizorul și un șir de conexiune la baza de date, astfel încât Entity Framework să poată ajunge la baza de date. Acest lucru înseamnă că, dacă trebuie să scrieți aplicații pentru mai multe baze de date diferite, nu va trebui să învățați toate detaliile fiecărei baze de date. Puteți scrie interogări cu sintaxa Entity Framework (fie LINQ to Entities, fie Entity SQL) și nu veți mai trebuie să vă faceți griji cu privire la diferențele dintre bazele de date. Dacă aveți nevoie să profitați de funcții sau operatori care sunt specifice unei baze de date, Entity SQL vă permite să faceți acest lucru, de asemenea.

Entity Framework Caracteristici: API-uri și instrumente

În plus față de EDM, Entity Framework oferă un set de API-uri de execuție .NET care vă permit să scrieți aplicații .NET utilizând EDM. Acesta include, de asemenea, un set de instrumente de proiectare pentru proiectarea modelului. În cele ce urmează este prezentat un rezumat al principalelor caracteristici ale Entity Framework.

Deși Entity Framework este conceput pentru a vă permite să lucrați direct cu clasele din EDM, acesta trebuie totuși să interacționeze cu baza de date. Modelul conceptual de date pe care îl descrie EDM este stocat într-un fișier XML a cărui schemă identifică entitățile și proprietățile acestora. În spatele schemei conceptuale descrise în EDM se află un alt fișier pereche de fișiere XML care mapează modelul de date înapoi în baza de date. Unul dintre acestea este un fișier XML care descrie baza de date, iar celălalt este un fișier care asigură corespondența între modelul dvs. conceptual și baza de date.

În timpul executării interogărilor și al executării comenzilor (pentru actualizări), Entity Framework își dă seama cum să transforme o interogare sau o comandă care este exprimată în termeni de model de date în una care este exprimată în termeni de bază de date. Face acest lucru prin citirea metadatelor.

Atunci când datele sunt returnate din baza de date, se utilizează metadatele pentru a modela baza de date. rezultatele bazei de date în entități și materializează în continuare obiecte din aceste rezultate.

Entity Framework dobândește capacitatea de a utiliza un model în memorie cu o caracteristică numită code first, care face parte din Entity Framework Community Technical Preview (CTP). Aceasta nu face încă parte din Entity Framework și trebuie descărcată separat. Code first vă permite să lucrați numai cu clase, iar metadatele XML necesare sunt generate în memorie din mers, în timpul execuției.

Object Services

Cel mai important set de caracteristici al Entity Framework și cel cu care este probabil să lucrați cel mai des este denumit Object Services. Object Services se află deasupra stivei Entity Framework, după cum se arată în figura de mai jos, și oferă funcționalitatea necesară pentru a lucra cu obiecte care se bazează pe entitățile dumneavoastră. Object Services oferă o clasă numită EntityObject și poate gestiona cu ușurință orice clasă care moștenește din EntityObject.

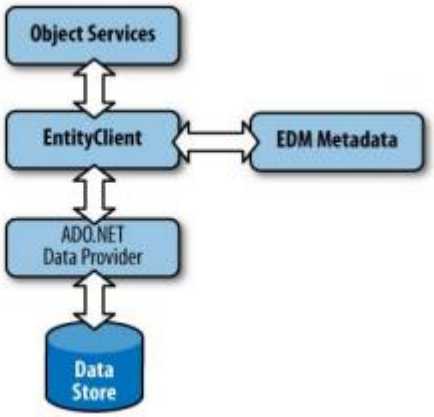

Figura 19. Relații dintre EntityClient și alte componente

Aceasta include materializarea obiectelor din rezultatele interogărilor în cadrul EDM, urmărirea modificărilor aduse acestor obiecte, gestionarea relațiilor dintre obiecte și salvarea modificărilor în baza de date.

Între interogare și actualizare, Object Services oferă o serie de capabilități pentru a interacțiune cu obiectele entității, cum ar fi lucrul automat cu un nivel inferior al Entity

Framework pentru a face toată munca necesară pentru a efectua apeluri către baza de date și a să se ocupe de rezultate. Object Services oferă, de asemenea, serializare (atât XML, cât și binară).

EntityClient

EntityClient este cealaltă API majoră din Entity Framework, deși este una pe care o puteți utiliza în mod obișnuit. mai puțin probabil să lucrați direct cu ea. Aceasta oferă funcționalitatea necesară pentru a lucra cu interogările și comenzile din magazin (împreună cu furnizorul de baze de date), conectarea la baza de date, executarea comenzilor, recuperarea rezultatelor din magazin și remodelarea rezultatelor pentru a se potrivi cu EDM.

Puteți lucra direct cu EntityClient sau puteți lucra cu Object Services, care se află pe deasupra EntityClient. EntityClient nu numai că este capabil să efectueze interogări, dar face și acest lucru în numele Object Services. Diferența constă în faptul că atunci când lucrați direct cu EntityClient, veți obține rezultate tabelare (deși rezultatele pot fi modelate). Dacă lucrați cu Object Services, acesta va transforma datele tabulare create de către EntityClient în obiecte.

Datele tabelare returnate de EntityClient sunt numai pentru citire. EntityClient este foarte potrivit pentru raportarea și mutarea datelor de la un mecanism de persistență la altul. Numai obiect Services oferă urmărirea modificărilor și posibilitatea de a salva modificările în memoria de date.

Operatii CRUD cu Entity Framework

Operațiunile CRUD (Create, Read, Update, Delete) sunt funcționalități de bază în Entity Framework care permit dezvoltatorilor să interacționeze cu baza de date. Iată un rezumat al modului în care se efectuează operațiile CRUD, fără exemple specifice:

Operațiuni de creare (inserare):

Pentru a crea noi înregistrări în baza de date, dezvoltatorii creează o instanță a clasei de entități corespunzătoare și o adaugă la proprietatea DbSet corespunzătoare a DbContext. DbContext urmărește modificările, iar apelarea metodei SaveChanges persistă noile entități în baza de date.

Operațiuni de citire (recuperare):

Entity Framework oferă diverse metode de recuperare a datelor din baza de date. Dezvoltatorii pot utiliza interogări LINQ pentru a defini criterii de filtrare, sortare și

proiecție pentru a prelua înregistrări specifice sau colecții de înregistrări. Interogările LINQ sunt executate în raport cu DbContext, iar rezultatele sunt returnate sub formă de obiecte.

Operațiuni de actualizare:

Pentru a actualiza înregistrările existente, dezvoltatorii recuperează entitatea din baza de date utilizând metode precum Find sau interogări LINQ. După ce se efectuează modificările necesare la entitate, modificările sunt urmărite automat de DbContext. Apelarea metodei SaveChanges aplică actualizările la înregistrările corespunzătoare din baza de date.

Operațiuni de ștergere:

Pentru a șterge înregistrări din baza de date, dezvoltatorii recuperează entitatea care urmează să fie ștearsă utilizând metode precum Find sau interogări LINQ. Entitatea este apoi eliminată din proprietatea DbSet corespunzătoare a DbContext, iar apelarea metodei SaveChanges aplică ștergerea în baza de date.

Merită menționat faptul că Entity Framework oferă caracteristici și capacități suplimentare pentru a gestiona scenarii complexe. De exemplu, suportă ștergerile în cascadă, în care entitățile conexe sunt șterse automat atunci când entitatea mamă este ștearsă. De asemenea, suportă încărcarea rapidă și încărcarea leneșă pentru a prelua în mod eficient entitățile conexe.

În plus, Entity Framework oferă mecanisme de lucru cu tranzacții, permițând dezvoltatorilor să grupeze mai multe operații ale bazei de date într-o singură unitate atomică. Acest lucru asigură fie că toate operațiunile sunt angajate, fie că niciuna dintre ele nu este angajată, menținând integritatea datelor.

