|  |  |  |
| --- | --- | --- |
|  | **UNIVERSITATEA „VASILE ALECSANDRI” DIN BACAU**  **Facultatea de Stiinte**  **Str. Calea Marasesti, nr. 157, Bacau, 600115**  **Tel. ++40-234-542411, tel./ fax ++40-234-571012** [www.ub.ro](http://www.ub.ro/); e-mail: [stiinte@ub.ro](mailto:stiinte@ub.ro) |  |

**PROGRAMUL DE STUDII INFORMATICA-IFR**

**LUCRARE DE LICENTA/DISERTATIE**

**Coordonator stiintific: Absolvent:**

Lector univ. dr. Iulian Marius Furdu Tanase Mihai

**Bacau**

**2023**

|  |  |  |
| --- | --- | --- |
|  | **UNIVERSITATEA „VASILE ALECSANDRI” DIN BACAU**  **Facultatea de Stiinte**  **Str. Calea Marasesti, nr. 157, Bacau, 600115**  **Tel. ++40-234-542411, tel./ fax ++40-234-571012** [www.ub.ro](http://www.ub.ro/); e-mail: [stiinte@ub.ro](mailto:stiinte@ub.ro) |  |

**PROGRAMUL DE STUDII INFORMATICA-IFR**

**Gestionarea programarilor online la cabinetele medicale**

**Coordonator stiintific: Absolvent:**

Lector univ. dr. Iulian Marius Furdu Tanase Mihai

**Bacau**

**2023**

Table of Contents

[www.ub.ro; e-mail: stiinte@ub.ro 1](#_Toc151640674)

[www.ub.ro; e-mail: stiinte@ub.ro 2](#_Toc151640675)

[Introducere 5](#_Toc151640676)

[Contextul si importanta gestionarii online a programarilor 5](#_Toc151640677)

[Problema si obiectivele studiului 5](#_Toc151640678)

[Solutii similare 6](#_Toc151640679)

[Prezentare Solutii Similare 6](#_Toc151640680)

[Asemanari dintre solutiile si aplicatia dezvoltata 7](#_Toc151640681)

[Asemanari dintre solutiile si aplicatia dezvoltata 7](#_Toc151640682)

[Despre Aplicatie 8](#_Toc151640683)

[Arhitectura backend 8](#_Toc151640684)

## 

# Introducere

## Contextul si importanta gestionarii online a programarilor

Intr-o era dominata de progresul tehnologic, digitalizarea serviciilor a devenit esentiala in numeroase domenii, inclusiv in cel medical. Dezvoltarea sistemelor de programare online raspunde acestei tendinte, oferind solutii eficiente pentru gestionarea interactiunilor dintre pacienti si furnizorii de servicii medicale.

Un astfel de sistem, orientat spre facilitarea rezervarilor medicale, ilustreaza cum inovatiile in tehnologia informatiei pot revolutiona procesele traditionale. Platforma permite utilizatorilor sa efectueze programari, modificari sau anulari ale vizitelor medicale intr-un mod simplu si eficient, depasind limitarile si ineficientele interactiunilor telefonice.

Optimizarea gestionarii timpului este un element cheie al acestui sistem. In domeniul medical, unde eficienta timpului este cruciala, imbunatatirea procesului de programare poate avea un impact major asupra calitatii ingrijirii. Reducerea sarcinilor administrative permite personalului medical sa se concentreze mai mult pe ingrijirea directa a pacientilor.

Sistemul de programare online aduce, de asemenea, un plus de accesibilitate. Oferind o alternativa digitala la metodele traditionale, platforma este o solutie convenabila pentru pacientii care prefera sau necesita mijloace digitale pentru gestionarea serviciilor medicale. Aceasta abordare este extrem de relevanta in contextul actual, unde tehnologia digitala capata o prezenta tot mai mare in viata cotidiana.

Din perspectiva tehnica, crearea unui astfel de sistem implica provocari specifice. Asigurarea securitatii datelor personale ale pacientilor este o prioritate, iar implementarea unor masuri de securitate robuste este esentiala. De asemenea, dezvoltarea unei interfete utilizator intuitive si eficiente este cruciala pentru asigurarea unei experiente utilizator pozitive.

In concluzie, dezvoltarea sistemelor de programare online in domeniul medical reprezinta un pas important spre modernizarea accesului la servicii medicale. Aceasta constituie nu doar o evolutie tehnologica, ci si o adaptare la nevoile si asteptarile societatii contemporane, deschizand calea catre un sistem de sanatate mai eficient si accesibil.

## Problema si obiectivele studiului

Problema Identificata:

In lumea medicala contemporana, unul dintre cele mai mari obstacole este gestionarea eficienta a programarilor. Sistemele traditionale, bazate pe apeluri telefonice si inregistrari manuale, sunt adesea ineficiente, conducand la pierderi de timp si resurse, atat pentru pacienti, cat si pentru personalul medical. Aceasta ineficienta poate duce la intarzieri in accesul la ingrijire medicala si la o utilizare suboptimala a resurselor clinice.

Obiectivele Studiului:

Analiza Necessitatii Digitalizarii: Explorarea modului in care digitalizarea proceselor de programare poate imbunatati eficienta in clinici si spitale. Studiul va evalua impactul sistemelor digitale asupra timpului de asteptare, satisfactiei pacientilor si eficientei administrative.

Dezvoltarea unei Solutii Tehnologice: Crearea unui sistem de programare online care sa adreseze problemele identificate. Obiectivul este de a dezvolta o platforma user-friendly, care sa permita pacientilor sa faca, sa modifice sau sa anuleze programari cu usurinta, imbunatatind astfel accesul la servicii medicale.

Securitatea Datelor: Asigurarea securitatii si confidentialitatii datelor personale ale pacientilor. Sistemul va incorpora tehnologii avansate pentru a proteja informatiile sensibile si pentru a respecta reglementarile privind protectia datelor.

Evaluarea Impactului Sistemului: Analiza modului in care implementarea sistemului de programare online influenteaza fluxul de lucru in cadrul institutiilor medicale. Studiul va include feedback-ul pacientilor si al personalului medical pentru a evalua eficacitatea si acceptarea solutiei propuse.

Sustenabilitatea si Scalabilitatea: Explorarea posibilitatilor de extindere si adaptare a sistemului pentru a raspunde nevoilor in continua schimbare ale sectorului medical. Sistemul trebuie sa fie flexibil si capabil sa se adapteze la diverse scenarii clinice si la evolutia tehnologica.

Prin abordarea acestor obiective, studiul isi propune sa contribuie la optimizarea proceselor de programare in domeniul medical, oferind o solutie tehnologica care sa faciliteze accesul la ingrijiri de calitate si sa imbunatateasca eficienta operationala a institutiilor medicale.

# Solutii similare

## Prezentare Solutii Similare

In contextul dezvoltarii sistemelor de programare online in domeniul medical, exista mai multe solutii similare care merita analizate. Acestea ofera perspective diferite asupra modului in care tehnologia poate fi utilizata pentru a imbunatati experienta pacientilor si eficienta administrativa in clinici si spitale.

Regina Maria: Unul dintre cele mai cunoscute sisteme de programare online din Romania, Regina Maria ofera o platforma complexa si bine integrata. Pacientii pot folosi website-ul sau aplicatia mobila pentru a gasi medici, a verifica disponibilitatea acestora si a programa vizite. Sistemul include si functii de gestionare a istoricului medical si de acces la rezultatele analizelor, ceea ce il face un exemplu relevant de solutie complexa si centrata pe pacient.

Zocdoc: Aceasta platforma internationala permite pacientilor sa caute medici in functie de specializare, locatie si asigurare de sanatate. Zocdoc ofera o interfata prietenoasa si eficienta, cu posibilitatea de a vedea recenziile altor pacienti, ceea ce ajuta la luarea unei decizii informate.

Doctolib: Sistemul francez Doctolib este un alt exemplu de platforma de programare online care a castigat popularitate datorita usurintei de utilizare si a interfetei intuitive. Ofera functii similare cu Zocdoc, inclusiv posibilitatea de a lasa recenzii si de a accesa istoricul programarilor.

## Asemanari dintre solutiile si aplicatia dezvoltata

In comparatie cu solutiile existente pe piata, cum ar fi Regina Maria, Zocdoc si Doctolib, aplicatia dezvoltata prezinta cateva asemanari notabile:

* Interfata Utilizator Prietenoasa: Similar cu celelalte platforme, aplicatia noastra pune accent pe o interfata simpla si intuitiva, facilitand accesul rapid la informatii si programari.
* Functionalitati de Programare Online: Ca si in cazul altor solutii, aplicatia permite utilizatorilor sa efectueze programari online, oferind o alternativa eficienta la metodele traditionale.
* Gestionarea Datelor Pacientilor: Aplicatia include functii pentru stocarea si gestionarea datelor personale si medicale ale pacientilor, asigurand confidentialitatea si securitatea acestora.
* Accesibilitate: La fel ca si alte platforme, aplicatia este accesibila de pe diverse dispozitive, inclusiv smartphone-uri si tablete, oferind flexibilitate in utilizare

## Asemanari dintre solutiile si aplicatia dezvoltata

Desi exista asemanari, aplicatia dezvoltata se distinge prin cateva caracteristici unice:

* Personalizare Avansata: Aplicatia ofera optiuni de personalizare mai avansate, permitand utilizatorilor sa isi adapteze experienta in functie de preferintele personale.
* Integrare cu Servicii Locale: Spre deosebire de alte platforme, aplicatia este special conceputa pentru a se integra eficient cu serviciile medicale locale, oferind informatii actualizate si relevante pentru utilizatori.
* Feedback si Evaluare: Aplicatia incorporeaza un sistem de feedback mai detaliat, permitand pacientilor sa ofere recenzii si sugestii specifice, contribuind astfel la imbunatatirea continua a serviciilor.
* Suport pentru Decizii Bazate pe Date: Un alt aspect distinctiv este utilizarea datelor analitice pentru a sprijini deciziile medicale si administrative, oferind astfel o valoare adaugata atat pentru pacienti, cat si pentru furnizorii de servicii.

