Titlul lucrării

PROIECT DE DIPLOMĂ

Autor: **Prenume NUME**

Conducător științific: **Titlu. ing. Prenume NUME**

|  |  |
| --- | --- |
| DECAN  **Prof. dr. ing. Liviu MICLEA** | Vizat,  DIRECTOR DEPARTAMENT AUTOMATICĂ  **Prof. dr. ing. Honoriu VĂLEAN** |

Autor: **Prenume NUME**

Titlul lucrării

1. **Enunțul temei:** *O scurtă descriere a temei proiectului de diplomă*
2. **Conținutul proiectului:** *(enumerarea părților componente) Pagina de prezentare, Declarație privind autenticitatea proiectului, Sinteza proiectului, Cuprins, Titlul capitolului 1, Titlul capitolului 2,… Titlul capitolului n, Bibliografie, Anexe.*
3. **Locul documentării:** *Universitatea Tehnică din Cluj-Napoca, alte locuri dacă este cazul*
4. **Consultanți:** *ing. Prenume Nume (dacă este cazul)*
5. **Data emiterii temei:**
6. **Data predării:**

Semnătura autorului

Semnătura conducătorului științific

**Declaraţie pe proprie răspundere privind**

**autenticitatea proiectului de diplomă**

Subsemnatul(a) **Prenume NUME**  , legitimat(ă) cu CI/BI seria nr. , CNP ,

autorul lucrării:

elaborată în vederea susținerii examenului de finalizare a studiilor de licență la **Facultatea de Automatică și Calculatoare**, specializareaChoose an item.**,** din cadrul Universității Tehnice din Cluj-Napoca, sesiunea Choose an item. a anului universitar 2022-2023, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

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

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data Prenume NUME

(semnătura)

**SINTEZA**

proiectului de diplomă cu titlul:

Titlul lucrării

Autor: **Prenume NUME**

Conducător științific: **Titlu. ing. Prenume NUME**

1. Cerințele temei:

2. Soluții alese:

3. Rezultate obținute:

4. Testări și verificări:

5. Contribuții personale:

6. Surse de documentare:

Semnătura autorului

Semnătura conducătorului științific

Cuprins

[1 Introducere 3](#_Toc138128918)

[1.1 Context general 3](#_Toc138128919)

[1.2 Obiective 3](#_Toc138128920)

[1.3 Specificații 3](#_Toc138128921)

[2 Studiu bibliografic 4](#_Toc138128922)

[2.1 Aplicație WEB 4](#_Toc138128923)

[2.2 Conceptele HTTP 5](#_Toc138128924)

[2.3 API 5](#_Toc138128925)

[2.4 MVC Framework 6](#_Toc138128926)

[2.4.1 Model 7](#_Toc138128927)

[2.4.2 View 7](#_Toc138128928)

[2.4.3 Controller 7](#_Toc138128929)

[2.5 Spring Boot 8](#_Toc138128930)

[2.5.1 Dependințele în Spring Boot 8](#_Toc138128931)

[2.6 Spring Security 9](#_Toc138128932)

[2.7 Spring Data 9](#_Toc138128933)

[2.8 HTML CSS JavaScript 10](#_Toc138128934)

[2.8.1 HTML 11](#_Toc138128935)

[2.8.2 CSS 11](#_Toc138128936)

[2.8.3 JavaScript 11](#_Toc138128937)

[2.9 Bootstrap 12](#_Toc138128938)

[2.10 React 12](#_Toc138128939)

[2.10.1 Axios 13](#_Toc138128940)

[2.10.2 Redux 14](#_Toc138128941)

[2.11 MySQL 15](#_Toc138128942)

[3 Analiză, proiectare, implementare 17](#_Toc138128943)

[3.1 Analiza 17](#_Toc138128944)

[3.2 Proiectare 18](#_Toc138128945)

[3.3 Proiectare 20](#_Toc138128946)

[3.3.1 Diagrama cazurilor de utilizare 20](#_Toc138128947)

[3.3.2 Diagrama secvențială 21](#_Toc138128948)

[3.3.3 Diagrama claselor 21](#_Toc138128949)

[3.3.4 Diagrama bazei de date 22](#_Toc138128950)

[3.4 Implementare 23](#_Toc138128951)

[3.4.1 Serverul 24](#_Toc138128952)

[3.4.2 Client 33](#_Toc138128953)

[3.4.3 Baza de date 37](#_Toc138128954)

[4 Testare 40](#_Toc138128955)

[4.1 Testarea server-ului 40](#_Toc138128956)

[4.2 Testarea Clientului 42](#_Toc138128957)

[5 Concluzii 43](#_Toc138128958)

[5.1 Rezultate obținute 43](#_Toc138128959)

[5.2 Direcții de dezvoltare 43](#_Toc138128960)

[5.3 Referințe bibliografice 45](#_Toc138128961)

[6 Bibliografie 45](#_Toc138128962)

# Introducere

## Context general

În introducere familiarizați cititorul cu motivația lucrării, plasați lucrarea într-un context care să permită cititorului să înțeleagă obiectivele.

Descrieți importanța lucrării, de ce merita să o faceți, plasați ideile într-un context larg.

Susțineți studiul: de ce exact aceasta aplicație/implementare. Comentați asupra aspectelor teoretice sau practice care v-au făcut să o alegeți.

În funcție de natura lucrării, ar putea fi necesar să prezentați informații de fond asupra domeniului în care se încadrează aplicația, mai ales dacă aveți o lucrare într-un domeniu multidisciplinar. În acest caz, puteți introduce terminologia pe care o utilizați în continuare.

Descrieți pe scurt lucrarea: ce conține fiecare capitol.

## Obiective

Enumerați și explicați obiectivele lucrării: ce v-ați propus să realizați în contextul prezentat anterior.

Obiectivele pot fi prezentate sub formă de listă care să evidențieze precis orientarea lucrării, să identifice conceptele fundamentale pe care le studiați, să stabilească scopul aplicației pe care o realizați, sau enunțați întrebările la care intenționați să răspundeți în lucrare.

## Specificații

În specificațiile lucrării detaliați cerințele. Descrieți ce intenționați să obțineți. Vă puteți referi la funcțiile aplicației, interfață, nivele de performanță, structuri de date, elemente, securitate, fiabilitate, calitate, limitări, etc.

# Studiu bibliografic

## 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 intr-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 stoacare. Cu alte cuvinte aplicația este mereu actualizată la ultima versiune crescând securitatea utilizatorilor.

Unul dinre avantajele importante este că aplicația poate sa 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 browser-e și cu o gamă largă de dispozitive, din orice loc, singura cerință fiind nevoia de conexiune la internet.

Aplicaștile Web folosesc arhitectura client-server, tot codul este împărțit în două componente, componenta clientului care conține interfața și componenta serverului care conține medodele, 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 clcik pe un buton, trimite o cerere (request) de la utilizator către sever.

Componenta serverului se ocupă cu procesarea de date. Serverul procesează cererea clientului si trimite înapo 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 primii 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 daca cererea s-a realizat cu succes sau nu, aceste cereri sunt de tip HTTP.

## 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 facută 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 sa fie accesată în mod direct.

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 creea 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.

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

Componenta headers care apare într-un raspuns trimis de către server pentru a oferii informații suplimentare despre rasounsul din care face parte cuim 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.

## API

API (Application Programming Interface) este un set de reguli și de protocoale careoferă aplicaților 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.

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 obieci folosesc principiile RESTful pentru a creea,citii, actualiza și șterge
* Există API-uri care sunt librării sau framework-uri deja construite pe care programatorii le pot folosii în aplicațile lor. Acestea oferă metode care simplifică dezvoltarea unei aplicații prin refolosirea codului
* Sistemele de operare dispun de API-uri care permit ca aplicațiile să interacționeze cu resursele sistemului cum ar fi: fișiere, procese sau conexiuni la internet. Aceste API-uri permit programatorilor să creeze aplicații care sunt specifice unei singure platforme.
* Există și 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 comezni specifice acestora ca preluarea sau modificarea informaților

Folosirea API-urilor vine cu niște beneficii importanate:

* Cu ajutorul acestor un sistem foarte complex se poate împărții în componente mai mici și mai usor de refolosit între mai multe simste, făcand codul mai ușor de urmărit
* Interoperabilitatea oferă posibilitatea pentru diferite aplicații, servicii sau sisteme de a lucra împreuna, promovând astfel integrarea între diferite tehnologii
* Cum exista API-uri care oferă componente deja construite care se pot integra într-o aplicație se reduce timpul de dezvoltare și se reduce efortul dezvoltării unei noi aplicașii nemaifiind nevoie de a scrie cod care deja este folosit pentru a altă aplicație
* API-urile au un ecosistem unde programatorii pot să își expună servicile care pot mai departe să fie folosite de alți programatori pentru diferite proiecte astfel se promovează inovațile

## MVC Framework

MVC este acronimul de la Model View Controller, acesta fiind un șablon de arhitectură pentru o aplicație software. Acesta oferă un mod de a structura modul prin care o aplicație este construită, separând aceasta în trei componente separate: Model, View și Controller. Fiecare din aceste componente este destinată pentru un aspect specific al aplicației.

Avantajul major penntru acest framework este că oferă o spearare clară pentru fiecare componentă a aplicației astfel programatorii pot sa lucreze independent pentru diferite aspecte, schimbari ale unei componente nu afeactează in mod direct restul componentelor doar dacă aceste schimbari urmaresc șablonul și regulile arhitecturii MVC. Acest framework oferă posibilitatea de a reutiliza codul, modularitatea acestuia.

Compromisul pentru aceste avantaje este creșterea complexitații programului fiind împărțit in diferite componente.

### Model

Model este componenta care se ocupă cu logica legată de datele/informațile? cu care utilizatorul poate să lucreze. Această componentă ține toată informația incapsulată pentru a putea fii acesată doar prin intermediul unor metode speciale care se ocupă cu manipularea acestor informații, cum ar fii obținerea informaților din baza de date sau validarea acestora.

### View

Componenta View este reprezentată de interfața utilizatorului. Aceasta definește modul in care informația de la componenta model este reprezentată pentru utilizatorul aplicației. O astfel de componentă de obicei este alcătuita din elemente de HTML, CSS si JavaScript conținând butoane, dropdown-uri și check box-uri cu care utilizatorul poate interacționa.

### Controller

Rolul componentei Controller este de a asigura un canal intre componenta Model și View.Aceasta componentă există pentru a procesa inputul primit de la utlilizator după care sa fie procesat si transmis componentei Model in funcție de request-ul primit. Cu ajutorul Controller-ului se asigură separarea si independența fiecaărei componente. Controllerul este componenta care decide modul în care informatia este folosită pentru întreaga aplicație folosinduse de View pentru a afișa aceste informații si de Model pentru a le obține din baza de date.

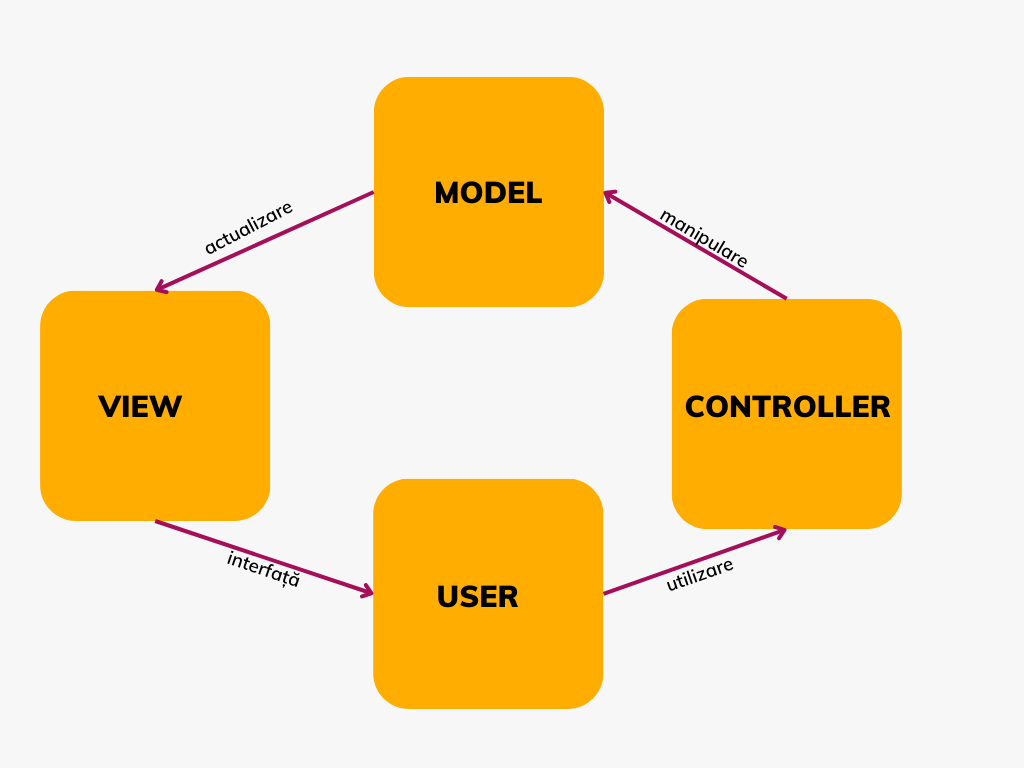


Figura 2.1 Componentele MVC

## Spring Boot

Spring Boot este un framework bazat pe limbajul de programare Java, deschis pentru publicaul larg/”open source”, care simplifică dezvoltarea aplicaților Java în special pentru aplicatile web și miroservicii. 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.