În general, Entity Framework simplifică procesul de efectuare a operațiunilor CRUD prin abstractizarea bazei de date subiacente și prin furnizarea unei API intuitive pentru lucrul cu datele sub formă de obiecte. Acest lucru permite dezvoltatorilor să se concentreze asupra logicii de afaceri a aplicației, în timp ce Entity Framework se ocupă în mod eficient de operațiunile bazei de date.

Interogare și filtrare avansată

Entity Framework oferă capabilități avansate de interogare și filtrare care permit dezvoltatorilor să recupereze date din baza de date în mod eficient și să efectueze operațiuni complexe. Iată câteva dintre caracteristicile avansate de interogare și filtrare cu Entity Framework:

Interogări LINQ:

Entity Framework acceptă interogări integrate în limbaj (LINQ), care permit dezvoltatorilor să scrie interogări sigure și puternic tipizate în raport cu modelul de obiecte. LINQ oferă o sintaxă fluentă și expresivă pentru interogarea datelor. Dezvoltatorii pot utiliza operatorii LINQ, cum ar fi Where, OrderBy, GroupBy și Select, pentru a filtra, sorta, grupa și proiecta datele.

Proiecție:

Entity Framework permite dezvoltatorilor să proiecteze rezultatele interogărilor în structuri de date personalizate sau în tipuri anonime. Această caracteristică permite selectarea doar a câmpurilor necesare din baza de date, îmbunătățind performanța și reducând cantitatea de date transferate în rețea. Dezvoltatorii pot utiliza operatorul Select în interogările LINQ pentru a specifica proiecția dorită.

Eager Loading (încărcare rapidă):

Entity Framework acceptă încărcarea rapidă, ceea ce permite dezvoltatorilor să specifice entitățile conexe care urmează să fie încărcate împreună cu entitatea principală într-o singură interogare a bazei de date. Acest lucru ajută la reducerea numărului de drumuri dus-întors către baza de date și îmbunătățește performanța. Dezvoltatorii pot utiliza metoda Include sau metoda de extensie Include în interogările LINQ pentru a specifica entitățile conexe care urmează să fie încărcate cu promptitudine.

Încărcare explicită:

Pe lângă încărcarea rapidă, Entity Framework oferă încărcare explicită, prin care dezvoltatorii pot încărca în mod explicit entitățile conexe la cerere. Acest lucru poate fi util atunci când dezvoltatorii doresc să încarce entități conexe în mod selectiv, pe baza unor scenarii sau condiții specifice. Metoda Load poate fi utilizată pentru a încărca în mod explicit entități conexe.

Filtrarea cu predicate:

Entity Framework permite dezvoltatorilor să aplice filtre dinamice și complexe la interogări folosind predicate. Predicatele sunt expresii care înglobează logica de filtrare și pot fi construite dinamic în timpul execuției. Dezvoltatorii pot utiliza predicte împreună cu interogările LINQ pentru a aplica filtre pe baza datelor introduse de utilizator sau a altor condiții de execuție.

Interogări compilate:

Entity Framework suportă interogări compilate, care permit dezvoltatorilor să precompileze interogările LINQ pentru o performanță îmbunătățită. Prin compilarea

interogărilor, se reduce sarcina de analiză și traducere a interogărilor, ceea ce duce la o execuție mai rapidă a interogărilor. Interogările compilate pot fi stocate și reutilizate în întreaga aplicație.

Interogări SQL brute:

Entity Framework oferă posibilitatea de a executa interogări SQL brute în baza de date. Această caracteristică este utilă atunci când dezvoltatorii trebuie să efectueze operații complexe sau să utilizeze funcționalități specifice bazei de date care nu sunt ușor de realizat cu ajutorul interogărilor LINQ. Interogările SQL brute pot fi executate utilizând metoda SqlQuery sau metoda de extensie FromSql.

Căutare în text complet:

Entity Framework suportă capabilități de căutare full-text pentru bazele de date care oferă această funcție (cum ar fi SQL Server). Căutarea full-text permite dezvoltatorilor să efectueze căutări eficiente și puternice bazate pe text pe coloane textuale din baza de date. Entity Framework oferă metode pentru a efectua căutări full-text utilizând cuvinte cheie, fraze și alte opțiuni de căutare avansate.

Aceste caracteristici avansate de interogare și filtrare permit dezvoltatorilor să efectueze operațiuni sofisticate de recuperare și manipulare a datelor utilizând Entity Framework. Utilizând aceste capacități, dezvoltatorii pot scrie interogări eficiente și optimizate ale bazei de date care să îndeplinească cerințele specifice ale aplicațiilor lor.

2.4.7 Node.js

Node.js este un mediu de execuție care permite dezvoltatorilor să ruleze codul JavaScript în afara unui browser web. Acesta este construit pe motorul JavaScript V8, același motor care alimentează Google Chrome. Node.js oferă un model de I/O ușor, bazat pe evenimente și care nu blochează, ceea ce îl face eficient pentru crearea de aplicații scalabile și de înaltă performanță.

Figura 20. Logo-ul Node.js

Una dintre caracteristicile cheie ale Node.js este capacitatea sa de a gestiona un număr mare de conexiuni concurente cu o eficiență ridicată. Realizează acest lucru prin utilizarea unei arhitecturi bazate pe evenimente, în care dezvoltatorii scriu cod care răspunde la evenimente declanșate de diverse surse, cum ar fi acțiunile utilizatorului sau sursele de date externe. Acest model permite Node.js să gestioneze mai multe cereri simultan fără a bloca execuția codului.

Node.js este utilizat pe scară largă pentru dezvoltarea de aplicații de server și de rețea. Dispune de un ecosistem vast de module și biblioteci, cunoscut sub numele de Node Package Manager (NPM), care permite dezvoltatorilor să integreze cu ușurință pachete de la terți în aplicațiile lor și să valorifice soluțiile existente. Acest ecosistem oferă o gamă largă de instrumente, cadre și utilități care îmbunătățesc experiența de dezvoltare și ajută dezvoltatorii să creeze aplicații mai eficient.

Cu Node.js, dezvoltatorii pot construi diverse tipuri de aplicații, inclusiv servere web, API-uri RESTful, aplicații în timp real, aplicații de streaming, instrumente de linie de comandă și multe altele. Este deosebit de potrivit pentru scenarii care necesită comunicare în timp real, gestionarea mai multor conexiuni simultane și efectuarea de sarcini intensive de I/O.

Node.js a câștigat o popularitate semnificativă datorită versatilității, scalabilității și capacității de a utiliza JavaScript atât pentru dezvoltarea front-end, cât și pentru cea back-end. Acesta permite dezvoltatorilor să utilizeze un limbaj și o paradigmă de programare consecventă pe întreaga stivă de aplicații, ceea ce poate simplifica dezvoltarea și îmbunătăți productivitatea.

Pe scurt, Node.js este un mediu de execuție puternic care le permite dezvoltatorilor să ruleze codul JavaScript în afara browserului, oferind un model de I/O bazat pe evenimente, fără blocaj, pentru crearea de aplicații scalabile și de înaltă performanță. Ecosistemul său extins și flexibilitatea îl fac o alegere populară pentru dezvoltarea pe partea de server, unde comunicarea în timp real și gestionarea eficientă a conexiunilor concurente sunt esențiale.

Construirea de servere web cu Node.js

Construirea de servere web cu Node.js implică utilizarea mediului de execuție pentru a crea aplicații de partea serverului care gestionează cereri HTTP și furnizează răspunsuri.

Node.js oferă o arhitectură ușoară, eficientă și bazată pe evenimente, ceea ce îl face foarte potrivit pentru crearea de servere web.

Pentru a construi servere web cu Node.js, dezvoltatorii utilizează adesea cadre precum Express.js, Nest.js sau Koa.js. Aceste cadre oferă un set de instrumente, utilități și convenții pentru a simplifica procesul de dezvoltare și pentru a gestiona funcționalitățile comune ale serverelor web.