# Despre Aplicatie

## Despre tehonlogiile folosite

**JavaScript**, cunoscut initial sub numele de Mocha, a fost creat de Brendan Eich in 1995 pentru Netscape Navigator. Scopul initial al limbajului era de a adauga interactivitate paginilor web, ceea ce era o noutate la acea vreme. JavaScript a evoluat rapid, devenind un standard de facto pentru scripting-ul pe partea de client in navigatoarele web.

In anii urmatori, JavaScript a devenit un limbaj tot mai popular, datorita flexibilitatii si usurintei de utilizare. Cu aparitia AJAX (Asynchronous JavaScript and XML) la inceputul anilor 2000, JavaScript a inceput sa fie folosit pentru crearea de aplicatii web mai complexe si dinamice.

Un moment important in istoria JavaScript a fost lansarea Node.js in 2009. Node.js a permis rularea JavaScript-ului pe server, extinzand utilizarea limbajului dincolo de browser. Acest lucru a deschis calea pentru JavaScript sa devina un limbaj full-stack, utilizat atat pentru frontend, cat si pentru backend.

Node.js a adus mai multe avantaje pentru dezvoltarea backend, inclusiv posibilitatea de a folosi acelasi limbaj atat pe client, cat si pe server, ceea ce a simplificat dezvoltarea si a imbunatatit eficienta. De asemenea, modelul sau non-blocking si orientat pe evenimente s-a dovedit a fi eficient pentru aplicatii web care necesita o mare scalabilitate si performanta.

In prezent, JavaScript este unul dintre cele mai utilizate limbaje de programare din lume, cu o comunitate vasta si o multime de biblioteci si cadre de lucru. Este folosit intr-o varietate larga de aplicatii, de la site-uri web simple la aplicatii complexe de server si chiar in dezvoltarea de aplicatii mobile si desktop.

In ceea ce priveste aplicatia backend mentionata, dezvoltarea acesteia in JavaScript, probabil folosind Node.js, este o alegere comuna in zilele noastre. Aceasta permite dezvoltatorilor sa creeze aplicatii robuste, eficiente si usor de intretinut, beneficiind de avantajele pe care JavaScript si ecosistemul sau le ofera.

**Express.js** este un framework pentru aplicatii web Node.js, recunoscut pentru simplitatea, flexibilitatea si scalabilitatea sa. Lansat in 2010, a devenit rapid unul dintre cele mai populare cadre de lucru pentru dezvoltarea backend in JavaScript.

Caracteristici Principale

Middleware: Express.js utilizeaza middleware pentru a procesa cererile. Middleware-urile sunt functii care au acces la obiectul cererii (req), obiectul raspunsului (res) si urmatoarea functie middleware in ciclul cerere-raspuns al aplicatiei. Acestea pot executa cod, modifica obiectele cerere si raspuns, incheia ciclul cerere-raspuns sau apela urmatoarea functie middleware.

Routing: Express ofera un sistem robust de rutare care permite definirea de rute pentru cererile HTTP. Acest lucru ajuta la gestionarea actiunilor in functie de URL si de metoda HTTP (GET, POST, etc.).

Performanta: Fiind construit pe Node.js, Express.js beneficiaza de performanta ridicata oferita de mediul de executie bazat pe evenimente si non-blocking al Node.js.

Simplu si Flexibil: Express nu impune o structura stricta, lasand dezvoltatorii liberi sa aleaga arhitectura si pachetele suplimentare conform necesitatilor proiectului.

Utilizare in Dezvoltarea Web

Express.js este folosit pentru a construi o varietate de aplicatii web, de la pagini simple, bloguri si portofolii, pana la aplicatii web complexe si API-uri RESTful. Este deosebit de popular in randul dezvoltatorilor care folosesc JavaScript pe intregul stack, permitandu-le sa foloseasca acelasi limbaj atat pe frontend, cat si pe backend.

Comunitate si Ecosistem

Unul dintre cele mai mari avantaje ale Express.js este comunitatea vasta si activa. Exista numeroase resurse, tutoriale si module terte parti disponibile, ceea ce face ca invatarea si utilizarea Express.js sa fie accesibila pentru dezvoltatori de toate nivelurile.

## Arhitectura backend

Backend-ul proiectului "Medical Booking" este construit folosind Node.js si Express, oferind o structura robusta si flexibila pentru gestionarea programarilor medicale online. Acesta este proiectat pentru a oferi functionalitati esentiale precum autentificarea utilizatorilor, gestionarea programarilor si a datelor utilizatorilor.

**1. Structura Generala si Configuratia Initiala**

app.js: Acest fisier serveste ca punct de intrare in aplicatie. Aici, serverul Express este configurat, inclusiv setarea middleware-urilor esentiale precum CORS pentru gestionarea cererilor cross-origin si cookie-parser pentru manipularea cookie-urilor. De asemenea, aici sunt definite rutele principale ale aplicatiei si se initiaza serverul.

src: Directorul 'src' este nucleul backend-ului, continand subdirectoarele si fisierele esentiale pentru functionarea aplicatiei.

Descrierea dependintelor(pachetelor) folosite in partea de backend:

**1.1 bcrypt (versiunea 5.1.1)**

Descriere: bcrypt este un pachet folosit pentru hash-uitul parolelor.

Ce Inseamna Hashing: Hashing-ul este un proces prin care o intrare (de exemplu, o parola) este convertita intr-o sir de caractere de lungime fixa, indiferent de lungimea intrarii initiale. Aceasta transformare este realizata de o functie de hash. Caracteristicile importante ale hashing-ului sunt unicitatea (doua intrari diferite nu ar trebui sa produca acelasi hash) si ireversibilitatea (nu se poate obtine intrarea initiala din hash). Hashing-ul este esential in securitatea informatiilor, fiind folosit pentru a proteja parolele si alte date sensibile. In contextul securitatii, un bun algoritm de hash trebuie sa fie rezistent la coliziuni (doua intrari diferite sa nu produca acelasi hash) si sa fie dificil de spart prin metode de forta bruta sau alte tehnici de hacking.

Utilizare: Folosit pentru hash-uitul si verificarea parolelor utilizatorilor.

Avantaje: Ofera un nivel ridicat de securitate, rezistent la atacuri brute-force.

Dezavantaje: Procesarea poate fi mai lenta comparativ cu alte metode de hash, ceea ce poate afecta performanta la volume mari de autentificari simultane.

**1.2. cookie-parser (versiunea 1.4.6)**

Descriere: cookie-parser este un middleware pentru Express care parseaza cookie-urile atasate cererilor HTTP.

Ce Inseamna Middleware: In dezvoltarea web, un middleware reprezinta un bloc de cod care se executa intre primirea unei cereri HTTP de la client (de exemplu, un browser web) si trimiterea unui raspuns de catre server. Middleware-urile sunt folosite pentru a executa diverse operatiuni cum ar fi verificarea autentificarii, logarea, manipularea datelor cererii sau raspunsului etc. In esenta, ele actioneaza ca un intermediar, procesand cererile inainte ca acestea sa ajunga la logica principala a aplicatiei sau dupa ce logica a fost executata, dar inainte de a trimite raspunsul inapoi la client.

Utilizare: Parseaza cookie-urile din request-uri HTTP pentru a usura gestionarea sesiunilor prin token-ul JWT

Avantaje: Simplifica lucrul cu cookie-uri, facilitand gestionarea autentificarii si a sesiunilor.

Dezavantaje: Nu ofera functionalitati de securitate pentru cookie-uri, necesitand implementari suplimentare pentru protectie.

**1.3. cors (versiunea 2.8.5)**

Descriere: cors este un pachet pentru gestionarea politicii de Same-Origin in aplicatiile web

Ce Inseamna CORS (Cross-Origin Resource Sharing): CORS este un mecanism care permite sau refuza resursele solicitate pe un server web, in functie de originea cererii. In contextul securitatii web, o politica stricta de same-origin previne un site web sa faca cereri catre un domeniu diferit de cel de unde a fost incarcat. CORS este o tehnica utilizata pentru a relaxa aceste restrictii, permitand cereri intre diferite origini, in conditii controlate. Acest lucru este crucial pentru aplicatiile web moderne, care adesea necesita incarcarea resurselor de pe diferite domenii sau subdomenii.

Utilizare: Permite sau restrictioneaza requesturile cross-origin.

Avantaje: Esential pentru API-uri accesibile public, ofera control flexibil asupra securitatii.

Dezavantaje: Configurarea necorespunzatoare poate duce la vulnerabilitati de securitate.

**1.4. dotenv (versiunea 16.3.1)**

Descriere: dotenv este un pachet care incarca variabile de mediu dintr-un fisier .env.

Ce Inseamna Variabile de Mediu: Variabilele de mediu sunt variabile externe aplicatiei, care sunt setate in sistemul de operare sau in contextul in care ruleaza aplicatia. Acestea sunt folosite pentru a stoca date care nu sunt hardcodate in codul sursa, cum ar fi stringuri de conexiune la baza de date, chei secrete, adrese de servere etc. Utilizarea variabilelor de mediu ajuta la cresterea securitatii si flexibilitatii aplicatiei, permitand schimbarea usoara a acestor setari fara a modifica codul sursa.

Utilizare: Incarca variabile de mediu dintr-un fisier .env.

Avantaje: Centralizeaza configurarile, facilitand managementul variabilelor de mediu.

Dezavantaje: Necesita gestionarea atenta a fisierului .env pentru a evita expunerea datelor sensibile.

**1.5. express (versiunea 4.18.2)**

Descriere: express este un framework web pentru Node.js.

Ce Inseamna Framework Web: Un framework web este o colectie de biblioteci si unelte care faciliteaza dezvoltarea aplicatiilor web. Acestea ofera structuri si modele pentru a crea aplicatii web, inclusiv gestionarea cererilor si raspunsurilor HTTP, rutarea, middleware, sabloane pentru interfete si multe altele. Framework-urile web sunt proiectate pentru a simplifica procesul de dezvoltare, permitand dezvoltatorilor sa se concentreze pe logica specifica aplicatiei, in loc sa reconstruiasca functionalitati comune de la zero.

Utilizare: Framework pentru crearea serverului web si gestionarea rutelor.

Avantaje: Usor de utilizat, flexibil, cu un ecosistem bogat de middleware-uri.

Dezavantaje: Poate necesita module suplimentare pentru functionalitati avansate.

