

# Proiect SCD

## Sistem distribuit de rezervări la clinici / cabinete medicale

### 1. Descriere generală

Proiectul reprezintă o platformă web distribuită care permite pacienților să efectueze programări online la clinici și cabinete medicale. Sistemul gestionează utilizatori, medici, specializari, cabinete, programări și notificări.

Platforma o sa fie implementată folosind arhitectură pe microservicii, containere Docker, un mecanism de autentificare centralizată (Keycloak), o bază de date relațională și un stack Docker Swarm. De asemenea, sistemul include două module avansate: un mecanism distribuit de gestionare a programărilor simultane și un sistem de notificări asincrone (pentru a notifica pacientul, remindere + salvarea informațiilor într-un pdf).

### 2. Arhitectura sistemului

Proiectul o sa fie construit pe baza unei arhitecturi cu minim 5 microservicii separate, fiecare având propriul rol în cadrul sistemului.

#### 2.1 Microservicii

1. **Auth-Service (Keycloak)** – gestionează autentificarea (SSO) și autorizarea.
2. **User-Service** – gestionează profilurile utilizatorilor și rolurile.
3. **Doctor-Service** – gestionează doctorii, specialitățile și cabinetele.
4. **Appointment-Service** – gestionează programările și integrează modulul avansat de queue distribuit.
5. **Notification-Service** – procesează notificări asincron.
6. **API Gateway / Frontend-Service** – centralizează accesul către API-uri.
7. **Database-Service** – stochează persistent datele.

#### 2.2 Rețele Docker

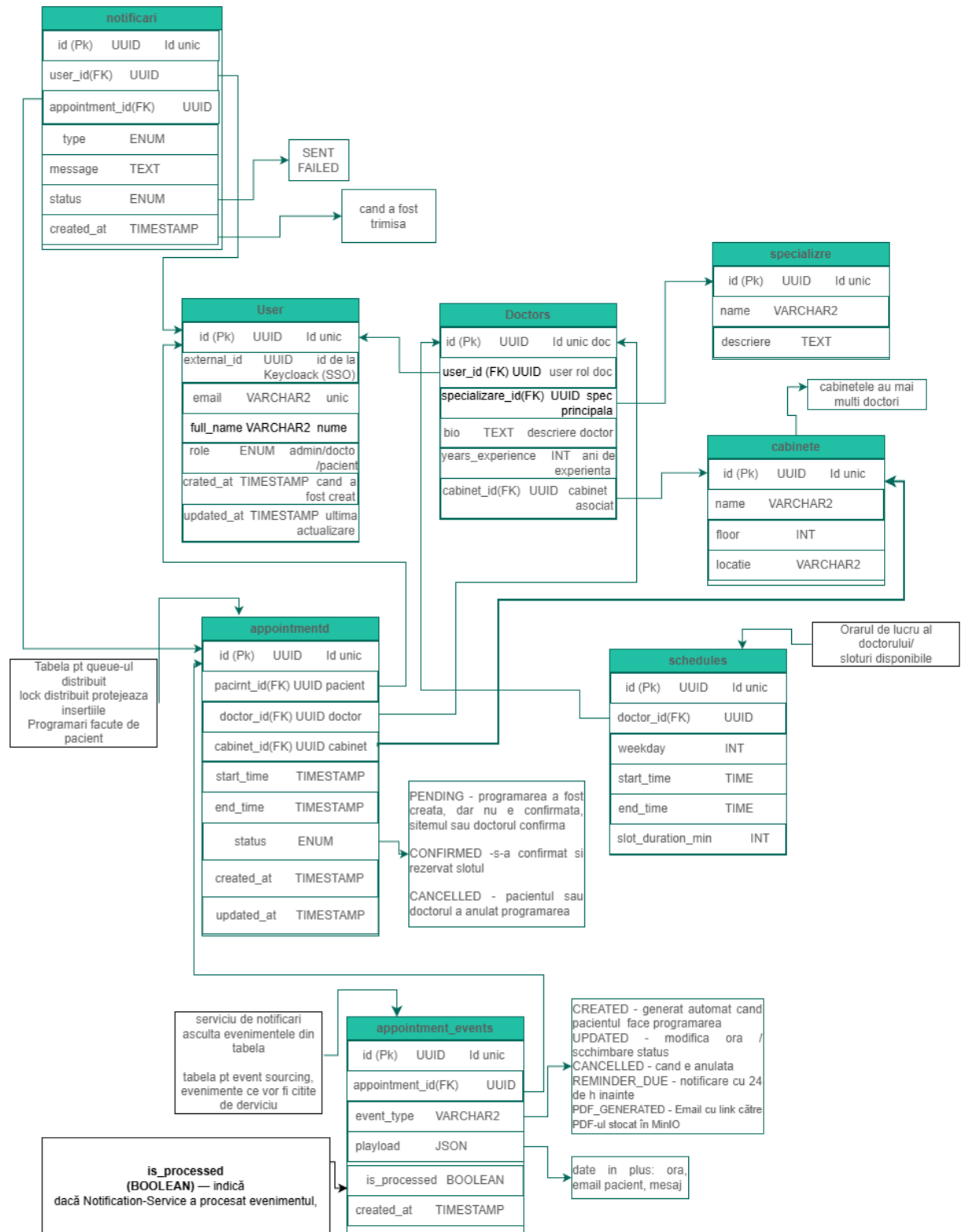
- **auth-net** – comunicare între gateway și Keycloak
- **internal-net** – comunicare între microservicii interne
- **db-net** – comunicație securizată între servicii și baza de date