Principalul scop al acestui framework este de a minimiza procesul nesesar 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 fară fara a aloca prea mult timp pe o schemă de amploare.



Figura 2.2 Dependență pentru Spring Boot

Unele din caracteristicile cheie ale acestui framework este:

* Auto-Configurarea: Spring Boot poate să configurează mai multe componente în funcție de dependințele din proprietațile 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 fii Tomcat care oferă posibilitatea aplicaților sa fie pornite fară a fi nevoie de instalarea pe un server extern
* Managementul Dependențelor: Spring Boot simplifică managementul versiunilor dependințelor astfel se asigura o buna 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 intr-un ecosistem se pot folosii necesitând un nivel minim de configurare.

### Dependințele în Spring Boot

În Spring Boot dependințele sunt genstionate de instrumente de automatizare cum ar fii: Apache Maven sau Gradle. Aceste dependințe se definec în aplicație in fișierul “pom.xml” in Maven sau în “build.gradle” în Gradle. Cele mai folosite dependințe pentru o aplicatie web construită cu Spring Boot sunt:

* Spring-Boot-Starter-Web: această dependtință include librariile necesare pentru o aplicație web, cum ar fi un server încorporat (Tomcat) și librarii pentru gestionarea cererilor și raspunsurilor HTTP
* Spring-Boot-Starter-Data-Jpa: această dependință oferă suport pentru folosirea Java Persistence API cu Spring Data, incluzând librariile 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

## Spring Security

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

* Autentificarea: acest framework pune la dispoziție diferite mecanisme pentru autentificarea utilizatorilor cum ar fi autentificarea bazată pe un formular, autentificarea simpla 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.

## 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 aplicațile care au la bază limbajul Java. Spring Data oferă un model de programare unificat și consistent pentru interacțiunile cu mai multe tehnologii pentru stocarea informaților incluzând și bazele de date relaționale.

Caracteristicile cheie pentru Spring Data sunt:

* Abstractizarea, Spring Data prevede abstractizări si interfețe pentru a ascunde complexitatea anumitor tehnologii de acces a informaților. În acest mod se permite ca programatorii să aibă opțiunea de a scrie cod care nu interacționează direct cu codul bazi de date
* Conceptul de “Repository Abstraction”. Acesta prevede interacțiunea cu un API care se ocupă de operațile de baza, creare, citire, actualizare, ștergere pentru interacțiunea cu informațile 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 creeare de query-uri. Spring Data oferă posibilitatea de a definii metode cu query-uri prin anumite convenții. Codul SQL sau NoSQL se generează automat bazânduse pe numele metodei, reducânduse nevoia de a scrie query-uri manual

## HTML CSS JavaScript

Într-o aplicație web componenta care se ocupă cu interfața 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 și JavaScript se realizează funcționalitatea componentelor create înainte.

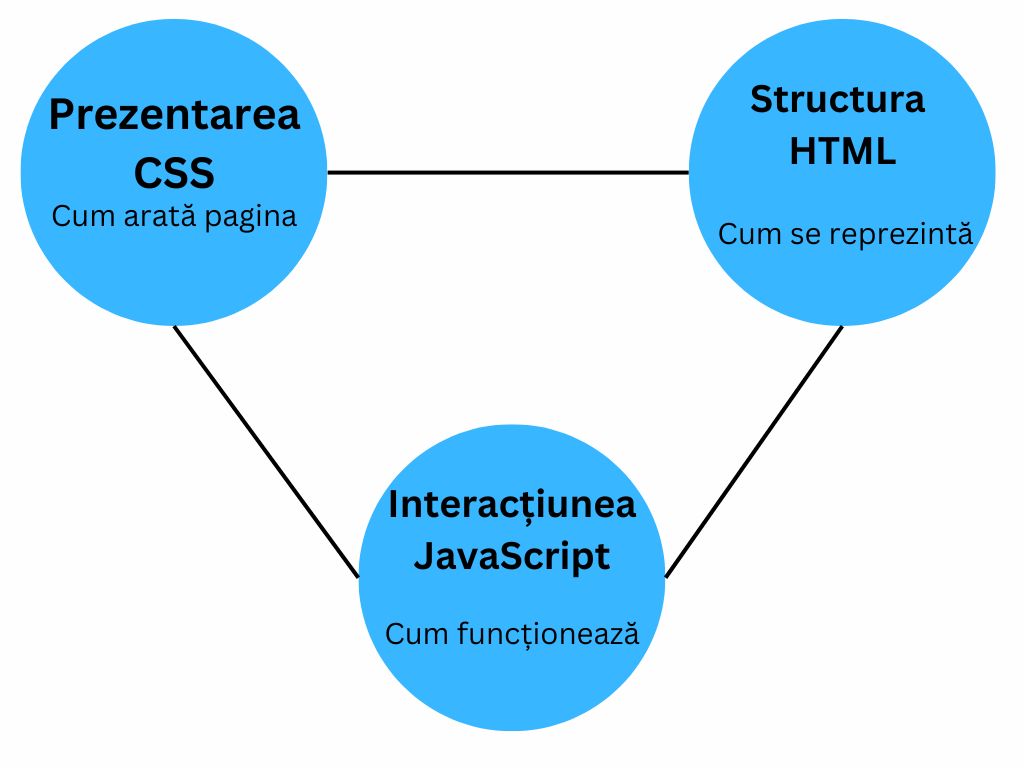


Figura 2.3 Scopul CSS, HTML, JavaScript

### HTML

Acronimul HTML vine de la “Hypertext Markup Language”, este un limbaj standard folosit pentru creearea de pagini și aplicații web. Structura și conținutul acestuia se reprezintă îm WWW(World Wide Web). Acest limbaj se foloseste de o serie de adnotații pentru a creea și pentru a structura o pagina web.

Codul pentru un document HTML se poate scrie în orice editor de text, singura condiție fiind ca extensia fișierului sa fie “.html”. Orice document HTML contine anotația: “<!DOCTYPE html>” care reprezintă locul de unde incepe codul HTML. Structura unei astfel de pagini se reprezintă prin anotațile „<html> </html>” care incadrează codul prorpiu-zis, „<head> </head>” care conține titlul paginii sau informații despre formatare, „<body> </body>” unde se reprezintă informația din pagină cum ar fi paragrafe cu text sau imagini, uneori din structura unei pagini web face parte și „<footer> </footer>”, reprezentând partea de jos a paginii unde se afla în general informații de contact de exemplu numarul de telefon sau adresa de e-mail.

### CSS

CSS (Cascading Style Sheets) este un limaj folosit împreuna cu limbajul HTML pentru a îmbunatății aspectul vizual pentru o pagina web. Cu ajutorul acestui limbaj programatorul poate să adauge sau să schimbe aspectele vizuale ale paginii după proprile preferințe, cum ar fi: font-urile, mărimea literelor, culorile textului, culorile de fundal sau redimensionarea imaginilor.

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

Acest limbaj pune la dispozitie si proprietăți prin care elementele se pot poziționa sau alinia intr-o pagină cum ar fi: “display” , “flexbox” sau “grid” . Cu aceste proprietăți se pot obține pagini care își schimbă aspectul în funcție de mărimea ecranului, aceeași aplicație poate să aibă un aspect diferit pe un Desktop față de aspectul de pe un telefon mobil.

### JavaScript

JavaScript este un limaj de programare de nivel înalt, orientat pe obiect care funcționează in browserele web. În cele mai multe cazuri este folosit pentru dezvoltarea aplicaților sau paginilor web. Cu acest limbaj se adaugă posibilitatea de a interacționa dinamic cu elementele create cu limajul HTML în pagină. Acest limaj se ocupă în principal cu gestionarea evenimentelor care apar într-o interfață web cum ar fi: click-uri, mișcări ale mouse-ului sau interacțiuni cu tastatura. Gestionarea evenimentelor oferă posibilitatea actualizării paginilor web fară a fi nevoie de reîncărcarea paginii.

Chiar daca la început a fost dezvoltat pentru creearea de cod pentru interfața aplicaților prin intermediul framework-ului Node.js acest limaj se poate utiliza și pentru construirea serverului, astfel acest limaj oferă suport pentru întreaga aplicație web.

Toate browserele din ziua de azi oferă suport și motoare pentru executarea codului JavaScript în mod eficient. În general aplicația trebuie să se muleze pentru nevoile utilizaorilor de aceea find important acest aspect în care se poate folosii orice browser după preferințe, JavaScript fiind cea mai bună alegere pentru dezvoltarea interfețelor aplicaților web.

JavaScript pune la dispoziție mecanisme pentru gestionarea informației primite de la un server pentru o interfață dintr-o pagina web. Fiind un limaj orientat pe obiecte suporta tipul de date obiect dar are si 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 sa aducă informație în pagina web din surse externe cum ar fi API-urile (Application Programming Interfaces) și astfel se poate afișa pe pagina conținut dinamic prin intermediul actualizărilor în timp real.

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

## Bootstrap

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

Unul din marile avantaje pentru folosirea acestui framework este că majoritatea componentelor predefinite oferă un design? care este adaptabil pentru diferite dispozitive. Bootstrap oferă si un sistem în care este împărțit ecranul sub forma de table cu 12/doisprezece coloane care se poate personaliza creeând diferite aspecte pentru simplificarea poziționării componentelor într-o pagină.

## React

React este una dintre cele mai populare framework-uri pentru dezvoltarea interfețelor destinate utilizatorilor și a aplicaților 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 pagina, 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 si de îmbunatățire 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 realizânduse mult mai rapid deoarece în momentul în care sunt schimbări în starea sau în proprietățile unei componente, React actualizează efficient doar părțile necesare din DOM și utlizează la un minim necesar acest proces.

Framework-ul React folosește o sintaxă declarative față de una imperativă.Prin sintaxa declarative programatorul scrie codul astfel încât să descriei aplicației modul în care vrea sa fie afișată informația primită iar React se ocupă de actualizarea interfeței pentru a arata schimbarile în funcție de ce informații primește, astfel procesul de programare a interfeței se simplifică nemaifiind nevoie ca programatorul sa se ocupe manual de fiecare component DOM pentru diferite informații primite ca intrare pentru componentă.

Față de o aplicație web clasică care nu folosește un framework pentru creearea interfeței unde in mod normal pentru a creea componente care se pot afișa pe ecran se folosește 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 combinație de structurare a codului HTML cu logica din spatele limajului JavaScript. În acest mod se pot definii toate structurile de componente și proprietățile deja existente în HTML dar intr-un mod intuitiv și mai ușor de recitit pentru alți programatori.

În acest framework se introduce conceptul de “React Hooks”. Această caracteristică pune la dispoziția programatorilor opțiunea de a folosii starea de utilizare a unei componente și alte proprietăți ale framework-ului fără a mai fi nevoie de componente Clasă. Există mai multe tipuri de “Hooks”, dar cele folosite în majoritatea aplicaților sunt “useState”, “useEffect” și “useContext” care simplifică managementul stării componentelor, maipularea evenimentelor secundare și împărțirea contextului între componente. De exemplu pentru a stoca și a seta starea unei componente se folosește useState, aceast “Hook” se declară astfel: const [componentă, setComponentă] = useState(‘’); astfel starea componentei este stocată in constanta numită “componentă” și se poate seta cu funcția “setComponentă”, valoarea de bază a acesteia fiind un string gol.

La fel ca și Spring, React are un ecosistem vast. La acest ecosistem contribuie foarte mult comunitatea care aduce constant componente refolosibile care se pot utiliza foarte ușor. Cele mai utilizate librării din acest ecosistem care ajută la îmbunătățirea productivității sunt: “React Router” care se ocupă cu rutarea pentru URL-urile paginilor din React, prin această librările se poate securiza accesul către pagini cu ajutorul rolurilor de utilizator, “Redux” librărie folosită pentru administrarea stării aplicației și “Axios” pentru cererile HTTP.