**1.6. jsonwebtoken (versiunea 9.0.2)**

Descriere: jsonwebtoken este un pachet pentru crearea si verificarea tokenurilor JWT.

Ce Inseamna JWT (JSON Web Token): JWT este un standard pentru tokenuri de securitate. Un JWT este un sir de caractere codificat care contine un set de afirmatii sau declaratii. JWT-urile sunt folosite in autentificarea si autorizarea utilizatorilor. Ele sunt structurate in trei parti: un header (care contine tipul tokenului si algoritmul de hashing folosit), un payload (care contine afirmatiile, cum ar fi identitatea utilizatorului) si o semnatura (pentru a verifica daca tokenul nu a fost modificat dupa emitere). JWT-urile sunt utile pentru transmiterea de informatii intre parti intr-un mod compact si verificabil.

Utilizare: Genereaza si verifica tokenuri JWT pentru autentificare.

Avantaje: Ofera un mecanism sigur si stateless pentru gestionarea sesiunilor.

Dezavantaje: Necesita gestionarea atenta a cheilor secrete si a expirarii tokenurilor.

**1.7. nodemon (versiunea 3.0.1)**

Descriere: nodemon este un utilitar care ajuta la dezvoltarea aplicatiilor bazate pe Node.js prin restartarea automata a serverului de fiecare data cand se detecteaza modificari in fisierele sursa.

Ce Este un Utilitar: In informatica, un utilitar este un tip de software care ofera functionalitati specifice pentru a facilita anumite operatiuni sau pentru a imbunatati fluxul de lucru. Utilitarele sunt adesea folosite pentru a automatiza procese repetitive sau pentru a imbunatati eficienta dezvoltarii software.

Utilizare: Restart automat al serverului la modificarea codului sursa.

Avantaje: Imbunatateste eficienta dezvoltarii prin actualizari rapide.

Dezavantaje: Destinat doar pentru dezvoltare, nu pentru productie.

**1.8. sequelize (versiunea 6.35.1)**

Descriere: sequelize este un ORM (Object-Relational Mapping) pentru Node.js. ORM-urile permit dezvoltatorilor sa interactioneze cu baze de date folosind obiecte si metode specifice limbajului de programare, in loc de a scrie interogari SQL brute.

Ce Este Object-Relational Mapping (ORM): ORM este o tehnica de programare care transforma datele intre sisteme incompatibile folosind obiecte orientate pe obiecte. In esenta, ORM permite reprezentarea datelor dintr-o baza de date relationala sub forma de obiecte in limbajul de programare folosit, facilitand astfel lucrul cu aceste date.

Utilizare: ORM pentru interactiunea cu baze de date SQL.

Avantaje: Abstractie puternica a bazei de date, usureaza interogarile.

Dezavantaje: Poate fi mai lent decat interogarile SQL native, complexitate crescuta pentru interogari avansate.  
  
**1.9. sqlite3 (versiunea 5.1.6)**

Descriere: sqlite3 este un modul care permite integrarea cu SQLite, o baza de date relationala usoara, care nu necesita un server de baze de date separat.

Ce Este SQLite: SQLite este un sistem de gestionare a bazelor de date relationale, cunoscut pentru simplitatea si eficienta sa. Este o baza de date "serverless", ceea ce inseamna ca nu necesita un server dedicat si este integrata direct in aplicatii.

Utilizare: Integrare cu bazai de date SQL

Avantaje: Usor de configurat si utilizat, ideal pentru aplicatii de dimensiuni mici si medii.

Dezavantaje: Nu este potrivit pentru aplicatii cu trafic intens sau cu cerinte complexe de scalabilitate.

**2. Controlere si Logica de Afaceri**

Controlere: Directorul 'controllers' este esential pentru gestionarea logicii de afaceri a aplicatiei. Fiecare controler, cum ar fi appointmentsController.js, authController.js, si userController.js, gestioneaza operatiuni specifice legate de entitatile respective. De exemplu, authController.js se ocupa de logica de autentificare, inregistrare si deconectare a utilizatorilor.

Fluxul de Lucru in Controlere: Fiecare controler primeste cereri de la frontend, proceseaza datele conform logicii de afaceri si interactioneaza cu modelul de date pentru a executa operatiuni precum citirea, scrierea sau actualizarea datelor in baza de date.

Controlerele in programare sunt o componenta cheie in arhitectura Model-View-Controller (MVC). MVC este un pattern de design folosit pentru a descompune aplicatiile, in special cele web, in trei componente interconectate: Modelul, View-ul si Controllerul. Acest pattern ajuta la organizarea codului, facandu-l mai usor de intretinut si de extins.

Modelul (M) in MVC reprezinta structura de date a aplicatiei. Acesta gestioneaza logica de afaceri si regulile aplicatiei si interactioneaza cu baza de date sau orice alt mecanism de stocare. Modelul raspunde la cererile de informatii din partea controllerului si trimite aceste informatii inapoi pentru a fi prezentate.

View-ul (V) este componenta care se ocupa de interfata cu utilizatorul. Acesta prezinta datele utilizatorilor si poate fi format din orice output reprezentativ, cum ar fi pagini HTML. View-ul primeste date de la controller si le afiseaza utilizatorilor. Aplicatia dezvoltata in aceasta lucrare nu se foloseste de view in structura aceasta, backend-ul si frontend-ul sunt separate.

Controllerul (C) este intermediarul dintre Model si View. Acesta primeste input de la utilizatori si decide ce actiune sa ia. In functie de aceasta actiune, controllerul va interactiona cu modelul pentru a prelua sau actualiza datele si apoi va alege view-ul corespunzator pentru a prezenta aceste date.

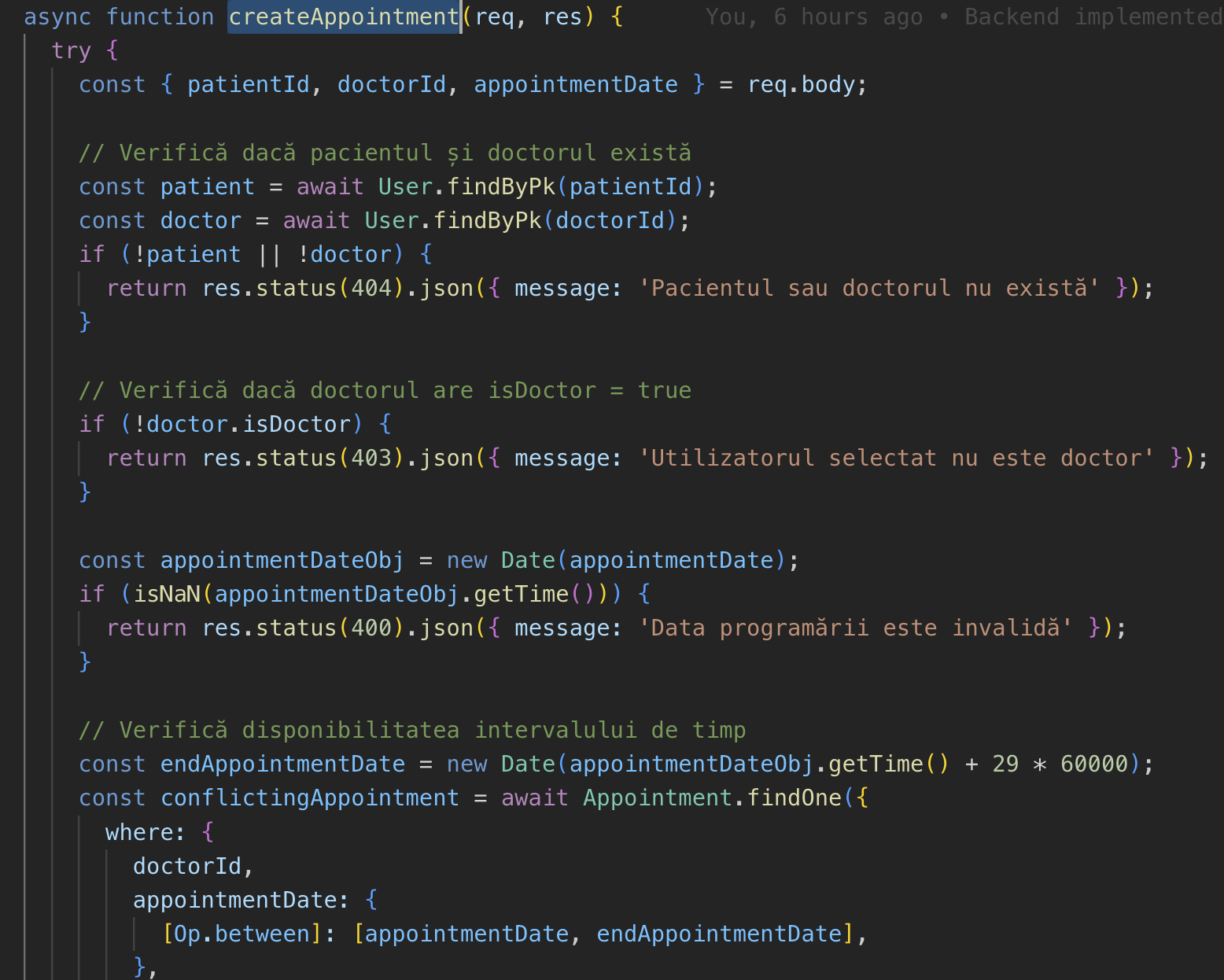
Folosirea MVC are mai multe avantaje. In primul rand, separa logica de afaceri de interfata cu utilizatorul, ceea ce face codul mai usor de intretinut si de testat. De asemenea, permite dezvoltatorilor sa lucreze independent pe diferite parti ale aplicatiei, ceea ce este util in echipe mari.

In cazul unei aplicatii unde view-ul este separat pe un server de frontend, cum ar fi o aplicatie cu un backend in Node.js si un frontend in React, controllerul si modelul sunt de obicei partea de backend. In acest caz, controllerul gestioneaza cererile HTTP, interactioneaza cu modelul pentru a prelua sau actualiza datele si trimite raspunsuri inapoi la client. View-ul este gestionat de frontend, care comunica cu backend-ul prin API-uri.