#### 2.3 Comunicarea între microservicii

Microserviciile comunică folosind DNS-urile generate de Docker și variabile de mediu.

### 3. Baza de date și schema tabelelor

Schema bazei de date include următoarele tabele:



## 4. Module avansate

Cele două module avansate pe care le am ales sunt:

### 4.1 Modul Avansat 1: Queue distribuit pentru programări simultane + Integrarea MinIO și generarea PDF-urilor

Într-un sistem medical, mai mulți utilizatori pot încerca în același timp să facă o programare pentru:

- același medic
- același cabinet
- același interval orar

Fără control distribuit → apar conflicte:

- două persoane pot primi același slot;
- baza de date poate avea inconsistențe;
- replicile microserviciului pot crea coliziuni.

Acest modul garantează ca un singur utilizator poate rezerva același interval la un moment dat, toate replicile microserviciului de programări văd aceleași reguli, consistență puternică ("no double bookings"), rezistență la load mare (mulți utilizatori simultan). Pentru asta o să folosesc un message broker precum RabbitMQ + coadă de programări.

Se garantează astfel că două programări pentru același doctor în același interval nu pot fi create simultan, deoarece toate replicile trimit cereri către o coadă, un singur worker procesează în ordine mesajele și nu pot apărea suprapuneri deoarece ordinea este garantat.

#### Generarea și stocarea PDF-urilor cu detalii despre programări:

- La confirmarea unei programări, Notification-Service generează un fișier PDF după ce primește acțiunea de la producătorul din appointment, care trimite cererea ca notification service să facă notificare legată de programare. PDF-ul se trimite la client doar când programarea e confirmată de doctor. PDF-ul conține:
  - numele pacientului
  - numele doctorului
  - specialitatea
  - data și ora programării
  - Locația/cabinetul
  - Ora la care s-a făcut documentul
- PDF-ul este apoi încărcat automat într-un bucket MinIO dedicat.
- Notification-Service trimite utilizatorului un link securizat către PDF.

Această funcționalitate demonstrează:

- procesare asincronă;
- integrarea unui serviciu de tip obiect-storage (MinIO);
- generarea de documente în paralel;
- izolarea responsabilităților între microservicii.