În procesul de construire a unui server web, dezvoltatorii definesc rute pentru a gestiona diferite metode HTTP (GET, POST, PUT, DELETE etc.) și specifică logica corespunzătoare care trebuie executată atunci când o cerere corespunde unei anumite rute. Aceștia pot defini funcții middleware pentru a intercepta și modifica cererile primite sau răspunsurile transmise. Funcțiile middleware pot îndeplini sarcini precum analiza corpurilor cererilor, gestionarea autentificării, înregistrarea, gestionarea erorilor și altele.

Dezvoltatorii pot utiliza obiectele de cerere și de răspuns furnizate de cadrul de lucru pentru a accesa parametrii de cerere, antetele, cookie-urile și pentru a trimite înapoi răspunsul corespunzător către client. Aceștia pot defini parametrii de rută și parametrii de interogare pentru a extrage date dinamice din cerere.

Serverele web Node.js pot servi fișiere statice, cum ar fi HTML, CSS, JavaScript, imagini și alte active. Dezvoltatorii pot specifica directorul în care se află fișierele statice, iar serverul va servi automat aceste fișiere atunci când sunt solicitate.

Serverele web construite cu Node.js pot, de asemenea, să gestioneze sesiuni și să mențină conexiuni cu stare cu clienții. Acestea pot utiliza tehnici precum cookie-urile sau autentificarea bazată pe token-uri pentru a gestiona sesiunile utilizatorilor și pentru a implementa funcții precum autentificarea și autorizarea utilizatorilor.

În plus, serverele web Node.js pot interacționa cu bazele de date, făcând posibilă persistența datelor și efectuarea de operații CRUD. Dezvoltatorii pot utiliza biblioteci precum Sequelize sau Mongoose pentru a stabili conexiuni cu bazele de date, a defini modele și a executa interogări ale bazelor de date.

Odată ce serverul web este construit, acesta trebuie să fie implementat și să fie accesibil pe internet. Serverele web Node.js pot fi găzduite pe diverse platforme, inclusiv pe furnizori de cloud precum Amazon Web Services (AWS), Microsoft Azure sau Heroku. De asemenea, acestea pot fi implementate în medii containerizate cu ajutorul unor instrumente precum Docker și pot fi gestionate cu ajutorul unor platforme de orchestrare a containerelor precum Kubernetes.

În general, construirea de servere web cu Node.js implică utilizarea de cadre, definirea de rute și middleware, gestionarea cererilor și răspunsurilor HTTP, servirea de fișiere statice, gestionarea sesiunilor, interacțiunea cu bazele de date și implementarea serverului pentru a-l face accesibil pe internet. Node.js oferă un mediu flexibil și eficient pentru dezvoltarea de servere web, permițând dezvoltatorilor să creeze aplicații scalabile și de înaltă performanță.

Extinderea și implementarea în Node.js

Extinderea și implementarea sunt aspecte critice în construirea și întreținerea aplicațiilor Node.js. Acest proces implică strategii și tehnici pentru a face față unui trafic crescut, pentru a îmbunătăți performanța, pentru a asigura o disponibilitate ridicată și pentru a simplifica procesul de implementare.

Scalarea în Node.js se referă la capacitatea de a gestiona un număr tot mai mare de cereri și utilizatori. Există două abordări principale ale scalării: scalarea verticală și scalarea orizontală. Scalarea verticală implică actualizarea hardware-ului serverului pentru a face față unei sarcini crescute prin adăugarea de mai multe resurse, cum ar fi CPU, memorie sau stocare. Pe de altă parte, scalarea orizontală implică distribuirea sarcinii pe mai multe servere, adesea realizată prin tehnici de echilibrare a sarcinii.

În contextul Node.js, scalarea implică adesea utilizarea mecanismelor de clusterizare și de echilibrare a sarcinii. Clusterizarea permite aplicației să utilizeze mai multe procese Node.js sau workers pentru a gestiona cereri simultane, îmbunătățind astfel performanța și utilizând eficient resursele de sistem disponibile. Echilibrarea încărcării distribuie cererile primite între aceste procese worker, asigurându-se că volumul de lucru este distribuit în mod egal și maximizând scalabilitatea aplicației.

Implementarea în Node.js presupune ca aplicația să fie disponibilă și accesibilă pentru utilizatori. Aplicațiile Node.js pot fi implementate pe diverse platforme, inclusiv pe furnizori de cloud precum Amazon Web Services (AWS), Microsoft Azure sau Google Cloud Platform (GCP). Aceste platforme oferă servicii și infrastructuri care facilitează procesul de implementare, cum ar fi mașinile virtuale, calculul fără server și containerizarea.

Containerizarea este o abordare de implementare populară în aplicațiile Node.js. Aceasta implică împachetarea aplicației și a dependențelor sale în containere cu ajutorul unor instrumente precum Docker. Containerele oferă un mediu de execuție ușor și coerent, facilitând implementarea aplicațiilor pe diferite platforme și asigurând un comportament coerent.

Implementarea cuprinde, de asemenea, configurarea componentelor de infrastructură adecvate, cum ar fi bazele de date, cozile de mesaje și sistemele de cache, în funcție de cerințele aplicației. Instrumentele de gestionare a configurației ajută la simplificarea procesului de configurare și gestionare a acestor componente în diferite medii, cum ar fi dezvoltarea, pregătirea și producția.

Monitorizarea și logarea joacă un rol vital în procesul de scalare și implementare. Instrumentele de monitorizare permit dezvoltatorilor să urmărească performanța și sănătatea aplicației, să colecteze măsurători și să identifice eventualele blocaje sau probleme. Instrumentele de logare ajută la capturarea și stocarea jurnalelor aplicației, permițând o depanare, depanare și analiză eficientă a performanței.

Conductele de integrare și implementare continuă (CI/CD) sunt esențiale pentru automatizarea proceselor de construire, testare și implementare. Instrumentele CI/CD, cum ar fi Jenkins, Travis CI sau GitLab CI/CD, permit dezvoltatorilor să automatizeze aceste procese, asigurându-se că modificările aduse aplicației sunt testate temeinic și implementate eficient.

În general, scalarea și implementarea în Node.js implică strategii pentru gestionarea traficului crescut, distribuirea volumului de lucru, optimizarea performanței și simplificarea procesului de implementare. Prin implementarea unor tehnici eficiente de scalare și desfășurare, dezvoltatorii se pot asigura că aplicațiile Node.js sunt capabile să facă față cererilor tot mai mari ale utilizatorilor, să ofere performanțe ridicate și să ofere o experiență de utilizare fiabilă.

2.4.8 Node Package Module (npm)

Figura 21. Logo-ul Node Package Module (npm)

Deși este dezvoltat independent și poate fi actualizat după un alt program decât Node, NPM (Node Package Manager) este un program însoțitor care se instalează alături de

Node. Acesta permite descărcarea pachetelor, care sunt module JavaScript reutilizabile (împreună cu toate materialele de suport necesare), dintr-un registru public de pachete (sau dintr-un depozit privat, dacă aveți unul). Puteți accesa depozitul principal la www.npmjs.com. Acesta poate fi accesat cu ajutorul unui browser de calculator și puteți vizualiza toate pachetele oferite, ceea ce face ca găsirea exact a ceea ce aveți nevoie să fie simplă.

Este ușor de utilizat NPM, deoarece este doar o comandă în plus care se execută din linia de comandă, la fel ca Node. Să ne imaginăm, pentru exemplificare, că ați creat un director numit MyFirstProject. Executați următoarele în el:

```
npm install express
```

Install este o comandă pe care o puteți lansa. În acest caz, npm este interfața de linie de comandă utilizată de NPM. Apoi, express este un argument al comenzii, iar acesta este un parametru general pe care majoritatea interacțiunilor dumneavoastră cu NPM îl vor necesita adesea.