Acesta separare intre backend si frontend permite o mai mare flexibilitate si scalabilitate. Frontend-ul poate fi dezvoltat si actualizat independent de backend, permitand echipei sa se concentreze pe experienta utilizatorului, in timp ce backend-ul se concentreaza pe logica de afaceri si manipularea datelor. Aceasta abordare este deosebit de utila in dezvoltarea aplicatiilor web moderne, unde experienta utilizatorului si performanta sunt esentiale.

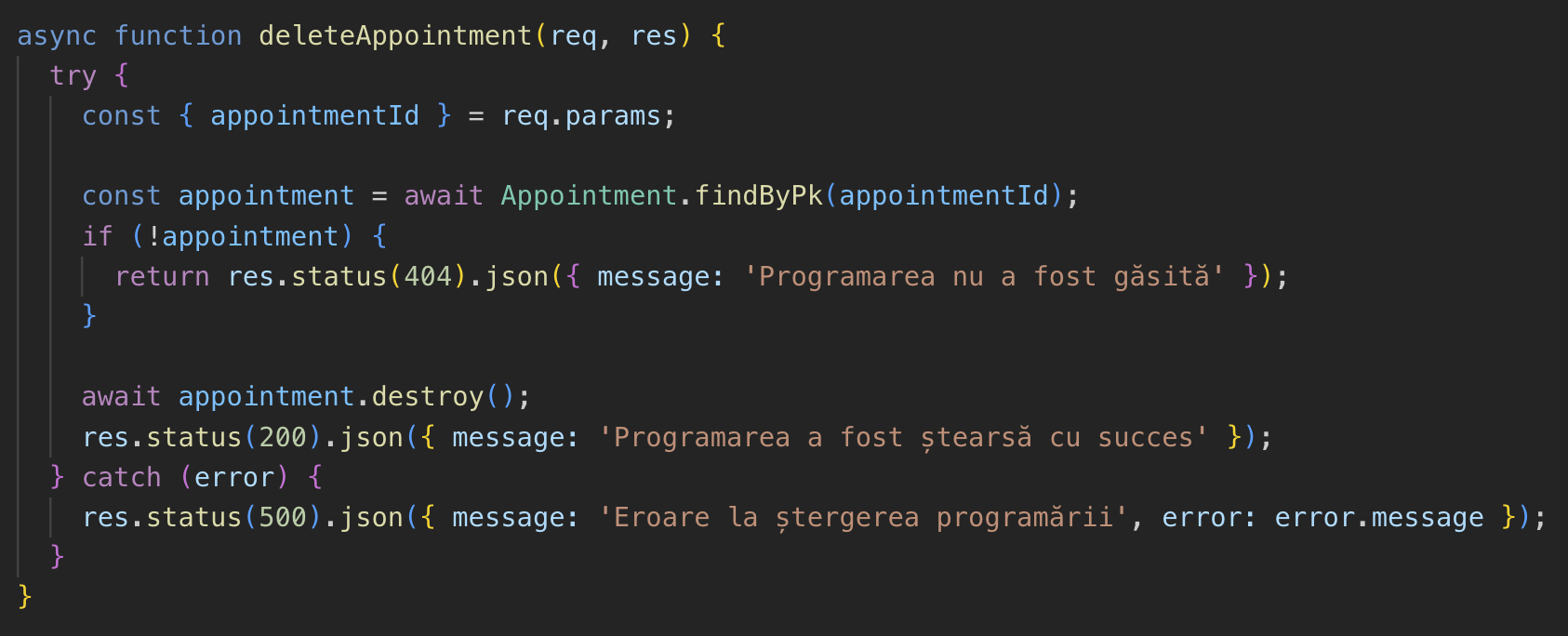
**2.1. Appointments Controller (appointmentsController.js)**

**createAppointment**: Aceasta functie creeaza o noua programare. Verifica existenta pacientului si a doctorului, asigura ca doctorul este valid, verifica validitatea datei programarii si disponibilitatea intervalului de timp, apoi creeaza programarea.

­­



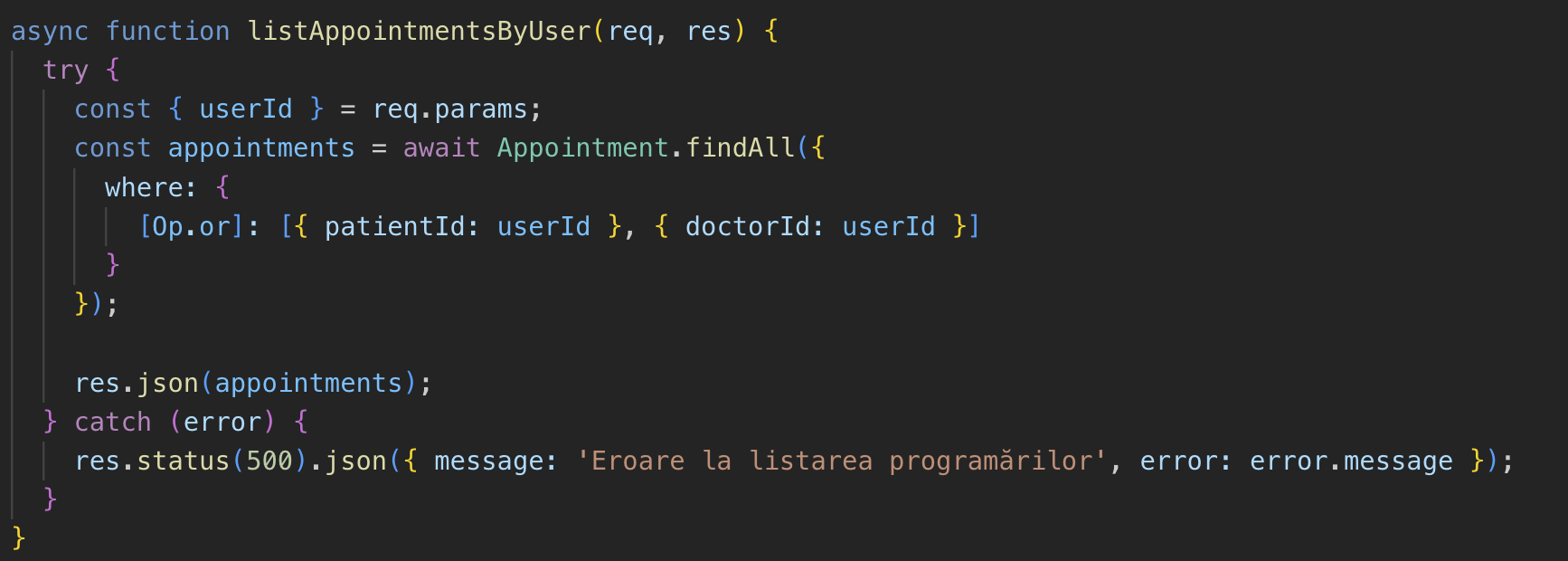
**deleteAppointment**: Permite stergerea unei programari existente, verificand mai intai existenta acesteia.