### 4.2 Modul Avansat 2: Sistem de notificări asincrone

Într-un sistem medical, după ce pacientul își face o programare:

- trebuie notificat prin email/SMS
- trebuie să primească reminder cu 24h înainte
- medicul trebuie informat asupra calendarului

Acest modul gestionează trimiterea notificărilor automate către utilizatori (pacienți și medici), pe baza evenimentelor generate în cadrul sistemului. Notificările sunt procesate asincron de un microserviciu separat, folosind un broker de mesaje.

### Tipuri de notificări gestionate

Notification-Service trimite automat email-uri în funcție de tipul evenimentului generat în sistem:

- **CREATED** → Email de confirmare a programării (cu detalii: doctor, specialitate, data/ora, cabinet și link PDF dacă este generat)
- **UPDATED** → Email care anunță modificarea programării (nouă dată/oră), trimis către pacient și doctor
- **REMINDER\_DUE** → Email de reamintire cu 24h înainte de programare
- **CANCELLED** → Email care anunță anularea programării pacientului
- **CONFIRMED** → Email care anunță pacientului ca doctorul a confirmat cererea + link către PDF-ul stocat în MinIO

### Funcționare

1. Appointment-Service generează un eveniment (ex: CREATED, CANCELLED, UPDATED).
2. Evenimentul este salvat în tabela `notification_events`.
3. Notification-Service consumă evenimentul și trimite notificarea corespunzătoare.

### Beneficii

- Îmbunătățește experiența utilizatorilor.
- Permite procesare distribuită și scalabilă a notificărilor.
- Creează un flux auditabil complet prin tabela `notifications`.

## 5. Fluxuri principale ale aplicației

### 5.1 Crearea unei programări

1. Pacientul se autentifică prin Keycloak.
2. User-service validează rolul.
3. Appointment-service primește cererea.
4. Se activează mecanismul de lock distribuit.
5. Dacă slotul este liber → programare creată.
6. Se generează un eveniment **CREATED** în `appointment_events`.
7. Notification-service trimite notificarea.

## 5.2 Reminder automat

1. Worker-ul din notification-service rulează la miezul nopții.
2. Identifică programările din ziua următoare.
3. Generează evenimente REMINDER\_DUE.
4. Trimite notificările către utilizatori.

## 6. Rutele API pentru funcționalități

### 6.1 User-Service – Route

Userul are acces doar la programările lui nu are voie să modifice nimic ce nu e al lui

**GET /users/register {"full\_name":..., "email":..., "password":...}**

Un user se poate autentifica, să-și facă cont fără token la început, el la rol poate să aibă doar rolul de PATIENT

**GET /users/me**

Returnează profilul utilizatorului autentificat.

**GET /users/{id} (ADMIN)**

Returnează profilul unui user în funcție de id

**GET /users (ADMIN)**

Listă completă de utilizatori.

**POST /users/sync-keycloak**

Creează/actualizează un user în baza de date locală pe baza tokenului adminului.

**PUT /users/me Body: { "full\_name": "...", "phone": "...", ... }**

Actualizare profil user de către user

**PUT /users/{id} { "full\_name": "...", "phone": "...", "email": "..." }**

Actualizare date de către admin la profilul user-ului

**PUT /users/{id}/role '{"role": "DOCTOR/PACIENT"}' (ADMIN)**

Actualizare doar rol user de către admin

**DELETE /users/{id} (ADMIN)**

Sterge un user după id

## 6.2 Doctor-Service – Rute HTTP

### **GET /doctors**

Lista tuturor doctorilor.

### **GET /doctors/{id}**

Detalii despre un doctor.

### **GET /doctors?specialization\_id={id}**

Returneaza lista doctorilor in functie de specializare

### **GET /doctors?cabinet\_id={id}**

Returneaza lista doctorilor in functie de cabinet

### **POST /doctors {"user\_id":..., "specialization\_id":..., "cabinet\_id":..., "bio":..., "years\_experience":...}(ADMIN)**

Creează un doctor nou, obligatorii in body sunt user\_id, specialization\_id, cabinet\_id.

