MINISTERUL EDUCAȚIEI
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

PLATFORMĂ CENTRALIZATĂ PENTRU TRIMITEREA ȘI ACCESAREA ELECTRONICĂ A PRESCRIPȚILOR ȘI TRIMITERILOR MEDICALE

PROIECT DE DIPLOMĂ

Autor:**Eduard Emil SZEBENI**

Conducător științific: **As. dr. ing. Claudiu DOMUȚA**

2023

MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Vizat,

DECAN
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT AUTOMATICĂ
Prof. dr. ing. Honoriu VĂLEAN

Autor: **Eduard Emil SZEBENI**

Platformă centralizată pentru trimiterea și accesarea electronică a prescripților și trimiterilor medicale

1. **Enunțul temei:** *O aplicație web care digitalizează procesul prin care, medicul de familie pote să trimită o prescriere pentru o anumită problemă de sănătate a unui pacient. Acesta, la rândul lui, poate să se programeze cu trimiterea la un medic specialist care în urma unei investigații, poate să creeze o rețetă, în continuare pacientul poate să o trimită mai departe unei farmacii. Farmacia va informa pacientul când medicamentele sunt pregătite și care este prețul acestora.*

2. **Conținutul proiectului:** *Pagina de prezentare, Declarație privind autenticitatea proiectului, Sinteza proiectului, Cuprins, Studiu Bibliografic, Analiză, proiectare, implementare, Testare, Concluzii, Bibliografie.*

3. **Locul documentării:** *Universitatea Tehnică din Cluj-Napoca*

4. **Consultanți:** *As. dr. ing. Claudiu Domuța*

5. **Data emiterii temei:** 09.01.2023

6. **Data predării:**

Semnătura autorului _____

MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Semnătura conducătorului ştiinţific _____

Declaraţie pe proprie răspundere privind

autenticitatea proiectului de diplomă

Subsemnatul(a) **Eduard Emil SZEBENI**,
legitimat(ă) cu CI/BI seria XM nr. 014242,CNP 5001222244480,
autorul lucrării:

Platformă centralizată pentru trimiterea şi accesarea electronică a prescripţilor şi trimiterilor medicale

elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la **Facultatea de Automatică şi Calculatoare**, specializarea Choose an item.**,** din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Choose an item. a anului universitar 2022-2023, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Eduard Emil SZEBENI

_____

(semnătura)

MINISTERUL EDUCAȚIEI
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SINTEZA

proiectului de diplomă cu titlul:

Platformă centralizată pentru trimiterea și accesarea electronică a prescripților și trimiterilor medicale

Autor: **Eduard Emil SZEBENI**

Conducător științific: **As. dr. ing. Claudiu DOMUȚA**

1. Cerințele temei: O aplicație web care digitalizează procesul prin care, medicul de familie pote să trimită o prescriere pentru o anumită problemă de sănătate a unui pacient. Acesta, la rândul lui, poate să se programeze cu trimiterea la un medic specialist care în urma unei investigații, poate să creeze o rețetă, în continuare pacientul poate să o trimită mai departe unei farmacii. Farmacia va informa pacientul când medicamentele sunt pregătite și care este prețul acestora.

2. Soluții alese: Implementarea sistemului de programare la medic, de eliberare a rețetelor și de ridicarea medicamentelor din farmacie, într-o aplicație web folosind framework-ul Spring Boot împreună cu Hibernate pentru partea serverului și React cu Redux pentru partea clientului.

3. Rezultate obținute: Completarea unui ciclu complet, care pornește de la medicul de familie, care eliberează o trimitere pentru medicul specialist, până la farmacistul care notifică pacientul că medicamentele au fost pregătite și prețul acestora.

4. Testări și verificări: Testarea manuală a tuturor endpoint-urilor, înregistrările din baza de date și funcționalitatea interfeței utilizatorului.

5. Contribuții personale: Ideea sistemului de digitalizare, proiectarea arhitecturii sistemului, contribuții la implementarea sistemului.

6. Surse de documentare: Cărți și documentații de specialitate în format fizic și electronic, articole științifice publicate pe internet.

Semnătura autorului _____

Semnătura conducătorului științific _____

Cuprins

1 Introducere

1.1 Context general

În ultima perioadă de timp, pentru a ne uşura existenţa, tot mai multe dintre sistemele pe care le folosim au ajuns să fie implementate în mediul online. În momentul actual există aplicaţii pentru facilitarea activităţilor zilnice, pentru economisirea timpului şi pentru reducerea numărului de deplasări.

Un domeniu foarte vast care se bucură de o rapidă creştere a digitalizării şi a intrării în mediul online este medicina. Cea mai mare dezvoltare din partea mediului privat, observându-se dorinţa de a fi mereu în pas cu tehnologia. Cu toate acestea sunt multe arii importante care nu au fost dezvoltate îndeajuns sau chiar deloc.

Un exemplu elocvent este că, majoritatea studenţilor studiază la universităţii din afara localităţilor de domiciliu, ceea ce face ca şi comunicarea cu medicul de familie să fie mult mai complicată chiar şi pentru o problemă banală, deoarece este dificil pentru pacient să intre în posesia unei trimiteri eliberate de mediul său de familie.

În prezent există doar moduri improvizate pentru a transmite aceste trimiteri, nefiind existente căi oficiale. Dar există unele site-uri unde se pot realiza programări pentru investigaţii efectuate de medici specialişti pentru anumite spitale, dezavantajul este că nu se pot ataşa trimiterile pacientului aici. Alte aplicaţii care îmbunătăţesc modul de creare a trimiterilor sunt cele folosite de medicii de familie pentru a printa fizic aceste trimiteri. Aici dezavantajul este că trimiterile nu se stochează într-o bază de date şi astfel nu pot să fie accesate în orice moment şi nu se poate crea un istoric pe baza acestora.

Este foarte important să avem cât mai mult timp la dispoziţie pentru activităţile noastre preferate şi să reducem cât mai mult timpul petrecut la cozi, prin faţa cabinetelor medicale sau prin farmacii. În momentul în care trebuie să mergem la farmacie după medicamente trebuie să aşteptăm până farmacistul citeşte reţeta, până pregăteşte medicamentele necesare, până notează indicaţile. S-ar putea reduce apropae în totalitate acest timp pierdut dacă pacientul ar merge doar pentru a ridica medicamentele gata pregătite alături de indicaţiile necesare.

Pentru realizarea cu uşurinţă a acestor nevoi, propun realizarea unei aplicaţii web care să ajute pacienţii să intre în posesia trimiterilor şi a reţetelor în format electronic şi să evite deplasările în plus şi timpul pierdut în farmacii. Prin această aplicaţie se mai urmăreşte şi eficientizarea modului în care medicii gestionează trimiterile şi reţetele medicale.

În al doilea capitol sunt descrise tehnologiile folosite în prezent pentru aplicaţiile web. Aceste tehnologii mai departe au fost folosite pentru construirea acestei aplicaţii web. Al treilea capitol descrie modul în care această aplicaţie a fost construită pentru componenta server şi componenta client. Capitolul următor descrie procedeul de testare

care a fost urmat pentru verificare funcționării corecte a metodelor din aplicație și a tuturor celorlalte interacțiuni.

1.2 Obiective

În această lucrare mi-am propus realizarea unei aplicații web prin care pacienții să poată primi trimiteri de la medicul de familie pentru investigații și de a putea realiza programări către medicii specialiști în care se pot atașa aceste trimiteri. După investigații medicii pot să genereze o prescriere care se completează în aplicație și se trimite pacientului. Cu această prescriere din contul pacientului, acesta poate să aleagă una dintre farmaciile din baza de date și să o trimită. Ca răspuns farmacistul anunță pacientul că medicamentele sunt pregătite, prețul acestora și perioada în care farmacia este mai puțin aglomerată.

Unul dintre obiective pentru această lucrare este de a îmbunătăți procesul de transmitere a trimiterilor dintre medicul de familie și pacient, în special pentru pacienții care nu au medicul de familie în orașul în care locuiesc în prezent, cum ar fi majoritatea studenților.

Un alt obiectiv este de a digitaliza trimiterile și prescrierile primite de la medici, în acest mod acestea se pot stoca mult mai ușor decât în format fizic, în acest mod fiind mult mai protejate.

Unul dintre obiectivele minore este de a economisi cât mai mult timp, în special pentru pacienții care evită procesul de procurare a trimiterii de la medicul de familie și statul la coadă la farmacie.

1.3 Specificații

Aplicația web propusă va avea cinci tipuri de utilizatori, fiecare dintre aceștia având o interfață puțin diferită dar care se bazează pe aceleași concepte. Tipul principal de utilizator în jurul căruia este construită aplicația este pacientul, care poate să-și adauge medicul de familie, poate să vizualizeze trimiterile eliberate de medic, poate crea programări, unde se pot atașa aceste trimiteri, de asemenea poate vizualiza prescrierile sale, pe care le poate trimite farmaciei alese.

Un alt tip de utilizator este medicul de familie care poate să vizualizeze o listă cu proprii lui pacienți și poate crea trimiteri pentru aceștia.

Asemănător medicului de familie avem medicul specialist care are o listă cu programări și trimiterile atașate acestora și poate elibera prescrieri medicale pentru pacienții investigați.

Farmacistul primește o listă cu prescrieri și posibilitatea de a anunța pacientul când medicamentele sale sunt pregătite, prețul medicamentelor și indicațiile adiționale.

Ultimul tip de utilizator este administratorul care are opțiunea de a confirma conturile medicilor și farmaciștilor, pentru a se evita conturile false și opțiunea de a înregistra farmacii noi în aplicație.

Aplicația ar trebuii să se bucure de o viteză mare deoarece intenția este de a folosii legături slăbite între entitățile aplicației, acestea putând fi ușor și rapid de accesat.

Toate conturile utilizatorilor vor fi securizate cu token-uri JWT, la fiecare sesiune în care sunt logați, aceste token-uri care expiră odată la fiecare 30 de minute, parolele de asemenea vor fi criptate și stocate astfel în baza de date.

Una din limitările acestei aplicații este că necesită conectivitate la internet în mod permanent, astfel fiind imposibil de accesat din zonele fără conexiune la internet.

2 Studiu bibliografic

2.1 Aplicație WEB

O aplicație web este un software care rulează într-un browser web, cum ar fi Google Chrome sau Mozilla FireFox, aceasta este stocată într-un server care se află la distanță și se conectează la interfața aflată în browser prin intermediul internetului.

Majoritatea caracteristicilor din aplicațile folosite în ziua de azi sunt la bază aplicații web, cum ar fi: cărucioarele de cumpărături de la magazinele online sau fluxurile de informații din rețelele de socializare.

Acest tip de aplicații oferă posibilitatea de a accesa anumite funcționalități complexe fără a fi nevoie de un software care să fie instalat sau configurat în prealabil. Este nevoie doar o conexiune la internet pentru a se putea accesa toate funcționalitățile disponibile pe serverul aplicației, acest avantaj simplifică aplicația pentru utilizator fiind mult mai ușor de folosit. În același timp se elimină și nevoia de mentenanță pentru spațiul de stocare. Cu alte cuvinte aplicația este mereu actualizată la ultima versiune crescând securitatea utilizatorilor.

Unul dinre avantajele importante este că aplicația poate să fie accesată concomitent de mai mulți utilizatori fără a fi nevoie de o infrastructură îmbunătățită a aplicației. Un alt avantaj este posibilitatea de a accesa o aplicație prin intermediul mai multor web browsere și cu o gamă largă de dispozitive, din orice loc, singura cerință fiind nevoie de conexiune la internet.

Aplicațiile Web folosesc arhitectura client-server, tot codul este împărțit în două componente, componenta clientului care conține interfața și componenta serverului care conține metodele, clasele și logica de funcționalitate a aplicației.

Componenta clientului se ocupă cu funcționalitatea interfeței cum ar fi: butoanele, check box-urile și dropdown-urile. Fiecare acțiune a userului cu interfața, de exemplu un click pe un buton, trimite o cerere (request) de la utilizator către sever.

Componenta serverului se ocupă cu procesarea de date. Serverul procesează cererea clientului și trimite înapoi un răspuns în funcție de modul în care este implementată logica de gestionare a datelor. Cererile primite de server, în general se referă la posibilitatea de a primi mai multe informații din baza de date, de a edita informații din baza de date sau pentru a salva informații noi. Răspunsul trimis de server este informația cerută sau un mesaj care arată dacă cererea s-a realizat cu succes sau nu, aceste cereri sunt de tip HTTP. [1]

2.2 Conceptele HTTP

Acronimul HTTP vine de la Hyper Text Transfer Protocol, este modul prin care browserul web adică componenta client comunică cu componenta sever. O astfel de cerere este făcută de client către un port care aparține de un server, cererea are ca

obiective accesarea unei resurse aflată în baza de date a sistemului. Această bază de date nu poate să fie accesată în mod direct. [2]

Acest protocol definește mai multe concepte cheie care exemplifică modul de funcționare. HTTP conține mai multe metode pentru interacțiunea cu resursele serverului, cum ar fi: metoda GET, care face o cerere pentru o resursă prin intermediul serverului, metoda POST trimite informații către server pentru a fi procesate, de cele mai multe ori această metodă este folosită pentru a crea resurse noi în baza de date, metoda PUT care este folosită pentru a actualiza o resursă deja existentă, metoda DELETE se ocupă de ștergerea uneia sau a mai multor resurse din baza de date. Aceste metode constituie CRUD care este un acronim pentru create, read, update, delete, acestea fiind acțiunile metodelor de bază. [3]

Un alt concept important poartă denumirea de HTTP headers care oferă informații adiționale despre cererile către server și răspunsurile de la server. Acest headers este o componentă dintr-o cerere trimisă de client pentru a primi informații despre autorizări sau cookie-uri într-o aplicație.

Componenta headers care apare într-un răspuns trimis de către server pentru a oferi informații suplimentare despre răspunsul din care face parte cum ar fi: un mesaj de succes sau o eroare, în funcție de cum a decurs cererea trimisă în prima parte. Ultima componentă a unei cereri HTTP este reprezentată de un Response Body (Corpul răspunsului) acesta reprezentând conținutul mesajului propriu-zis. [4]

2.3 API

API (Application Programming Interface) este un set de reguli și de protocoale care oferă aplicațiilor software posibilitatea de a comunica și interacționa între ele.

Un API se bazează pe modelul cerere-răspuns (request-response) în care o aplicație face o cerere pentru o anumită informație sau acțiune către o altă aplicație care răspunde cu informația cerută sau realizează acțiunile cerute.

API-urile oferă un nivel de abstractizare care ascunde complexitatea programelor cu care lucrează, astfel programatorul poate să lucreze cu o interfață simplificată fără a fi nevoie ca acesta să înțeleagă toată logica de programare din spatele metodelor pe care le folosește.

API-urile se construiesc după anumite standarde și protocoale cum ar fi HTTP sau REST pentru a asigura consecvență și interoperabilitate în toate sistemele în care sunt folosite. [5]

Există mai multe tipuri de API-uri:

- API-urile Web care sunt folosite pentru comunicarea prin intermediul internetului. Acestea sunt bazate pe protocolul HTTP și de obicei folosesc principiile RESTful pentru a crea, citi, actualiza și șterge
- Există API-uri care sunt librării sau framework-uri deja construite pe care programatorii le pot folosi în aplicațiile lor. Acestea oferă metode care simplifică dezvoltarea unei aplicații prin refolosirea codului

- Sistemele de operare dispun de API-uri care permit ca aplicaţiile să interacţioneze cu resursele sistemului cum ar fi: fişiere, procese sau conexiuni la internet. Aceste API-uri permit programatorilor să creeze aplicaţii care sunt specifice unei singure platforme.
- Există şi API-uri pentru bazele de date cum ar fi: JDBC pentru JAVA ODBC pentru alte limbaje de programare. Prin aceste API-uri se poate comunica cu baza de date pentru a se executa comenzi specifice acestora ca preluarea sau modificarea informaţiilor [6]

Folosirea API-urilor vine cu nişte beneficii importanate:

- Cu ajutorul acestora, un sistem foarte complex se poate împărţi în componente mai mici şi mai uşor de refolosit între mai multe sisteme, făcând codul mai uşor de urmărit
- Interoperabilitatea oferă posibilitatea pentru diferite aplicaţii, servicii sau sisteme de a lucra împreună, promovând astfel integrarea între diferite tehnologii
- Cum există API-uri care oferă componente deja construite care se pot integra într-o aplicaţie se reduce timpul de dezvoltare şi se reduce efortul dezvoltării unei noi aplicaţii nemaifiind nevoie de a scrie cod care deja este folosit pentru o altă aplicaţie
- API-urile au un ecosistem unde programatorii pot să îşi expună serviciile care pot mai departe să fie folosite de alţi programatori pentru diferite proiecte, astfel se promovează inovaţiile.

2.4 MVC Framework

MVC este acronimul de la Model View Controller, acesta fiind un şablon de arhitectură pentru o aplicaţie software. Acesta oferă un mod de a structura modul prin care o aplicaţie este construită, separând aceasta în trei componente distincte: Model, View şi Controller. Fiecare din aceste componente este destinată pentru un aspect specific al aplicaţiei. [7]

Avantajul major pentru acest framework este că oferă o separare clară pentru fiecare componentă a aplicaţiei, astfel programatorii pot să lucreze independent pentru diferite aspecte, schimbări ale unei componente şi nu afectează în mod direct restul componentelor doar dacă aceste schimbări urmăresc şablonul şi regulile arhitecturii MVC. Acest framework oferă posibilitatea de a reutiliza codul, modularitatea acestuia.

Compromisul pentru aceste avantaje este creşterea complexităţii programului fiind împărţit în diferite componente.

2.4.1 Model

Model este componenta care se ocupă cu logica legată de datele cu care utilizatorul poate să lucreze. Această componentă ţine toată informaţia încapsulată pentru a putea fi accesată doar prin intermediul unor metode speciale care se ocupă cu manipularea

acestor informaţii, cum ar fi obţinerea informaţiilor din baza de date sau validarea acestora.

2.4.2 View

Componenta View este reprezentată de interfaţa utilizatorului. Aceasta defineşte modul în care informaţia de la componenta model este reprezentată pentru utilizatorul aplicaţiei. O astfel de componentă de obicei este alcătuită din elemente de HTML, CSS şi JavaScript conţinând butoane, dropdown-uri şi check box-uri cu care utilizatorul poate interacţiona.