**getAppointmentDetails**: Ofera detaliile unei programari specifice, identificate prin ID.

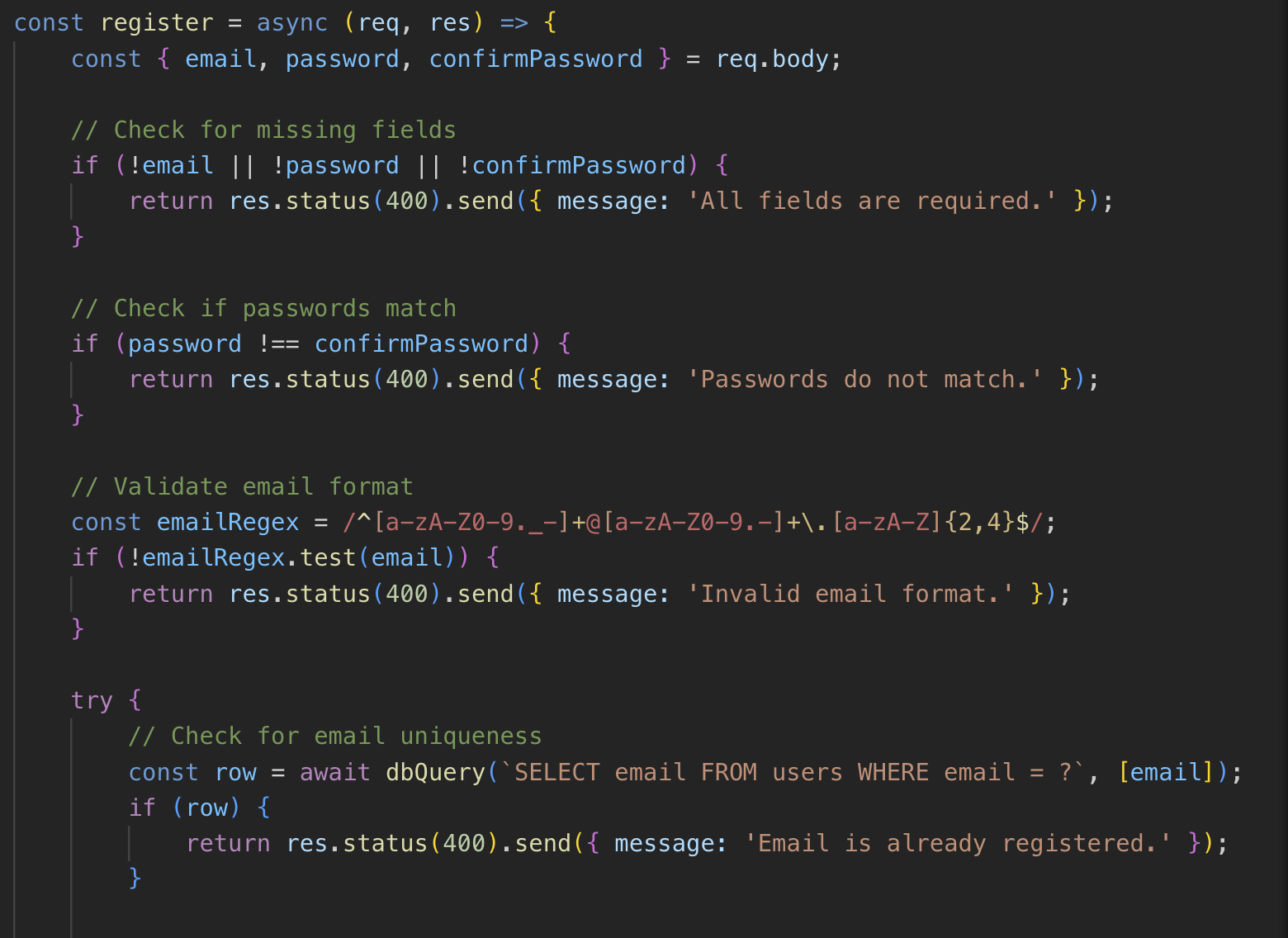
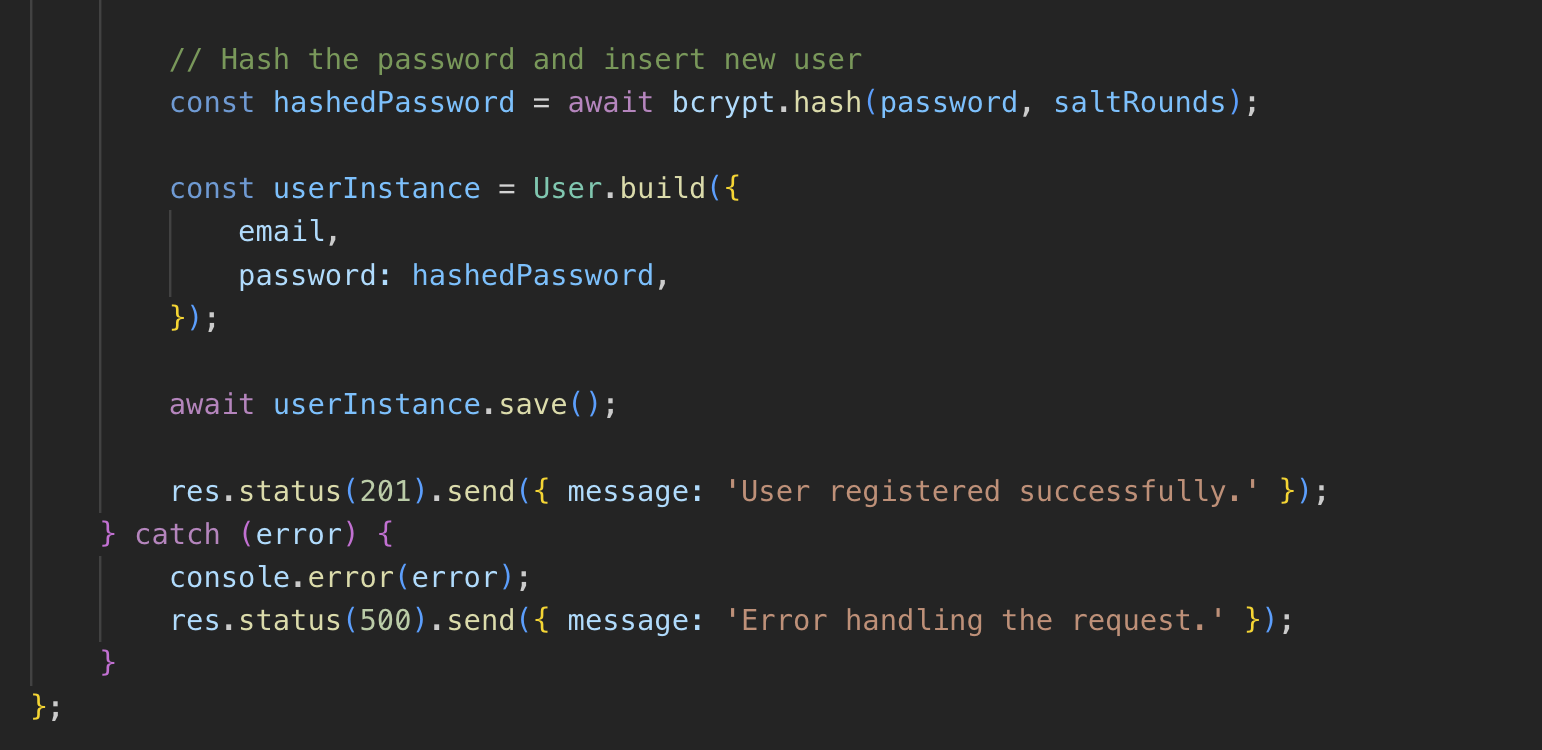


**listAppointmentsByUser**: Listeaza toate programarile asociate unui utilizator, fie ca pacient, fie ca doctor.

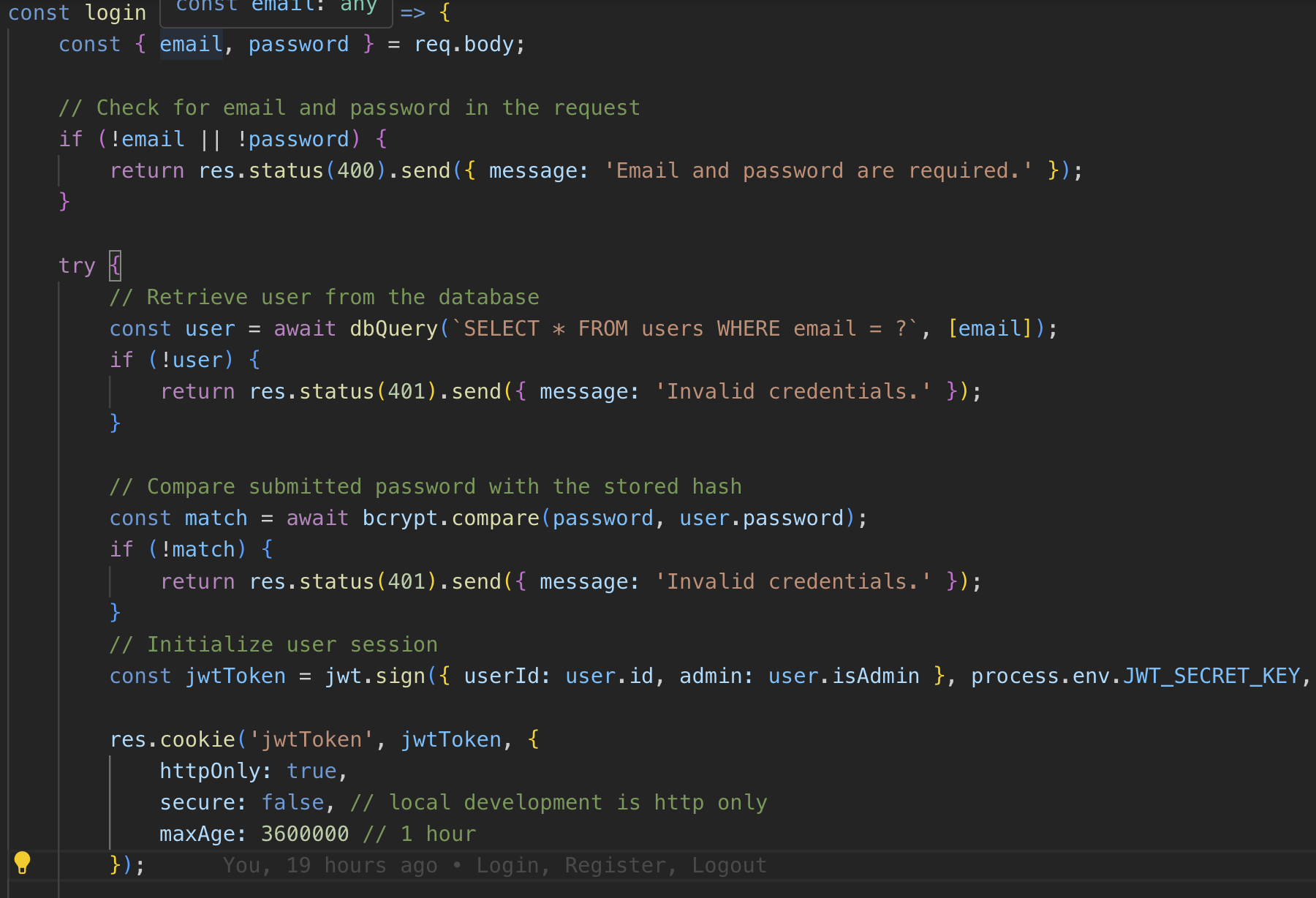


**2.2. Auth Controller (authController.js)**

**register**: Permite inregistrarea unui nou utilizator. Verifica completitudinea campurilor, conformitatea parolelor, formatul emailului si unicitatea acestuia, apoi inregistreaza utilizatorul.

**login**: Autentifica un utilizator. Verifica existenta emailului si corectitudinea parolei, apoi initiaza sesiunea utilizatorului.

