

**SISTEME DISTRIBUITE**

**Tema 1**

**Documentație**

Online Medication Platform

Pop Claudia Dorina

Cuprins

[1. Arhitectura conceptuală a sistemului distribuit 3](#_Toc56033113)

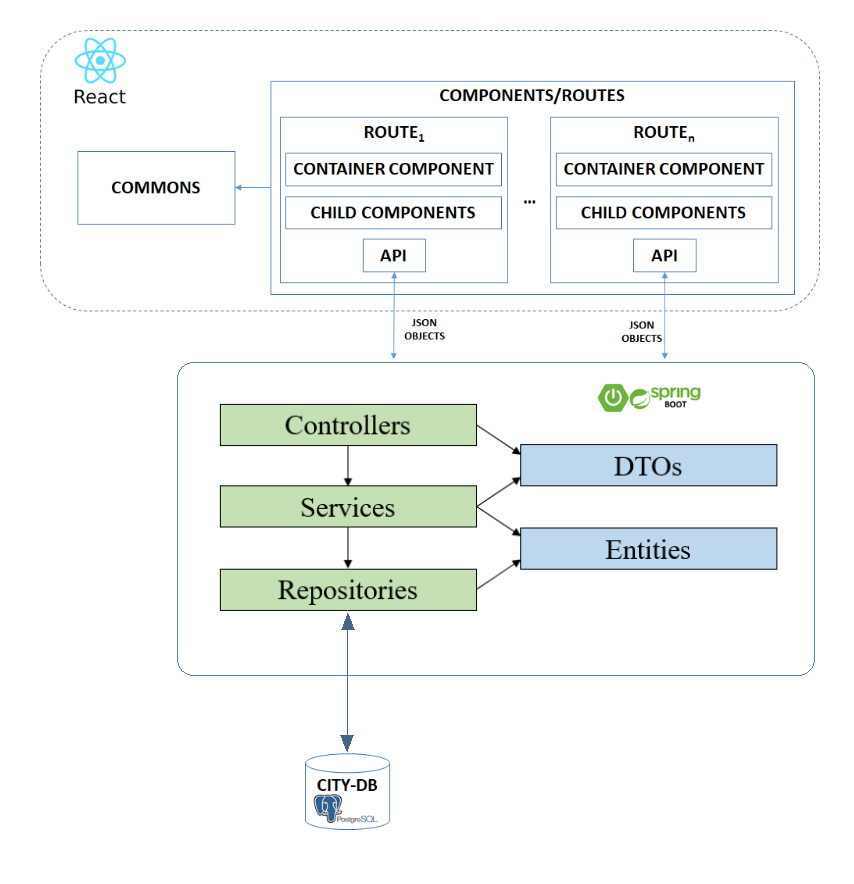
[2. Designul Bazei de Date 4](#_Toc56033114)

[3. Diagrama UML de Deployment 5](#_Toc56033115)

[4. Build and execution considerations 5](#_Toc56033116)

# Arhitectura conceptuală a sistemului distribuit

Sistemul distribuit este alcătuit din două aplicații de sine stătătoare care comunică între ele, astfel realizându-se funcționalitățile cerute.