2.4.3 Controller

Rolul componentei Controller este de a asigura un canal între componenta Model şi View. Această componentă există pentru a procesa inputul primit de la utlilizator după care să fie procesat şi transmis componentei Model în funcţie de request-ul primit. Cu ajutorul Controller-ului se asigură separarea şi independenţa fiecărei componente. Controllerul este componenta care decide modul în care informaţia este folosită pentru întreaga aplicaţie folosindu-se de View pentru a afişa aceste informaţii şi de Model pentru a le obţine din baza de date.

În Figura 2.1 se prezintă legătura dintre componentele unei aplicaţii bazate pe arhitectura MVC (Model, View, Controller).

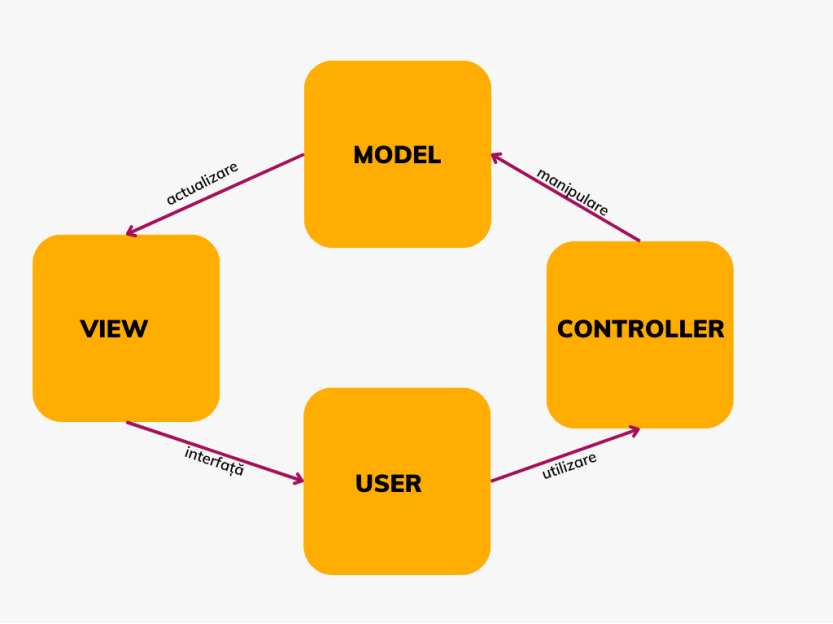

Figura 2.1 Componentele MVC

2.5 Spring Boot

Spring Boot este un framework bazat pe limbajul de programare Java, deschis „open source”, care simplifică dezvoltarea aplicaţiilor Java în special pentru aplicaţiile

web și microservicii. Este construit cu ajutorul framework-ului Spring, oferind un mod de a configura și a lansa aplicații Java care pot să ajungă direct în stadiul de producție. [8]

Principalul scop al acestui framework este de a minimiza procesul necesar de configurare și înființare pentru o aplicație Spring, prin adoptarea unui set de configurări logice implicite, permițând astfel programatorilor să înceapă dezvoltarea unei aplicații noi fără a aloca prea mult timp pe o schemă de amploare. [8]

Figura 2.2 arată componenta parent care definește configurațile de bază într-un proiect cu Spring Boot.

Figura 2.2 Dependență pentru Spring Boot

Unele din caracteristicile cheie ale acestui framework sunt:

- Auto-Configurarea: Spring Boot poate să configureze mai multe componente în funcție de dependințele din proprietățile aplicației. Astfel se elimină nevoia configurării manuale și se reduce probabilitatea apariției unor erori
- Server Încorporat: Spring Boot are în componență un server încorporat cum ar fi Tomcat, care oferă posibilitatea aplicațiilor să fie pornite fără a fi nevoie de instalarea pe un server extern
- Managementul Dependințelor: Spring Boot simplifică managementul versiunilor dependințelor, astfel se asigură o bună compatibilitate între acestea
- Integrarea ecosistemului Spring: Spring Boot integrează alte componente din ecosistemul Sping cum ar fi: Spring Security, Spring Cloud și Spring Data. Toate acestea fiind integrate într-un ecosistem, se pot folosi necesitând un nivel minim de configurare. [8]

2.5.1 Dependințele în Spring Boot

În Spring Boot dependințele sunt gestionate de instrumente de automatizare cum ar fi: Apache Maven sau Gradle. Aceste dependințe se definesc în aplicație în fișierul "pom.xml" în Maven sau în "build.gradle" în Gradle. Cele mai folosite dependințe pentru o aplicație web construită cu Spring Boot sunt:

- Spring-Boot-Starter-Web: această dependință include librăriile necesare pentru o aplicație web, cum ar fi un server încorporat (Tomcat) și librării pentru gestionarea cererilor și răspunsurilor HTTP

- Spring-Boot-Starter-Data-Jpa: această dependință oferă suport pentru folosirea Java Persistence API cu Spring Data, incluzând librăriile necesare pentru legătura cu baze de date relaționale.
- MySql-Connector-Java: este o dependință specifică bazei de date relaționale MySql. Include driver-ul JDBC care oferă posibilitatea aplicației construită cu Spring Boot să se conecteze la baza de date MySql [9]

2.6 Spring Security

Spring Security este un framework folosit pentru aplicațiile Java create cu ecosistemul Spring, oferind un mod de a securiza aplicația. Acest framework pune la dispoziția programatorului mai multe caracteristici și opțiuni pentru a manipula autentificarea, autorizarea și alte nevoi legate de securitate pentru aplicație, într-un mod flexibil și modular. Unele din obiectivele pe care Spring Security le îndeplinește sunt următoarele: [10]

- Autentificarea: acest framework pune la dispoziție diferite mecanisme pentru autentificarea utilizatorilor cum ar fi autentificarea bazată pe un formular, autentificarea simplă HTTP sau cu ajutorul token-urilor JWT(Json Web Token)
- Autorizarea: Spring Security oferă mecanisme pentru a controla accesul utilizatorilor pentru diferite părți ale aplicației și poate securiza URL-uri sau anumite metode pentru a fi accesate din exteriorul aplicației. Aceste mecanisme pot să fie bazate pe rolurile utilizatorilor, permisiuni sau reguli predefinite.

2.7 Spring Data

Spring Data este un sub framework din framework-ul Spring care are ca scop simplificarea accesului la diferite nivele de acces de informații pentru aplicațiile care au la bază limbajul Java. Spring Data oferă un model de programare unificat și consistent pentru interacțiunile cu mai multe tehnologii pentru stocarea informațiilor incluzând și bazele de date relaționale. [11]

Caracteristicile cheie pentru Spring Data sunt:

- Abstractizarea, Spring Data prevede abstractizări și interfețe pentru a ascunde complexitatea anumitor tehnologii de acces a informațiilor. În acest mod se permite ca programatorii să aibă opțiunea de a scrie cod care nu interacționează direct cu codul bazei de date
- Conceptul de “Repository Abstraction”. Acesta prevede interacțiunea cu un API care se ocupă de operațiile de bază, creare, citire, actualizare, ștergere pentru interacțiunea cu informațiile din baza de date reducând nevoia de a scrie o parte semnificativă de cod
- Posibilitatea de a pagina și de a sorta informația din baza de date fiind simplificată prin anumite abstractizări

- Metode pentru crearea de query-uri. Spring Data oferă posibilitatea de a defini metode cu query-uri prin anumite convenţii. Codul SQL sau NoSQL se generează automat bazându-se pe numele metodei, reducându-se nevoia de a scrie query-uri manual

2.8 HTML CSS JavaScript

Într-o aplicaţie web componenta care se ocupă cu interfaţa utilizatorului este construită din 3 limbaje. Aceste limbaje se folosesc cu orice framework pentru front-end.

Cu ajutorul limbajului HTML se construiesc componentele dintr-o pagină cum ar fi butoane sau tabele, cu CSS se stilizează aceste componente construite şi JavaScript se realizează funcţionalitatea componentelor create înainte.

În Figura 2.3 arată rolul fiecărui limbaj din componenta care se ocupă de interfaţa utilizatorului.

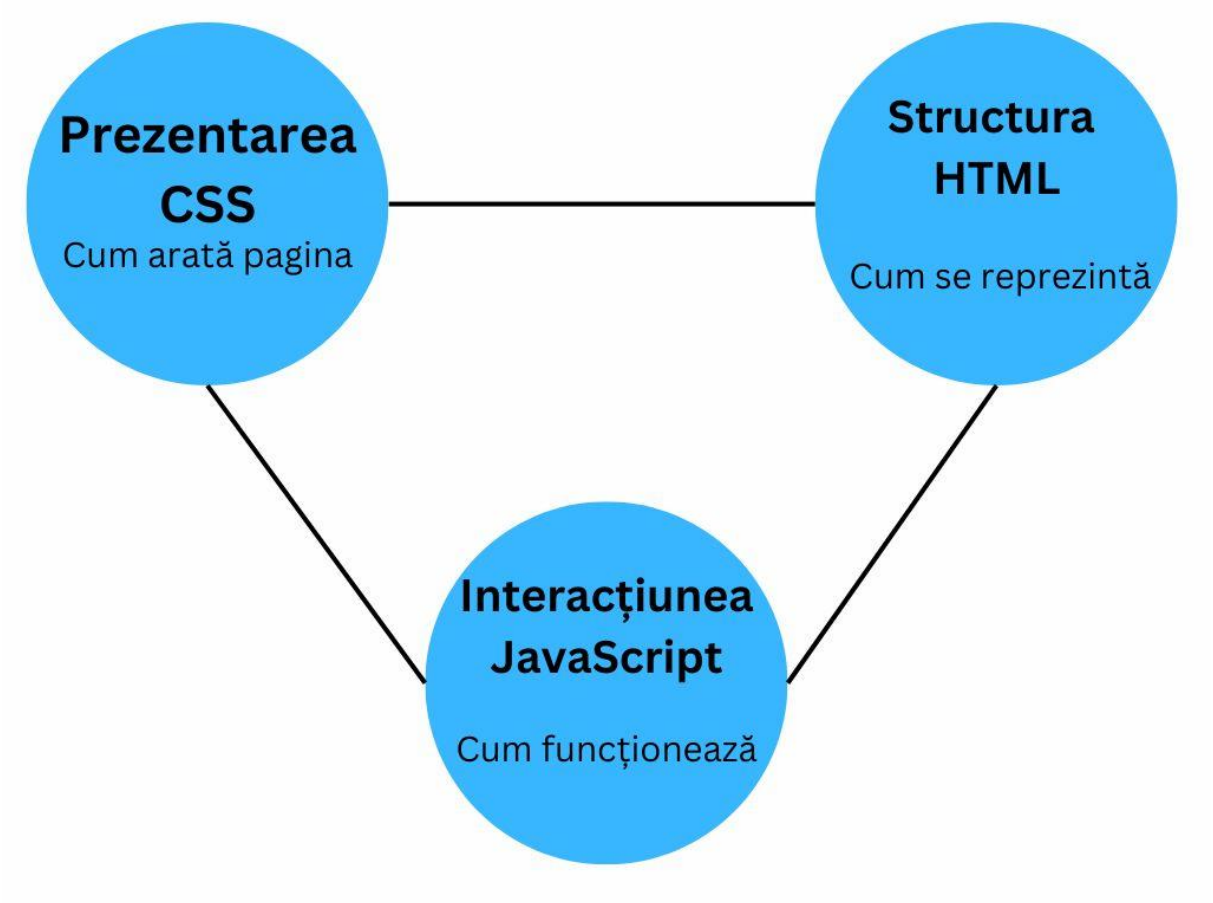

Figura 2.3 Scopul CSS, HTML, JavaScript

2.8.1 HTML

Acronimul HTML vine de la "Hypertext Markup Language", este un limbaj standard folosit pentru crearea de pagini şi aplicaţii web. Structura şi conţinutul acestuia este

reprezentat în WWW(World Wide Web). Acest limbaj se folosesc de o serie de adnotaţii pentru a crea şi pentru a structura o pagina web.

Codul pentru un document HTML se poate scrie în orice editor de text, singura condiţie fiind ca extensia fişierului sa fie ".html". Orice document HTML contine adnotaţia: "<!DOCTYPE html>" care reprezintă locul de unde începe codul HTML. Structura unei astfel de pagini se reprezintă prin adnotaţiile „<html> </html>" care încadrează codul propriu-zis, „<head> </head>" care conţine titlul paginii sau informaţii despre formatare, „<body> </body>" unde se reprezintă informaţia din pagină cum ar fi paragrafe cu text sau imagini, uneori din structura unei pagini web face parte şi „<footer> </footer>", reprezentând partea de jos a paginii unde se află în general informaţii de contact de exemplu numărul de telefon sau adresa de e-mail. [12]

2.8.2 CSS

CSS (Cascading Style Sheets) este un limbaj folosit împreună cu limbajul HTML pentru a îmbunătăţi aspectul vizual pentru o pagina web. Cu ajutorul acestui limbaj programatorul poate să adauge sau să schimbe aspectele vizuale ale paginii după propriile preferinţe, cum ar fi: fonturile, mărimea literelor, culorile textului, culorile de fundal sau redimensionarea imaginilor.

Pentru a stiliza un element specific din HTML, CSS pune la dispoziţie o serie de selectori pentru componentele HTML. Aceşti selectori se pot baza pe numele elementelor (h1{}), id-urile acestora (#id{}) sau pe relaţile dintre elemente(body > p {}).

Acest limbaj pune la dispoziţie şi proprietăţi prin care elementele se pot poziţiona sau alinia într-o pagină cum ar fi: "display" , "flexbox" sau "grid" . Cu aceste proprietăţi se pot obţine pagini care îşi schimbă aspectul în funcţie de mărimea ecranului, aceeaşi aplicaţie poate să aibă un aspect diferit pe un Desktop faţă de aspectul de pe un telefon mobil. [13]

2.8.3 JavaScript

JavaScript este un limbaj de programare de nivel înalt, orientat pe obiect care funcţionează în browserele web. În cele mai multe cazuri este folosit pentru dezvoltarea aplicaţiilor sau paginilor web. Cu acest limbaj se adaugă posibilitatea de a interacţiona dinamic cu elementele create cu limbajul HTML în pagină. Acest limbaj se ocupă în principal cu gestionarea evenimentelor care apar într-o interfaţă web cum ar fi: click-uri, mişcări ale mouse-ului sau interacţiuni cu tastatura. Gestionarea evenimentelor oferă posibilitatea actualizării paginilor web fără a fi nevoie de reîncărcarea paginii.

Chiar dacă la început a fost dezvoltat pentru crearea de cod pentru interfaţa aplicaţiilor, prin intermediul framework-ului Node.js acest limbaj se poate utiliza şi pentru construirea serverului. Astfel acest limbaj oferă suport pentru întreaga aplicaţie web.

Toate browserele din ziua de azi oferă suport şi motoare pentru executarea codului JavaScript în mod eficient. În general aplicaţia trebuie să se muleze pentru nevoile utilizatorilor, de aceea este important acest aspect în care se poate folosi orice browser

după preferințe. JavaScript este cea mai bună alegere pentru dezvoltarea interfețelor aplicațiilor web. [14]

JavaScript pune la dispoziție mecanisme pentru gestionarea informației primite de la un server pentru o interfață dintr-o pagina web. Fiind un limbaj orientat pe obiecte, suportă tipul de date obiect dar are și tipuri de date simple cum ar fi: "string" în care se pot stoca litere, cuvinte și numere; "numbers" unde se pot stoca doar numere și "arrays" unde se pot stoca diferite tipuri de date sub formă de șiruri unidimensionale sau bidimensionale. JavaScript poate să aducă informație în pagina web din surse externe cum ar fi API-urile (Application Programming Interfaces) și astfel se poate afișa pe pagină conținut dinamic prin intermediul actualizărilor în timp real.

S-au dezvoltat mai multe framework-uri pentru o dezvoltare mai eficientă și mai rapidă a interfețelor web, toate acestea având la bază limbajul JavaScript, unele dintre cele mai populare framework-uri care se ocupă cu așa ceva sunt: React, Angular și jQuery.

2.9 Bootstrap

Bootstrap este un framework destinat pentru partea clientului adică pentru interfață. Acesta dispune de o serie de colecții de componente HTML, CSS si JavaScript predefinite astfel procesul de dezvoltare a unei pagini sau aplicații web este mult mai rapid și mai simplu de realizat. [15]

Unul din marile avantaje pentru folosirea acestui framework este că majoritatea componentelor predefinite oferă un design adaptabil pentru diferite dispozitive. Bootstrap oferă și un sistem în care ecranul este împărțit sub formă de tabel cu douăsprezece coloane care se poate personaliza creând diferite aspecte pentru simplificarea poziționării componentelor într-o pagină.

2.10 React

React este una dintre cele mai populare framework-uri pentru dezvoltarea interfețelor destinate utilizatorilor și a aplicațiilor web, arhitectura pe care este bazată se folosește de reutilizarea codului și modularitatea acestuia.

Proprietatea de modularitate este atinsă prin divizarea interfețelor în componente independent construite care conțin propria logică de gestionarea a datelor, propriul mod de aranjare și stilizare a elementelor în pagină, chiar și propriul set de teste. Prin această modularizare se pot obține mult mai ușor aplicați complexe în care este nevoie de o mulțime de elemente la care se poate lucra independent și în paralel aducând un plus de eficientizare a procesului și de îmbunătățire a vitezei de lucru.

React introduce o reprezentare virtuală pentru DOM(Document Object Model) care se numește "Virtual DOM". În loc să se folosească DOM-ul propriu-zis, React construiește o reprezentare virtuală a acestuia în memorie care îmbunătățește performanța programului, tot procesul realizându-se mult mai rapid deoarece în momentul în care sunt schimbări în starea sau în proprietățile unei componente, React

actualizează eficient doar părţile necesare din DOM şi utilizează la un minim necesar acest proces.

Framework-ul React foloseşte o sintaxă declarativă faţă de una imperativă. Prin sintaxa declarativă programatorul scrie codul astfel încât să descrie aplicaţiei modul în care vrea să fie afişată informaţia primită iar React se ocupă de actualizarea interfeţei pentru a arăta schimbările în funcţie de ce informaţii primeşte. Astfel procesul de programare a interfeţei se simplifică nemaifiind nevoie ca programatorul să se ocupe manual de fiecare componentă DOM pentru diferite informaţii primite ca intrare pentru componentă.