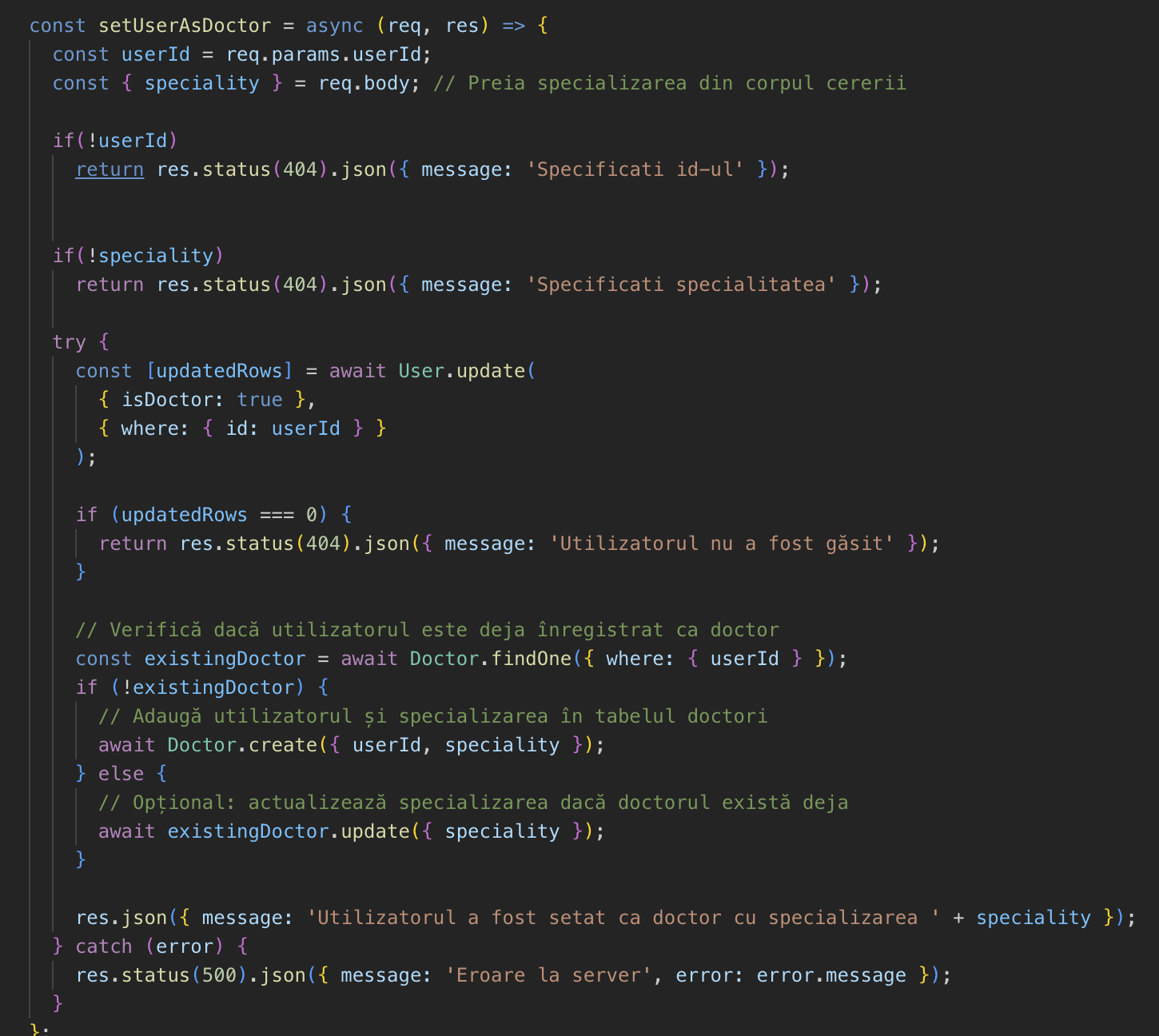
 

**logout**: Deconecteaza utilizatorul, eliminand tokenul JWT din cookie-uri.

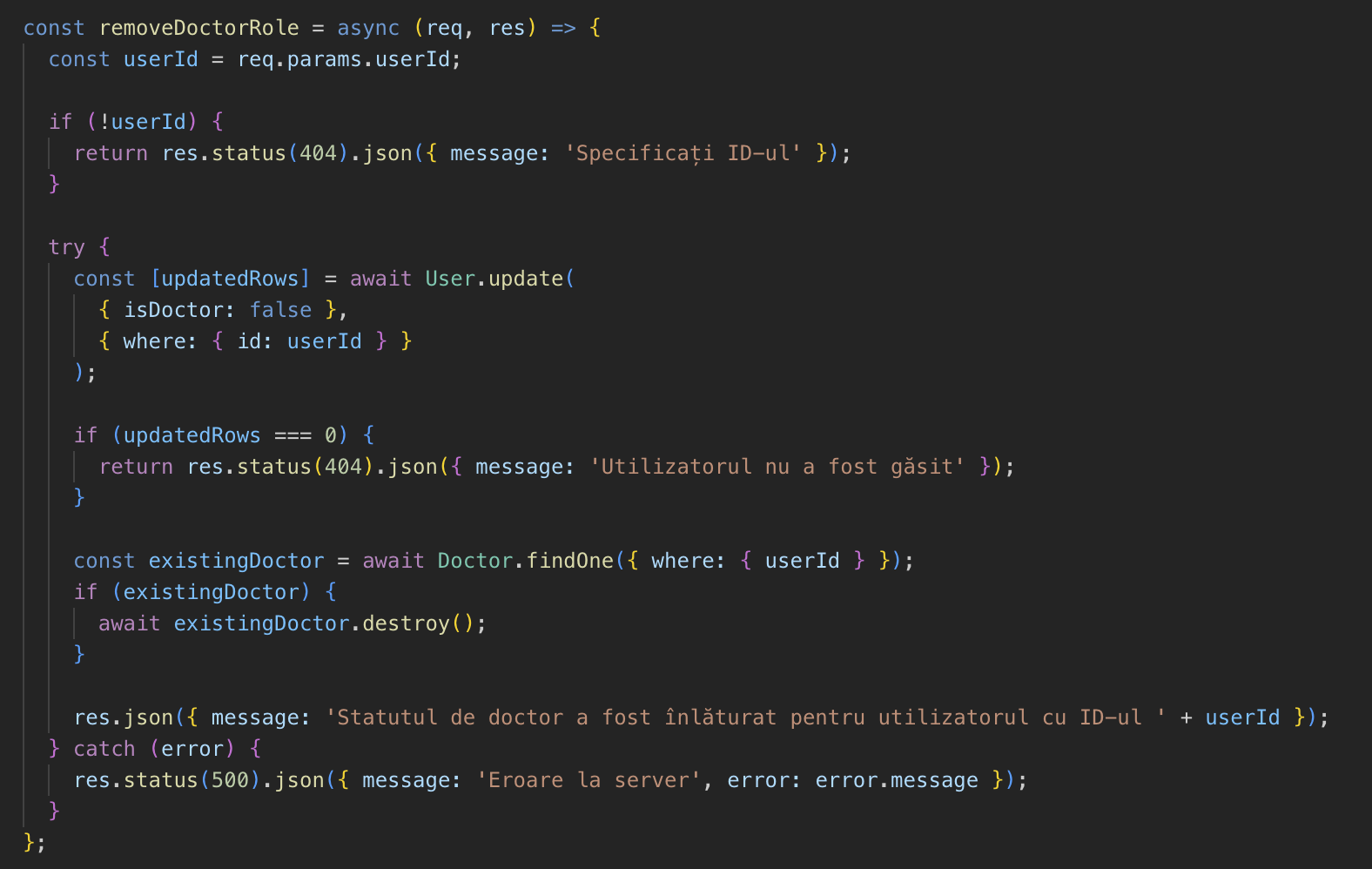


**2.3. User Controller (userController.js)**

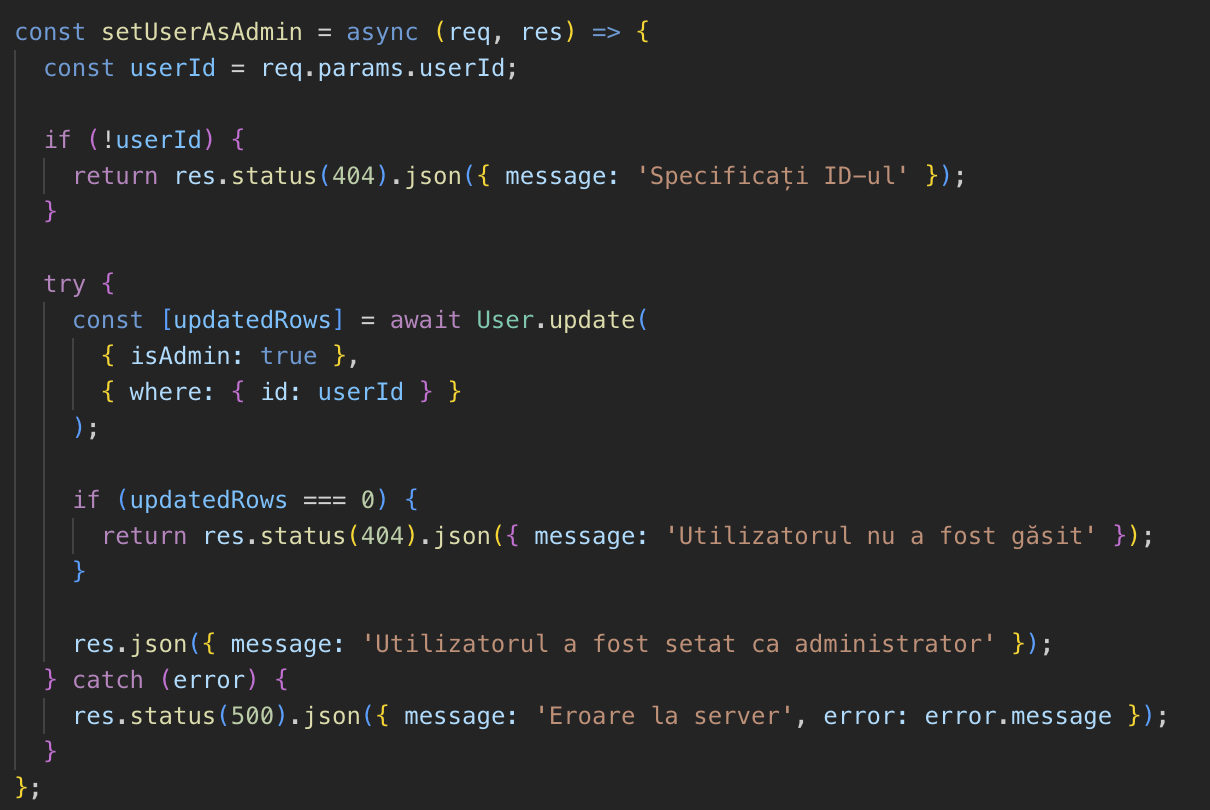
**setUserAsDoctor**: Seteaza un utilizator ca doctor, verificand existenta acestuia si adaugandu-l in tabelul doctorilor cu specialitatea specificata.