### Axios

Axios este o librările React folosită pentru a face cereri HTTP dintr-un browser web la un server. Oferă un mod simplu si intuitiv pentru programatori de a apela un API și a de a se ocupa cu operații asincrone în comunicarea cu un server.

Axios se bazează pe un model bazat pe Promises (promisiuni), ceea ce însemnă ca pentru fiecare cerere HTTP facută astfel se returnează un Promise. Un promise este un obiect care reprezintă modul în care o operație asincronă a decurs, în general aducerea informației de la un server, adică poate să rezulte intr-un răspuns de la server sau o eroare în momentul în care ceva nu a decurs bine. Mai departe un Promise pune la dispoziția programatorului un mod în care poate să gestioneze raspunsul primit și să fie salvat intr-o componentă sau un mod în care poate vedea eroarea primită împreună cu alte detalii legate de aceasta pentru a putea fi rezolvată cat mai ușor. Pentru cazul în care se primește răspunsul de la server se utilizează funcția “.then()” iar pentru cazul în care se primește o eroare există funcția “.catch()” . Aceste funcții se folosesc concomitent pentru a se acoperii ambele cazuri la fiecare cerere HTTP.

Pentru a putea folosii 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. Acestă librărie are suport pentru toate browserele importante și funcționează după aceleași reguli în fiecare penru o consistență 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()” și asa mai departe. Configurările cererilor se pot face prin opțiunile adiționale: headers, query parameters, request data și timeouts.

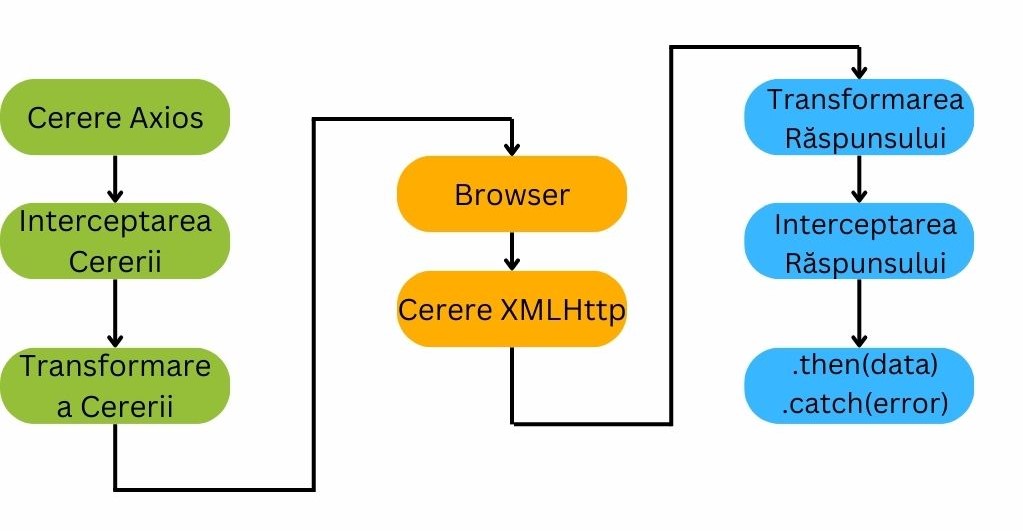


Figura 2.4 Modul de funcționare Axios

### Redux

Redux este o librărie JavaScrit pentru managementul stării unei aplicații care se folosește în general cu framework-uri ca și React sau Angular. Cu această librărie se creează un container pentru starea aplicației în care se rețin diferite detalii cum ar fi datele utilizatorului curent pentru fiecare sesiune de pe fiecare dispozitiv unde funcționează aplicația.

Store este singurul obiect în care se găsește starea aplicației care folosește Redux în orice moment, atât timp cat 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 informațile derivate din cele de bază având opțiunea de a accesa doar anumite părți cheie.

Librăria Redux se bazează pe 3/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 protejare împotriva duplicării informaților și accseul rapid la acestea.
* Starea aplicație în Redux este invariabilă, asta însemnând ca nu poate fi modificată în mod direct. Pentru a se putea interacționa cu stare aplicaței există așa numitele “Actions” care descriu schimbările de stare necesare, după “Reducers” gestionează aceste acțiuni creeând o stare nouă care se bazează pe acțiunile trimis 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 stare aplicației este ușor de testat și este predictibilă deoarece aceste funcții trebuie sa returneze aceeași ieșire pentru aceeași intrare

## MySQL

MySQL este o bază de date relațională deschisă pentru publicul larg?(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.

O bază de date relațională înseamnă că informația este organizată sub formă de tabele cu linii si coloane. Stabiliarea relaților între tabele se realizează prin intermediul cheilor oferind integritate informaților 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 operațile de baza CRUD primite de la un server cum ar fi: inserare, actualizare, citire, ștergere și multe altele, dispunând și de operațile SQL de subinterogări, alăturări, agregări și sortări pentru manipularea datelor într-un mod eficient. Pentru a putea folosii aceste metode este nevoie de unul sau mai multe query-uri. Acestea sunt comenzi dedicate în genral bazelor de date pentru a manipula informația implementând metodele de mai sus. Un query se construiește din una sau mai multe clauze care specifică ce operație trebuie executată și care sunt criteriile pentru obținerea informației respective. Cele mai folosite instrucțiuni sunt: “SELECT” este folosită pentru a obține informașie din una sau mai multe baze de date și se poate specifica numele coloanelor de care este nevoie, “FROM” indentifică tabelul sau tabelele din care informația este extrasă, “WHERE” această clauză este folosită pentru a filtra informația extrasă în funcție de criterile dorite, “INSERT” este folosită pentru a adăuga noi informații într-un tabel specificând numele tabelului și valorile care urmează să fie inserate, “DELETE” este folosită pentru a șterge informație din baza de date specificând condițile pentru a determina ce se dorește pentru ștergere.

Securitatea în această bază de date este oferită de posibilitatea de a creea conturi pentru utilizatori multiplii, fiind nevoie de a te autentifica înainte de a accesa baza de date. Fiecare utilizator având diferite privilegii și nivele de acces, fiind sigur că doar cine are autorizarea potrivită poate sa execute anumite acțiuni. În plus pentru a securiza informația aflată în baza de date MySQL are suport pentru mecanisme de encriptare a informației cand este stocată dar și in momentul în care se află în tranzit. Există posibilitatea de a encripta comunicarea folosind protocoale SSL/TLS. Astfel chiar dacă apare acces unautorizat, informația ramane criptată fara a avea cheia necesară pentru decriptare.

Scalarea bazei de date MySQL funcționează pe verticală, însemnând că prind adaugarea mai multor resurse hardware se poate mării capacitatea sau pe orizontală prin distribuirea informației între servere multiple.

Indexarea îmbunătățește performanța query-urilor deoarece aceștia activează posibilitatea de a primii informațile din tabele mult mai rapid. MySQL oferă suport pentru mai multe tipuri de indexare cum ar fi: hash, graf sau text în funcție de cerințele necesare pentru query-uri.

# Analiză, proiectare, implementare

## 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 indeplinirea următoarelor obiective:

* Posibilitatea tuturor utilizatorilor de a creea 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 dupa înregistare pacientul poate sa își actualizeze contul cu datele medicului de familie de care aparține, având posibilitatea de schimba aceste date în orice moment dorit
* Medicul de familie are posibilitatea să vizualizeze pacienții de care aparține și poate creea 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 creeată în prealabil către medicul specialist
* Medicul specialist poate sa vizualizeze trimiterile primite de la pacienți acestea fiind ordonate crescător în ordinea 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 își vizualizeze prescrierile medicale în cont la secțiunea ,,Prescrieri”
* Pacientul poate să aleagă una dintre farmaciile prezente, adresa acesteia fiind afișată, în aplicație și să îsi trimită prescrierea
* Farmacistul vizualizează prescrierile primite și poate să pregăteasca medicamentele, după poate să notifice pacientul că meidcamentele 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 sa revină oricând să vizualizeze indicațile primite de la farmacist

Scopul general al acestei aplicații este acela de a facilita tot procesul care pornește de la nevoia de o investigație facută de un medic specialist pentru o anuită problemă de sănătate. Astfel un medic de familie poate sa realizeze o trimitere mult mai ușor fiind salvată în format electronic astfel se ușurează modul în care pacientul poate să își primească trimiterea de la distanță.

Această parte a procesului desfășurându-se în mediul online și trimiterea fiind în format electronic este necesar ca și următoare parte a procesului să se desfăsoare la fel, astfel și prescrierile care vin de la medicul specialist se vor creea în format electronic.

Prin aceată aplicație se digitalizează tot procesul care pornește de la medicul de famile care creează o trimitere și se termină în momentul în care farmacisul notifică pacientul ca medicamentele de care are nevoie sunt pregătite.

Formularele de trimiterii și de prescrieri sunt construite astfel încat să conțină toate câmpurile necesare din formularele de trimitere și prescriere din format fizic. Întregul proces realizat în această aplicație se mulează după procesul de trimiteri și prescrieri deja existent din sistemul medical din România.

## Proiectare

Această aplicație 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 și poate să fie accesat și manipulat prin metodele standard din HTTP. Resursele sunt reprezentate de utilizatori, trimiteri, prescrieri sau de notificările farmaciștilor.

Această athitectură se folosește pentru construirea de servicii web. Se bazează pe principile arhitecturale “Representational State Transfer” (REST) care se folosesc de protocoalele și standardele web existente, resursele sunt identificate de adrese URL unice și clienții interacționează cu aceste resurse folosing standardul HTTP prin metodele de bază CURD (GET, POST, PUT, DELETE).

Astfel interfața, partea pe care o vede orice utilizator care folosește aplicația, comunică cu serverul prin aceste meotode din standardul HTTP. În acest mod se asigură separare dintre client și sever, acestea nu pot să comunice în mod direct ci doar prin intermediul metodelor definite cu acest scop.

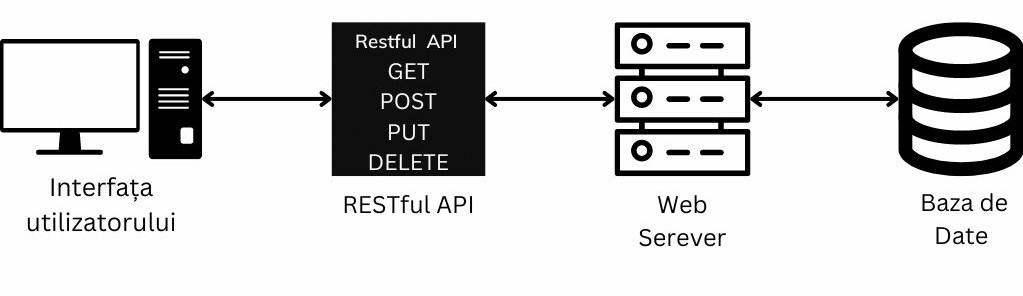


Figura 3.1 RESTful API

Arhitectura folosită penru realizarea părții de server este MVC (Model-View-Controller). După cum îi spune și numele această arhitectură împarte aplicație în trei componente: Model (modele), View (interfețe) și Controller (controlori). Modelele reprezintă datele și logica aplicației, tot aici se implementează și accesul la date.

În această componentă se definește modelul utilizatorilor, pacienți, doctori, farmaciști dar și a trimiterilor, prescrierilor sau a programărilor și administratori împreună cu proprietățile acestora cum ar fi rolurile, numele sau instituțile din care fac parte. Tot în această componentă se creează și servicile pentru fiecare din aceste componente, servicii unde sunt definite metodele prin care se poate interacționa cu proprietățile entităților.

Componeta View în această aplicație reprezintă antetul metodelor create pentru interacționarea cu resursele. Acestea nu conțin deloc implementarea metodelor și pot să fie doar accesate din componenta controller care folosește metodele.

Componenta Controller se ocupă de interacțiunea dintre Model și View. Aceasta primește datele utilizatorului prin intermediul componentei View și actualizează componenta Model în funcție de această intrare.

Arhitectura parții clientului se bazează în mare parte pe tiparul Flux care este implementat prind Redux. Această arhitectură se bazează pe acțiuni care declanșează schimbări într-o componentă centrală a aplicației numită “store”. Schimbările care apar în interfața utilizatorului se actualizează în funcție de schimbările care apar în această componentă centrală. Aceste schimbări în store constau in cereri HTTP din client către server deoarce întrega aplicație este construită pe baza arhitecturii RESTful.