### **PUT /doctors/{id} {"specialization\_id":..., "cabinet\_id":..., "bio":..., "years\_experience":...} (ADMIN)**

Actualizează detalii despre doctor.

### **DELETE /doctors/{id} (ADMIN)**

Șterge un doctor.

### **GET /specializations**

Returnează lista specialităților.

### **POST /specializations (ADMIN)**

Creează o specialitate nouă.

### **PUT /specializations/{id} {"name":..., "description":...} (ADMIN)**

Actualizeaza datele unei specializari anume

### **DELETE /specializations/{id} (ADMIN)**

Sterge o specializare

### **GET /cabinets**

Returnează toate cabinetele.

### **POST /cabinets (ADMIN)**

Creează un cabinet.

### **PUT /cabinets/{id} {"name":..., "location":..., "floor":...} (ADMIN)**

Actualizeaza datele unui cabinet

**DELETE /cabinets/{id} (ADMIN)**

Sterge un cabinet dupa id

### 6.3 Schedules-Service – Rute HTTP

**GET /doctors/{id}/schedule**

Returnează orarul unui doctor.

**POST /doctors/{id}/schedule (ADMIN/DOCTOR)**

Adaugă program de lucru pentru doctorul respectiv-orar.

**DELETE /doctors/{doctor\_id}/schedule/{schedule\_id} (ADMIN/DOCTOR)**

Șterge un program orar pentru doctor (de catre el sau admin) din programele pe care le are

**GET /doctors/{id}/available-slots?date=YYYY-MM-DD**

Returneaza sloturile libere pacientilor inainte de a face o programare

Calculează sloturile libere pe baza: Programului doctorului (schedules), Programărilor existente (appointments), Data selectată

### 6.4 Appointment-Service – Rute HTTP

**GET /appointments (ADMIN/DOCTOR)**

**?status=PENDING&doctor\_id=...&patient\_id=...&date\_from=...&date\_to=...**

Listă programări filtrată după user, doctor sau dată.

**GET /appointments/my (PATIENT)**

Programările active ale pacientului.

**GET /appointments/my/history?status=COMPLETED|CANCELLED|rejected**

Programările trecute ale pacientului.

**GET /appointments/{id}**

Detalii programare.

**POST /appointments**

Creează o programare nouă (activează mecanismul de lock distribuit).

Body: { "doctor\_id": "...",  
          "start\_time": "...",

"end\_time": "..." }

#### **PUT /appointments/{id}/cancel**

Anulează o programare (generează eveniment CANCELLED).

- Doar evenimentele în starea de PENDING/CONFIRMED

#### **PUT /appointments/{id}/confirm (ADMIN/DOCTOR)**

Confirmă programarea de către doctor/admin.

#### **PUT /appointments/{id} (ADMIN/DOCTOR)**

Actualizează detaliile unei programări existente (dată, oră, cabinet). Body: { "start\_time": "...", "end\_time": "...", "cabinet\_id": "..." (opțional) }

- Se generează un eveniment UPDATED în appointment\_events
- Pacientul primește notificare despre modificare
- Doar programările cu status PENDING pot fi modificate

### **6.5 Appointment-Events – Rute HTTP**

#### **GET /events/pending (admin)(NOTIFICATION-SERVICE)\*\***

Returnează evenimentele neprocesate ( is\_processed = false ). INTERN doar pentru notification service

#### **PUT /events/{id}/processed (admin)**

Marchează evenimentul ca procesat.

### **6.6 Notification-Service – Rute HTTP**

#### **POST /notifications/send**

Trimite o notificare manuală.

#### **GET /notifications/user/{id}**

Listă notificări trimise unui utilizator.

#### **GET /notifications/appointment/{id}**

Notificări legate de o programare.

## **7. COMPONENTA REPLICATĂ**



La mine componenta replicata v-a fi APPOINTMENT-SERVICE care este si la MODUL 1 AVANSAT, pentru a scoate in evidenta faptul ca o singura replica v-a putea face rezervare pe un slot, nu mai multe in acelasi timp.