Dacă executați această comandă, veți vedea un director numit node-modules care este umplut cu diverse elemente. În esență, modulul Express - un modul JavaScript - este alcătuit în întregime din cod. Toate aceste dependențe vor fi gestionate de NPM. În plus, puteți vedea că a fost creat un fișier cu numele package-lock.json; acesta nu trebuie salvat, deoarece NPM are nevoie de el pentru a funcționa corect.

Atunci când comanda install este utilizată în acest mod, modulele pe care le-ați numit sunt instalate în directorul curent; acest lucru este cunoscut sub numele de memoria cache a proiectului sau memoria cache locală. Opțional, puteți instala modulul în memoria cache globală adăugând un argument la comandă:

```
npm install -g express
```

Acum că Express a fost descarcat în afara directorului curent, acesta este disponibil pentru toate proiectele Node și este partajat de toate proiectele Node.

Node, ca proiect, nu va utiliza un modul instalat la nivel global decât dacă i se cere acest lucru). Veți dori să instalați dependențele în memoria cache a proiectului în cea mai mare parte a timpului, astfel încât diferite dependențe să fie instalate în memoria cache a proiectului.

CAPITOLUL 3

PREZENTAREA APLICAȚIEI

- **Aplicație web Cabinet Medical – "Bentea Medical"**
- **Structura bazei de date**
- **Pagina de întâmpinare**
- **Înregistrarea**
- **Autentificarea**
- **Interfața pacientului**
- **Interfața doctorului**
- **Interfața administratorului**

3.1 Aplicație web Cabinet Medical – "Bentea Medical"

Aplicația web de față își propune să îmbunătățească eficiența și calitatea serviciilor medicale oferite în cabinetele medicale. Aceasta aduce o serie de beneficii importante, printre care se numără:

Gestiunea centralizată a pacienților: Aplicația web permite o administrare eficientă a datelor pacienților, inclusiv programări, istoric medical, prescripții medicale și alte informații relevante. Prin accesul rapid și ușor la aceste informații, personalul medical poate reduce erorile și îmbunătăți fluxul de lucru. Doctorii își pot vedea calendarul cu programări pentru o mai ușoară gestiune a timpului.

Programări online și rezervări: Pacienții pot programa și gestiona propriile programări prin intermediul aplicației web. Aceasta elimină necesitatea de a petrece timp în așteptarea telefonică sau fizică și le permite pacienților să rezerve rapid un slot convenabil pentru ei.

Aceste funcționalități ale aplicației web au scopul de a spori eficiența și calitatea serviciilor medicale oferite în cabinetele medicale, facilitând atât munca personalului medical, cât și accesul și implicarea pacienților în procesul de îngrijire medicală.

3.2 Structura bazei de date

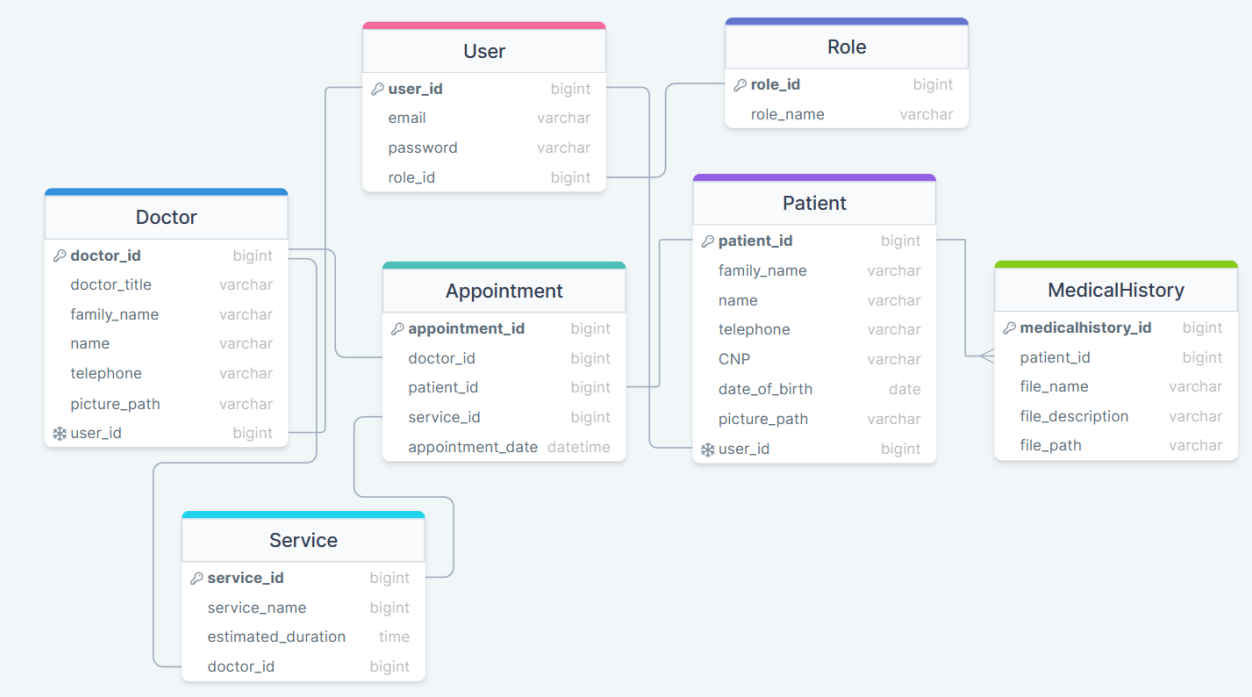

Figura 22. Structura bazei de date a aplicației web

Pentru baza de date am folosit Microsoft SQL Server, structura bazei de date arată în felul următor:

User este tabela de bază, în aceasta reținem email-ul și parola utilizatorului, ne deservește la funcționalitatea de autentificare, de asemenea înregistrează și rolul pe care îl deservește acel utilizator față de aplicația noastră. Avem 3 roluri: pacient, doctor și administrator. De acest User se leagă tabelele Doctor și Patient (pacient) printr-o relație de unu-la-unu (unui User îi corespunde un Doctor, unui User îi corespunde un Patient).

În tabela Patient găsim informații despre pacient, nume de familie, prenume, număr de telefon, CNP, data nașterii, și o cale în care salvăm poza de profil a acestui utilizator. De tabela Patient se leagă tabela MedicalHistory, în tabela MedicalHistory (istoric medical) vom reține fișiere ce reprezintă istoricul medical al pacientului in cauză (controale, medicamentație, ș.a.m.d). Relația dintre aceste două tabele este de unu-la-mai-mulți, un pacient va avea mai multe astfel de fișiere.

Tabela Doctor înglobulează detalii despre doctor, asemănător cu ce avem la pacient.

Aceasta alături de tabela Patient și tabela Service formează tabela Appointment (programare), această e o tabelă de legătură și reprezintă o relație de tip mai-mulți-la-mai-mulți, pot exista mai multe programări între diferiți doctori și pacienți. De asemenea în această tabelă avem stocată și data programării pentru utilizarea în vizualizarea calendarului.

Tabela Service (serviciu) deservește rolul de a stoca seviciile pe care le oferă un doctor (fiecare doctor are anumite servicii pe care le oferă, în funcție de specializare), este inclusă și ea în tabela de programare cu rol de a verifica că doctorul oferă serviciul dorit de pacient.