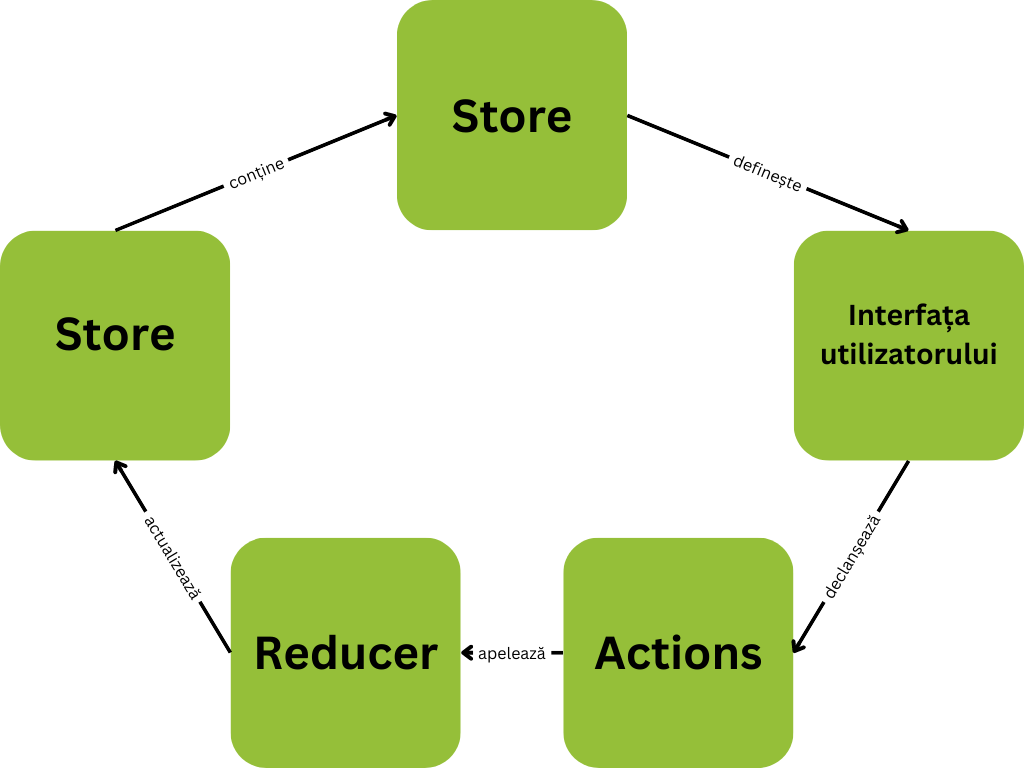


Figura 3.2 Librăria Redux

## 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 relațile dintre acestea.

### 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.

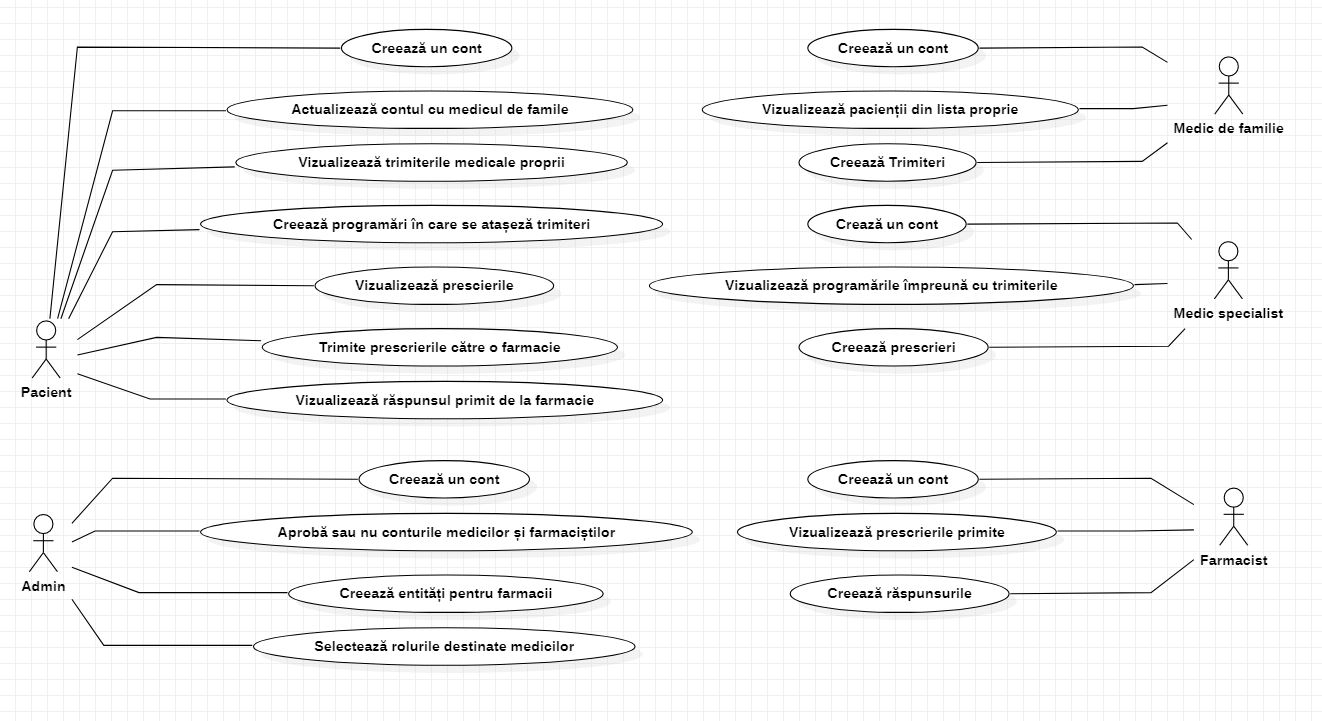


Figura 3.3 Diagrama cazurilor de utiliza

În această diagramă se poate observa că fiecare tip de utilizator poate să își creeze un cont. Medicii în funcție de rol pot să creeze trimiteri sau prescrieri în funcție de rolul pe care îl au și pot să vadă o listă cu pacienții, respectiv o listă cu programări. Se poate observa că pacientul are cele mai multe opțiuni. După ce își creează un cont acesta poate să își actualizeze contul pe a adauga mediul de familie. Poate să vizaulizeze trimiterile ptrimite de la doctorul de familie, poate să creeze programări cu aceste trimiteri, poate să vizualizeze prescrierile primite în urma unei investigații, aceste prescrieri le poate trimite către o farmacie și poate să vizaulizeze răspunsul primit în momentul în care medicamentele sunt pregătite. Administratorul poate să aprobe doctorii și farmaciștii care își creează un cont nou doar dacă instituția din care fac parte confirmă ca lucrează acolo și atribuie roluri pentru medicii de familie sau pentru medicii specialiști. Farmaciștii pot să vizualizeze prescrierile primite de la pacienți și să creeze răspunsuri pentru aceștia.

### Diagrama secvențială

Diagrama secvențială este un tip de fiagramă UML (Unified Modeling Language) care descrie modul în care diferite componente, interfețe și obiecte din aplicație interacționează între ele. Modul în care se descriu aceste interacțiuni este ordonat în modul cronologic de apariție a evenimentelor și a secvențelor de apelare a metodelor.

Componentele din aplicație sunt reprezentate ca fiind niște blocuri cu o linie punctată care reprezintă linia de viață a componente și mesajele sau interacțiunile către aceste componente sub formă de săgeți deasupra cărora este specificată acțiunea realizată. Informațile primite sau răspunsurile care reprezintă succesul sau eșecul acțiunilor sunt reprezentate cu săgeți cu linii punctate. Locul în care se realizează o astfel de acțiune cum ar fi apelul unei metode este reprezentat sub formă de dreptunghi care arată durata unei astfel de metode.

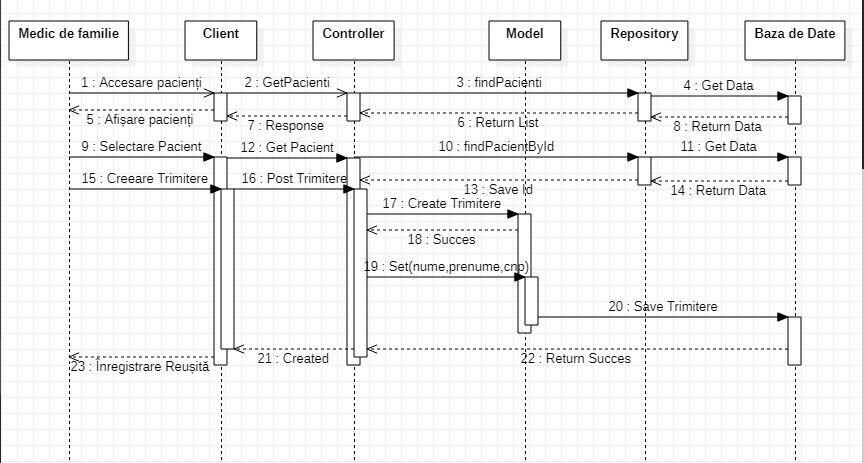


Figura 3.4 Diagrama secvențială

### Diagrama claselor

Diagrama claselor este una din diagramele folosite pentru pentru proiectarea aplicaților și descrie foarte bine modul în care componetele din server interacționează între ele. Într-o astfel de diagramă se reprezintă structura logicii din spatele aplicației prin legăturile făcute între diferite clase și interfețe.

Fiecare dreptunghi reprezintă o clasă, în partea de sus se află numele, în prima parte sunt toate proprietățile pe care le conține și prin ce tip de date este reprezentat fiecare și indetificatorul de acces. În a doua parte a dreptunghiului sunt reprezentate toate metodele care țin de clasa respectivă, împreună cu parametrii fiecărei metode, cu tipul de date pe care le returnează metoda și identificatorul de acces. Fiecare interfață este reprezentata de câte un cerc care respectă aceeleași norme de descriere la fel cu cele pentru clase.