**Figura 1.** Arhitectura conceptuală a sistemului distribuit

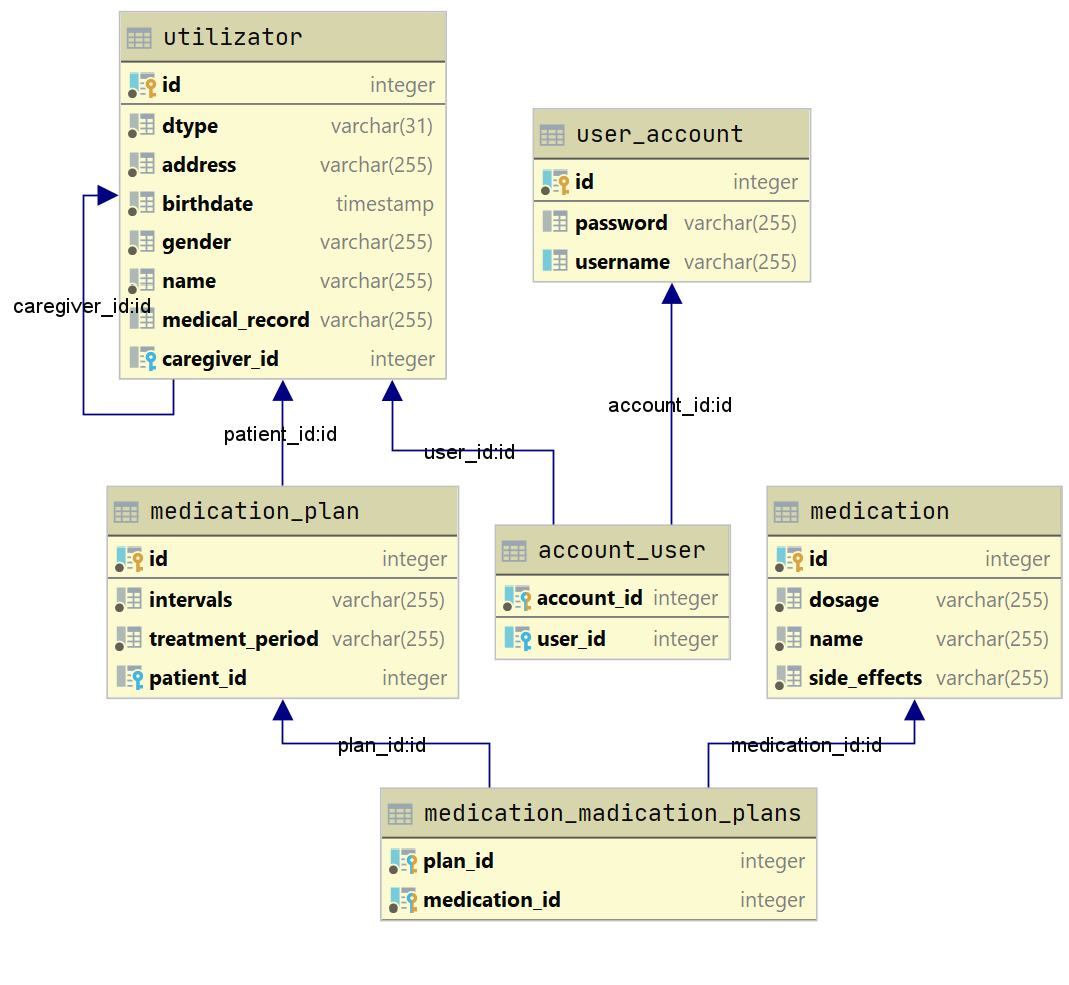
Din arhitectura aplicației React reies principalele componente. Directorul Commons conține fișiere ce se utilizează global în aplicație. Fiecare pagină diferită ce va fi vizibilă în browser este reprezentată de un container care conține mai multe componente. Componentele API sunt cele care trimit cererile înspre backend și primesc răspuns.

Partea de backend are o arhitectură bazată pe layere. Controllerele sunt componentele capabile să primească cererile de tip HTTP REST de la partea de frontend a sistemului distribuit și să trimită răspunsuri. Acestea comunică in mod direct cu Serviciilefolosind obiecte de tip Data Transfer. Aceste obiecte conțin o parte din informațiile din entități, pot să îmbine datele din mai multe entități și pot exista mai multe clase DTO pentru aceeași entitate. În servicii se fac operații mai complexe asupra datelor înainte de a accesa stratul Repositories. Tot aici se transformă DTO-urile în entități și invers, în funcție de direcția în care merg datele, spre și dinspre baza de date. Serviciile comunică cu etajul Repositories folosind entități. Entitățile reprezintă maparea în Java a tabelelor din baza de date sub formă de clase. Clasele din Repositories sunt cele care accesează baza de date, trimit date, respectiv iau date din tabele.