3.3 Pagina de întâmpinare

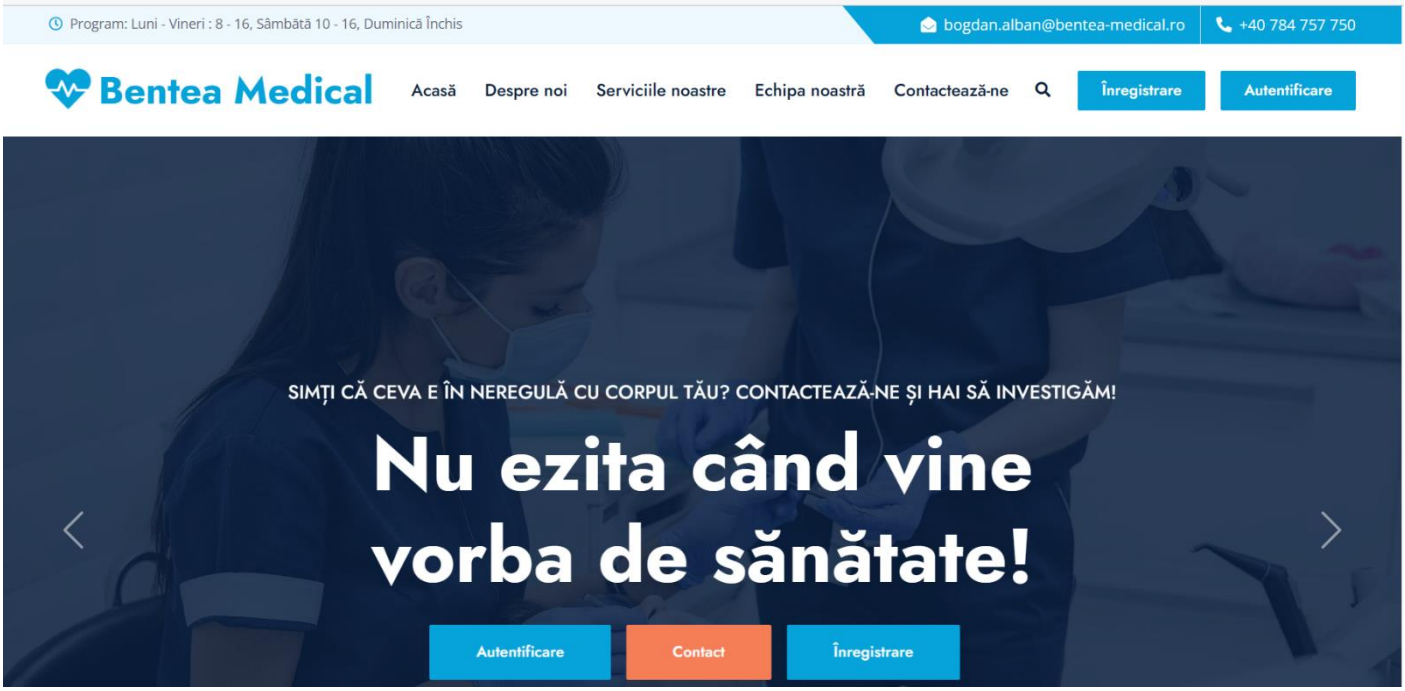

Figura 23. Pagina de întâmpinare din aplicația web

Pe lângă funcționalitățile ce vor fi prezentate ulterior, aplicația web are și rol de prezentare: putem vedea diferite detalii despre cabinet fără a ne autentifica. Cum ar fi: echipa de doctori și serviciile oferite.

La click pe oricare dintre etichetele din bara de sus vom fi redirecționați către elementele specifice, acestea se află deja pe pagina principală, dar prin această metodă facilităm tranziția între elementele paginii.

Figura 24. Detalii despre serviciile oferite în aplicația web

Pe click pe butonul cu iconița de săgeată suntem redirecționați la începutul paginii, fiind o pagină destul de lungă ca și întindere acest mic element ne ajută să ne mișcăm mai ușor prin elementele paginii.

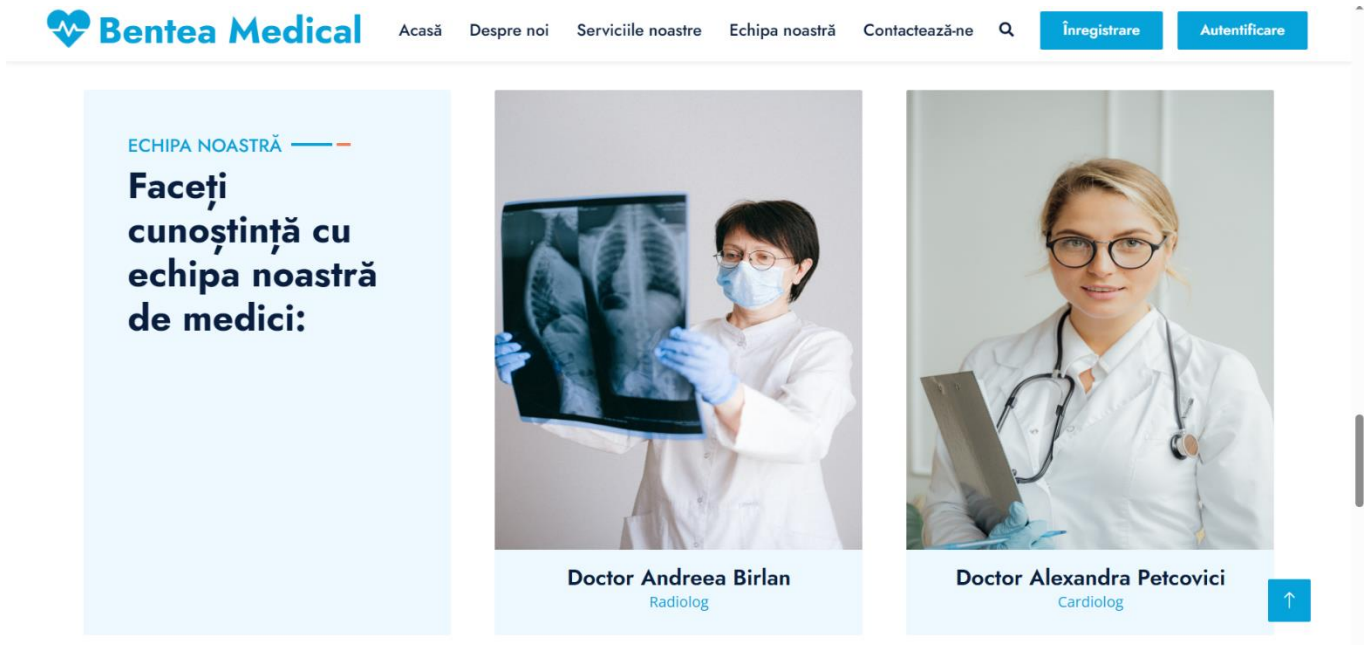

Figura 25. Detalii despre echipă în aplicația web

Datele despre servicii și echipa de doctori sunt preluate din baza de date și deci poti fi modificate cu ușurință de un administrator pentru a păstra verosimilitatea datelor (în cazul în care un doctor părăsește cabinetul, datele sale aferente și serviciile care i se atribuie nu vor mai fi afișate).

3.4 Înregistrarea

La apăsarea pe butonul de "Înregistrare" de pe pagina principală vom avea de a face cu o pagină ce ne prezintă un formular în care ne putem înregistra, înregistrarea este exclusiv pentru pacienți, doctorii vor fi adăugați de către un administrator. Dacă datele sunt valide (fiecare câmp este completat corect și conform așteptărilor) și dacă nu există deja un user cu același email vom fi înregistrați în aplicație ceea ce înseamnă că ne vom putea autentifica.