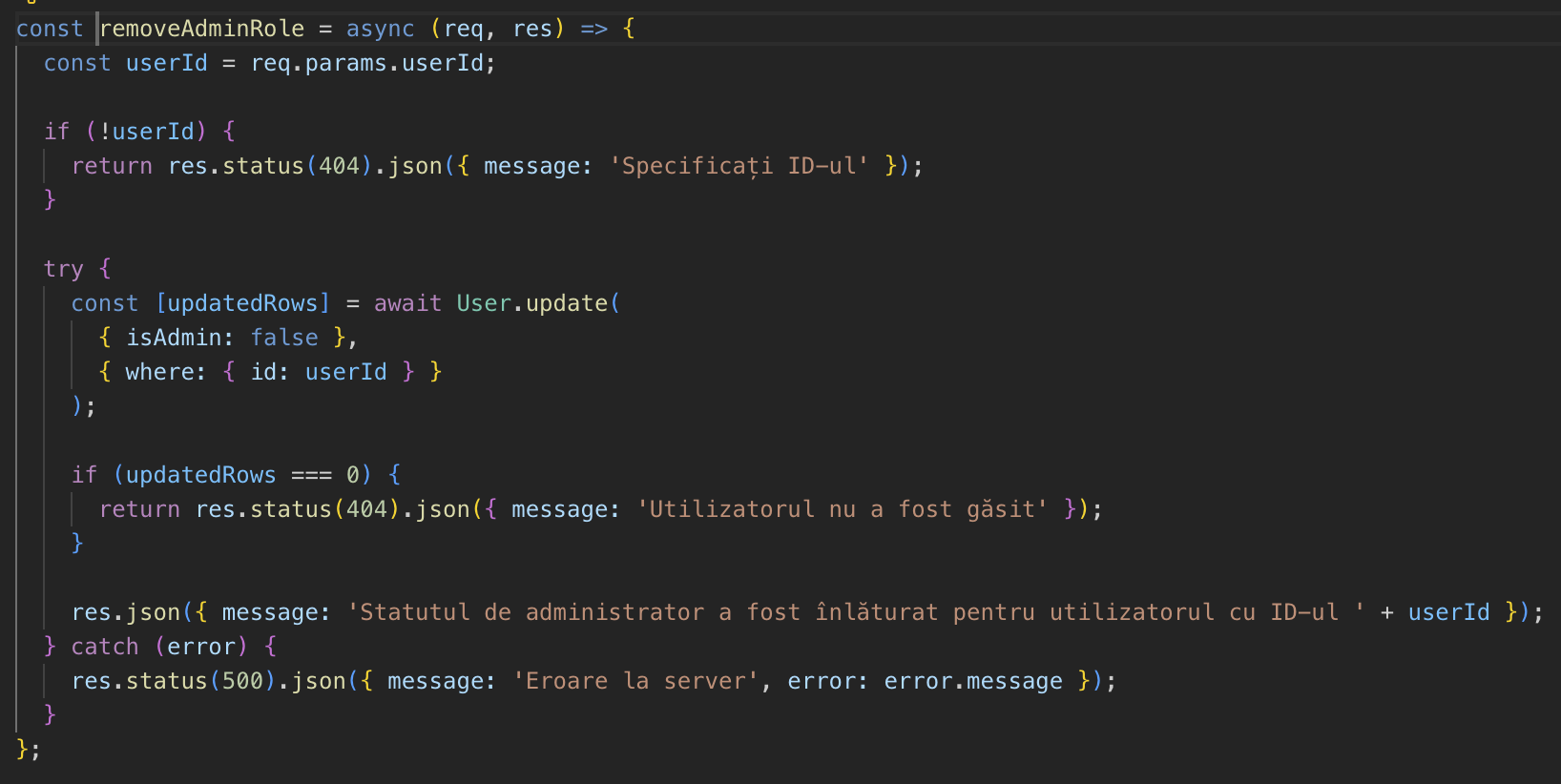
**removeDoctorRole**: Inlatura rolul de doctor al unui utilizator, actualizand statutul acestuia si eliminandu-l din tabelul doctorilor.

****

**setUserAsAdmin**: Promoveaza un utilizator la statutul de administrator.



**removeAdminRole**: Inlatura statutul de administrator al unui utilizator.

****

**3. Gestionarea Bazei de Date**

Database: Directorul 'database' contine setarile si scripturile necesare pentru initializarea si configurarea bazei de date. Acesta include fisiere precum database.js si initializeDB.js, care sunt esentiale pentru stabilirea conexiunii cu baza de date si pentru configurarea initiala a acesteia.

ORM si Modele: Sequelize, un ORM (Object-Relational Mapping) popular pentru Node.js, este folosit pentru a defini modelele de date si pentru a facilita interactiunea cu baza de date. Modelele, definite in directorul 'models', reprezinta structura tabelelor din baza de date si sunt esentiale pentru operatiunile CRUD (Create, Read, Update, Delete).

O baza de date SQL este un sistem pentru stocarea si manipularea datelor structurate. SQL, sau Structured Query Language, este limbajul standard folosit pentru interogarea si modificarea bazelor de date. SQL permite utilizatorilor sa creeze tabele, sa insereze, sa actualizeze, sa interogheze si sa stearga datele.

SQLite, spre deosebire de alte sisteme de gestionare a bazelor de date, este o biblioteca software care ofera un sistem de gestionare a bazelor de date relationale. Principala diferenta intre SQLite si alte sisteme SQL este ca SQLite este o baza de date incorporata, ceea ce inseamna ca ruleaza integrat in aplicatie, fara a necesita un server separat. Acest lucru o face ideala pentru aplicatii mobile sau pentru aplicatii desktop care au nevoie de o baza de date locala, simpla si eficienta.

Un ORM, sau Object-Relational Mapping, este un instrument care permite programatorilor sa lucreze cu baze de date folosind paradigma orientata pe obiecte. ORM transforma datele din bazele de date, care sunt de obicei tabulare, in obiecte care pot fi folosite in limbaje de programare. Acest lucru simplifica mult munca programatorilor, deoarece le permite sa scrie cod in limbajul lor preferat si sa lase ORM-ul sa se ocupe de interogarile SQL.

Folosirea unui ORM are mai multe avantaje. In primul rand, creste productivitatea, deoarece reduce cantitatea de cod SQL scris manual. De asemenea, un ORM poate ajuta la reducerea erorilor si la cresterea securitatii, deoarece interogarile sunt generate automat, ceea ce reduce riscul de injectii SQL. ORM-urile faciliteaza de asemenea lucrul cu baze de date complexe si pot face mai usoara migrarea intre diferite baze de date.

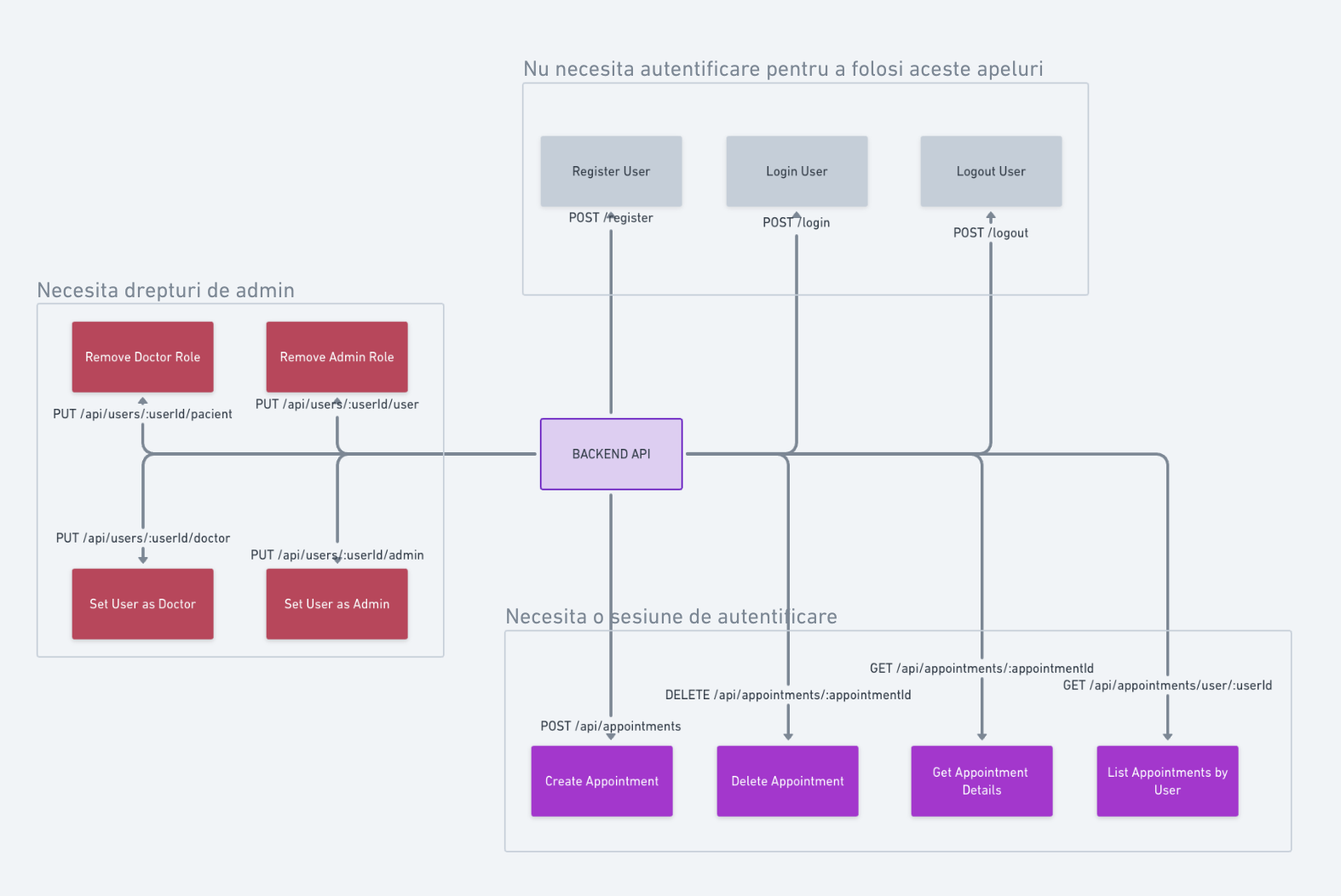
Cu toate acestea, exista si dezavantaje in folosirea unui ORM. Unul dintre ele este ca poate reduce performanta, deoarece generarea automata a interogarilor SQL poate fi mai putin eficienta decat interogarile scrise manual. De asemenea, ORM-urile pot fi complexe si pot necesita un timp de invatare pentru a fi folosite eficient.