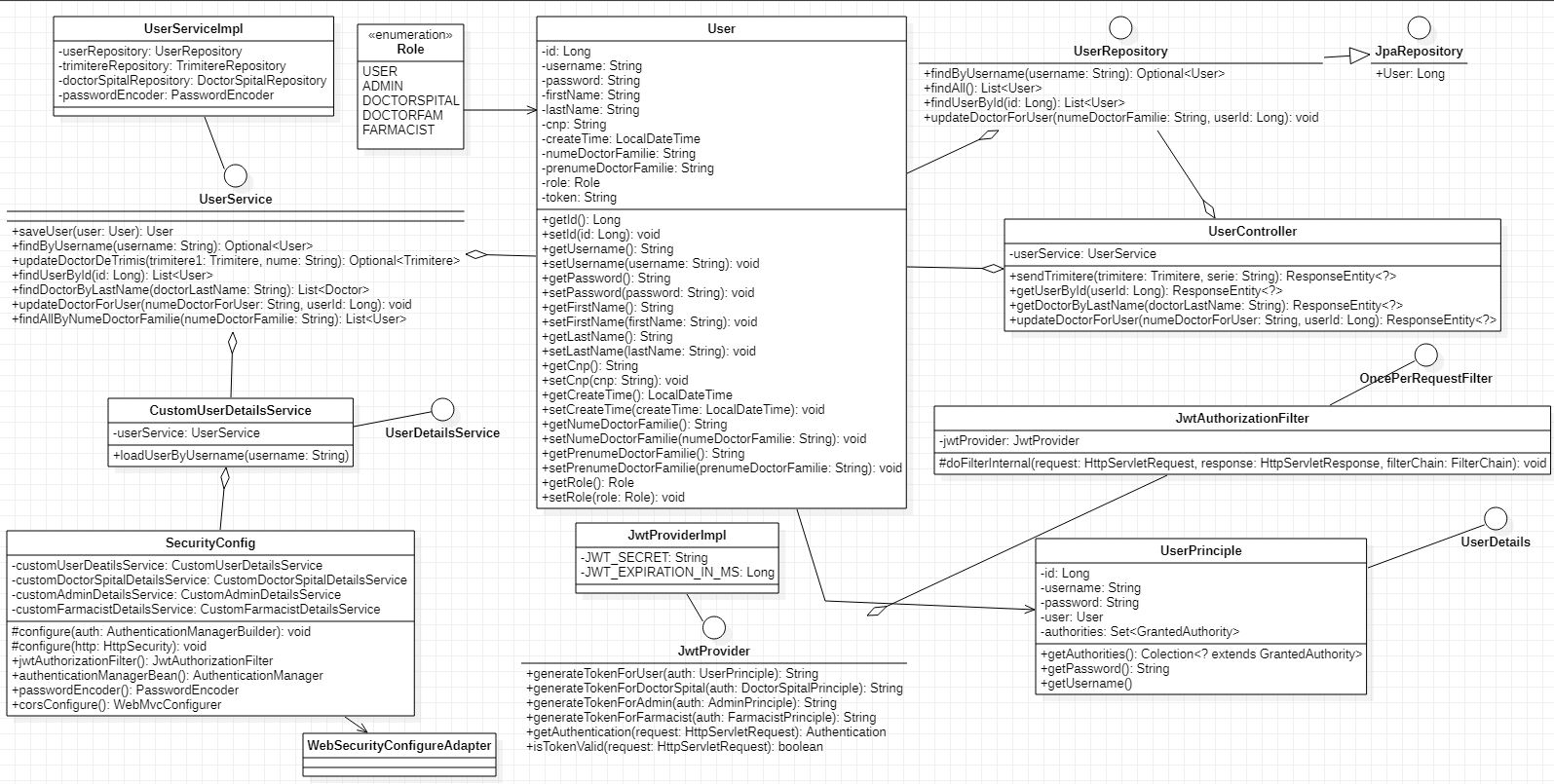


Figura 3.5 Diagrama claselor

### Diagrama bazei de date

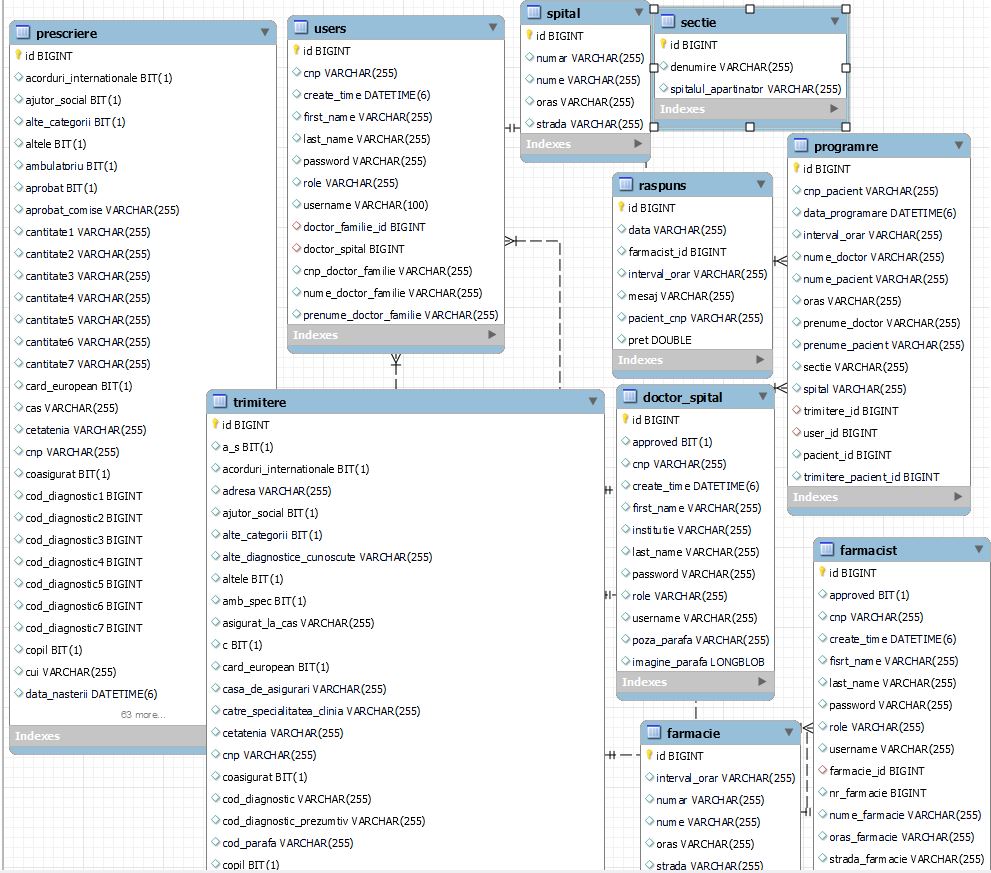


Figura 3.6 Diagrama bazei de date

Diagrama bazei de date descrie modul în care baza de date dintr-o aplicație este structurată. Asemănătoare cu diagrama claselor aici se reprezintă entitățile descrise în componenta server dar sub formă de tabele deoarece astfel sunt reprezentate într-o bază de date relațională.

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

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

## Implementare

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

Întrega aplicație se bazează pe arhitectura RESTful care separă codul în două mari componente Clientul care este destinat utilizatorului și se ocupă de interfața acestuia și Serverul care implementează logica de lucru cu informațile.

Componenta serverului este implementată cu limbajul de programare Java și prin ajutorul framework-ului Spring Boot care are la bază arhitectura Spring MVC. Această arhitectura împarte codul din server în 3 componente model, view și controller, ficare 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 interfața utilizatorului este implementată cu ajutorul framework-ului React care se bazeză pe o librăria Redux. Această librărie se ocupă de administrarea stării aplicației în momentul în care este folosită de către un utilizator. Limbajul de programare folosit este JavaScript pentru a adăuga funcționalitate componentelor dintr-o pagină web, limbajele folosite pentru construirea interfeței sunt HTML și JSX, acest limbaj din urmă este o combinație între HTML și JavaScript cu ajutorul căruia se construiesc componente menite să implementeze funcționalitatea dată de JavaScript. Limbajul CSS este folosit pentru stilizarea și poziționarea componentelor creeate înainte în paginile folosite. Această componentă comunică cu serverul prin intermediul protocolului HTTP.

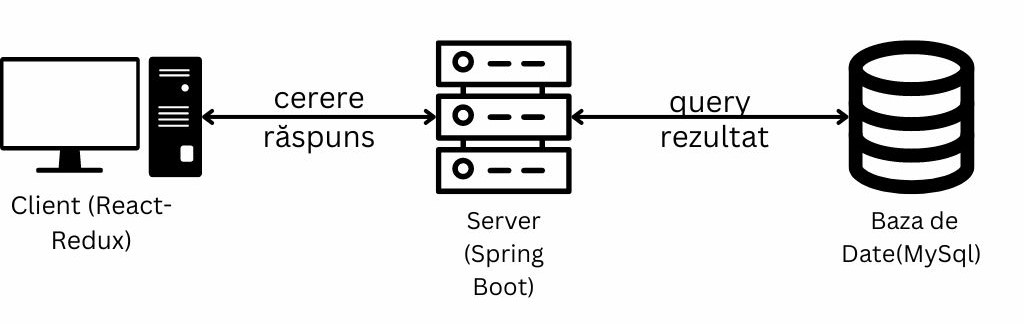


Figura 3.7 Arhitectura aplicației

### Serverul

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

Pentru a putea face ca codul scris în limbajul Java din componenta de back-end s-a folosit Apache Tomcat. Acesta este un server web și un container servlet deschis publicului larg/ open source care este foarte folosit pentru a ține o aplicație 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 și generează un raspuns dinamic. Tomcat gestionează ciclul de viață a acestor clase, incluzând inițializarea, tratarea cererilor și alocarea resurselor. Tomcat funcționând ca un server web poate să trateze cereri HTTP în mod direct.

Tomcat oferă un mod direct pentru implementarea și configurarea aplicaților web de care trebuie să se ocupe. Aplicațile pot să fie implementate prin copierea fișierelor WAR (Web Application Archive) în directorul potrivit. Tomcat detectează automat și implementează aceste aplicații. Pentru configurare instrucțiunile se scriu in fișierele server.xml și web.xml, aici se pot oferii detalii despre conectori și 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 construii automat codul proiectului și pentru a administra dependințele proiectului. Oferind un model standardizat pentru construirea proiectelor se realizează un mod organizat de a compila codul sursă, executa testele necesare și pentru folosiarea dependințelor. Conceptul pe care se bazează acest tool este Project Object Model (POM), este un fișier XML care este folosit pentru configurarea proiectului. În el se află instrucțiuni pentru Maven care arată modul în care proiectul să fie construit și modul în care să fie administrat mai departe. În pom.xml există informații cu privire la id-ul proiectului, id-ul artefactului, versiunea și tipul pachetului, în acest caz fiin JAR, astfel acești identificatori identifică în mod unic proiectul. În acest fișier se specifică și versiunea framework-urilor folosite, cum ar fi Spring Boot sau Spring Security și conectorul pentru baza de date.

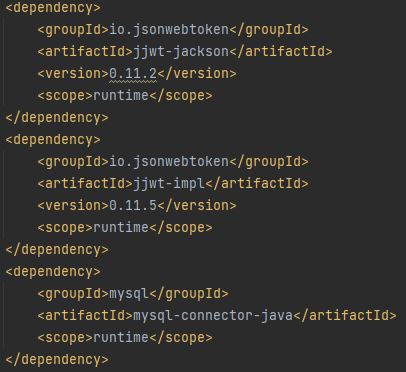


Figura 3.8 Dependențe pentru server

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

Prima componentă cu care s-a început este Model. Aici avem definite toate entitățile care fac parte din aplicația web. În Spring Boot o entitate se referă la o clasă Java care reprezintă obiect cu informații. Pentru a semnaliza pentru Spring Boot care clase sunt entități trebuie să folosim adnotarea @Entity deasupra numelui clasei. Cand Spring Boot gasește această adnotare construiește progarmul astfel încat 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 relațională (ORM) pentru a reprezenta obiecte Java în tabelele din baza de date relațională. Pentru fiecare din aceste entități urmează adnotarea “@Table”. Prin această adnotare cu atributul “name=” se specifică numele tabelului, în ghilimele, prin care urmează să fie reărezentată entitatea în baza de date astfel “@Table(name = “numele\_entității)”“. Primul atribut din orice entitate este identificatorul unic prin care proprietățile unei entități se inregistrează in baza de date. Acesta are numele de id și este de tipul “Long” pentru a putea stoca cat mai multe înregistrări unice în baza de date, acest tip de date nu suportă numerele cu virgula și este reprezentat pe 8 octeți. Pentru a semnaliza în Spring Boot că acesta este un identifictor unic se folosește adnotația “@Id”, dupa aceasta urmează adnotația care descrie modul în care se generează aceste valori id.

Modul de generare este specificat cu adnotația “@GeneratedValue” care primește 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 dupa o coloană identitate care se auto incrementează, cand o noua line cu informații se inserează în baza de date valoarea id-ului nou crește cu o unitate față de inserarea precedentă, prima inserare având valoarea unu, astfel se asigură ca nu se vor repeta valorile id-urilor într-un tabel.

Pentru restul proprietăților dintr-o entitate se specifică cu adnotația “@Column” numele sub care proprietatea respectivă se va înregistra în baza de date, de exemplu pentru “@Column(name = “nume”)” vom aveam o coloana în baza de date care se numește “nume”.

La fiecare proprietate dintr-o entitate se specifică numele, tipul de date (Long, Boolean sau String) și modificatorul de acces “private”. Acest modficator de acces restricționează vizibilitatea și accesibilitatea proprietăților. Modificatorul “private” face ca proprietatea să poată să fie accesată dor din aceeași clasă în care a fost declarată. În acest mod proprietățile unei clase nu se pot accesa în mod direct menținând integritatea datelor. Aceste proprietăți se pot accesa sau modifica doar prin intermediul unor metode speciale care se numesc “Getters” pentru metodele de accesare și “Setters” pentru metodele prin care se pot seta aceste proprietăți.

Pentru a creea automat aceste metode de acces s-a folosit adnotația “@Data” deasupra numelui clasei. Această adnotație face parte din librăria Lombok și generează metode de acces pentru toate prprietățile unei clase pentru a putea fi accesate și in exteriorul claselor în care au fost declarate.

Printre modelele entităților există și 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 asemanatoare ca valori și ca scop. Acest tip de date se poate definii asemănător cu o clasă pentru a organiza și a putea lucra cu o colocție de valori constante. În această aplicație 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 colecție de constamte.

În componenta model există două tipuri de de entități, cele care se referă tipurile de utilizatori, cum ar fi Farmacistul sau Administratorul și entitățile care se referă la obiectele cu care se lucreză în interiorul aplicației cum ar fi trimiterea sau prescrierea medicală. Diferența majoră dintre aceste doua 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 detalilel 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 gasim proprietatea în baza de date. În mod inplicit 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 usor de interpretat și pentru a îmbunătății lizibilitatea informaților stocate în baza de date s-a folosit adnotația “@Enumerated” în care sub formă de parametru se specifică tipul de date care se salveză, 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 sau rolul acestuia în aplicație, aceste informații sunt ecriptate 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. Informațile din acest token expiră după o anumită periodă de timp setată, acest fapt îmbunătățeste securitatea utilizatorilor deoarce și dacă acest token se află? pe langă faptul ca trebuie decodat acesta va expira și se generează unull nou, utilizatorul fiind nevoit să se reînregistreze 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 ca pentru o proprietate nu trebuie crreată o coloană nouă se folosește adnotarea “@Transient”. Această adnotație este folosită cand avem nevoie de accea valoare pentru anumite operații dar nu trebuie și salvată. Meotdele de acces și de actualizare a proprietăților pot să acceseze această proprietate în mod normal fără rastricții.