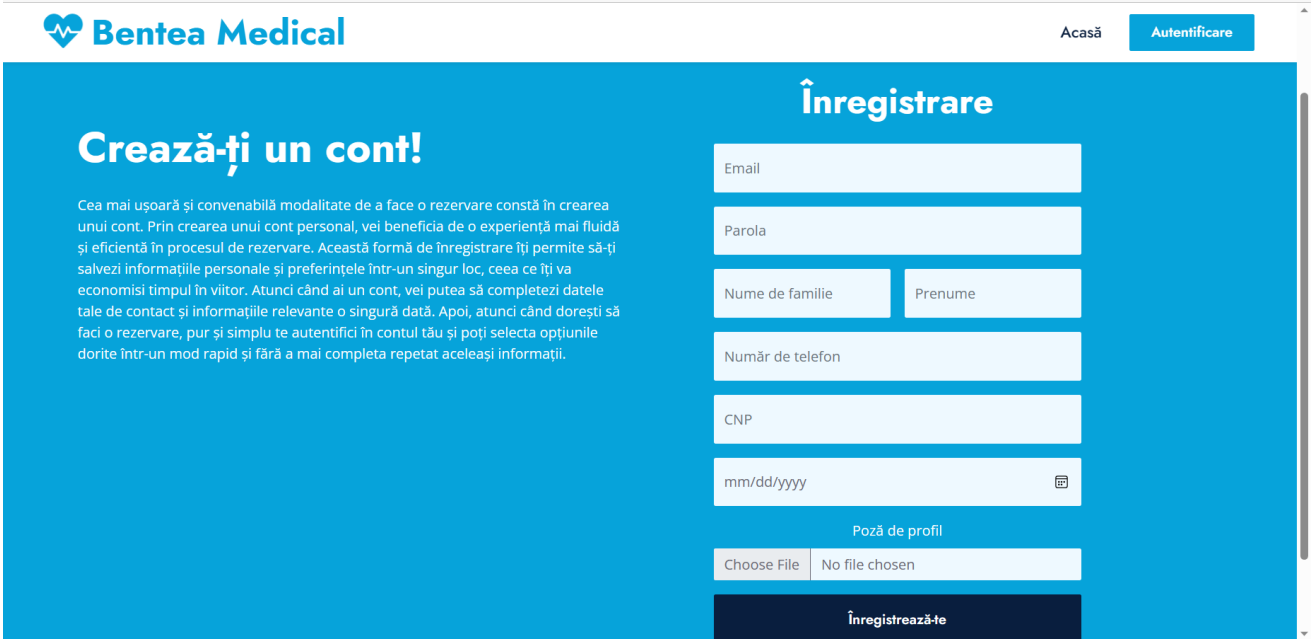

Figura 26. Interfața de înregistrare din aplicația web

3.5 Autentificarea

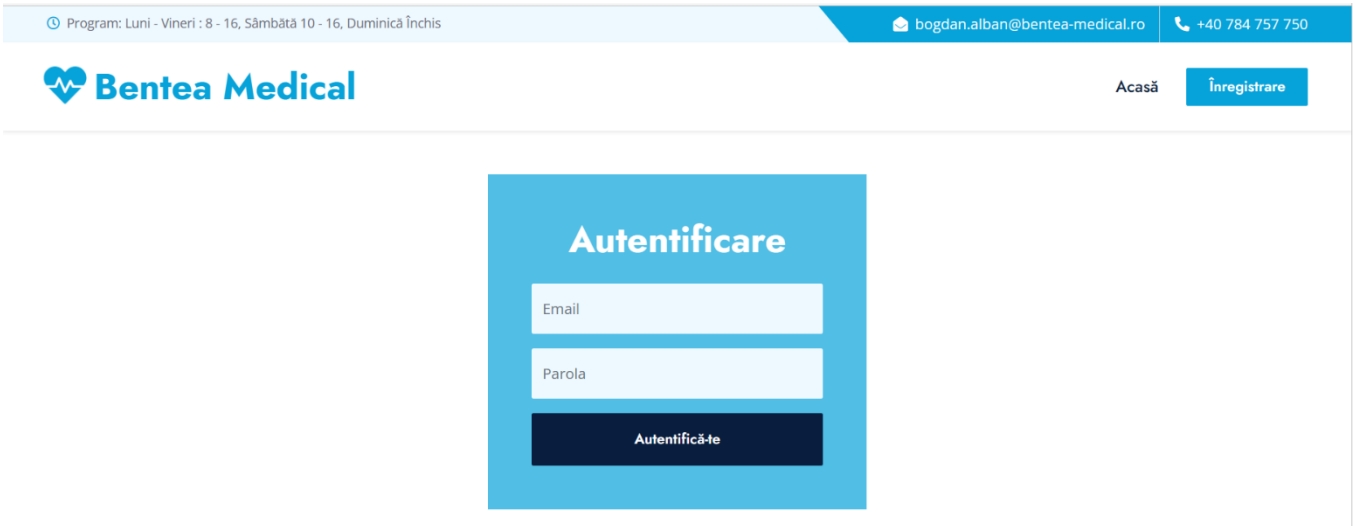

Figura 27. Interfața de autentificare din aplicația web

După ce ne-am creat un cont, sau dacă aveam în prealabil un cont cu un alt rol, avem opțiunea de a ne autentifica, după rolul pe care îl deținem vom avea diferite interfețe afișate către noi.

3.6 Interfața pacientului

Pacientului i se va afișa o interfață de programare, după ce își alege doctorul dorit și serviciul oferit de acest doctor, i se va afișa un calendar ce reprezintă calendarul doctorului selectat și acesta va avea opțiunea de a alege o dată și o oră la care dorește să aibă loc programarea, dacă aceasta nu se suprapune cu altele deja existente ale doctorului, acesta va putea să se programează în perioada specificată.

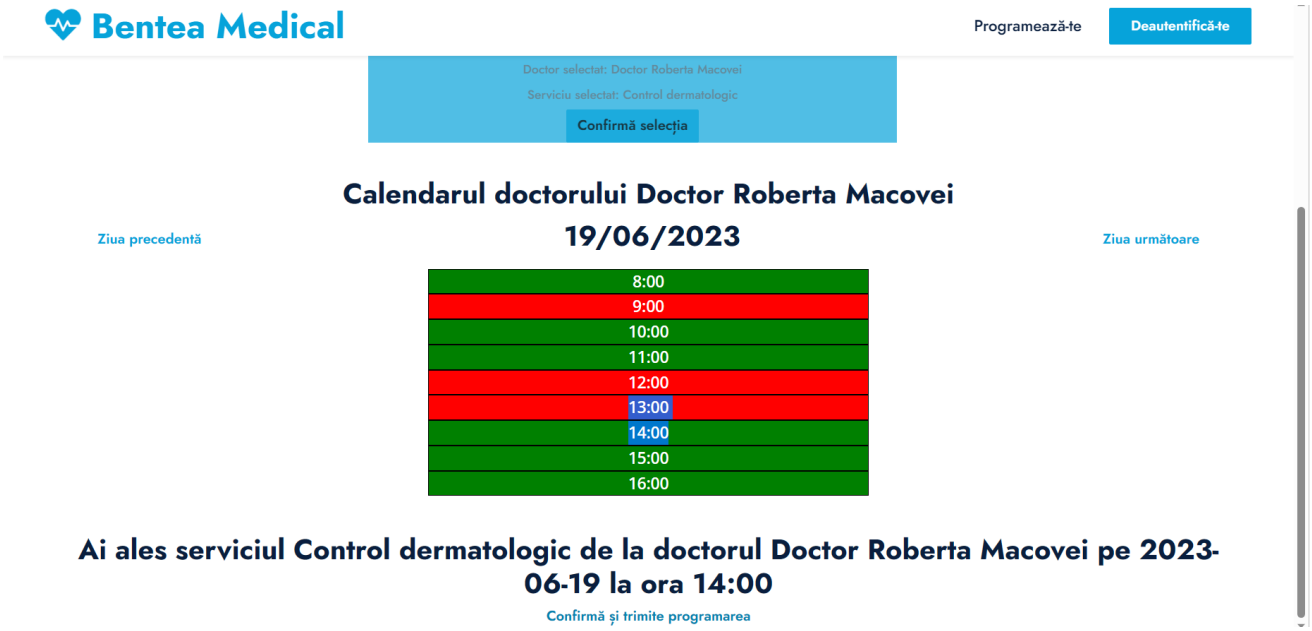

Figura 28. Interfața pacientului din aplicația web

3.7 Interfața doctorului

Doctorul va avea două opțiuni, prima ar fi să vadă lista de pacienți și să le poată atribui fișiere în istoricul medical, acesta dispune de o bară de cautare pentru a găsi cu ușurință pacientul căruia își dorește să-i adauge un fișier în istoricul medical. Căutarea nu e restrictivă și va facilita găsirea oricărui pacient.