In concluzie, o baza de date este un sistem esential pentru stocarea si gestionarea datelor in aplicatii. SQL este limbajul standard pentru interactiunea cu bazele de date, in timp ce SQLite ofera o solutie simpla si eficienta pentru baze de date incorporate. ORM-urile ofera o modalitate de a lucra cu baze de date intr-un mod mai natural pentru programatori, dar pot avea un impact asupra performantei si complexitatii. Alegerea de a folosi sau nu un ORM depinde de cerintele specifice ale proiectului si de preferintele dezvoltatorului.

**4. Rute si Endpoint-uri API**

Rute: Directorul 'routes' contine fisiere precum appointmentsRoutes.js, authRoutes.js, si userRoutes.js, fiecare definind rutele specifice pentru diferitele functionalitati ale aplicatiei. Aceste rute sunt legaturi intre cererile frontend-ului si logica de afaceri gestionata de controlere.

Securitate si Middleware: Securitatea este un aspect crucial, iar in directorul 'session' sunt definite middleware-uri precum verifyToken.js si verifyAdmin.js, care sunt folosite pentru a verifica autenticitatea si autorizatia utilizatorilor inainte de a permite accesul la anumite rute sau functionalitati.



1. Arhitectura endpoint-urilor API

**CE ESTE UN API?**

Un API este un set de reguli, protocoale si definitii care permite diferitelor aplicatii software sa comunice intre ele. Este o interfata prin care diferite programe software pot interactiona, schimba date si functionalitati. API-urile sunt fundamentale in lumea tehnologiei moderne, fiind omniprezente in aproape orice aplicatie software.

In esenta, un API defineste modul in care un programator poate solicita servicii de la un program sau sistem de operare. Acesta stabileste regulile si conventiile care trebuie urmate pentru a accesa functionalitatile oferite de un software. De exemplu, un API pentru un sistem de gestionare a bazelor de date va defini cum se pot face interogari pentru a extrage sau actualiza date.

API-urile pot fi de diferite tipuri, in functie de utilizarea lor. Exista API-uri pentru sisteme de operare, biblioteci software, aplicatii web si multe altele. De exemplu, API-urile sistemelor de operare, cum ar fi Windows API sau POSIX, permit aplicatiilor sa foloseasca functii precum crearea si gestionarea fisierelor sau a conexiunilor de retea.

API-urile web, cum ar fi cele oferite de Google Maps sau Twitter, permit dezvoltatorilor sa integreze functionalitati ale acestor servicii in propriile lor aplicatii. De exemplu, un dezvoltator poate folosi API-ul Google Maps pentru a afisa harti si informatii de localizare intr-o aplicatie de calatorie.

API-urile sunt esentiale pentru dezvoltarea software moderna, deoarece permit reutilizarea codului si functionalitatilor. In loc sa dezvolte de la zero anumite functii, programatorii pot folosi API-uri pentru a accesa functii deja existente si bine testate. Acest lucru economiseste timp si resurse si permite dezvoltatorilor sa se concentreze pe aspectele unice ale aplicatiilor lor.

Un API bun este bine documentat, ceea ce inseamna ca ofera instructiuni clare despre cum poate fi folosit. Documentatia unui API este cruciala pentru programatori, deoarece le ofera informatiile necesare pentru a intelege si a folosi corect API-ul.

API-urile pot fi de asemenea clasificate in functie de nivelul de acces pe care il ofera. Exista API-uri publice, care sunt disponibile oricarui dezvoltator, si API-uri private, care sunt destinate utilizarii interne in cadrul unei companii sau organizatii. API-urile publice sunt adesea folosite pentru a crea ecosisteme in jurul unui anumit serviciu sau produs, incurajand inovatia si dezvoltarea de aplicatii terte.

Securitatea este un aspect crucial in gestionarea API-urilor, mai ales in cazul API-urilor web expuse pe internet. Este important ca un API sa fie securizat pentru a preveni accesul neautorizat si abuzul. Acest lucru se realizeaza prin autentificare, autorizare si criptare.

Un alt aspect important al API-urilor este gestionarea versiunilor. Pe masura ce un API evolueaza, este esential sa se mentina compatibilitatea cu versiunile anterioare pentru a nu intrerupe functionarea aplicatiilor care il folosesc. Gestionarea versiunilor unui API implica crearea de noi versiuni intr-un mod care minimizeaza impactul asupra aplicatiilor existente.

In concluzie, API-urile sunt un element fundamental in dezvoltarea software, facilitand interactiunea intre diferite sisteme si aplicatii. Ele permit dezvoltatorilor sa construiasca aplicatii mai complexe si mai bogate in functionalitati, reutilizand cod si servicii existente. In lumea tehnologiei in continua schimbare, API-urile joaca un rol esential in inovatie si progres.

**5. Securitate si Autentificare**

JWT (JSON Web Tokens): Pentru autentificare si securizarea sesiunilor, sistemul foloseste JWT-uri. Aceste token-uri criptate sunt emise utilizatorilor la autentificare si sunt verificate la fiecare cerere pentru a asigura securitatea si integritatea sesiunii.

JWT, sau JSON Web Token, este un standard pentru transmiterea in siguranta a informatiilor intre parti sub forma unui obiect JSON. Aceasta tehnologie este folosita in special in domeniul autentificarii si autorizarii in aplicatiile web. Un JWT este un string codat care este trimis in headerul HTTP, permitand serverului sa verifice si sa valideze cererea clientului.

Un JWT este format din trei parti: un header, un payload si o semnatura. Headerul contine de obicei tipul tokenului (JWT) si algoritmul de criptare folosit. Payloadul contine claims, adica declaratii despre utilizator si alte informatii necesare. Semnatura este creata prin criptarea headerului si payloadului cu o cheie secreta, asigurand astfel ca tokenul nu a fost modificat pe parcursul transmiterii.

JWT-urile sunt utile pentru ca sunt usor de folosit si pot fi integrate rapid in majoritatea aplicatiilor web. Ele permit aplicatiilor sa evite stocarea informatiilor de autentificare pe server, ceea ce reduce riscul de atacuri de tip XSS (Cross-Site Scripting) si CSRF (Cross-Site Request Forgery). De asemenea, JWT-urile sunt scalabile, ceea ce inseamna ca pot fi folosite in sisteme mari fara a afecta performanta.

Totusi, JWT-urile au si dezavantaje. De exemplu, daca tokenul este interceptat, un atacator poate avea acces la toate informatiile pe care le contine. De aceea, este esential ca transmiterea JWT-urilor sa fie facuta printr-o conexiune securizata, cum ar fi HTTPS. De asemenea, JWT-urile nu sunt potrivite pentru stocarea unor cantitati mari de date, deoarece marimea lor creste odata cu cantitatea de informatii pe care le contin.

In concluzie, JWT este o tehnologie puternica si flexibila care ofera o metoda sigura si eficienta de autentificare si autorizare in aplicatiile web. Este important, insa, ca dezvoltatorii sa inteleaga riscurile asociate cu folosirea JWT-urilor si sa implementeze masuri de securitate corespunzatoare pentru a proteja datele si aplicatiile.

**6. Interactiunea cu Frontend-ul**

API-uri si Endpoint-uri: Backend-ul expune o serie de API-uri si endpoint-uri care sunt consumate de frontend. Acestea includ endpoint-uri pentru autentificare, gestionarea utilizatorilor, programarilor si multe altele.

Comunicarea cu Frontend-ul: Comunicarea intre backend si frontend se realizeaza prin cereri HTTP, unde frontend-ul trimite cereri catre endpoint-urile definite in backend, iar backend-ul raspunde cu datele solicitate sau cu confirmarea executarii unei operatiuni.

Interactiunea dintre backend si frontend in cadrul unei aplicatii web este fundamentala si se bazeaza pe protocolul HTTP (Hypertext Transfer Protocol). HTTP este coloana vertebrala a comunicarii pe internet, facilitand transferul de informatii intre client (frontend) si server (backend).

HTTP functioneaza pe baza unui model cerere-raspuns. Clientul, adica browserul sau aplicatia frontend, trimite o cerere HTTP catre server, iar serverul raspunde la aceasta cerere. Fiecare cerere HTTP include o metoda sau un "verb", care indica tipul actiunii dorite. Cele mai comune metode HTTP sunt GET, POST, PUT, DELETE si PATCH.

GET: Este folosit pentru a solicita date de la server. De exemplu, cand un utilizator acceseaza o pagina web, browserul trimite o cerere GET pentru a obtine continutul paginii. GET este o operatiune "safe", adica nu modifica starea serverului.

POST: Aceasta metoda este folosita pentru a trimite date catre server, de obicei pentru a crea o noua resursa. De exemplu, atunci cand un utilizator completeaza un formular pe o pagina web si apasa butonul de trimitere, datele formularului sunt trimise serverului printr-o cerere POST.

PUT: Este utilizat pentru a actualiza o resursa existenta pe server. Diferenta fata de POST este ca PUT este idempotent, ceea ce inseamna ca efectuarea aceleiasi cereri de mai multe ori va avea acelasi rezultat.

DELETE: Cum sugereaza si numele, aceasta metoda este folosita pentru a sterge o resursa de pe server.

PATCH: Este similar cu PUT, dar este folosit pentru a face actualizari partiale la o resursa.

Fiecare cerere HTTP contine, de asemenea, un header si un body. Headerul poate include diverse informatii, cum ar fi tipul de continut al cererii, cookie-uri sau informatii de autentificare. Body-ul cererii contine datele efective trimise serverului, cum ar fi informatiile dintr-un formular.

Cand frontend-ul trimite o cerere HTTP catre backend, serverul proceseaza cererea in functie de metoda si resursa solicitata. Serverul poate interoga baza de date, poate efectua operatiuni de logica de afaceri si apoi trimite un raspuns inapoi la client. Raspunsul include un cod de status care indica daca cererea a fost procesata cu succes (de exemplu, 200 OK pentru succes, 404 Not Found pentru o resursa care nu a fost gasita) si, de obicei, include si date in body-ul raspunsului.

Aceasta comunicare prin cereri si raspunsuri HTTP este esentiala pentru functionarea aplicatiilor web. Ea permite separarea clara a responsabilitatilor intre frontend, care se ocupa de interfata cu utilizatorul si experienta acestuia, si backend, care gestioneaza logica de afaceri, accesul la date si securitatea. Prin utilizarea API-urilor si endpoint-urilor, dezvoltatorii pot crea aplicatii web puternice, flexibile si usor de intretinut.