În continuare s-a creeat componenta care se ocupă de accesul la informațile din baza de date către întrega 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 informaților de logica lucuru a programului, oferă o cale modulară pentru interacțiunile cu baza de date. Beneficile acesti șablon este că încapsulează operațile 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 excepților.

Modelul Repository este un nivel de abstractizare pentru șablonul DAO, acesta oferă o interfață la nivel mai înalt pentru interacțiunea cu informațile din baza de date dar în același timp și simplifică aceste interacțiuni. Implicit Repository oferă un set de metode de bază CRUD (creat, read, update, delete) pentru operațiunile cu baza de date. Aceste metode de bază se creează printr-o convenție pentru denumirea metodelor dintr-un repository.

Pentru a putea folosii acest model de interacțiuni, în prpgram se creează interfețe pentru fiecare entitate facută inainte, pentru a organiza informațiile și metodele care țin de câte o entitate. Spring Boot poate să recunoască aceste interfețe ca fiind repository daca folosim adnotația “@Repository” arătând că în aceste interfețe se află metode pentru interacțiunea cu baza de date.

Metodele de bază CRUD se pot folosii în interfețele create daca extindem JpaRepository. Aceasta este o interfață din framework-ul Spring Data JPA care face parte din proiectul Spring Data. În această interfață avem metoda POST daca creeăm o metodă cu numele “save()”, avem o metoda de GET în funcție de cheia primară a unor entități prin numele “findById()” sau metoda DELETE după cheia primară prin metoda cu numele “deleteById()”. Aceste metode abstractizează complexitatea acestor operații de bază si se pot implementa dor prin numirea metodelor prin aceste regului. Această interfață oferă și posibilitatea de a folosii metode speciale pentru fiecare entitate care urmăresc aceleași reguli de numire și pentru proprietăți. Astfel dacă avem câmpul cnp în entitatea User putem să creeăm o metodă “findByCnp”, această metodă funcționează ca un GET care filtrează utilizatorii în funcție 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 entității dorite și tipul de date folosit de cheia primară care în cele mai multe cazuri este Long.

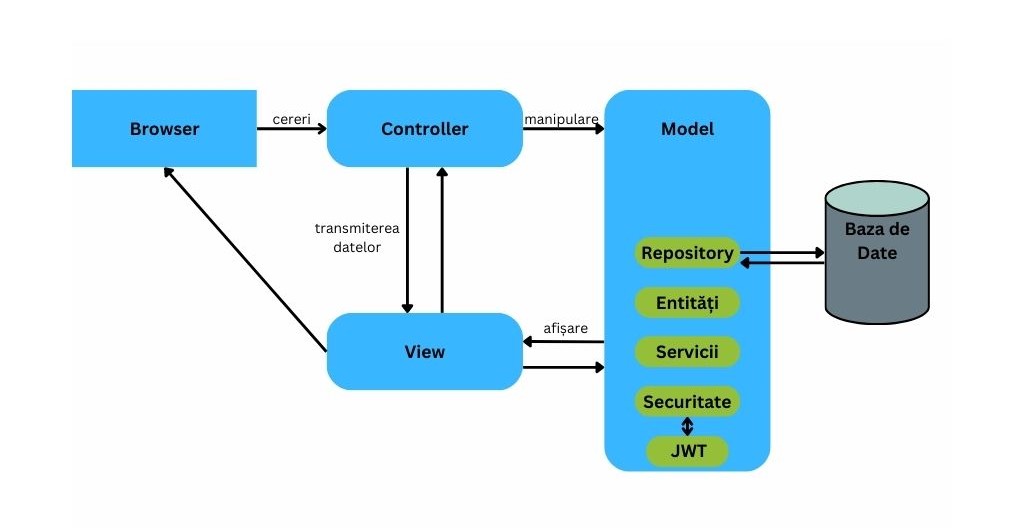


Figura 3.9 Componentele serverului

În momentul în care este nevoie de metode de interacțiune, cu informațile, mai avansate există adnotația “@Query”. Cu ajutorul acestei adnotații 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 și mai performante decat metodele de bază CRUD. În acest proiect s-au folosit astfel de metode în cea mai mare parte pentru funcția de actualizare, pentru a putea actualiza câmpuri specifice în funcție de cheia primară a utilizatorului curent. Parametrii folosiți in acest limbaj trebuie declarați în antetul metodelor prind adnotația “@Param”. Avem nevoie și de adnotația “@Modifying” pentru metodele care realizează operația de actualizare semnalând astfel în Spring Boot ca metoda schimbă anumite informații din baza de date.

Service este un nivel în care se află toate metodele care se ocupă cu logica de interacțiunie a datelor din aplicație, aici se folosesc funcțiile definite în componenta repository, prin intermediul acestor funcții, service are acces la informațiile din baza de date. În această componentă se definesc funcțiile care conțin metodele de interacțiune cu informațiile din baza de date. Service este o nivel care aparține de componenta model din arhitectura MVC. Pentru a îmbunătății modularitatea și refolosirea codului se creează aceast nivel, fiind un intermediar între componenta controller și nivelul de acces la informații, repository.

În general se creează un nou pachet pentru definirea claselor care compun nivelul service. La fel ca la repository și aici există câte o clasă care este dedicată pentru fiecare entitate creeată la început. În fiecare clasă există toate operațile complexe care se ocupă cu o anumită entitate. Creearea unui astfel de intermediar este benefică deoarece aici se efectuează validarea datelor, se aplică regului și constrângeri pentru a se asigura integritatea și corectitudinea informațiilor. În service se aplică și politicile de securitate dar și autentificare și autorizarea utilizatorilor. Pentru această aplicație există o clasă care se ocupă doar de autentificare utilizatorilor în funcție de entitatea de care aparțin, această clasă implementând metodele de autentificare cu ajutorul token-urilor definite în nivelul de securitate al aplicației. Folosind un nivel de service se ușurează gestioneara excepțiilor și gestionarea erorilor deoarce aici se pot converti în mesaje care oferă mai multe detalii despre erorile apărute.

În momentul în care se creează o clasă care ține de nivelul service aceasta trebuie sa aibă în componență adnotația “@Service”, care semnalează framework-ului Spring Boot că aceasta este o clasă care se ocupă de logica de utilizare a datelor în interiorul aplicației.

Pentru a putea folosii metodele deja definite în repository trebuie să importăm aceea interfață. Spring Boot oferă o metodă specială de importare care se numește injecția dependințelor (dependency injection). Acesta este un model de proiectare și un concept fundamental în dezvoltarea softwarre care promovează cuplarea liberă între componentele unui program și designul modular. Acest model implică oferirea dependențelor unui obiect care se află într-o sursă externă. Astfel obiectul nu trebuie să își gestioneze dependințele sau să le creeze, oferind o mai mare flexibilitatea și posibilitate de reutilizare a codului. Folosirea acestui model se poate realiza printr-un constructor sau prin metoda recomandată de Spring Boot printr-o adnotație. Implementarea cu ajutorul adnotației seamană cu declararea unei proprietăți, 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 nevie, de exemplu interfața creată pentru interacțiunea cu proprietățile din entitatea farmacist și numele variabilei care de obieci 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 multe metode fiecare având diferite seturi de instrucțiuni. Pentru a putea organzia mai usor aceste metode, pentru a se putea oberva 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 servicilor conține adnotația “@Override”. Această adnotație arată ca 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 celelate 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 creearea de servicii web pentru arhitectura RESTful deoarece nu mai trebuie ca fiecare metodă din controller să conțină adnotația ,,@ResponseBody” pentru răaspunsul din aplicație și este destul să existe adnotația ,,@RestController” pentru întrega 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 este transformat automat în formatul JSON (JavaScript Object Notation), format standardizat care se folosește pentru transmiterea informaților 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 creearea 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 definii 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 definii și parametrii necesari pentru funcționarea metodei sau headers pentru autorizare 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 penntru 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 de ce tip este în funcție de operațile 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 metodelor în care se pot include și parametrii necesari pentru apelarea corectă a metodelor. Pentru fiecare parametru specificat în URL trebuie să existe un parametru speficicat și în antetul metodei. Un parametru se specifică cu numele acestuia, care trebuie să coincidă cu numele specificat în URL, tipul acestuia cum ar fi: Long, String, Object și pentru a se specifica în Spring Boot ca este un parametru și ca 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 aplicației 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 serice, 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 metoe trebuie să interacționeze cu interfața utilizatorului este nevoie de astfel de clase care folosesc întregul răspuns HTTP. Pentru a putea folosii orice tip de date pentru corpul răspunsului fără a fi nevoie să fie specificat cu exacitate există posibilitatea de a fi declarat astfel: ,,ResponseEntity<?>”. În acest mod se îmbunătățește productivitatea pentru că se poate folosii 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 nicelul de serivice.

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

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

Aceste token-uri sunt structurate în trei părți, fiecare separate cu ,,.”, header, conținutul și semnătura. Toate aceste componente sunt encodate cu Base64 pentru o securitate îmbunătățită. În header se află tipul token-ului, în acest caz jwt și algoritmul folosit pentru partea de semnătură, cu ajutorul acestei componente se specifică cum se validează și cum trebuie procesat acest token. În partea de conținut se află toate datele utilizatorului, numele, id-ul și rolurile acestuia prin intermediul cărora funcționează permisiunile din program. Semnătura este creeată dintr-o combinație dintre header, conținut și o cheie secretă. Prin această componentă se asigură integritatea token-ului și asigură informațile aflate în conținut.

În momentul în care un utilizator intră în cont se apeleză interfața JwtProvider din pachetul de securitate pentru a genera un token nou. Metodele acestei interfețe sunt implementate într-o clasă cu același nume.

În clasa care generează aceste token-uri există doua variabile, prima este un string care se numește jwt\_secret unde prin intermediul adnotației ,,@Value)” se injectează o valoare din fișierul de proprietăți ale aplicației unde este specificată cheia pentru criptare și decriptare. Această aplicație folosește encodarea HMC care folosește aceeași cheie și pentru criptare și pentru decriptare. A doua variabilă din clasă este de tipul Long și la fel ca înainte se importă o valoare din fișierul de proprietăți ale aplicației. Aici se reține o valoare care este reprezentată în sistem în milisecunde și reprezintă perioada dupa care token-ul expiră și nu mai poate să fie folosit, după această perioadă utilizatorul trebuie sa reintre în cont pentru a putea folosii funcționalitatea aplicației. Este necesar un timp de expirare pentru token pentru a sporii securitatea și pentru a împiedica folosirea acestuia în cazul în care a fost obținut. În mod normal decodarea acestui token este imposibilă deoarce cheia necesară decodării se află în componenta server și nu există acces direct spre această componentă.

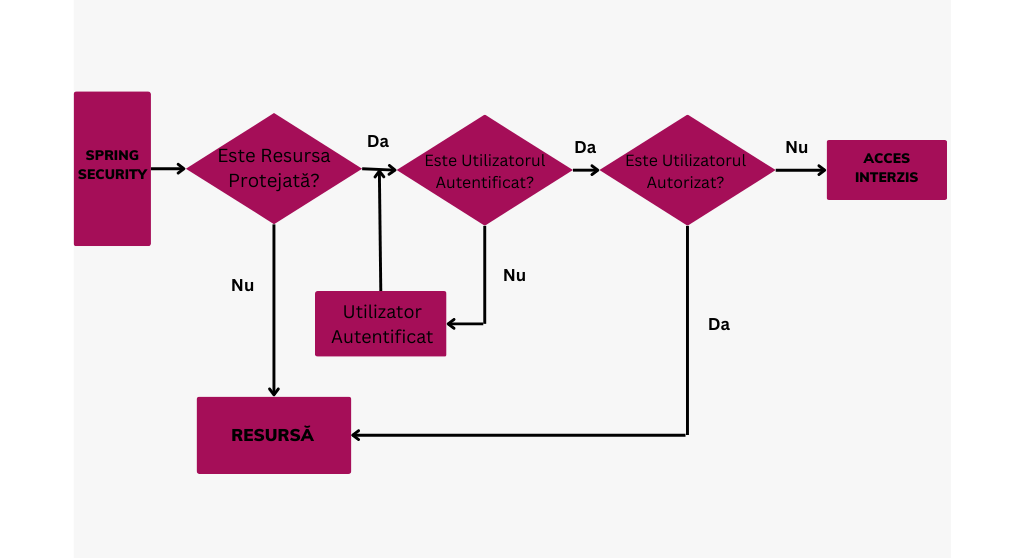


Figura 3.10 Logica Spring Security

Pentru entitățile care au nevoie de un cont, există clase numite Principle, acestea implementează interfața UserDetails car 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 aplicație și alte detalii despre utilizator. Asemănător claselor din nivelul service, aici la nivelul de securitate există clase de service și pentru informațiile despre utilizator. Aceste clase folosesc adnotația ,,@Service” și implementează interfața UserDetailsService care face parte din Spring Security. Singura metoda de implementat din această interfață este ,,loadUserByUsername” și primeste ca parametru username-ul utilizatorului. Această metodă caută utilizatorul în baza de date și returnează un ,,UserPrinciple” care este construit cu șablonul ,,Builder”. Aici sunt integrate informații despre utilizator cum ar fi parolă, id, username și nivelul de autoritate pe care îl are în aplicație.