Faţă de o aplicaţie web clasică care nu foloseşte un framework pentru crearea interfeţei unde în mod normal pentru a crea componente care se pot afişa pe ecran se foloseşte HTML, în React se utilizează JSX (JavaScriptXML) care este o extensie de sintaxă unde se încorporează cod asemănător cu cel HTML în JavaScript. Extensia JSX este formată dintr-o combinaţie de structurare a codului HTML cu logica din spatele limbajului JavaScript. În acest mod se pot defini toate structurile de componente şi proprietăţile deja existente în HTML dar într-un mod intuitiv şi mai uşor de citit pentru alţi programatori.

În acest framework se introduce conceptul de "React Hooks". Această caracteristică pune la dispoziţia programatorilor opţiunea de a folosi starea de utilizare a unei componente şi alte proprietăţi ale framework-ului fără a mai fi nevoie de componente Clasă. Există mai multe tipuri de "Hooks", dar cele folosite în majoritatea aplicaţiilor sunt "useState", "useEffect" şi "useContext" care simplifică managementul stării componentelor, manipularea evenimentelor secundare şi împărţirea contextului între componente. De exemplu pentru a stoca şi a seta starea unei componente se foloseşte useState, acest "Hook" se declară astfel: const [componentă, setComponentă] = useState(''); astfel starea componentei este stocată in constanta numită "componentă" şi se poate seta cu funcţia "setComponentă", valoarea de bază a acesteia fiind un string gol. [16]

La fel ca şi Spring, React are un ecosistem vast. La acest ecosistem contribuie foarte mult comunitatea care aduce constant componente refolosibile care se pot utiliza foarte uşor. Cele mai utilizate librării din acest ecosistem care ajută la îmbunătăţirea productivităţii sunt: "React Router" care se ocupă cu rutarea pentru URL-urile paginilor din React, prin aceasta librăriile se poate securiza accesul către pagini cu ajutorul rolurilor de utilizator, "Redux" librărie folosită pentru administrarea stării aplicaţiei şi "Axios" pentru cererile HTTP.

2.10.1 Axios

Axios este o librărie React folosită pentru a face cereri HTTP dintr-un browser web la un server. Oferă un mod simplu şi intuitiv pentru programatori de a apela un API şi de a se ocupa cu operaţii asincrone în comunicarea cu un server. [17]

Axios se bazează pe un model bazat pe Promises (promisiuni), ceea ce înseamnă că pentru fiecare cerere HTTP făcută astfel se returnează un Promise. Un Promise este un obiect care reprezintă modul în care o operaţie asincronă a decurs, în general aducerea informaţiei de la un server, adică poate să rezulte într-un răspuns de la server sau o eroare

în momentul în care ceva nu a decurs bine. Mai departe un Promise pune la dispoziţia programatorului un mod în care poate să gestioneze răspunsul primit şi să fie salvat într-o componentă sau un mod în care poate vedea eroarea primită împreună cu alte detalii legate de aceasta pentru a putea fi rezolvată cât mai uşor. Pentru cazul în care se primeşte răspunsul de la server se utilizează funcţia ".then()" iar pentru cazul în care se primeşte o eroare există funcţia ".catch()" . Aceste funcţii se folosesc concomitent pentru a se acoperi ambele cazuri la fiecare cerere HTTP.

Pentru a putea folosi Axios este necesară instalarea, care se poate realiza printr-un manager de pachete cum ar fi: npm. După instalare se poate importa în paginile în care este nevoie de cererile HTTP. Această librărie are suport pentru toate browserele importante şi funcţionează după aceleaşi reguli în fiecare pentru o consistenţă în răspunsuri.

Modul în care se fac cererile HTTP este intuitiv, astfel că pentru metodele de bază CRUD (POST, GET, PUT, DELETE) sintaxa axios arată astfel: "axios.post()" , "axios.get()" şi aşa mai departe. Configurările cererilor se pot face prin opţiunile adiţionale: headers, query parameters, request data şi timeouts.

În Figura 2.4 este sunt reprezentaţi paşii de funcţionare a librăriei Axios din momentul în care se face cererea până se primesc informaţile

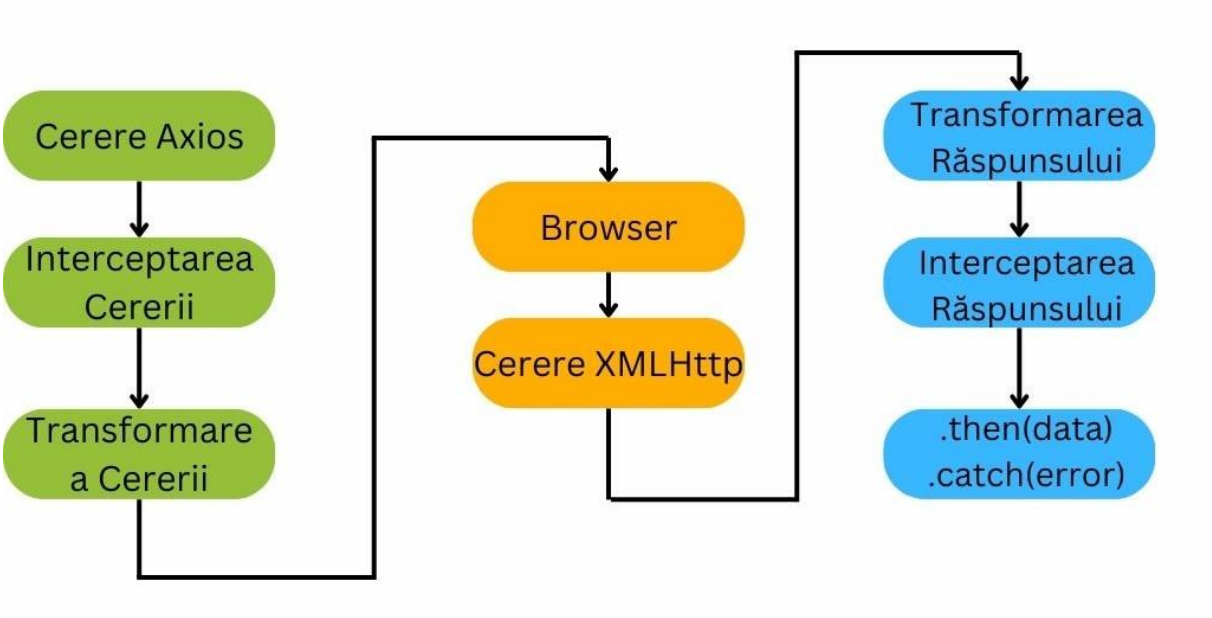

Figura 2.4 Modul de funcţionare Axios

2.10.2 Redux

Redux este o librărie JavaScript pentru managementul stării unei aplicaţii care se foloseşte în general cu framework-uri ca şi React sau Angular. Cu această librărie se creează un container pentru starea aplicaţiei în care se reţin diferite detalii cum ar fi datele utilizatorului curent pentru fiecare sesiune de pe fiecare dispozitiv unde funcţionează aplicaţia.

Store este singurul obiect în care se găsește starea aplicației care folosește Redux în orice moment, atât timp cât aceasta funcționează.

Actions sunt obiecte JavaScript simple care descriu schimbări pentru starea unei aplicații.

Reducers sunt funcții care se ocupă de actualizarea stării aplicației în funcție de informația primită din Actions.

Dispatch se ocupă de procesul de trimitere a unei Action pentru a actualiza starea aplicației prin apelarea unui Reducer.

Selectors sunt funcții care au rolul de a obține informații specifice din store-ul Redux. Aceste funcții pun la dispoziția programatorului un mod structurat de a accesa informația oferind posibilitatea de a accesa și informațiile derivate din cele de bază având opțiunea de a accesa doar anumite părți cheie.

Librăria Redux se bazează pe trei principii fundamentale:

- Redux stochează starea întregii aplicații într-un singur obiect JavaScript numit "store". Acest obiect store asigură faptul că toate componentele aplicației au acces la aceleași informații în orice moment, astfel se asigură protejarea împotriva duplicării informațiilor și accesul rapid la acestea.
- Starea aplicației în Redux este invariabilă, asta însemnând că nu poate fi modificată în mod direct. Pentru a se putea interacționa cu starea aplicației există așa-numitele "Actions" care descriu schimbările de stare necesare, după "Reducers" gestionează aceste acțiuni creând o stare nouă care se bazează pe acțiunile trimise și starea precedentă.
- Reducers sunt doar niște funcții care primesc starea trecută a aplicației și un Action ca și parametrii de funcție, după se returnează o nouă stare a aplicației. Astfel starea aplicației este ușor de testat și este predictibilă deoarece aceste funcții trebuie să returneze aceeași ieșire pentru aceeași intrare [18]

2.11 MySQL

MySQL este o bază de date relațională open-source fiind foarte folosită pentru a stoca informație structurată. Oferă un sistem robust, scalabil și flexibil pentru o gamă largă de tipuri de aplicații. Limbajul SQL (Structured Query Language) stă la baza acestui sistem de baze de date. [19]

O bază de date relațională înseamnă că informația este organizată sub formă de tabele cu linii si coloane. Stabilirea relațiilor între tabele se realizează prin intermediul cheilor oferind integritate informațiilor stocate.

MySQL are suport pentru stocarea a diferite tipuri de date: numeric, string, dată. Această bază de date dispune de mai multe motoare de stocare cum ar fiȘ MyISAM sau InnoDB fiecare având avantaje și dezavantaje.

În MySQL se pot realiza toate operaţiile de bază CRUD primite de la un server cum ar fi: inserare, actualizare, citire, ştergere şi multe altele, dispunând şi de operaţiile SQL de subinterogări, alăturări, agregări şi sortări pentru manipularea datelor într-un mod eficient. Pentru a putea folosi aceste metode este nevoie de unul sau mai multe query-uri. Acestea sunt comenzi dedicate în general bazelor de date pentru a manipula informaţia implementând metodele de mai sus. Un query se construieşte din una sau mai multe clauze care specifică ce operaţie trebuie executată şi care sunt criteriile pentru obţinerea informaţiei respective. Cele mai folosite instrucţiuni sunt: „SELECT" este folosită pentru a obţine informaţie din una sau mai multe baze de date şi se poate specifica numele coloanelor de care este nevoie, "FROM" indentifică tabelul sau tabelele din care informaţia este extrasă, "WHERE" această clauză este folosită pentru a filtra informaţia extrasă în funcţie de criteriile dorite, "INSERT" este folosită pentru a adăuga noi informaţii într-un tabel, specificând numele tabelului şi valorile care urmează să fie inserate, "DELETE" este folosită pentru a şterge informaţiile din baza de date, specificând condiţiile pentru a determina ce se doreşte pentru ştergere.

Securitatea în această bază de date este oferită de posibilitatea de a crea conturi pentru utilizatori multiplii, fiind nevoie de a te autentifica înainte de a accesa baza de date. Fiecare utilizator are diferite privilegii şi nivele de acces, fiind sigur că doar cine are autorizarea potrivită poate sa execute anumite acţiuni. În plus pentru a securiza informaţia aflată în baza de date MySQL are suport pentru mecanisme de encriptare a informaţiei când este stocată dar şi în momentul în care se află în tranzit. Există posibilitatea de a encripta comunicarea folosind protocoale SSL/TLS. Astfel chiar dacă apare acces neautorizat, informaţia rămâne criptată fără a avea cheia necesară pentru decriptare. [20]

Scalarea bazei de date MySQL funcţionează pe verticală, însemnând că prin adăugarea mai multor resurse hardware se poate mări capacitatea, sau pe orizontală prin distribuirea informaţiei între servere multiple.

Indexarea îmbunătăţeşte performanţa query-urilor deoarece aceştia activează posibilitatea de a primi informaţiile din tabele mult mai rapid. MySQL oferă suport pentru mai multe tipuri de indexare cum ar fi: hash, graf sau text în funcţie de cerinţele necesare pentru query-uri.

3 Analiză, proiectare, implementare, testare și implementare

3.1 Analiza

Această aplicație este dedicată în cea mai mare parte pacienților, în special pacienții care locuiesc într-o localitate diferită de cea în care se află medicul de familie. Prin intermediul aplicației se simplifică procesul prin care un pacient poate să intre în posesia unei trimiteri medicale. Cu această aplicație se dorește îndeplinirea următoarelor obiective:

- Posibilitatea tuturor utilizatorilor de a crea un cont securizat cu diferite nivele de acces.
- Pentru medici și farmaciști aprobarea înregistrării în contul creat doar după ce datele personale sunt confirmate de instituția de care acesta aparține.
- Chiar după înregistrare, pacientul poate să își actualizeze contul cu datele medicului de familie de care aparține, având posibilitatea de a schimba aceste date în orice moment dorește.
- Medicul de familie are posibilitatea să vizualizeze pacienții de care aparține și poate crea trimiteri medicale pentru aceștia, trimiterile fiind deja completate cu datele personale a pacientului selectat.
- Pacientul își poate vizualiza trimiterile creeate cu numele său și le poate atașa într-o programare creată în prealabil către medicul specialist.
- Medicul specialist poate să vizualizeze trimiterile primite de la pacienți, acestea fiind ordonate crescător după data și ora programării.
- În funcție de trimitere și de investigația pe care o face medicul specialist poate să creeze o prescriere în care specifică medicamentele de care pacientul are nevoie
- Pacientul poate să-și vizualizeze prescrierile medicale în cont la secțiunea „Prescrieri”.
- Pacientul poate să aleagă una dintre farmaciile prezente, adresa acesteia fiind afișată, în aplicație și să-și trimită prescrierea.
- Farmacistul vizualizează prescrierile primite și poate să pregătească medicamentele. După poate să notifice pacientul că medicamentele sunt pregătite, specifică prețul, ora recomandată de ridicare pentru a nu creea aglomerație și poate să ofere detalii suplimentare pentru modul de administrare
- Pacientul poate să revină oricând să vizualizeze indicațiile primite de la farmacist.

Scopul general al acestei aplicaţii este acela de a facilita tot procesul care porneşte de la nevoia unei investigaţii făcută de un medic specialist, pentru o anumită problemă de sănătate. Astfel un medic de familie poate să elibereze o trimitere mult mai uşor, fiind salvată în format electronic, astfel se uşurează modul în care pacientul poate să îşi primească trimiterea de la distanţă.

Această parte a procesului desfăşurându-se în mediul online şi trimiterea fiind în format electronic este necesar ca şi următoarea parte a procesului să se desfăşoare la fel. Astfel şi prescrierile care vin de la medicul specialist se vor crea în format electronic.

Prin această aplicaţie se digitalizează tot procesul care porneşte de la medicul de famile care creează o trimitere şi se termină în momentul în care farmacistul notifică pacientul că medicamentele de care are nevoie sunt pregătite.

Formularele de trimiteri şi de prescrieri sunt construite astfel încât să conţină toate câmpurile necesare din formularele de trimitere şi prescriere din format fizic. Întregul proces realizat în această aplicaţie se mulează după procesul de trimiteri şi prescrieri deja existent din sistemul medical din România.

3.2 Proiectare

Această aplicaţie web are la bază arhitectura de proiectare RESTful. La bază arhitectura este orientată pe modul de gestionare a resurselor, fiecare resursă are un identificator unic URI şi poate să fie accesat şi manipulat prin metodele standard din HTTP. Resursele sunt reprezentate de utilizatori, trimiteri, prescrieri sau de notificările farmaciştilor.

Această arhitectură se foloseşte pentru construirea de servicii web. Se bazează pe principiile arhitecturale “Representational State Transfer” (REST) care se folosesc de protocoalele şi standardele web existente, resursele sunt identificate de adrese URL unice şi clienţii interacţionează cu aceste resurse folosind standardul HTTP prin metodele de bază CURD (GET, POST, PUT, DELETE).

Astfel interfaţa, partea pe care o vede orice utilizator care foloseşte aplicaţia, comunică cu serverul prin aceste metode din standardul HTTP. În acest mod se asigură separarea dintre client şi server, acestea nu pot comunica în mod direct ci doar prin intermediul metodelor definite cu acest scop.

În Figura 3.1 sunt reprezentate componentele dintr-o aplicaţie construită cu arhitectura RESTful.

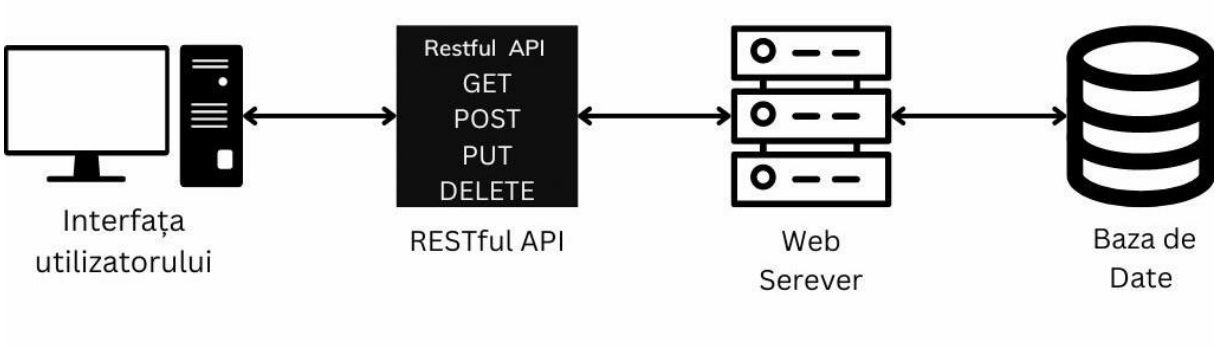

Figura 3.1 RESTful API

Arhitectura folosită pentru realizarea părţii de server este MVC (Model-View-Controller). După cum îi spune şi numele această arhitectură împarte aplicaţia în trei componente: Model (modele), View (interfeţe) şi Controller (controlori). Modelele reprezintă datele şi logica aplicaţiei, tot aici se implementează şi accesul la date.

În această componentă se defineşte modelul utilizatorilor, pacienţi, medici, farmacişti, administratori dar şi a trimiterilor, prescrierilor şi programărilor împreună cu proprietăţile acestora cum ar fi rolurile, numele sau instituţia din care fac parte. Tot în această componentă se creează şi serviciile pentru fiecare din aceste componente, servicii unde sunt definite metodele prin care se poate interacţiona cu proprietăţile entităţilor.

Componeta View în această aplicaţie reprezintă antetul metodelor create pentru interacţionarea cu resursele. Acestea nu conţin deloc implementarea metodelor şi pot să fie doar accesate din componenta controller care foloseşte metodele.

Componenta Controller se ocupă de interacţiunea dintre Model şi View. Aceasta primeşte datele utilizatorului prin intermediul componentei View şi actualizează componenta Model în funcţie de această intrare.