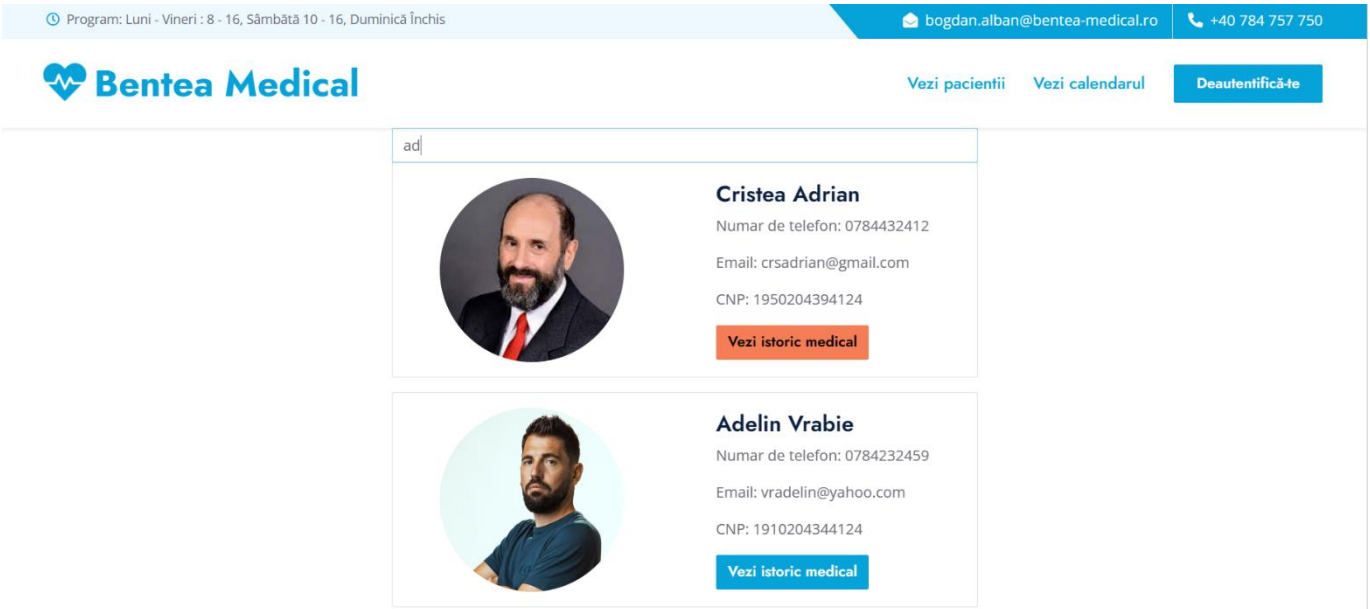

Figura 29. Interfața de vizualizare pacienți, a doctorului, în aplicația web

Pe apăsarea butonului intitulat "Vezi istoric medical", doctorul va avea posibilitatea de a vedea o listă cu fișierele atribuite acelui pacient, listă din care-și poate descărca fișierele sau poate adăuga unele noi.

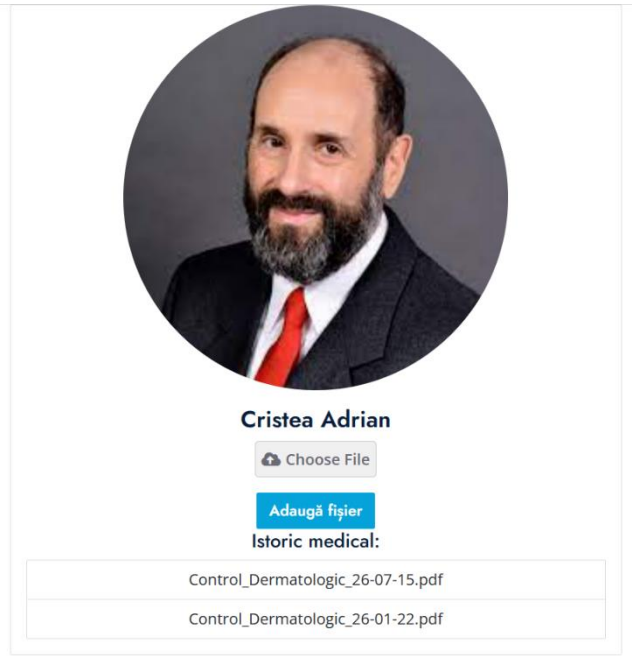

Figura 30. Interfața de adăugare/descărcare de fișiere în istoricul medical al unui pacient

Cea de a doua opțiune pe care o are doctorul este de a-și vedea propriul calendar, unde va putea vedea care este orarul personal, cu ce pacienți va avea de a face și serviciul pe care îl va presta, astfel asigurând o mai ușoară administrare a timpului.

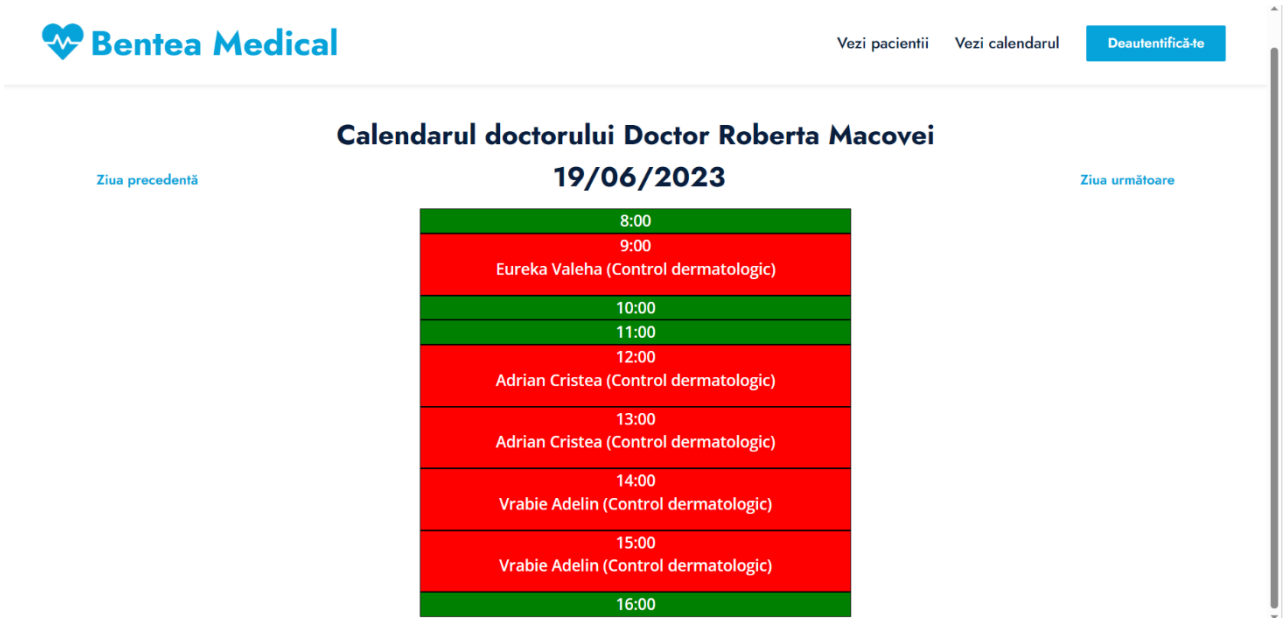

Figura 31. Interfața de calendar propriu a doctorului

3.8 Interfața administratorului

Administratorul va putea vedea lista de pacienți, de doctori, de servicii și de programări, acesta va putea efectua operațiile de adăugare, editare și ștergere pe oricare dintre acestea.