Toate cererile HTTP sunt interceptate de clasa ,,JwtAuthorizationFilter” pentru a autoriza aceste cereri în funcție de token-ul jwt pe care îl conțin. Această clasă extinde superclasa ,,OncePerRequestFilter”. Aici se verifică token-urile din aplicație daca sunt valide, daca îndeplinesc condițile se adaugă în obiectul de autentificare și se pune în contextul oferit de Spring Security unde se află toate token-urile cu sesiuni active. Cand un token expiră acesta este șters din context și nu mai este cunoscut în interiorul aplicației.

### Client

Utilizatorul interacționează în mod direct cu componenta client, astfel trimițând anumite cereri către server pentru a creea seturi noi de informații, pentru a schimba unele informații sau pentru a primii informații deja existente din baza de date. Componenta client în cele mai multe cazuri este redată prin intermediul unui browser web.

Această componentă a fost creeată în editorul de cod Visual Studio Code cu ajutorul limbajului de programare JavaScript, cu HTML, limbaj folosit pentru construirea paginilor web și CSS, limbaj folosit pentru stilizarea componentelor din paginile web. O interfață web este constituită din tot ce poate să vadă utilizatorul pe ecran în momentul în care folosește aplicația. Obiectele cu care poate să interacționeze în interfață un utilizator sunt în general butoane, selectarea opțiunilor dintr-un dropdown sau selectarea anumitor checkbox-uri. Toate aceste interacțiuni creează o acțiune în componentele din interfață care mai apoi se transmite sub formă de cerere HTTP pentru server de la care se așteaptă un răaspuns 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 creearea interfețelor în aplicațile 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 refolosii cu difeite 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 creearea 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 separată deschisă în aplicație de către fiecare utilizator la un moment dat de timp.



Figura 3.11 Paginile utilizatorului

O intrfață construită în React utilizează o singură pagina de JavaScript pentru a renda 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 deoarcece 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 interfeței 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 interfața, pentru fiecare componentă. În acest mod se poate folosii doar una din aceste căii de acees la un moment dat pentru fiecare utilizator împarte și browser-ul știe exact de care pagină este nevoie pentru a fi rendată.

O parte din aceste URL-uri care reprezintă adresa pentru o anumită pagină din aplicație este nevoie de autorizare pentru a putea fi accesat doar de o anumită entitate sau pentru a bloc 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 alicației. În acest caz autoritatea se verifcă prin intermediul rolurilor pe care fiecare utilizator le primește în momentul în care își crează un cont în aplicație, 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 cunoaște 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 anumiți utilizatori din aplicație este necesar accesul la containerul centralizat numit store. Acest store face parte din librăria Redux. Se folosește componenta ,,Provider” pentru a oferii 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 crează toate rutele spre pagini.

Asemănător cu modelele din server și pentru partea de client se crează modele pentru anumite entități. Aceste modele se creează pentru entitățile care sunt reprezentate printr-un cont, de exemplu pacient sau medic de familie sau entitățile care se pot creea cum ar fi trimiterea sau prescrierea medicală. Modelele create în componenta client sunt în strânsă legătură cu modelel din server. Acestea se construiesc sub formă de constructori și poartă numele entității de care sunt legați. 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 proprietăție claselor.

Creearea containerului store se realizează în aplicație în directorul ,,redux-store”. În acest director se află mai multe fișiere, fiecare cu un rol bine definit. Într-un fișier se definesc tipurile de acțiuni posibile, aceste tipuri constau în setarea utilizatorului curent sau în ștergerea acestuia, penru fiecare rol posibil din aplicație. În fișierul pentru acțiuni se definesc obiecte JavaScript care reprezintă intenția de a schimba starea aplicației, aici reprzintă acțiunea de a seta utilizatorul curent sau de a fi șters în momentul în care acesta decide să iasă din cont sau token-ul expiră. În interiorul acestor obiecte se definește tipul obiectului, importat din fișierul în care sunt definite tipurile și pentru obiectele destinate setării utiizatorului are ca și conținut datele utilizatorului. Aceste obiecte definite ca acțiuni se folosesc mai departe în directorul pentru reducers. Aici se definesc funcțile care definesc modul în care aceste obiecte se folosesc. Aceste funcții primesc ca parametru o starea curentă a aplicației și o acțiune, in funcție de acești parametrii se decide modul în care următoarea stare a aplicației se actualizează.

De exemplu dacă starea curentă conține un utilizator conectat la aplicație si ca acțiune primște obiectul în care acesta este șters din container-ul store se va executa această acțiune, în cazul în care nu există un utilizator în aplicație și se primește acțiunea de a șterge utilizatorul curent dintr-o acțiune nu se va întâmpla nimic și se merge pe cazul implicit definit în funcție, astfel toate cazurile posibile de actualizare a stării aplicației sunt acoperite. La sfârșit toate aceste funcții sunt configurate penru a forma container-ul store. Acesta se formează cu ajutorul funcției definite în ,,redux-toolkit” și combină toți reducers pentru a se forma store.

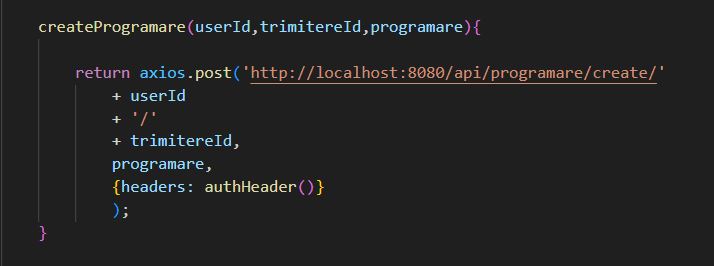


Figura 3.12 Metoda pentru creearea programărilor importată în front-end

Există un director denumit ,,service” care este asemănător cu cel din componenta server dar ca și comportament funcționează diferit. În directorul service sunt definite mai multe clase care au legătură cu tipurile de utilizatori din aplicație și cu modul de autentificare și 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 definește numele și parametrii necesari. Importarea se realizează prin librăria Axios. Această librărie se folosește pentru a face cereri HTTP și pentru operații asincrone, se bazează pe conceptul numit Promise. Astfel axios returnează un promise folosind metode ca și ,,.then()” pentru raspunsul primit și ,,.catch()” pentru cazul în care se primește o eroare. Axios oferă suport pentru toate metodele de bază CRUD. În momentul în care se face importul se precizează metoda CRUD și URL metodei definit în server, împreună cu parametrii și cu portul pe care serverul transmite informațile, în acest caz portul 8080. La final se definesc headers unde se folosește autorizarea prin intermediul token-urilor.

Majoritatea codului pentru interfață 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 conține anumite componente definite în react care se numesc ,,Hooks”. Aceste componente ajută la construirea acestor pagini și la modul de gestionarea a informaților care se folosesc. Fiecare câmp care conține informație sau așteaptă să primească informații 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 își primește valoarea printr-o funcție proprie care se ocupă doar cu schimbarea informației din acea variabilă.

Un alt ,,hook” folosit des în aplicație este ,,useEffect()” , acesta aduce informațile de la server către client în momentul în care o pagină nouă este accesată sau un buton este apăsat. Informațile sunt primite printr-o metodă definită pentru diferite tipuri de informații în fiecare serviciu și 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 in componentele pe care le conține și astfel se populeză un tabel. Un astfel de procedeu este folosit pentru trimiterile sau prescrierile medicale.

Pentru paginile care sunt destinate creării unei trimiteri, prescrieri sau programări se folosește o interfață sepcială numită ,,Modal”. Aceasta este este o componentă care este afișată peste interfața deja existentă. Aceasta este folosită pentru a completa o componentă de tipul formular. Aceste modele sunt definite separat de paginile aplicației deoarce nu sunt pagini de sine stătătoare. Se apelează din paginile destinate creării de astfel de formulare și se instanțiază ca o componentă care primește parametrii din variabilele primite de la server, acestea fiind informații diferse sau date despre utilizatorul curent, dar și o funcție care este destinată deschiderii acestor modele speciale de pagini. În funcție de rolul lor se apelează cu o referință care este dedicată creării unui nou formular sau destinat pentru vizionarea unui formular. Această referință din urmă se folosește pentru paginile care au doar drept de vizualizare și nu se pot schimba informațile, cum ar fi un pacient care dorește să îsi vadă trimiterea primită de la doctorul de familie.

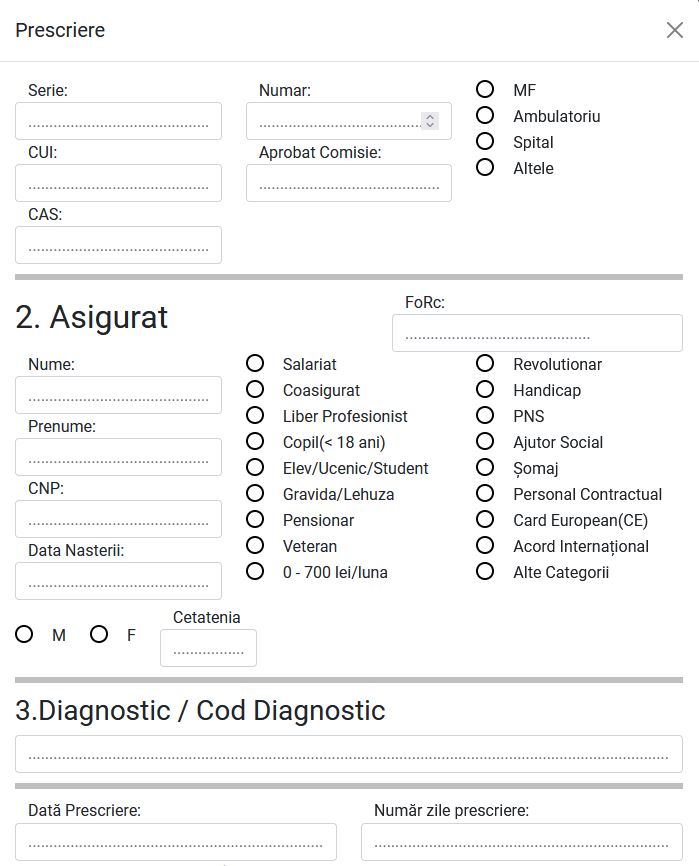


Figura 3.13 Formularul pentru prescriere

### Baza de date

Baza de date în care se stochează informațile în această aplicație a fost creeată cu ajutorul serverului MySql. Această bază de date este una relațională însemnând că informația este organizată în tabele în care pot să formeze relații între aceste informații. În aplicație există informații numite ,,one-to-one” între o programare și o trimitere, într-o programare poate să existe doar o trimitere și o trimitere poate să se afle doar într-o programare. Relația de ,,one-to-many” apare la medicii de familie și pacienți, unde un medic de familie poate să aibă mai mulți pacienți dar un pacient poate să aparțină 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 șterge informații pentru testarea aplicației. După ce fiecare entitate a fost definită și primește toate adnotațile necesare, în momentul în care se pornește componenta server prin intermediul web server-ului Tomcat câte un tabel pentru fiecare entitate se definește automat în baza de date cu atributele specificate în editorul IntelliJ. Această automatizare se datorează framework-ului Hibernate.