Arhitectura părţii clientului se bazează în mare parte pe tiparul Flux care este implementat prind Redux. Această arhitectură se bazează pe acţiuni care declanşează schimbări într-o componentă centrală a aplicaţiei numită “store”. Schimbările care apar în interfaţa utilizatorului se actualizează în funcţie de schimbările care apar în această componentă centrală. Aceste schimbări în store constau în cereri HTTP ale clientului către server, deoarece întreaga aplicaţie este construită pe baza arhitecturii RESTful.

În Figura 3.2 sunt reprezentate componentele prin care librăria Redux stochează starea aplicaţiei şi legăturile dintre aceste componente.

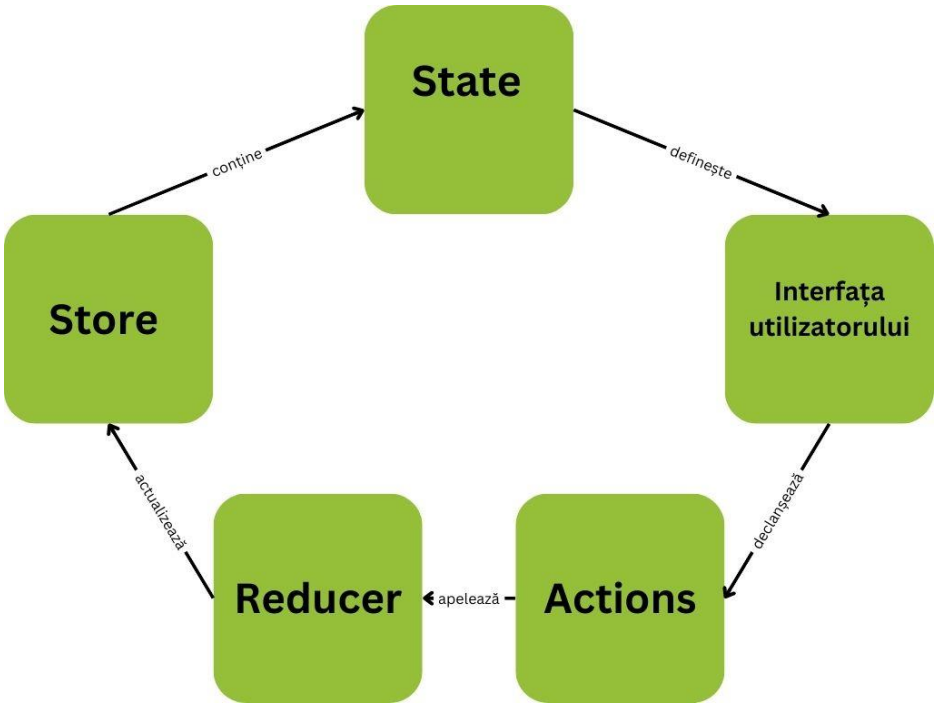

Figura 3.2 Librăria Redux

3.3 Proiectare

Pentru a se putea pune baza aplicației am realizat următoarele diagrame care au ajutat la conceperea funcționalității: diagrama claselor, diagrama cazurilor de utilizare, diagrama secvențială și diagrama bazei de date unde sunt reprezentate tabele și relațiile dintre acestea.

3.3.1 Diagrama cazurilor de utilizare

Diagrama cazurilor de utilizare este una dintre cele mai importante diagrame realizate pentru o aplicație deoarece face posibilă vizualizarea opțiunilor pe care un utilizator le poate face prin intermediul aplicației. Astfel se pot observa caracteristicile care s-au dezvoltat în aplicație pentru fiecare tip de utilizator în parte. [21]

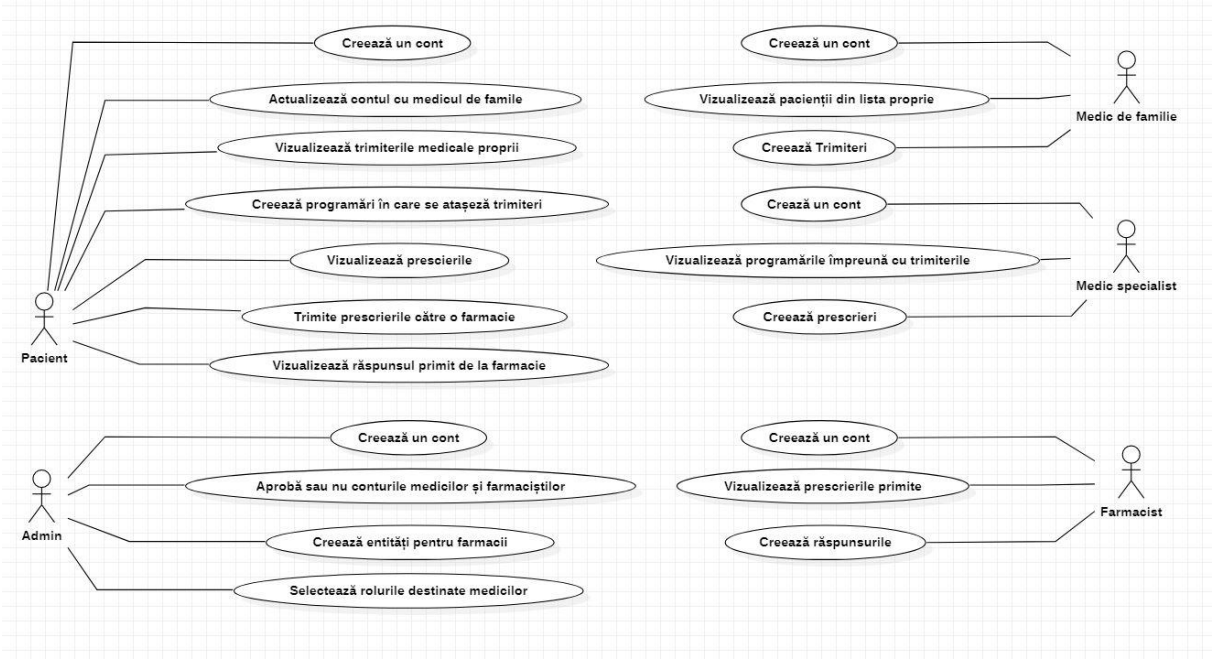

Figura 3.3 Diagrama cazurilor de utilizare

În această diagramă se poate observa că fiecare tip de utilizator poate să îşi creeze un cont. Medicii în funcţie de rol pot să creeze trimiteri sau prescrieri în funcţie de rolul pe care îl au şi pot să vadă o listă cu pacienţii, respectiv o listă cu programări. Se poate observa că pacientul are cele mai multe opţiuni. După ce îşi creează un cont acesta poate să îşi actualizeze contul pentru a adauga medicul de familie, poate să vizaulizeze trimiterile ptrimite de la medicul de familie, poate să creeze programări cu aceste trimiteri, poate să vizualizeze prescrierile primite în urma unei investigaţii, aceste prescrieri se poate trimite către o farmacie şi poate să vizaulizeze răspunsul primit în momentul în care medicamentele sunt pregătite. Administratorul poate să aprobe medicii şi farmaciştii care îşi creează un cont nou doar dacă instituţia din care fac parte confirmă că lucrează acolo şi atribuie roluri pentru medicii de familie sau pentru medicii specialişti. Farmaciştii pot să vizualizeze prescrierile primite de la pacienţi şi să creeze răspunsuri pentru aceştia.

3.3.2 Diagrama secvenţială

Diagrama secvenţială este un tip de diagramă UML (Unified Modeling Language) care descrie modul în care diferite componente, interfeţe şi obiecte din aplicaţie interacţionează între ele. Modul în care se descriu aceste interacţiuni este ordonat în modul cronologic de apariţia evenimentelor şi a secvenţelor de apelare a metodelor.

Componentele din aplicaţie sunt reprezentate ca fiind nişte blocuri cu o linie punctată care reprezintă linia de viaţă a componentei şi mesajele sau interacţiunile către aceste componente sub formă de săgeţi deasupra cărora este specificată acţiunea realizată. Informaţile primite sau răspunsurile care reprezintă succesul sau eşecul acţiunilor sunt reprezentate cu săgeţi cu linii punctate. Locul în care se realizează o astfel

de acţiune cum ar fi apelul unei metode este reprezentat sub formă de dreptunghi care arată durata unei astfel de metode. [22]

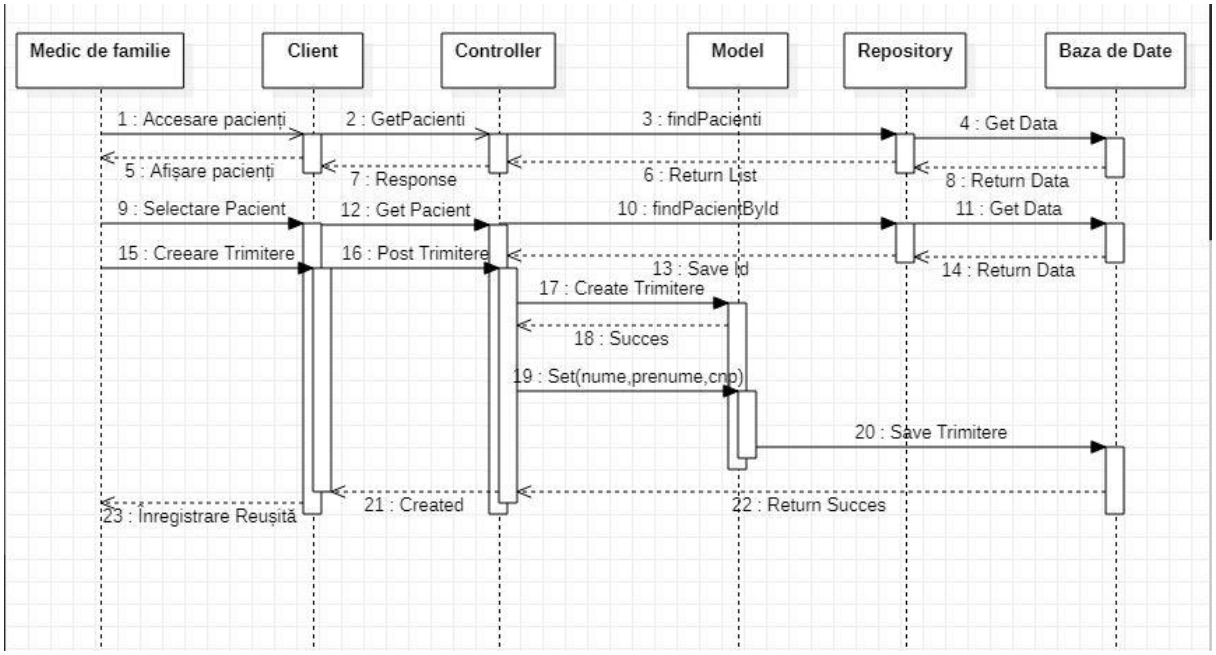

Figura 3.4 Diagrama secvenţială

3.3.3 Diagrama claselor

Diagrama claselor este una din diagramele folosite pentru pentru proiectarea aplicaţiilor şi descrie foarte bine modul în care componetele din server interacţionează între ele. Într-o astfel de diagramă se reprezintă structura logici din spatele aplicaţiei prin legăturile făcute între diferite clase şi interfeţe. [23]

Fiecare dreptunghi reprezintă o clasă, în partea de sus se află numele, în prima parte sunt toate proprietăţile pe care le conţine şi prin ce tip de date este reprezentată fiecare proprietate şi indetificatorul de acces. În a doua parte a dreptunghiului sunt reprezentate toate metodele care ţin de clasa respectivă, împreună cu parametrii fiecărei metode, cu tipul de date pe care le returnează metoda şi identificatorul de acces. Fiecare interfaţă este reprezentată de câte un cerc care respectă aceeleaşi norme de descriere la fel cu cele pentru clase.

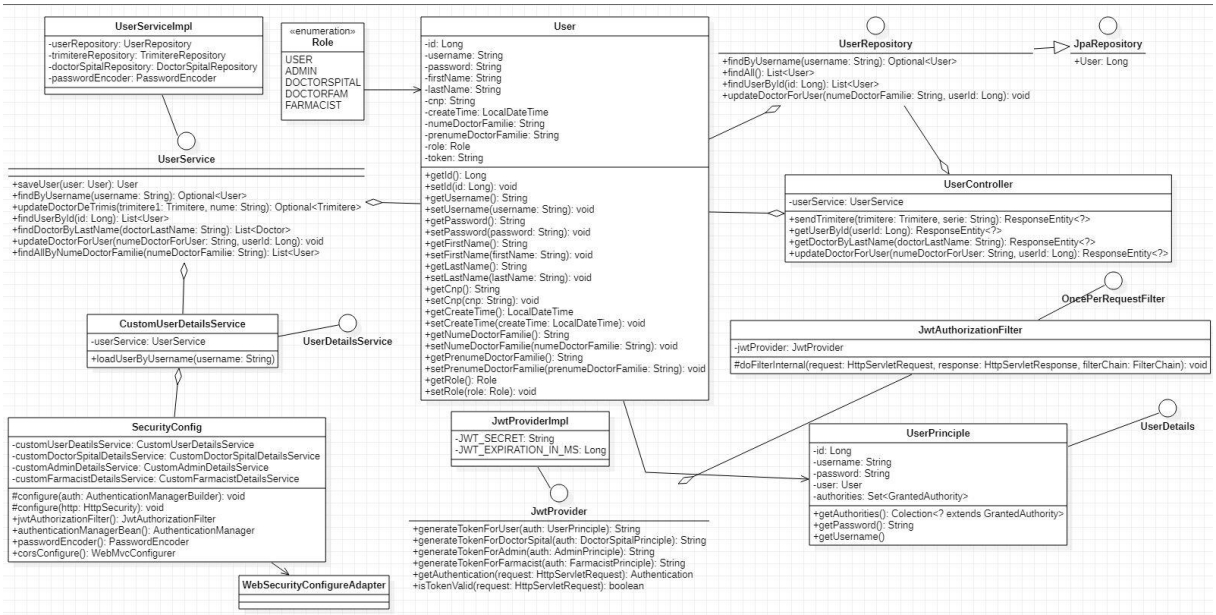

Figura 3.5 Diagrama claselor

3.3.4 Diagrama bazei de date

Diagrama bazei de date descrie modul în care baza de date dintr-o aplicaţie este structurată. Asemănătoare cu diagrama claselor aici se reprezintă entităţile descrise în componenta server dar sub formă de tabele deoarece astfel sunt reprezentate într-o bază de date relaţională. [24]

În partea de sus a fiecărui tabel se află numele, după urmează toate atributele entităţilor. Pentru fiecare proprietate se specifică tipul de date, în această bază de date există tipul „BIT” pentru proprietăţile care pot să aibă o valoare adevărată sau falsă, „VARCHAR” care este un tip care suportă text şi numere şi „BIGINT” care suportă doar numere dar de dimensiuni mari.

Fiecare tabel are un atribut reprezentat cu o cheiţă, aceasta arată cheia primară care este identificatorul unic din fiecare tabel, în toate tabelel din această aplicaţie cheia primară este reprezentată de proprietatea „id”.

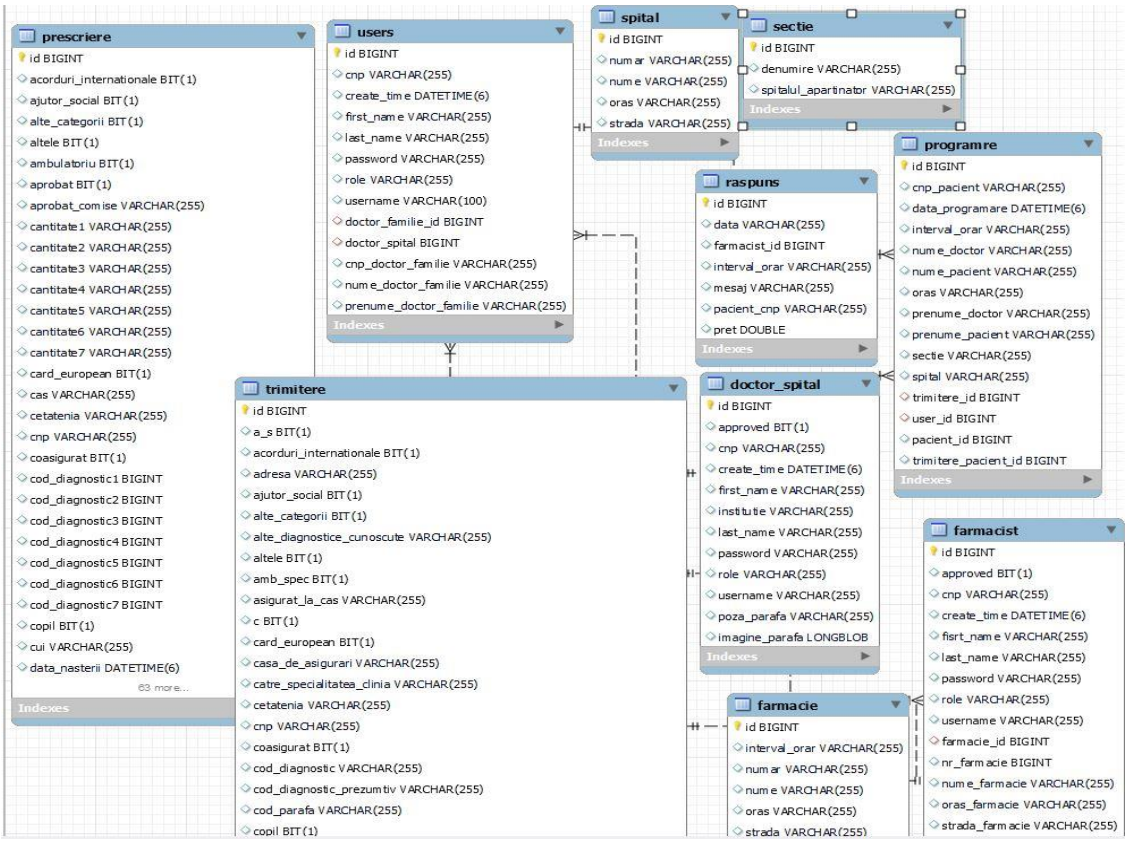

Figura 3.6 Diagrama bazei de date

3.4 Implementare

Aplicaţia web care se ocupă de digitalizarea sistemului, care se ocupă de trimiterile şi prescrierile medicale în format electronic şi de folosirea acestora prin intermediul internetului s-a folosit mediul de dezvoltare Intellij IDEA pentru partea serverului şi Visual Studio Code pentru partea de client.