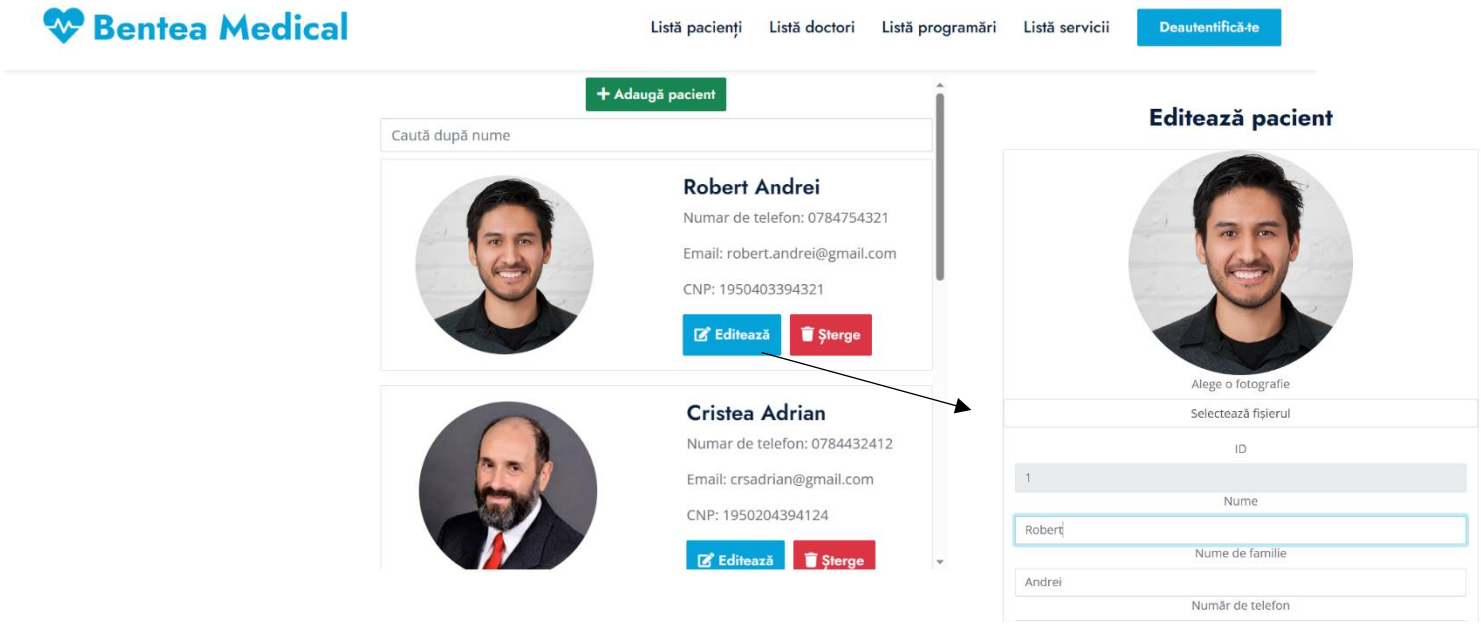

Figura 32. Interfață a administratorului de vizualizare și opțiuni asupra clienților

CAPITOLUL 4

CONCLUZII ȘI POTENȚIALE DEZVOLTĂRI

Cabinetele medicale au un rol esențial în asigurarea bunăstării și sănătății populației. Acestea reprezintă puncte de acces primar la serviciile medicale și sunt considerate prima linie de apărare în îngrijirea și tratarea pacienților.

Aplicația de față își atinge scopul de a facilita programarea la astfel de cabinete medicale, devenind astfel o unealtă importantă pentru toți actorii implicați în această acțiune, pacienții primesc o platformă care ușurează comunicarea cu cabinetul medical, nemaifiind nevoie ca timpul să fie pierdut pe diferite apeluri telefonice, sincronizări ale doctorilor cu programul și cu celelalte programări, iar doctorii își pot direcționa atenția înspre scopul lor principal și vital, tratarea și diagnosticarea pacienților.

Pe lângă partea funcțională, aplicația își atinge și scopul de a deservi ca o prezentare a cabinetului medical, a doctorilor, a serviciilor oferite și detalii despre așezarea geografică a cabinetului, pe lângă oferirea de metode alternative către pacienți de a se programa.

Aplicația este, totuși, doar un prototip, există multe puncte cheie care ar îmbunătăți semnificativ utilizarea acesteia de către toate părțile, printre acestea amintim:

- Posibilitatea de a se genera statistici în legătură cu fișierele din istoricul medical, astfel ca doctorii să poată vedea parcursul sănătății unui pacient de-a lungul timpului și să poată sugera diverse controale sau ajustări pentru a menține sănătatea pacienților

- Programările să se facă într-un sistem de cerere, doctorii să poată primi cereri de programări din direcția pacienților și să le accepte, să le respingă sau să poată propune o nouă dată pentru acestea

- Sistem de notificari pe email sau telefon, aplicația să poată trimită diferite mesaje informative către pacienți, fie că a trecut mult timp de la un ultim control, fie că programarea lor începe curând și ar trebui să se grăbească să ajungă la cabinet, fie că programarea curentă durează mai mult decât a fost estimat și pot să-și mai întârzie venirea la cabinet, posibilitățile sunt multe

- Interfață mai modernă (în special pentru calendar, care arată relativ primitiv în comparație cu restul componentelor)

Deși este într-un stadiu relativ incipient, aplicația își atinge scopurile de bază, și, cu câteva ajustări, poate deveni o aplicație care este folosită de sute de oameni.

CAPITOLUL 5

BIBLIOGRAFIE

[1] Cerf, V. G. (2012). The dawn of the internet. In A brief history of the future (pp. 11-42). Morgan & Claypool.

[2] Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., ... & Wolff, S. (2009). Brief history of the internet. ACM SIGCOMM Computer Communication Review, 39(5), 22-31.

[3] Berners-Lee, T., Cailliau, R., Groff, J. F., & Pollermann, B. (1992). World-wide web: The information universe. Electronic Networking: Research, Applications and Policy, 1(2), 52-58.

[4] Duckett, J. (2014). JavaScript and jQuery: Interactive Front-End Web Development. Wiley Publishing.

[5] Johnson, B. (2019). Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. John Wiley & Sons.

[6] Robbins, J. N. (2012). Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. O'Reilly Media.

[7] Lerman, J. (2012). Programming Entity Framework: Code First (2nd ed.). O'Reilly Media.

[8] GitHub, URL: https://en.wikipedia.org/wiki/GitHub.

[9] Angular – Introduction to Angular concepts, URL: https://angular.io/guide/architecture#introduction-to-angular-concepts

[10] Shklar, L., & Rosen, R. (2009). Web Application Architecture: Principles, Protocols, and Practices. Wiley.

[11] Introduction to Git and Types of Version Control Systems, URL: https: //serengetitech.com/tech/introduction- to- git- and- typesof-version-control-system

[12] Mardan, A., Corrigan, S. I. (2018). Practical Node.js. Springer.

[13] Sullivan, R. T., & Gibson, P. (2019). SQL Server 2017 Administration Inside Out. Microsoft Press.

[14] Dewhurst, S. (2018). C# 7.1 and .NET Core 2.0 - Modern Cross-Platform Development (3rd ed.). Packt Publishing.

[15] HTTP response status codes, URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[16] npm Docs, URL: https://docs.npmjs.com/

[17] Angular documentation, URL: https://angular.io/docs

[18] Kauffman, A., & Sullivan, S. (2020). Entity Framework Core in Action. Manning Publications.

[19] Powell, T., & Derbyshire, J. (2020). HTML, CSS & JavaScript Web Publishing in One Hour a Day, Sams Teach Yourself (8th ed.). Pearson.

[20] W3Schools, URL: https://www.w3schools.com/