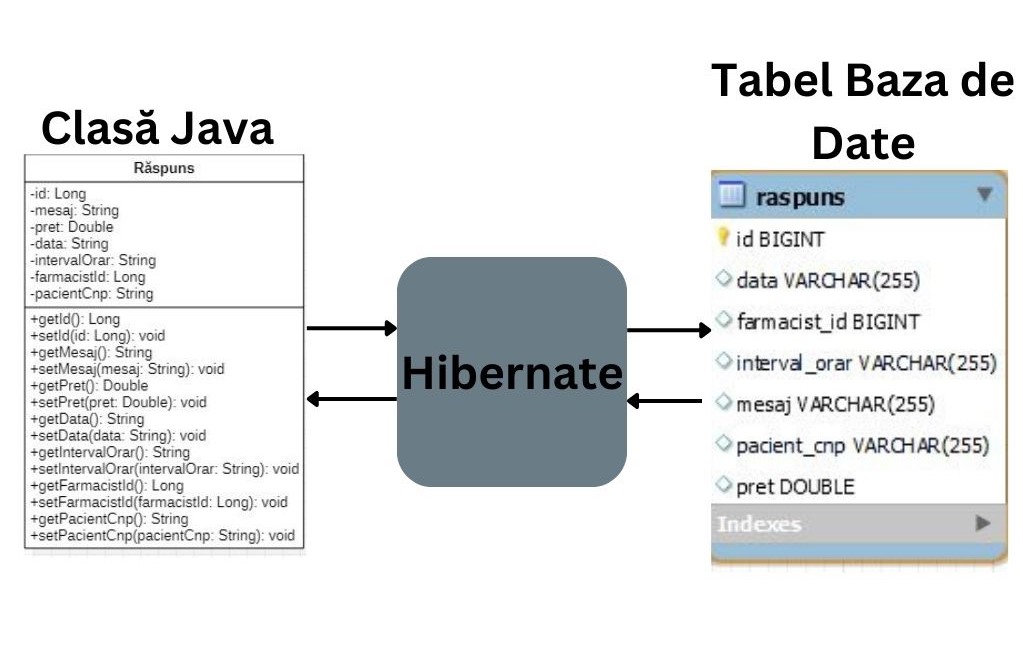


Figura 3.13 Modul de funcționare Hibernate

Hibernate este un framework bazat pe limbajul Java care simplifică modul în care se interacționează cu bazele de date relaționale în aplicațile 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 îmbunătățește performanța prin reducerea de informații care se obține 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 relațile dintre acestea în timp ce acestea interacționeză cu informațile din tabelele din baza de date. Astfel modul de inițializare pentru interacțiunile dintre Hibernate și baza de date dar și conexiunea cu aceasta se realizează în directorul numit ,,resources”, în fișierul cu proprietățile aplicației. Pe prima linie se specifică URL-ul pentru sursa de informații folosită. Se specifică serverul, în acest caz mysql, adresa acestuia, dacă se află pe același dispozitiv cu servrul atunci este localhost, numărul portului care în mod implicit pentru MySql este 3306 și numele bazei de date. Mai departe în acest URL se specifică că se dorește creearea bazei de date dacă aceasta nu există deja astfel se automatizează modul în care se creează baza de date și acest lucru se realizează prima data cand se pornește 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 și parola pentru serverul în care este conținută baza de date pentru a se putea conecta.

Împreună cu Hibernate lucrează și JPA (Java Persistance API) care este un standard pentru bazele de date relaționale. Jpa oferă un set de interfețe și adnotații care definesc un API pentru accesarea și gestionarea bazelor de date relaționale în aplicațille Java. Acest standard oferă metode și API-uri pentru a realiza operațiile de bază CRUD cu informațile din baza de date. Astfel se pot extrage sau creea informații pe baza anumitor proprietăților din entități care sunt transformate în tabele iar proprietățile în câmpuri care compun aceste tabele.

# Testare

Testarea într-o aplicație web se ocupă de evaluarea mai multe aspecte din aplicație cum ar fi calitatea, securitatea și performanța. Există mai multe tipuri de testare, fiecare fiind menită pentru testarea unui comportament specific a aplicației. Pentru această aplicație s-a folosit testarea funcțională pentru testarea diferitelor funcționalități cum ar fi interacțiunile utilizatorului cu diferite pagini web sau modul în care utilizatorii cu rol de doctor interacționează cu formularele. Testarea securității constă în verificare funcționalității pentru atribuirea corectă a autorizaților de accesare a resurselor și validarea token-urilor de sesiune pentru utilizatori.

## Testarea server-ului

Testarea server-ului se ocupă în cea mai mare parte cu verificarea funcționării corecte a tuturor metodelor create penru operarea logicii întregii aplicații. Aici se testează și funcționarea corectă pentru baza de date și comunicarea dintrea aceasta și server pentru obținerea informaților.

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 și testarea API-urilor. Cu ajutorul acestei platforme se poate simula componenta client și astfel se pot trimite cereri HTTP și se pot primii răspunsuri de la server în funcție de metodele apelate, se oferă și suport penru utilizarea token-urilor de autentificare deci se poate testa și securitatea metodelor.

Testarea acestei aplicații web în Postman se realizează prin intermediul a doua metode. În prima metodă se testează fiecare metodă manual și se verifică răspunsul și codul HTTP primit pentru a se putea valida funcționalitatea.

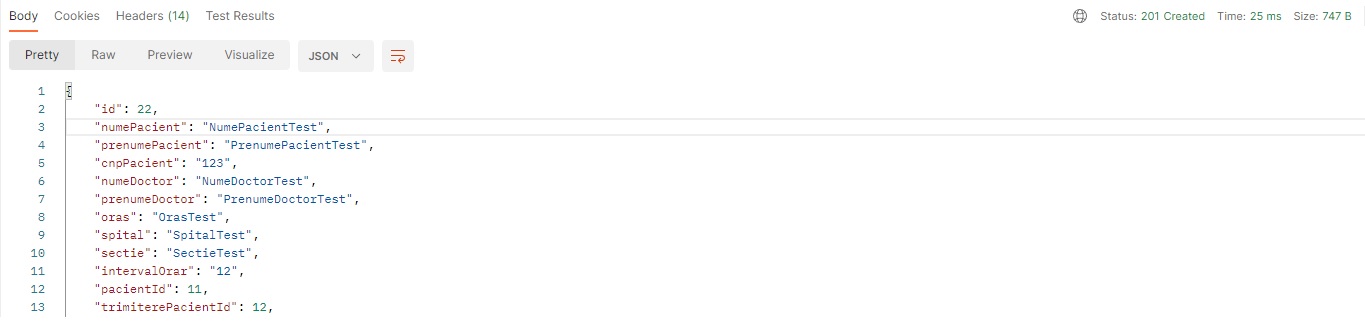


Figura 4.1 Răspunsul din postman

În această poză se poate vedea răspunsul primit pentru metoda de creeare a unei programări de către un utilizator. Metoda se aplelează de pe portul 8080 unde funcționează serverul și se continua cu URL-ul metodei, în acest caz ,,/api/programare/create”. În URL-ul metodei se specifică și id-urile pentru userul care creează programarea (11) și id-ul trimiteri pe care trebuie să o conțină programare (12), toate restul informaților trimise pentru corpul programării sunt date de test. În partea dreaptă se vede codul primit ca răspuns de la server, timpul de exectuție a metodei și dimensiunea informaților. Cu ajutorul codului primit din răspuns se constată ca metoda funcționează corect, ,,201” arătând ca cererea s-a realizat cu succes și resursa a fost creeată cu datele arătate în Postman. Daca metoda funcționează corect se poate verifica și în baza de date daca înregistrarea s-a făcut corect. Asemănător cu această metodă se testează și pentru celelalte metode pentru a obține, a actualiza sau pentru a creea 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 asa ceva care rulează testele de fiecare data cand se face o cerere în Postman și se primește răspunsul.

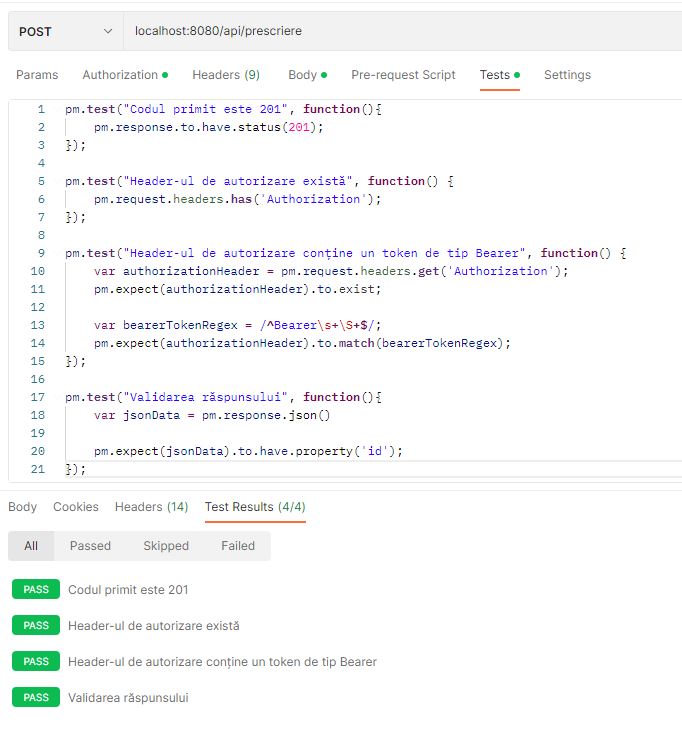


Figura 4.2 Testarea în postman

Cu ajutorul acestor teste se verifică daca codul primit este corect, aici se testează metoda care trebuie să creeze o prescriere și se așteptă codul ,,201” care arată ca o resursă nouă a fost creeată. 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 conține și rolul acestuia. La verificarea existenței token-ului se verifică daca primul cuvânt este ,,Bearer”, acesta indicând tipul de token. Această aplicație este construită astfel îmcât să accepte doar token-uri de tip ,,Bearer”. Ultimul test verifică daca răspunsul primit conține proprietatea ,,id” aceasta fiind identificatorul unic pentru fiecare înregistrare din baza de date. În interiorul acestui test se adaugă verificare diferitelor proprietăți în funcție de răspunsul care se așteptă de la fiecare metodă.

## Testarea Clientului

Pentru testarea clientului s-au testat componentele care formează interfața utilizatorului dar și funcționalitatea componentelor. Prima parte din această testare constă în verificarea modului în care se așează componentele în pagina web pentru diferite tipuri și mărimi de ecrane. Aici se verifică dacă marginile și spațile din pagină se paăstrează chiar dacă aplicația este folosită cu un ecran mai mare sau mai mic. Se verifică și daca informația este reprezentată corect în tabele, astfel încât datele de pe o coloană să nu depăsească o anumită limită și să ajungă ca poziționare într-o altă coloană.

După ce se testează aspectul și modul de așezare a componentelor se trece la verificarea ca fiecare tip de utilizator să poată vizualiza doar interfața destinată acestuia. De exemplu un medic de familie care are opțiunea de a creea trimiteri medicale nu trebuie să primească și opțiunea de a creea și prescrieri medicale, acestea fiind dedicate doar pentru medicii specialiști. Fiecare tip de utilizator are la dispoziție un set diferit de butoane care duc la pagininile pe care doar acesta le poate accesa.

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

# Concluzii

## Rezultate obținute

Această aplicație web a fost realizată pentru digitalizarea procesului în care funcționeză 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 face 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 pacientilor 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 fac realizează programările prin intermediul aplicației depind 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 după ce a fost investigat de un medic specialist

## Direcții de dezvoltare

De la acest circuit de bază realizat în aplicație se poate dezvolta mai departe în funcție de nevoile pacienților, a medicilor sau a farmaciștilor. Mai departe se pot digitaliza mai multe proceduri deja existente până ar rămâne ca interacțiune fizică doar momentul în care doctorul specialist consultă pacientul.

O direcție de dezvoltare ar fi implementarea unui sistem centralizat de plăți cu cardul pentru pacienți pentru a 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 imobilul în care locuiesc 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 ce fel de intervenție a fost și cauza acesteia. Mai departe 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.

## Referințe bibliografice

Se recomandă ca citarea referințelor bibliografice să fie făcută în formatul IEEE.

În secțiunea Bibliografie sunt prezentate exemple pentru: o citare a unui capitol dintr-o carte [1], un articol publicat într-o revistă [2] și un articol publicat la o conferință [3].

Detalii cu privire la formatul citării diverselor tipuri de referințe pot fi găsite în [4] sau [5].

Referințele bibliografice se pot insera în text utilizând facilitățile Word de a adăuga surse și bibliografie unui document (References -> Citations & Bibliography). Dacă formatul IEEE pentru bibliografie nu este instalat implicit în Word, se poate descărca gratuit de la:

<https://bibword.codeplex.com/wikipage?title=Styles&referringTitle=Home>

Instrucțiunile de instalare pentru diferite versiuni de Word se pot obține de la aceeași adresă.

# Bibliografie

|  |  |
| --- | --- |
| [1] | P. Nume, "Titlul capitolului," în *Titlul cartii*, Oras, Editura, 2016, pp. 1-24. |
| [2] | P. Nume, "Titlul articolului," *Titlul revistei,* vol. 1, no. 2, pp. 22-30, 2016. |
| [3] | P. Nume, "Titlul articolului," în *Numele conferintei*, Oras, 2015. |
| [4] | "IEEE Citation Reference," 2009. [Online]. Available: https://www.ieee.org/documents/ieeecitationref.pdf. |
| [5] | "IEEE Editorial Style Manual," 2016. [Online]. Available: https://www.ieee.org/documents/style\_manual.pdf. |