Întreaga aplicaţie se bazează pe arhitectura RESTful care separă codul în două mari componente, Clientul, care este destinat utilizatorului şi se ocupă de interfaţa acestuia şi Serverul, care implementează logica de lucru cu informaţiile.

Componenta serverului este implementată cu limbajul de programare Java şi prin ajutorul framework-ului Spring Boot care are la bază arhitectura Spring MVC. Această arhitectură împarte codul din server în trei componente model, view şi controller, fiecare cu un rol bine definit pe care îl are de îndeplinit.

Baza de date este implementată folosind serverul MySQL iar schimbările făcute în baza de date s-au observat prin programul MySQL Workbench, limbajul folosit pentru baza de date este SQL. Pentru comunicarea cu baza de date în server există o componentă numită Repository.

Componenta care se ocupă cu interfaţa utilizatorului este implementată cu ajutorul framework-ului React care se bazează pe o librăria Redux. Această librărie se ocupă de

administrarea stării aplicaţiei în momentul în care este folosită de către un utilizator. Limbajul de programare folosit este JavaScript pentru a adăuga funcţionalitate componentelor dintr-o pagină web, limbajele folosite pentru construirea interfeţei sunt HTML şi JSX, acest limbaj din urmă este o combinaţie între HTML şi JavaScript cu ajutorul căruia se construiesc componente menite să implementeze funcţionalitatea dată de JavaScript. Limbajul CSS este folosit pentru stilizarea şi poziţionarea componentelor creeate înainte în paginile folosite. Această componentă comunică cu serverul prin intermediul protocolului HTTP.

În figura 3.7 este sunt reprezentate framework-urile prin care s-a construit fiecare componentaă aplicaţie şi modul în care comunică acestea între ele.

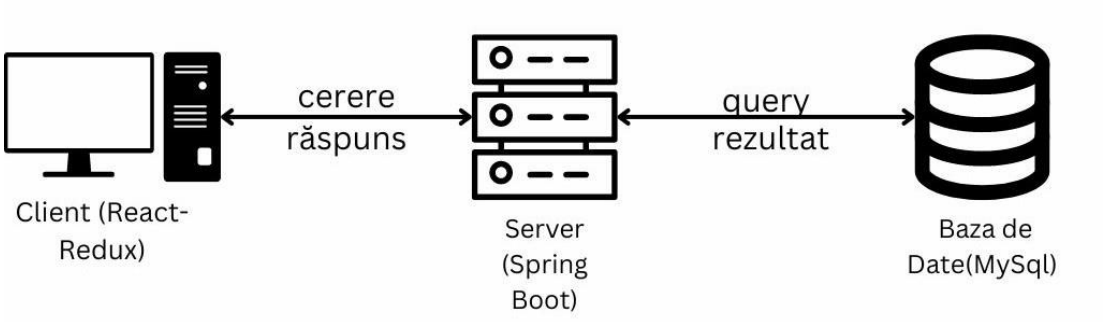

Figura 3.7 Arhitectura aplicaţiei

3.4.1 Serverul

Într-o aplicaţie web construită cu arhitectura RESTful, serverul se referă la componenta back-end care se ocupă de cererile clienţilor, procesarea de date şi generarea de răspunsuri pebtru client.

Pentru a putea face codul scris în limbajul Java din componenta de back-end s-a folosit Apache Tomcat. Acesta este un server web şi un container servlet deschis publicului larg/ open source care este foarte folosit pentru a ţine o aplicaţie bazată pe limbajul Java pentru a putea fi folosită dintr-un server. Servlets sunt clase Java care se ocupă de cererile care vin de la client şi generează un răspuns dinamic. Tomcat gestionează ciclul de viaţă a acestor clase, incluzând iniţializarea, tratarea cererilor şi alocarea resurselor. Tomcat funcţionează ca un server şi web poate să trateze cereri HTTP în mod direct.

Tomcat oferă un mod direct pentru implementarea şi configurarea aplicaţilor web de care trebuie să se ocupe. Aplicaţiile pot să fie implementate prin copierea fişierelor WAR (Web Application Archive) în directorul potrivit. Tomcat detectează automat şi implementează aceste aplicaţii. Pentru configurare instrucţiunile se scriu în fişierele server.xml şi web.xml. Aici se pot oferii detalii despre conectori şi securitate. Prin securitatea oferită de Tomcat s-au implementat roluri care permit accesul la diferite metode.

Pentru construirea proiectului în partea de back-end s-a folosit tool-ul Apache Maven. Acest tool este folosit pentru a construi automat codul proiectului şi pentru a

administra dependinţele proiectului. Oferind un model standardizat pentru construirea proiectelor se realizează un mod organizat de a compila codul sursă, efectua testele necesare şi pentru folosirea dependinţelor. Conceptul pe care se bazează acest tool este Project Object Model (POM), este un fişier XML care este folosit pentru configurarea proiectului. În el se află instrucţiuni pentru Maven care arată modul în care proiectul să fie construit şi modul în care să fie administrat mai departe. În pom.xml există informaţii cu privire la id-ul proiectului, id-ul artefactului, versiunea şi tipul pachetului, în acest caz fiind JAR, astfel aceşti indicatori identifică în mod unic proiectul. În acest fişier se specifică şi versiunea framework-urilor folosită, cum ar fi Spring Boot sau Spring Security şi conectorul pentru baza de date.

În Figura 3.8 sunt trei dependinţe din fişierul pom.xml, primele două se ocupă de folosirea token-urilor JWT şi al treilea de conectarea cu baza de date MySql.

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.2</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Figura 3.8 Dependinţele pentru server

Componenta serverului se împarte în trei componente principale, Model, View şi Controller şi în doua componente secundare, Repository care comunică cu baza de date şi Security care se ocupă de securitatea metodelor şi asigură diferite nivele de securitate în funcţie de anumite roluri.

Prima componentă cu care s-a început este Model. Aici avem definite toate entităţile care fac parte din aplicaţia web. În Spring Boot o entitate se referă la o clasă Java care reprezintă obiectul cu informaţii. Pentru a semnaliza pentru Spring Boot care clase sunt entităţi trebuie să folosim adnotaţie @Entity deasupra numelui clasei. Cand Spring Boot găseşte această adnotaţie, construieşte programul astfel încât clasa să fie reprezentată ca un obiect de date persistent, acestea vor fi reprezentate în baza de date

sub formă de tabele. Adnotarea "@Entity" face parte din Java Persistance API (JPA), care oferă o mapare obiect relaţională (ORM) pentru a reprezenta obiecte Java în tabelele din baza de date relaţională. Pentru fiecare din aceste entităţi urmează adnotarea "@Table". Prin această adnotaţie cu atributul "name=" se specifică numele tabelului, în ghilimele, prin care urmează să fie reprezentată entitatea în baza de date astfel "@Table(name = "numele_entităţii)"". Primul atribut din orice entitate este identificatorul unic prin care proprietăţile unei entităţi se înregistrează în baza de date. Acesta are numele de id şi este de tipul "Long" pentru a putea stoca cât mai multe înregistrări unice în baza de date. Acest tip de date nu suportă numerele cu virgula şi este reprezentat pe 8 octeţi. Pentru a semnaliza în Spring Boot că acesta este un identificator unic se foloseşte adnotaţia "@Id", după aceasta urmează adnotaţia care descrie modul în care se generează aceste valori id.

Modul de generare este specificat cu adnotaţia "@GeneratedValue" care primeşte un parametru numit "strategy" care specifică tipul de generare. În acest caz s-a ales "@GenerateValue(strategy = GenerationType.IDENTITY)" arată bazei de date să genereze id-ul după o coloană identitate care se auto incrementează. Când o nouă line cu informaţii se inserează în baza de date valoarea id-ului nou creşte cu o unitate faţă de inserarea precedentă, prima inserare având valoarea unu, astfel se asigură că nu se vor repeta valorile id-urilor într-un tabel.

Pentru restul proprietăţilor dintr-o entitate se specifică cu adnotaţia "@Column" numele sub care proprietatea respectivă se va înregistra în baza de date, de exemplu pentru "@Column(name = "nume")" vom aveam o coloană în baza de date care se numeşte "nume".

La fiecare proprietate dintr-o entitate se specifică numele, tipul de date (Long, Boolean sau String) şi modificatorul de acces "private". Acest modificator de acces restricţionează vizibilitatea şi accesibilitatea proprietăţilor. Modificatorul "private" face ca proprietatea să poată să fie accesată doar din aceeaşi clasă în care a fost declarată. În acest mod proprietăţile unei clase nu se pot accesa în mod direct menţinând integritatea datelor. Aceste proprietăţi se pot accesa sau modifica doar prin intermediul unor metode speciale care se numesc "Getters" pentru metodele de accesare şi "Setters" pentru metodele prin care se pot seta aceste proprietăţi.

Pentru a crea automat aceste metode de acces s-a folosit adnotaţia "@Data" deasupra numelui clasei. Această adnotaţie face parte din librăria Lombok şi generează metode de acces pentru toate proprietăţile unei clase pentru a putea fi accesate şi in exteriorul claselor în care au fost declarate.

Printre modelele entităţilor există şi un tip de date special numit Enum (Enumeration). Acest tip de date reprezintă un set de date fix de constante predefinite, în general este folosit pentru a definii un grup de date asemănătoare ca valori şi ca scop. Acest tip de date se poate defini asemănător cu o clasă, pentru a organiza şi a putea lucra cu o colecţie de valori constante. În această aplicaţie s-a folosit acest tip de date pentru a definii rolurile tuturor posibililor utilizatori. În acest Enum avem definite numele rolurilor utilizatorilor sub formă de colecţie de constante.

În componenta model există două tipuri de de entităţi, cele care se referă tipurile de utilizatori, cum ar fi Farmacistul sau Administratorul şi entităţile care se referă la

obiectele cu care se lucrează în interiorul aplicației cum ar fi trimiterea sau prescrierea medicală. Diferența majoră dintre aceste două entități sunt proprietățile pe care le conțin, cum ar fi "Username" sau "Password" dar mai important proprietățile care stochează rolul utilizatorilor și de cele care stochează token-ul care reține toate detaliile de autentificare și de autorizare pentru fiecare utilizator.

Proprietățile care țin de rol au ca identificator de acces tot "private" dar tipul de date este cel al Enum-ului din care face parte, în acest caz "Role". Pentru acest tip de date este necesară o adnotație în plus față de cea în care specificăm denumirea coloanei după care găsim proprietatea în baza de date. În mod implicit informația pentru aceste roluri se salvează sub formă de numere, primul rol are valoarea unu, al doilea rol are valoare doi și așa mai departe. Pentru a putea fi mai ușor de interpretat și pentru a îmbunătăți lizibilitatea informațiilor stocate în baza de date s-a folosit adnotația "@Enumerated" în care sub formă de parametru se specifică tipul de date care se salvează, în acest caz este nevoie de numele rolurilor exact cum au fost specificate deci se folosește tipul de date String, astfel adnotația devine "@Enumerated(EnumType.String)".

În proprietatea token se stochează detalii despre utilizator cum ar fi username-ul, numele său rolul acestuia în aplicație, aceste informații sunt criptate cu ajutorul criptării Base64 și datele nu sunt vizibile cu ochiul liber. Este folosit pentru a se putea înregistra în contul personal și ca tip de date este un string. Informațiile din acest token expiră după o anumită perioadă de timp setată, acest fapt îmbunătățește securitatea utilizatorilor deoarece și dacă acest token se află pe lângă faptul ca trebuie decodat acesta va expira și se generează unul nou, utilizatorul fiind nevoit să se înregistreze cu noul token generat. Această proprietate nu trebuie salvată în baza de date deoarece se schimbă foarte des și conține informații care sunt deja salvate în baza de date dar sub alte nume dar și din motive de securitate.

Pentru a arăta că pentru o proprietate nu trebuie creată o coloană nouă se folosește adnotarea "@Transient". Această adnotație este folosită când avem nevoie de acces pentru anumite operații dar nu trebuie și salvată. Metodele de acces și de actualizare a proprietăților pot să acceseze această proprietate în mod normal fără restricții.

În continuare s-a creat componenta care se ocupă de accesul la informațiile din baza de date către întreaga aplicație. Această operație se realizează printr-un concept care se numește DAO (Data Access Object).

Acesta este un șablon, care se folosește pentru separarea logii de accesare a informațiilor de logica lucru a programului, oferă o cale modulară pentru interacțiunile cu baza de date. Beneficile acestui șablon este că încapsulează operațiile și query-urile specifice bazei de date, oferă aceeași interfață pentru diferite surse de informații, îmbunătățește testabilitatea, mentenanța și posibilitatea reutilizării codului și oferă suport pentru tratarea excepțiilor.

Modelul Repository este un nivel de abstractizare pentru șablonul DAO, acesta oferă o interfață la nivel mai înalt pentru interacțiunea cu informațiile din baza de date dar în același timp și simplifică aceste interacțiuni. Implicit Repository oferă un set de metode de bază CRUD (create, read, update, delete) pentru operațiunile cu baza de date. Aceste

metode de bază se creează printr-o convenţie pentru denumirea metodelor dintr-un repository.

Pentru a putea folosi acest model de interacţiuni, în program se creează interfeţe pentru fiecare entitate făcută înainte, pentru a organiza informaţiile şi metodele care ţin de câte o entitate. Spring Boot poate să recunoască aceste interfeţe ca fiind repository dacă folosim adnotaţia "@Repository" arătând că în aceste interfeţe se află metode pentru interacţiunea cu baza de date.

Metodele de bază CRUD se pot folosii în interfeţele create dacă extindem JpaRepository. Aceasta este o interfaţă din framework-ul Spring Data JPA care face parte din proiectul Spring Data. În această interfaţă avem metoda POST dacă creăm o metodă cu numele "save()", avem o metoda de GET în funcţie de cheia primară a unor entităţi prin numele "findById()" sau metoda DELETE după cheia primară prin metoda cu numele "deleteById()". Aceste metode abstractizează complexitatea acestor operaţii de bază şi se pot implementa dor prin numirea metodelor prin aceste reguli. Această interfaţă oferă şi posibilitatea de a folosii metode speciale pentru fiecare entitate care urmăresc aceleaşi reguli de numire şi pentru proprietăţi. Astfel dacă avem câmpul CNP în entitatea User putem să creăm o metodă "findByCnp", această metodă funcţionează ca un GET care filtrează utilizatorii în funcţie de ce se specifică în câmpul CNP.

Pentru a specifica tabelul din baza de date pentru care sunt folosite aceste metode, în JpaRepository se specifică numele entităţii dorite şi tipul de date folosit de cheia primară care în cele mai multe cazuri este Long.

În Figura 3.9 sunt reprezentate componentele serverului şi legăturile dintre acestea împreună cu legăturile cu componentele externe, baza de date şi browserului.

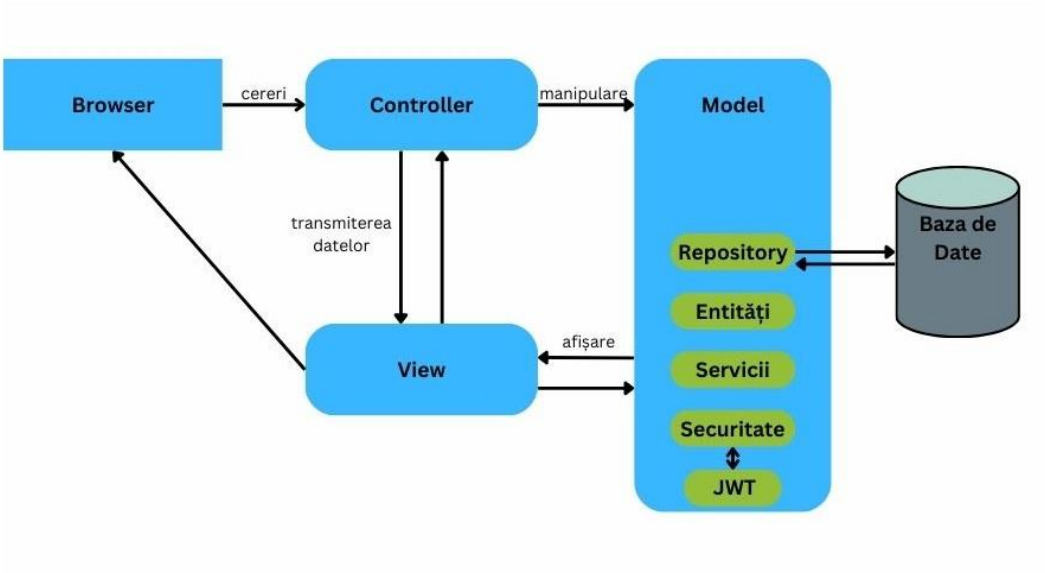

Figura 3.9 Componentele serverului

În momentul în care este nevoie de metode de interacţiune, cu informaţiile, mai avansate există adnotări "@Query". Cu ajutorul acestei adnotaţii putem să construim metode avansate cu limbajul JPQL (Java Persistance Query Language) sau limbajul nativ SQL. Metodele realizate în acest mod sunt mult mai specifice şi mai performante decât metodele de bază CRUD. În acest proiect s-au folosit astfel de metode în cea mai mare parte pentru funcţia de actualizare, pentru a putea actualiza câmpuri specifice în funcţie de cheia primară a utilizatorului curent. Parametrii folosiţi în acest limbaj trebuie declaraţi în antetul metodelor prind adnotaţia "@Param". Avem nevoie şi de adnotaţia "@Modifying" pentru metodele care realizează operaţia de actualizare semnalând astfel în Spring Boot că metoda schimbă anumite informaţii din baza de date.

Service este un nivel în care se află toate metodele care se ocupă cu logica de interacţiune a datelor din aplicaţie, aici se folosesc funcţiile definite în componenta repository, prin intermediul acestor funcţii, service are acces la informaţiile din baza de date. În această componentă se definesc funcţiile care conţin metodele de interacţiune cu informaţiile din baza de date. Service este un nivel care aparţine de componenta model din arhitectura MVC. Pentru a îmbunătăţii modularitatea şi refolosirea codului se creează acest nivel, fiind un intermediar între componenta controller şi nivelul de acces la informaţii, repository.