# Designul Bazei de Date

Structura bazei de date este ilustrată în figura care urmează. Tipurile de utilizatori ai aplicației, doctori, pacienți și îngrijitori, au fost modelați ca subclase ale clasei *User,* astfel entitățile au fost salvate în baza de date într-un singur tabel, *utilizator.* Între entități există diferite relații de cardinalitate, după cum urmează:

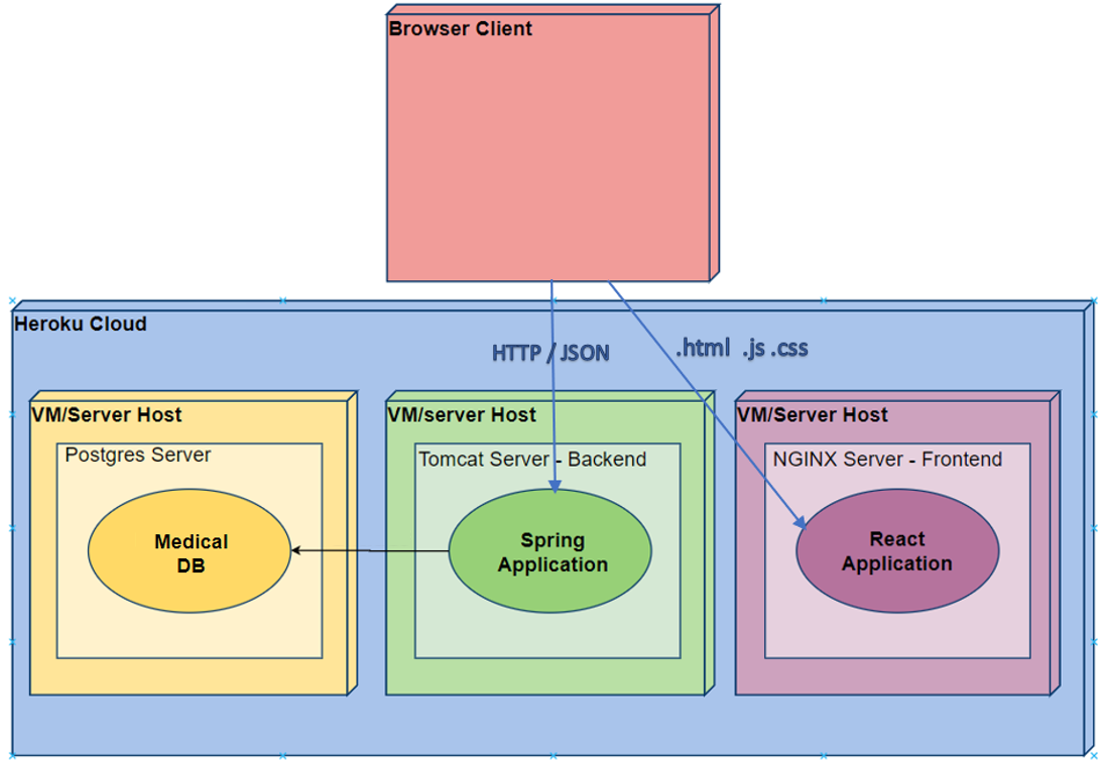
* Patient – Caregiver: Many-to-One
* User – UserAccount: One-to-One → cu tabel intermediar
* Medication – MedicationPlan: Many-to-Many → cu tabel intermediar
* Patient – MedicationPlan: One-to-Many



**Figura 2.** Diagrama Bazei de Date

# Diagrama UML de Deployment

Modulele aplicației finale vor fi stocate în trei servere principale, unul pentru baza de date, unul pentru aplicația de frontend și unul pentru aplicația de backend. Aceste trei părți comunică între ele. Partea de backend va comunica în mod direct cu baza de date, trimițând date pentru a fi stocate sau extrăgând date prin intermediul interogărilor. Browserul web este cel care facilitează comunicarea dintre frontend și backend. Comunicarea se face prin *HTTP requests,* iar datele care circulă sunt obiecte de tip JSON, astfel arhitectura aplicației este una de tipul client-server, browser-ul trimite cereri către server, iar serverul trimite înapoi un răspuns.



**Figura 3.** Diagrama UML de Deployment

# Build and execution considerations

Pentru Backend:

* Java 11;
* Server PostgresSQL pentru baza de date (v13.0.1) + pgAdmin, se creează o nouă bază de date;
* Configurarea unui server web–Apache Tomcat;
* Mediu de dezvoltare: IntelliJ
* Rulare: se rulează fișierul *Ds2020Application*

Pentru Frontend:

* Instalare pachet npm
* Rulare: comenzi: *npm install, npm start*