În general se creează un nou pachet pentru definirea claselor care compun nivelul service. La fel ca la repository şi aici există câte o clasă care este dedicată pentru fiecare entitate creată la început. În fiecare clasă există toate operaţiile complexe care se ocupă cu o anumită entitate. Crearea unui astfel de intermediar este benefică deoarece aici se efectuează validarea datelor, se aplică reguli şi constrângeri pentru a se asigura integritatea şi corectitudinea informaţiilor. În service se aplică şi politicile de securitate dar şi autentificarea şi autorizarea utilizatorilor. Pentru această aplicaţie există o clasă care se ocupă doar de autentificarea utilizatorilor în funcţie de entitatea de care aparţin. Această clasă implementează metodele de autentificare cu ajutorul token-urilor definite în nivelul de securitate al aplicaţiei. Folosind un nivel de service se uşurează gestionarea excepţiilor şi gestionarea erorilor deoarece aici se pot converti în mesaje care oferă mai multe detalii despre erorile apărute.

În momentul în care se creează o clasă care ţine de nivelul service aceasta trebuie sa aibă în componenţă adnotaţia "@Service", care semnalează framework-ului Spring Boot că aceasta este o clasă care se ocupă de logica de utilizare a datelor în interiorul aplicaţiei.

Pentru a putea folosi metodele deja definite în repository trebuie să importăm acea interfaţă. Spring Boot oferă o metodă specială de importare care se numeşte injecţia dependinţelor (dependency injection). Acesta este un model de proiectare şi un concept fundamental în dezvoltarea software care promovează cuplarea liberă între componentele unui program şi designul modular. Acest model implică oferirea dependinţelor unui obiect care se află într-o sursă externă. Astfel obiectul nu trebuie să îşi gestioneze dependinţele sau să le creeze, oferind o mai mare flexibilitatea şi posibilitate de reutilizare a codului. Folosirea acestui model se poate realiza printr-un constructor sau prin metoda recomandată de Spring Boot printr-o adnotaţie. Implementarea cu ajutorul adnotaţiei, seamănă cu declararea unei proprietăţi, avem modificatorul de acces, în acest

caz "private" pentru a nu putea interacționa cu alte clase în mod direct, după urmează tipul variabilei care ține de interfața de care avem nevoie, de exemplu interfața creată pentru interacțiunea cu proprietățile din entitatea farmacist și numele variabilei care de obicei este la fel cu cel al interfeței doar că începe cu literă mica. Deasupra se afla adnotația "@Autowired" folosită pentru injectarea dependințelor Cu ajutorul acestei variabile se pot apela metodele care aparțin de interfața respectivă în toată clasa în care a fost importată. O clasă care face parte din nivelul de service poate să conțină mai multe astfel de instanțieri care realizează injectarea dependințelor.

În general fiecare clasă din service are mai multe metode, fiecare având diferite seturi de instrucțiuni. Pentru a putea organiza mai ușor aceste metode, pentru a se putea observa de ce tip este fiecare și de ce tipuri de parametrii este nevoie pentru implementarea în componenta controller, se creează câte o interfață pentru fiecare clasă din service. Astfel în aceste interfețe avem doar antetul metodelor și în controller importăm metodele necesare din interfețe și nu din clase. Legătura dintre interfață și clasă se face prin cuvântul cheie "implements" (implementează) care specifică programului că metodele declarate în interfață sunt implementate în clasa respectivă. Pentru a realiza această modularizare fiecare metodă din clasele de implementare a serviciilor conține adnotația "@Override". Această adnotație arată că metoda respectivă se suprascrie din interfață, acolo unde există doar antetul metodei.

În aceeași manieră ca la celelalte componente și pentru componenta controller se definește câte o clasă separată pentru fiecare entitate din model care are nevoie de interacțiuni cu clientul. Componenta Controller există pentru a face legătura între interfața utilizatorului și celelalte componente ale aplicației cum ar fi model sau service. Controller-ul primește cereri de la utilizator prin intermediul interfeței și acesta procesează aceste cereri după care apelează metodele necesare pentru procesarea datelor din nivelul de service.

Pentru crearea unei clase din componenta controller este nevoie de a specifica pentru framework-ul Spring Boot că face parte din controller printr-o adnotație. „@RestController" este o variantă îmbunătățită a adnotației „@Controller'' adăugând și funcționalitatea adnotației „@ResponseBody". Astfel, această adnotație specializată este perfectă pentru crearea de servicii web pentru arhitectura RESTful deoarece nu mai trebuie ca fiecare metodă din controller să conțină adnotația „@ResponseBody" pentru răspunsul din aplicație și este destul să existe adnotația „@RestController" pentru întreaga clasă.

Într-o astfel de clasă controller se primește cererea HTTP de la un client și se returnează un răspuns potrivit înapoi către client. Aceste cereri pot să fie de mai multe tipuri cum ar fi: Get, Post, Put și Delete. Controller-ul apelează pentru fiecare, împarte metoda de care este nevoie cu tipul necesar. Răspunsurile care se returnează de la aceste metode sunt transformate automat în formatul JSON (JavaScript Object Notation), format standardizat care se folosește pentru transmiterea informațiilor între server și client, acesta fiind și marele avantaj pentru adnotația „@RestController'', nemaifiind nevoie ca programatorul să specifice transformarea necesară.

O altă adnotație necesară pentru crearea unei clase în componenta controller este „@RequestMapping". Această adnotație este folosită pentru a mapa cererile web către metodele specifice dintr-o clasă care face parte din componenta controller. Prin intermediul adnotației se poate defini un șablon URL pentru a se putea apela setul de metode respectiv doar de către cererile care respectă acest șablon URL. În acest șablon se pot defini și parametri necesari pentru funcționarea metodei sau headers pentru autorizarea utilizatorului care dorește să folosească metoda.

La fel ca la nivelul service este nevoie de injectarea dependințelor. Tot prin intermediul adnotației „@Autowired" se realizează această operație dar aici este nevoie de o instanță a interfeței de servicii pentru apelarea metodelor necesare.

În continuare se definesc metodele pe care clientul le are la dispoziție pentru a interacționa cu datele din aplicație. Pentru fiecare metodă trebuie specificat tipul, în funcție de operațiile de bază CRUD. Pentru cererea HTTP Get avem adnotația „@GetMapping", pentru Post avem „@PostMapping" și la fel pentru fiecare metodă. Pentru fiecare din aceste metode se poate specifica un URL pentru accesarea acestora în care se pot include și parametri necesari pentru apelarea corectă a metodelor. Pentru fiecare parametru specificat în URL trebuie să existe un parametru specificat și în antetul metodei. Un parametru se specifică cu numele acestuia și trebuie să coincidă cu numele specificat în URL. Tipul acestuia cum ar fi: Long, String, Object și pentru a se specifica în Spring Boot că este un parametru și că este legat de parametrul specificat înainte în URL se folosește adnotația „@PathVariable".

În funcție de necesitățile metodei poate să fie nevoie și de alte adnotații în antetul metodei. „@AuthenticationPrincipal" este folosită în aplicație pentru a accesa utilizatorul autentificat în aplicație, astfel se pot obține informații despre utilizator fără a fi nevoie de accesarea obiectului de autentificare sau de „@RequestBody" care se folosește pentru a indica că parametrul metodei trebuie să fie legat de corpul unei cereri HTTP, astfel se face automat conversia informației în formatul JSON.

La fel ca în clasele care aparțin de service, mai departe se definește tipul de date al metodei, numele și corpul propriu-zis în care se prelucrează informația. Pentru aceste metode care interacționează direct cu clientul este nevoie de un tip special pentru metode care se numește „ResponseEntity". Mai exact aceasta este o clasă din Spring Boot și reprezintă întregul răspuns HTTP, incluzând codurile de status, headers și response body. Deoarece aceste metode trebuie să interacționeze cu interfața utilizatorului este nevoie de astfel de clase care folosesc întregul răspuns HTTP. Pentru a putea folosi orice tip de date pentru corpul răspunsului fără a fi nevoie să fie specificat cu exactitate, există posibilitatea de a fi declarat astfel: „ResponseEntity<?>". În acest mod se îmbunătățește productivitatea pentru că se poate folosi direct orice tip de date. Tipul returnat de metode implicit trebuie să fie tot ResponseEntity dar pot să conțină orice tip primitiv de date, obiecte sau colecții de date. În funcție de ce valori returnează metoda se pot seta diferite răspunsuri.

În general aceste metode trebuie sa fie construite cât mai simplu posibil și partea complexă de logică să fie conținută doar la nivelul de service.

Nivelul de securitate este unul din cele mai importante din aplicaţie deoarece conturile utilizatorilor conţin date personale importante cum ar fi CNP-ul. Securitatea în această aplicaţie este construită prin intermediul framework-ului Spring Security. Acesta oferă posibilitatea de a folosii tokenuri pentru sesiuni securizate în aplicaţie.

JWT (JSON Web Token) este un mod compact de transmitere securizată a informaţiei între două componente, în acest caz între server şi client, ca obiect de tip JSON. De cele mai multe ori este folosit pentru autentificare şi autorizare în aplicaţiile web.

Aceste token-uri sunt structurate în trei părţi, fiecare separate cu „.", header, conţinutul şi semnătura. Toate aceste componente sunt encodate cu Base64 pentru o securitate îmbunătăţită. În header se află tipul token-ului, în acest caz jwt şi algoritmul folosit pentru partea de semnătură. Cu ajutorul acestei componente se specifică cum se validează şi cum trebuie procesat acest token. În partea de conţinut se află toate datele utilizatorului, numele, id-ul şi rolurile acestuia prin intermediul cărora funcţionează permisiunile din program. Semnătura este creată dintr-o combinaţie dintre header, conţinut şi o cheie secretă. Prin această componentă se asigură integritatea token-ului şi asigură informaţiile aflate în conţinut.

În momentul în care un utilizator intră în cont se apelează interfaţa JwtProvider din pachetul de securitate pentru a genera un token nou. Metodele acestei interfeţe sunt implementate într-o clasă cu acelaşi nume.

În clasa care generează aceste token-uri există două variabile, prima este un string care se numeşte „jwt_secret", unde prin intermediul adnotaţiei „@Value()" se injectează o valoare din fişierul de proprietăţi ale aplicaţiei unde este specificată cheia pentru criptare şi decriptare. Această aplicaţie foloseşte encodarea HMC care foloseşte aceeaşi cheie şi pentru criptare şi pentru decriptare. A doua variabilă din clasă este de tipul Long şi la fel ca înainte se importă o valoare din fişierul de proprietăţi ale aplicaţiei. Aici se reţine o valoare care este reprezentată în sistem în milisecunde şi reprezintă perioada după care token-ul expiră şi nu mai poate să fie folosit, după această perioadă utilizatorul trebuie sa reintre în cont pentru a putea folosii funcţionalitatea aplicaţiei. Este necesar un timp de expirare pentru token pentru a spori securitatea şi pentru a împiedica folosirea acestuia în cazul în care a fost obţinut. În mod normal decodarea acestui token este imposibilă deoarece cheia necesară decodării se află în componenta server şi nu există acces direct spre această componentă.

Pentru entităţile care au nevoie de un cont, există clase numite Principle, acestea implementează interfaţa UserDetails care face parte din Spring Security. Aici există metode care se ocupă de parolă, username, autoritatea pe care o poate avea un tip de cont în aplicaţie şi alte detalii despre utilizator. Asemănător claselor din nivelul service, aici la nivelul de securitate există clase de service şi pentru informaţiile despre utilizator. Aceste clase folosesc adnotaţia „@Service" şi implementează interfaţa UserDetailsService care face parte din Spring Security. Singura metoda de implementat din această interfaţă este „loadUserByUsername" şi primeşte ca parametru username-ul utilizatorului. Această metodă caută utilizatorul în baza de date şi returnează un „UserPrinciple" care este construit cu şablonul „Builder". Aici sunt integrate informaţii despre utilizator cum ar fi parolă, id, username şi nivelul de autoritate pe care îl are în aplicaţie.

În Figura 3.10 este reprezentat modul în care Spring Security gestionează resursele aplicaţiei în funcţie de autentificarea şi de autorizarea utilizatorilor care doresc să le folosească.

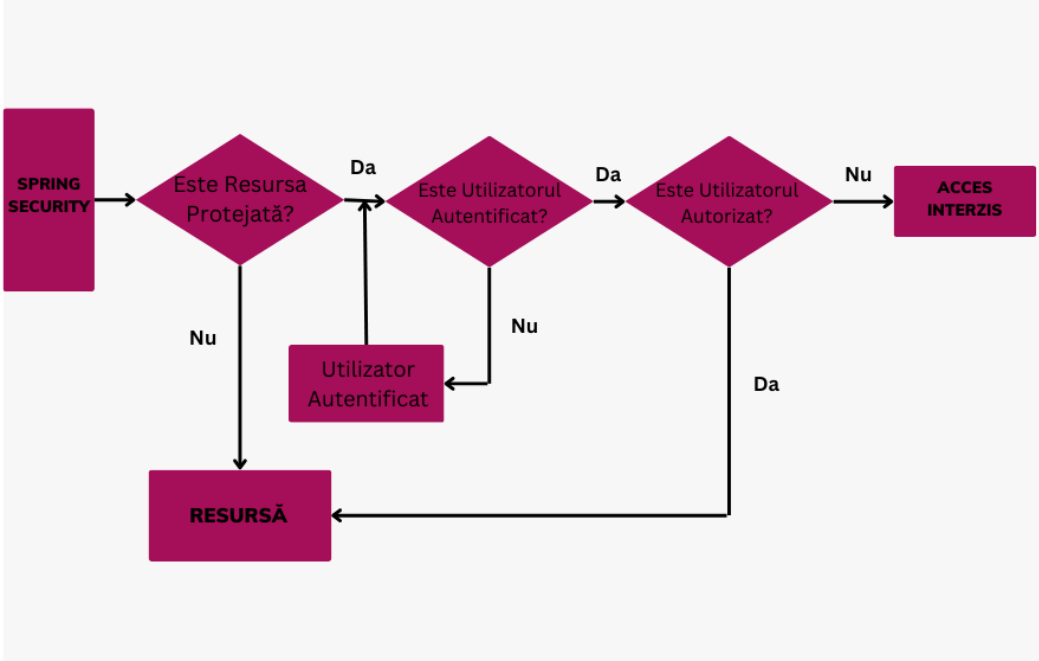

Figura 3.10 Logica Spring Security

Toate cererile HTTP sunt interceptate de clasa „JwtAuthorizationFilter" pentru a autoriza aceste cereri în funcţie de token-ul jwt pe care îl conţin. Această clasă extinde superclasa „OncePerRequestFilter". Aici se verifică token-urile din aplicaţie dacă sunt valide. Dacă îndeplinesc condiţiile se adaugă în obiectul de autentificare şi se pune în contextul oferit de Spring Security unde se află toate token-urile cu sesiuni active. Cand un token expiră acesta este şters din context şi nu mai este cunoscut în interiorul aplicaţiei.

3.4.2 Client

Utilizatorul interacţionează în mod direct cu componenta client, astfel trimiţând anumite cereri către server pentru a crea seturi noi de informaţii, pentru a schimba unele informaţii sau pentru a primi informaţii deja existente din baza de date. Componenta client în cele mai multe cazuri este redată prin intermediul unui browser web.

Această componentă a fost creată în editorul de cod Visual Studio Code cu ajutorul limbajului de programare JavaScript, cu HTML, limbaj folosit pentru construirea paginilor web şi CSS, limbaj folosit pentru stilizarea componentelor din paginile web. O interfaţă web este constituită din tot ce poate să vadă utilizatorul pe ecran în momentul în care foloseşte aplicaţia. Obiectele cu care poate să interacţioneze în interfaţă un utilizator sunt în general butoane, selectarea opţiunilor dintr-un dropdown sau selectarea anumitor checkbox-uri. Toate aceste interacţiuni creează o acţiune în componentele din interfaţă

care mai apoi se transmite sub formă de cerere HTTP pentru server de la care se așteaptă un răspuns pentru a fi redat pe ecran.

Framework-ul pe baza căruia s-a realizat această componentă este React. Acest framework este folosit în general pentru crearea interfețelor în aplicațiile web. Arhitectura unei interfețe construite în react este bazată pe componente. Aici interfața este divizată în componente care conțin propria logică de construire a unei pagini, au o stare proprie în raport cu restul componentelor și se pot refolosi cu diferite scopuri în interiorul aplicației. În acest framework componentele care se redau pe ecran sunt construite cu ajutorul limbajului specializat din react numit JSX. Acesta este o combinație între limbajul pentru crearea componentelor din interfață HTML și limbajul de programare care creează logica de interacțiune din spatele acestor componente JavaScript.

Pentru a se utiliza datele cu care utilizatorul lucrează în momentul în care aceste este în cont, în aplicație se utilizează librărie Redux. Această librărie este construită pentru a gestiona starea globală a aplicației prin utilizarea unui container numit store care centralizează toate datele folosite în aplicație, pentru fiecare sesiune separat deschisă în aplicație de către fiecare utilizator la un moment dat.

În Figura 3.11 sunt reprezentate toate paginile din aplicație pe care un pacient le poate accesa.

Figura 3.11 Paginile utilizatorului

O interfață construită în React utilizează o singură pagina de JavaScript pentru a reda pe ecran componentele. Fiecare componentă realizată cu diferite scopuri care este necesară utilizatorilor este importată în aceasta pagină numită „App.js”. Tot codul necesar pentru stilizarea și poziționarea tuturor componentelor din această pagină se află în „App.css”, astfel se creează un mod simplu de a schimba poziționarea sau modul în care arată fiecare pagină deoarece nu există cod CSS în alt loc în aplicație. Pentru a se alege care componente apar pe ecran la un moment dat se utilizează librăria React Router. Această librărie pune la dispoziție componenta Browser Router care este responsabilă de

sincronizarea interfeţei utilizatorului cu adresa URL din browser-ul web. Prin utilizarea acestei componente se poate defini o rută care se integrează în adresa URL a browser-ului în care este folosită interfaţa, pentru fiecare componentă. În acest mod se poate folosi doar una din aceste căi de acces la un moment dat pentru fiecare utilizator înparte şi browser-ul ştie exact de care pagină este nevoie pentru a fi redată.

O parte din aceste URL-uri, care reprezintă adresa pentru o anumită pagină din aplicaţie, este nevoie de autorizare pentru a putea fi accesat doar de o anumită entitate sau pentru a bloca accesul din exterior. Pentru a putea securiza aceste pagini se utilizează „AuthGuard". Aceasta este o componentă care verifică autoritatea pe care o are un utilizator în interiorul aplicaţiei. În acest caz autoritatea se verifică prin intermediul rolurilor pe care fiecare utilizator le primeşte în momentul în care îşi crează un cont în aplicaţie. O pagină poate să ofere posibilitatea de a fi accesată cu mai multe roluri. Astfel un pacient nu poate să acceseze paginile destinate pentru un medic de familie chiar dacă acesta cunoaşte URL-ul pe care pagina respectivă îl are. Singurele pagini care nu au nevoie de securizare sunt cele în care se pot înregistra conturi noi sau paginile de logare.

Pentru toate aceste pagini care lucrează cu anumiţi utilizatori din aplicaţie este necesar accesul la containerul centralizat numit store. Acest store face parte din librăria Redux. Se foloseşte componenta „Provider" pentru a oferi acces la store. Are un parametru care poartă chiar denumirea store unde se specifică numele cu care s-a importat acest container în pagina App, unde se creează toate rutele spre pagini.

Asemănător cu modelele din server şi pentru partea de client se creează modele pentru anumite entităţi. Aceste modele se creează pentru entităţile care sunt reprezentate printr-un cont, de exemplu pacient sau medic de familie sau entităţile care se pot crea, cum ar fi trimiterea sau prescrierea medicală. Modelele create în componenta client sunt în strânsă legătură cu modelele din server. Acestea se construiesc sub formă de constructori şi poartă numele entităţii de care sunt legaţi. Deoarece există o strânsă legătura cu serverul, numele variabilelor folosite în acest constructor trebuie să coincidă cu numele folosite în server, pentru a se face legătura între proprietăţile claselor.

Crearea containerului store se realizează în aplicaţie în directorul „redux-store". În acest director se află mai multe fişiere, fiecare cu un rol bine definit. Într-un fişier se definesc tipurile de acţiuni posibile, aceste tipuri constau în setarea utilizatorului curent sau în ştergerea acestuia, pentru fiecare rol posibil din aplicaţie. În fişierul pentru acţiuni se definesc obiecte JavaScript care reprezintă intenţia de a schimba starea aplicaţiei, aici reprezintă acţiunea de a seta utilizatorul curent sau de a fi şters în momentul în care acesta decide să iasă din cont sau token-ul expiră. În interiorul acestor obiecte se defineşte tipul obiectului importat din fişierul în care sunt definite tipurile şi pentru obiectele destinate setării utilizatorului, are ca şi conţinut datele utilizatorului. Aceste obiecte definite ca acţiuni se folosesc mai departe în directorul pentru reducers. Aici se definesc funcţiile care definesc modul în care aceste obiecte se folosesc. Aceste funcţii primesc ca parametru o starea curentă a aplicaţiei şi o acţiune. În funcţie de aceşti parametrii se decide modul în care următoarea stare a aplicaţiei se actualizează.

De exemplu dacă starea curentă conţine un utilizator conectat la aplicaţie şi ca acţiune primeşte obiectul în care acesta este şters din container-ul store se va executa

această acţiune, în cazul în care nu există un utilizator în aplicaţie şi se primeşte acţiunea de a șterge utilizatorul curent dintr-o acţiune nu se va întâmpla nimic şi se merge pe cazul implicit definit în funcţie, astfel toate cazurile posibile de actualizare a stării aplicaţiei sunt acoperite. La sfârşit toate aceste funcţii sunt configurate pentru a forma containerul store. Acesta se formează cu ajutorul funcţiei definite în „redux-toolkit" şi combină toţi reducers pentru a se forma store.

În Figura 3.12 este reprezentată o metodă destinată pentru pacienţi pentru crearea unei programări importate cu librăria Axios din server.

```
createProgramare(userId,trimitereId,programare){

    return axios.post('http://localhost:8080/api/programare/create/'
        + userId
        + '/'
        + trimitereId,
        programare,
        {headers: authHeader()}
        );
}
```

Figura 3.12 Metoda pentru creearea programărilor importată în front-end

Există un director numit „service" care este asemănător cu cel din componenta server dar ca şi comportament funcţionează diferit. În directorul service sunt definite mai multe clase care au legătură cu tipurile de utilizatori din aplicaţie şi cu modul de autentificare şi autorizare. Fiecare din aceste clase importă toate metodele din componenta server de la nivelul controller. Fiecare clasă importă metodele care se referă la entitatea respectivă. Metodele nu au un tip anume, se definesc numele şi parametrii necesari. Importarea se realizează prin librăria Axios. Această librărie se foloseşte pentru a face cereri HTTP şi pentru operaţii asincrone, se bazează pe conceptul numit Promise. Astfel axios returnează un promise folosind metode ca şi „.then()" pentru răspunsul primit şi „.catch()" pentru cazul în care se primeşte o eroare. Axios oferă suport pentru toate metodele de bază CRUD. În momentul în care se face importul se precizează metoda CRUD şi URL metodei definit în server, împreună cu parametrii şi cu portul pe care serverul transmite informaţile, în acest caz portul 8080. La final se definesc headers unde se foloseşte autorizarea prin intermediul token-urilor.

Majoritatea codului pentru interfaţă se află la următorul nivel unde se află toate paginile care pot să apară pe ecran pentru diferite tipuri de utilizatori. Fiecare din aceste pagini conţine anumite componente definite în react care se numesc „Hooks". Aceste componente ajută la construirea acestor pagini şi la modul de gestionare a informaţiilor care se folosesc. Fiecare câmp care conţine o informaţie sau aşteaptă să primească informaţii de la utilizator este definit în spate prin unul din aceste „hooks" numit useState(). De exemplu pentru fiecare câmp care urmează să fie completat de medicul de familie într-o trimitere medicală există o variabilă declarată sub formă de state care îşi

primeşte valoarea printr-o funcţie proprie care se ocupă doar cu schimbarea informaţiei din acea variabilă.

Un alt „hook” folosit des în aplicaţie este „useEffect()” , acesta aduce informaţiile de la server către client în momentul în care o pagină nouă este accesată sau un buton este apăsat. Informaţiile sunt primite printr-o metodă definită pentru diferite tipuri de informaţii în fiecare serviciu şi se stochează în una din variabilele declarate ca state înainte. Mai departe în partea de cod JSX care se rendează pe ecran această variabilă este mapată în componentele pe care le conţine şi astfel se populeză un tabel. Un astfel de procedeu este folosit pentru trimiterile sau prescrierile medicale.

În Figura 3.13 este reprezentat un formular pentru crearea prescrierilor din aplicaţie. Acest formular poate să fie accesat doar de medicii de familie.

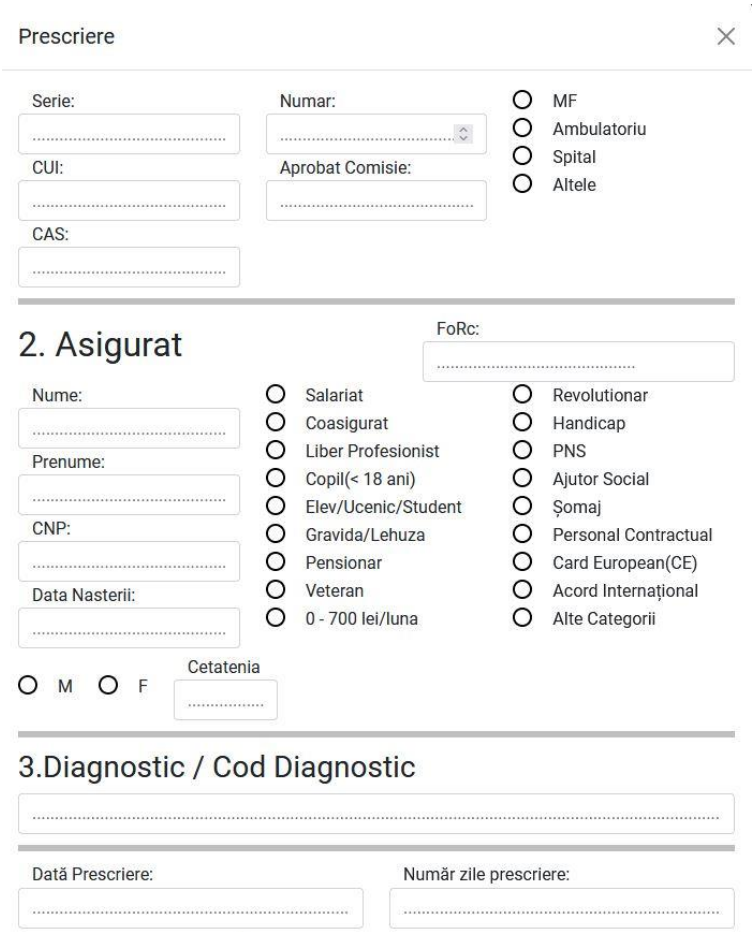

Figura 3.13 Formularul pentru prescriere

Pentru paginile care sunt destinate creării unei trimiteri, prescrieri sau programări se foloseşte o interfaţă specială numită „Modal”. Aceasta este o componentă care este afişată peste interfaţa deja existentă şi este folosită pentru a completa o componentă de tipul formular. Aceste modele sunt definite separat de paginile aplicaţiei deoarece nu sunt pagini de sine stătătoare. Se apelează din paginile destinate creării de astfel de formulare şi se instanţiază ca o componentă care primeşte parametrii din variabilele primite de la server, acestea fiind informaţii diverse sau date despre utilizatorul curent, dar şi o funcţie care este destinată deschiderii acestor modele speciale de pagini. În funcţie de rolul lor

se apelează cu o referinţă care este dedicată creării unui nou formular sau destinat pentru vizionarea unui formular. Această referinţă din urmă se foloseşte pentru paginile care au doar drept de vizualizare şi nu se pot schimba informaţiile, cum ar fi un pacient care doreşte să-şi vadă trimiterea primită de la medic de familie.

3.4.3 Baza de date

Baza de date în care se stochează informaţiile în această aplicaţie a fost creeată cu ajutorul serverului MySql. Această bază de date este una relaţională însemnând că informaţia este organizată în tabele în care pot să se formeze relaţii cu aceste informaţii. În aplicaţie există relaţii numite „one-to-one" între o programare şi o trimitere, într-o programare poate să existe doar o trimitere şi o trimitere poate să se afle doar într-o programare. Relaţia de „one-to-many" apare la medicii de familie şi pacienţi, unde un medic de familie poate să se ocupe de mai mulţi pacienţi dar un pacient poate să aparţină de un singur medic.

Pentru a putea vizualiza înregistrările din baza de date pentru fiecare entitate se utilizează programul MySql Workbench. În interiorul acestui program se poate executa cod SQL pentru a insera sau a şterge informaţii pentru testarea aplicaţiei. După ce fiecare entitate a fost definită şi primeşte toate adnotaţile necesare, în momentul în care se porneşte componenta server prin intermediul web serverului Tomcat câte un tabel pentru fiecare entitate se defineşte automat în baza de date cu atributele specificate în editorul IntelliJ. Această automatizare se datorează framework-ului Hibernate.

În Figura 3.13 este reprezentat modul în care framework-ul Hibernate creează automat tabele în baza de date din entităţile create în aplicaţia Java.

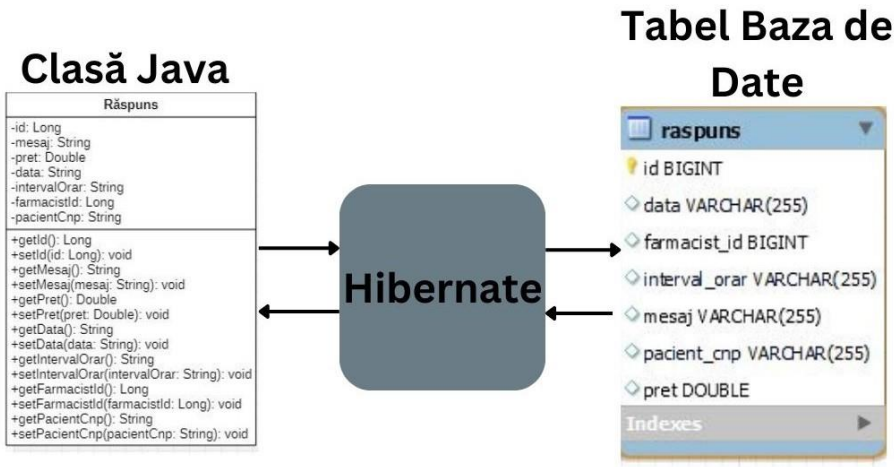

Figura 3.13 Modul de funcţionare Hibernate

Hibernate este un framework bazat pe limbajul Java care simplifică modul în care se interacţionează cu bazele de date relaţionale în aplicaţile dezvoltate cu limbajul de

programare Java. Hibernate oferă suport pentru „lazy loading" ceea ce însemnă că obiectele asociate sunt încărcate din baza de date doar în momentul în care acestea sunt accesate, ceea ce îmbunătăţeşte performanţa prin reducerea de informaţii care se obţine din baza de date.

Acest framework se bazează pe ORM (object-relational mapping) care oferă posibilitatea programatorilor de a lucra cu obiecte Java împreună cu relaţile dintre acestea în timp ce acestea interacţionează cu informaţiile din tabelele din baza de date. Astfel modul de iniţializare pentru interacţiunile dintre Hibernate şi baza de date dar şi conexiunea cu aceasta se realizează în directorul numit „resources", în fişierul cu proprietăţile aplicaţiei. Pe prima linie se specifică URL-ul pentru sursa de informaţii folosită. Se specifică serverul, în acest caz MySql, adresa acestuia, dacă se află pe acelaşi dispozitiv cu servrul atunci este „localhost", numărul portului care în mod implicit pentru MySql este 3306 şi numele bazei de date. Mai departe în acest URL se specifică faptul că se doreşte crearea bazei de date dacă aceasta nu există deja astfel se automatizează modul în care se crează baza de date, acest lucru se realizează prima data cand se porneşte componenta server pe un dispozitiv. Mai departe se setează diferite detalii cum ar fi fusul orar al serverului sau permisiunea pentru prelucrarea cheilor publice din baza de date. Mai departe se specifică username-ul şi parola pentru serverul în care este conţinută baza de date pentru a se putea conecta.

Împreună cu Hibernate lucrează şi JPA (Java Persistance API) care este un standard pentru bazele de date relaţionale. Jpa oferă un set de interfeţe şi adnotaţii care definesc un API pentru accesarea şi gestionarea bazelor de date relaţionale în aplicaţille Java. Acest standard oferă metode şi API-uri pentru a realiza operaţiile de bază CRUD cu informaţile din baza de date. În acest mod se pot extrage sau creea informaţii pe baza anumitor proprietăţi din entităţi care sunt transformate în tabele iar proprietăţile în câmpuri care compun aceste tabele.

În Figura 3.14 este o imagine cu tabela în care se stochează medicii, imaginea este din aplicaţia MySql Workbench în care se poate vizualiza baza de date.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	approved	cnp	create_time	first_name	institutie	last_name	password	role	username	poza_parafa
14	1	12345	2023-06-15 09:14:07.409385	PrenumeDoctorTest11	Spitalul Municipal	NumeDoctorTes...	$2a$10$x/ys2hP7R9fs91vKbbuxJOqlpTdHWDr13HR...	DOCTORSPITAL	DoctorTest11	MS890123
17	0	18908233...	2023-06-25 18:17:30.153828	firstName1	Spitalul Judeţean de Urgenţă Alba	Popescu	$2a$10$wtm79a4tR7yKhLXNJy6CcuZT/jV2X0COBD...	DOCTORSPITAL	PopAna1985	SP123456
19	1	29311011...	2023-06-25 18:19:12.652527	Mihai	Spitalul Orăşenesc Ineu	Ionescu	$2a$10$v4SsuJj4BlaZhX3Ekzm/sOE.ElGT7mbtpaSt6...	DOCTORSPITAL	IonescuMiha...	MS789012
20	1	50604222...	2023-06-25 18:19:46.231023	Elena	Spitalul Judeţean de Urgenţă Piteş...	Vasilescu	$2a$10$D1hCw9o8p6d.FE2th77j/uXKhy3NdO2KjEc....	DOCTORSPITAL	VasiElena7	DR345678
21	1	67803101...	2023-06-25 18:20:17.010983	Andrei	Spitalul Clinic Judeţean de Urgenţă...	Stanescu	$2a$10$h9cLd2kZppKfUHnPcLpfdOricieYtMC91t7vg...	DOCTORFAM	StanAndrei99	SP901234
22	0	64310234...	2023-06-25 18:21:13.214561	Ioana	Spitalul Judeţean de Urgenţă Bistriţa	Dumitrescu	$2a$10$eL/Ct4Wmd5oOvnRooD7rq.iLpVOfvdPvD3x...	DOCTORFAM	IoanaDumitr...	MS567890
23	0	52506041...	2023-06-25 18:21:49.383452	Alexandru	Spitalul de Recuperare Sfântul Ghe...	Georgescu	$2a$10$FKummJ0hu619k1oFPoV9w.pGB8g9ntL.Lue...	DOCTORSPITAL	AlexGeorges...	DR234567
24	1	66011236...	2023-06-25 18:22:23.545537	Maria	Spitalul Clinic Judeţean de Urgenţă...	Mihaiu	$2a$10$JZME/WMWLdIdBibkLGmrfOy8iS9S3VzS3HC...	DOCTORFAM	MariaMihaiu89	SP678901
25	1	51703111...	2023-06-25 18:22:59.992062	Ion	Spitalul Judeţean de Urgenţă Buzău	Radulescu	$2a$10$leyzmoGqmF4cIa8MQ4d7EuwXpOIzdunpW...	DOCTORFAM	IonRadulesc...	MS012345
26	1	64109250...	2023-06-25 18:23:29.868610	Laura	Spitalul Clinic Judeţean de Urgenţă...	Enescu	$2a$10$o8DUBv4lEP8e.3jLD9PYgOPTV6T0T7IqZfGe...	DOCTORSPITAL	LauraEnescu22	MS012345
27	0	52212121...	2023-06-25 18:27:19.148973	Adrian	Spitalul Clinic Judeţean de Urgenţă...	Andreescu	$2a$10$SSH1xIger0CUCAS527oMRugVf0H01NI4uF...	DOCTORSPITAL	AdrianAndre...	DR567890
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.14 Tabela din baza de date pentru medici

3.5 Testare şi validare

Testarea într-o aplicaţie web se ocupă de evaluarea mai multor aspecte din aplicaţie cum ar fi calitatea, securitatea şi performanţa. Există mai multe tipuri de testare, fiecare fiind menită pentru testarea unui comportament specific a aplicaţiei. Pentru această

aplicaţie s-a folosit testarea funcţională pentru testarea diferitelor funcţionalităţi cum ar fi interacţiunile utilizatorului cu diferite pagini web sau modul în care utilizatorii cu rol de medic interacţionează cu formularele. Testarea securităţii constă în verificare funcţionalităţii pentru atribuirea corectă a autorizaţilor de accesare a resurselor şi validarea token-urilor de sesiune pentru utilizatori. [25]

3.5.1 Testarea server-ului

Testarea server-ului se ocupă în cea mai mare parte cu verificarea funcţionării corecte a tuturor metodelor create penru operarea logicii întregii aplicaţii. Aici se testează şi funcţionarea corectă pentru baza de date şi comunicarea dintrea aceasta şi server pentru obţinerea informaţiilor.

Prima dată s-a început cu testarea manuală unde fiecare endpoint din componenta server se testează pentru un set diferit de date. Aceste teste s-au realizat cu ajutorul platformei Postman. Această platformă este folosită pentru dezvoltarea şi testarea API-urilor. Cu ajutorul acestei platforme se poate simula componenta client şi astfel se pot trimite cereri HTTP şi se pot primii răspunsuri de la server în funcţie de metodele apelate, se oferă şi suport penru utilizarea token-urilor de autentificare deci se poate testa şi securitatea metodelor.

Testarea acestei aplicaţii web în Postman se realizează prin două moduri. În primul mod se testează fiecare metodă manual şi se verifică răspunsul şi codul HTTP primit pentru a se putea valida funcţionalitatea.

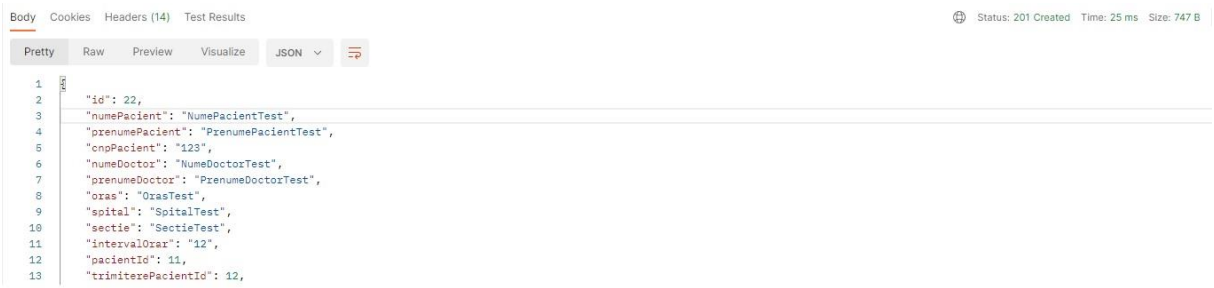

Figura 4.1 Răspunsul din postman

În Figura 4.1 se poate vedea răspunsul primit în format JSON pentru metoda de creeare a unei programări de către un utilizator. Metoda se aplelează de pe portul 8080 unde funcţionează serverul şi se continua cu URL-ul metodei, în acest caz „/api/programare/create". În URL-ul metodei se specifică şi id-urile pentru userul care creează programarea (11) şi id-ul trimiteri pe care trebuie să o conţină programare (12), toate restul informaţilor trimise pentru corpul programării sunt date de test. În partea dreaptă se vede codul primit ca răspuns de la server, timpul de exectuţie a metodei şi dimensiunea informaţilor. Cu ajutorul codului primit din răspuns se constată că metoda funcţionează corect, codul „201" arată că cererea s-a realizat cu succes şi resursa a fost creată cu datele arătate în Postman. Daca metoda funcţionează corect se poate verifica şi în baza de date dacă înregistrarea s-a realizat cu succes. Asemănător cu această metodă

se testează și pentru celelalte metode pentru a obține, a actualiza sau pentru a crea resurse în baza de date.

Postman oferă un framework pentru automatizarea testelor pentru metodele dintr-un API. Aceste teste automate sunt scrise în limbajul JavaScript într-o componentă specializată pentru așa ceva care efectuează testele de fiecare dată când se face o cerere în Postman și se primește răspunsul.

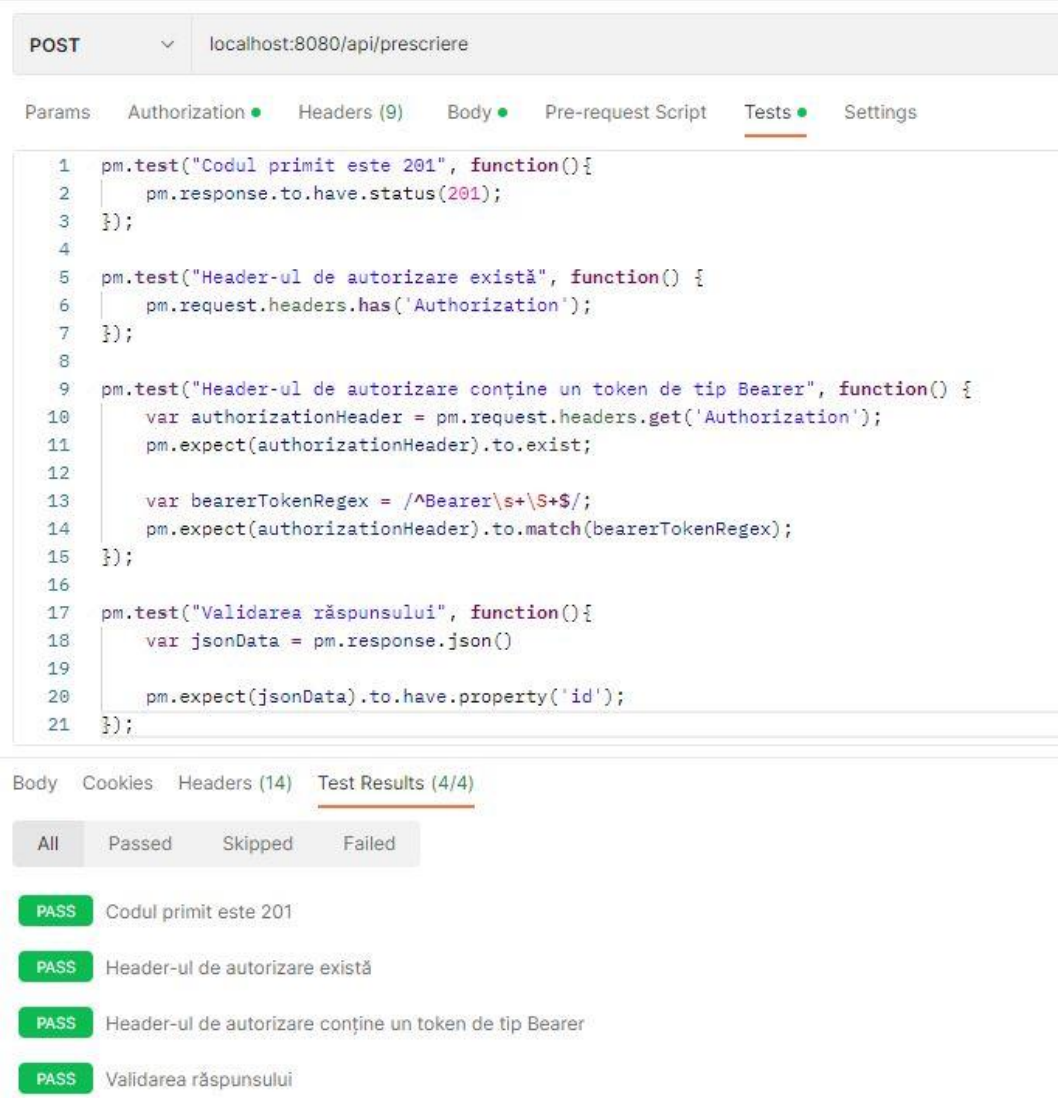

Figura 4.2 Testarea în postman

În Figura 4.2 sunt reprezentate testele și rezultatele testelor efectuate pentru crearea unei prescrieri. Cu ajutorul acestor teste se verifică dacă codul primit este corect, aici se testează metoda care trebuie să creeze o prescriere și se așteptă codul „201” care arată că o resursă nouă a fost creată. Mai departe se verifică dacă există un header de autorizare deoarece această metodă trebuie să fie accesată doar de utilizatorii care au rol de medic specialist. În acest header trebuie să existe un token de autorizare care pe lângă

alte detalii ale utilizatorului conţine şi rolul acestuia. La verificarea existenţei token-ului se verifică dacă primul cuvânt este ,,Bearer", acesta indică tipul de token. Această aplicaţie este construită astfel încât să accepte doar token-uri de tip ,,Bearer". Ultimul test verifică dacă răspunsul primit conţine proprietatea ,,id", aceasta fiind identificatorul unic pentru fiecare înregistrare din baza de date. În interiorul acestui test se adaugă verificare diferitelor proprietăţi în funcţie de răspunsul care se aşteptă de la fiecare metodă.

3.5.2 Testarea Clientului

Pentru testarea clientului s-au testat componentele care formează interfaţa utilizatorului dar şi funcţionalitatea componentelor. Prima parte din această testare constă în verificarea modului în care se aşează componentele în pagina web pentru diferite tipuri şi mărimi de ecrane. Aici se verifică dacă marginile şi spaţile din pagină se rămân la fel chiar dacă aplicaţia este folosită cu un ecran mai mare sau mai mic. Se verifică şi dacă informaţia este reprezentată corect în tabele, astfel încât datele de pe o coloană să nu depăşească o anumită limită şi să ajungă ca poziţionare într-o altă coloană.

După ce se testează aspectul şi modul de aşezare a componentelor se verifică ca fiecare tip de utilizator să poată vizualiza doar interfaţa destinată acestuia. De exemplu un medic de familie care are opţiunea de a creea trimiteri medicale nu trebuie să primească şi opţiunea de a creea şi prescrieri medicale, acestea fiind dedicate doar pentru medicii specialişti. Fiecare tip de utilizator are la dispoziţie un set diferit de butoane care duc la paginile pe care doar acesta le poate accesa.

Ultima parte din testarea clientului constă în verificare funcţionalităţilor tuturor componentelor din paginile web. În această parte se testează dacă toate butoanle, dropdown-urile şi checkbox-urile funcţionează corect şi dacă se apelează metodele corecte. De exemplu pentru testarea butonului care este destinat creării de trimiteri se verifică dacă se deschide corect formularul pentru creare şi dacă în momentul salvării formularului acesta apelează metoda corectă din API şi apare o înregistrare corectă în baza de date destinată trimiterilor medicale. Aceste verificări contiinuă înlănţuite pentru că în mod natural următoare verificare constă în verificare butonului din interfaţa utilizatorului cu rol de pacient care oferă posibilitatea de a vizualiza trimiterile primite.

4 Concluzii

4.1 Rezultate obținute

Această aplicație web a fost realizată pentru digitalizarea procesului în care funcționează trimiterile și prescrierile medicale pentru pacienți dar și pentru medicii care se ocupă de acestea. Un alt avantaj al acestei aplicații este că pacienții care nu locuiesc în aceeași localitate cu medicii lor de familie pot să își primească trimiterile pentru a putea merge la un control specializat mult mai ușor. Unul dintre avantajele obținute pentru termen lung este că se creează o baza de date centralizată unde se pot stoca toate trimiterile și prescrierile pacienților și în acest mod se pot păstra pentru un timp îndelungat. În aceelași timp se poate accesa mai ușor istoricul medical pentru un pacient.

Modul în care se realizează programările prin intermediul aplicației depinde de modul în care acestea se realizează în mometul de față pentru fiecare instituție. S-a ales acest mod de realizare a programărilor pentru ca nu se urmărește scopul de a înlocuii celelalte aplicații prin care deja pacienții se pot programa pentru o investigație la un medic specialist. Am considerat că un mod nou de a realiza programări nu este necesar deoarcece fiecare instituție poate să aibă un mod diferit de a se ocupa de aceste programări în momentul de față. Așadar această aplicație urmărește să integreze sistemele deja existente realizate pentru programări, dar și pentru alte componente și nu să le înlocuiască.

Aplicația urmărește să construiască un circuit complet în care se pornește de la nevoia pacientului pentru o trimitere medicală sau pentru o investigație și se termină la momentul în care acesta poate să meargă la farmacie dupa medicamente dacă în prealabil a fost investigat de un medic specialist.

4.2 Direcții de dezvoltare

De la acest circuit de bază realizat în aplicație se poate dezvolta mai departe în funcție de nevoile pacienților, medicilor sau farmaciștilor. În continuare se pot digitaliza mai multe proceduri deja existente care să eficientizeze modul de lucru al medicilor și timpul pacienților.

O direcție de dezvoltare ar fi implementarea unui sistem centralizat de plăți cu cardul pentru pacienți. Aceștia ar putea achita prețul medicamentelor direct din aplicație. De aici s-ar putea implementa și o funcție pentru livrări de medicamente de la farmacii către pacienții care nu se pot deplasa după acestea. În acest mod s-ar putea ajuta o parte și mai mare din pacienții care nu pot să părăsească locuința personală din diferite motive.

În aplicație ar fi benefic dacă fiecare pacient ar avea o pagină în care este descris trecutul medical al acestora. Medicul de familie ar putea adăuga după fiecare intervenție o nouă înregistrare care ar descrie tipul intervenției și cauza acesteia. Mai departe acesta ar putea să adauge într-o listă toate afecțiunile ereditare pe care pacientul le-ar putea avea pentru a fi luate în considerare la umrătoarele investigații.

O altă direcţie de dezvoltare pentru aplicaţie ar fi implementarea API-ului de la Google Maps. Aceasta ar fi o caracteristică folositoare pentru a arăta pacientului exact locul în care se află farmacia pe care o caută şi în momentul în care doreşte să ridice medicamentele să primească cel mai rapid traseu din locul în care se află până la farmaice. Această implementare ar putea să fie folosită şi pentru a afişa spitalele din baza de date a aplicaţiei pe o hartă.

Pentru a acoperii mai multe cazuri posibile care ar putea să apară, aplicaţia ar putea să fie îmbunătăţită pentru a avea prescrieri medicale de tipul „TAB1” şi „TAB2” pentru meidcamentele compensate. Aceste prescrieri au un format specific şi după ce prescrierea ajunge la un farmacist acesta trebuie să o păstreze şi nu se poate înapoia pacientului deoarece acele medicamente au un risc ridicat de a provoca dependenţă.

5 Bibliografie

[1] M. K. J. Sharma, „The arhitecture of a web application," în *Web Application Development*, Pune, KMRO, 2015, pp. 3-6.

[2] D. G. &. B. Totty, „Overview of HTTP," în *HTTP: The Definitive Guide*, vol. 1, O'Reilly Media, Inc, 2002, pp. 4-5.

[3] D. G. &. B. Totty, „HTTP Messages," în *HTTP: The Definitive Guide*, O'Reilly Media, Inc, 2002, pp. 8-9.

[4] D. G. &. B. Totty, „HTTP Messages," în *HTTP: The Definitive Guide*, O'Reilly Media, Inc, 2002, pp. 51-52.

[5] B. Cooksey, „Protocols," în *An Introduction to APIs*, zapier, 2009, pp. 18-19.

[6] W. Hoogenraad, „ITpedia," 2011-2022. [Interactiv]. Available: https://ro.itpedia.nl/2018/11/02/wat-zijn-apis-application-programming-interface/. [Accesat 16 4 2023].

[7] „Web MVC framework," [Interactiv]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html.

[8] F. Gutierrez, „Introduction to Spring Boot," în *Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices*, Albuquerque, Apress, 2019, pp. 31-35.

[9] F. Gutierrez, „Spring Framework 5 - Adding Dependencies," în *Pro Spring Boot 2: An Authoritative Guide to Building MicroServices, Web and Enterprise Applications, and Best Practices*, Apress, 2019, pp. 4-10.

[10] F. Gutierrez, „Security with Spring Boot," în *Pro Spring Boot 2: An Authoritative Guide to Building MicroServices, Web and Enterprise Applications, and Best Practices*, Apress, 2019, pp. 219-231.

[11] „Data Access with Spring Boot," în *Pro Spring Boot 2: An Authoritative Guide to Building MicroServices, Web and Enterprise Applications, and Best Practices*, Apress, 2019, pp. 127-129.

[12] B. Lutkevick, „What is HTML and How Does Hypertext Markup Languag work," [Interactiv]. Available: https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language.

[13] „CSS: Cascading Style Sheets," [Interactiv]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS.

[14] „JavaScript language," Tutorials Point, 2015.

[15] A. Zola, „What is a Bootstrap and how does it work?," Tech Target, [Interactiv]. Available: https://www.techtarget.com/whatis/definition/bootstrap.

[16] A. Satyal, „DESIGNING AND DEVELOPING A WEBSITE WITH REACTJS," OAMK, 2020.

[17] „Getting Startted | Axios Docs," [Interactiv]. Available: https://axios-http.com/docs/intro.

[18] B. D. &. I. Gelman, „Core Concepts of Flux and Redux," în *The complete Redux Book*, Leanpub, 2017, pp. 6-22.

[19] M. Delisle, „Introducing MySQL Design," în *Creating your MySQL Database: Practical Design Tips*, Packt, 2006, pp. 5-8.

[20] M. Delisle, „Security and Privileges," în *Creating your MySQL Database:*, Packt, 2006, pp. 53-54.

[21] S. Popa, „Diagrama cazurilor de utilizare (Use Case Diagram)," 2018.

[22] „Crearea unei diagrame secvenţă UML," Microsoft, [Interactiv]. Available: https://support.microsoft.com/ro-ro/office/crearea-unei-diagrame-secven%C8%9B%C4%83-uml-c61c371b-b150-4958-b128-902000133b26.

[23] „Crearea unei diagrame de clasă UML," Microsoft, [Interactiv]. Available: https://support.microsoft.com/ro-ro/office/crearea-unei-diagrame-de-clas%C4%83-uml-de6be927-8a7b-4a79-ae63-90da8f1a8a6b.

[24] „Crearea unui model de bază de date (numit și diagramă Relații entități)," Microsoft, [Interactiv]. Available: https://support.microsoft.com/ro-ro/topic/crearea-unui-model-de-baz%C4%83-de-date-numit-%C8%99i-diagram%C4%83-rela%C8%9Bii-entit%C4%83%C8%9Bi-%C3%AEn-visio-7042e719-384a-4b41-b29c-d1b35719fc93.

[25] „Ghid complet de testare a aplicațiilor web," [Interactiv]. Available: https://ro.myservername.com/web-application-testing-complete-guide.