Universitatea POLITEHNICA din București
Facultatea de Automatică și Calculatoare

Documentație inițială - Proiect SCD

MediHelp – Platformă distribuită pentru gestionarea rețetelor medicale și a stocurilor din farmacii

Bogdan-Vasile Petrea
342C3

Sisteme Concurente și Distribuite

Data: 28 noiembrie 2025

1 Introducere

Platforma propusă, **MediHelp**, este o aplicație web pentru managementul rețetelor medicale și al stocurilor de medicamente din farmacii. Sistemul permite emiterea și urmărirea rețetelor de către doctori, procesarea lor de către farmacii, precum și verificarea rapidă a disponibilității medicamentelor în farmaciile implicate.

Modelul de date și regulile de business sunt, în linii mari, următoarele:

- **Doctor**:
  - emite rețete medicale pentru pacienți;
  - selectează farmacia (sau farmaciile) unde rețeta poate fi procesată;
  - poate urmări starea rețetelor emise (emisă, în curs, eliberată, respinsă).
- **Farmacie**:
  - primește rețete adresate farmaciei;
  - verifică stocul medicamentelor prescrise;
  - eliberează sau respinge rețeta, actualizând automat stocurile.
- **Gestionarea stocurilor**:
  - fiecare farmacie are propriul inventar de medicamente;
  - pentru fiecare medicament sunt păstrate cantitățile disponibile și un prag critic;
  - interogările frecvente ale stocurilor sunt optimizate prin caching distribuit (Redis).

2 Obiectivele proiectului

Obiectivul principal este dezvoltarea unei platforme web care:

- gestionează rețete medicale, farmacii și stocuri de medicamente;
- permite doctorilor să emită rețete și farmaciilor să le elibereze/respinga;
- oferă autentificare SSO (OAuth2/OIDC cu Keycloak) și roluri (Admin / Doctor / Farmacist);
- utilizează caching distribuit (Redis) pentru interogările de stoc frecvente;
- include un modul de monitorizare (Prometheus + Grafana) pentru metrici tehnice relevante;
- rulează ca un stack Docker Swarm cu servicii separate și cel puțin un serviciu replicat care interacționează cu o funcționalitate avansată (caching).

3 Arhitectura soluției

3.1 Componente Docker (servicii)

Arhitectura aplicației va conține următoarele servicii:

1) **gateway-service** (componentă proprie)

   Microserviciu backend scris în Python (Flask), care acționează ca *API Gateway*. Rolurile principale:

- punct unic de intrare în backend pentru `frontend-web`;
- validează token-urile de autentificare emise de Keycloak (JWT / OIDC);
- face rutarea request-urilor către microserviciile interne: `user-profile-service`, `prescription-service`, `inventory-service`;
- poate implementa reguli simple de rate limiting și logare centralizată a apelurilor.

2) **frontend-web** (componentă proprie)

   Aplicație web (HTML+JS) care oferă interfața utilizator:

   - logare prin Keycloak (SSO);
   - afișarea listelor de rețete, farmacii și stocuri;
   - formulare pentru emiterea rețetelor (Doctor) și pentru procesarea lor (Farmacist);
   - afișarea metricilor relevante (sau link către dashboard-ul Grafana pentru Admin).

3) **user-profile-service** (componentă proprie)

   Microserviciu backend (REST) responsabil cu profilurile utilizatorilor:

   - primește meta-date din token-ul Keycloak (username, email, rol);
   - creează și actualizează profiluri interne (Admin / Doctor / Farmacist);
   - stochează asocieri precum farmacist–farmacie, specializări ale doctorilor;
   - expune endpoint-uri pentru consultarea și actualizarea profilurilor (în special pentru Admin).

4) **prescription-service** (componentă proprie)

   Microserviciu backend (REST) care gestionează rețetele:

   - creează rețete noi pentru pacienți (doar Doctor);
   - modifică statusul rețetelor (în curs, eliberată, respinsă) la cererea farmacistului;
   - oferă liste și detalii de rețete filtrate după doctor, farmacie, status;
   - persistă datele în PostgreSQL printr-un ORM;
   - expune metrici tehnice pentru Prometheus (număr rețete, latență, erori).

5) **inventory-service** (componentă proprie, *replicată*)

   Microserviciu backend (REST) responsabil de gestionarea stocurilor de medicamente pe farmacie:

   - păstrează inventarul per farmacie (medicament, cantitate disponibilă, prag critic);
   - răspunde la interogări de tip "există suficient stoc pentru această rețetă?";
   - actualizează stocurile la eliberarea unei rețete sau la recepția unui transport;
   - folosește `redis-cache` pentru a citi rapid stocurile frecvent accesate;
   - expune metrici pentru Prometheus (număr interogări, latență, rate de cache hit/miss).

   Componenta `inventory-service` va fi configurată în modul *replicated* în Docker Swarm, interacționând direct cu funcționalitatea avansată de caching (Redis).

6) **postgres-db** (open-source)

   Bază de date PostgreSQL utilizată pentru persistența entităților:

- utilizatori și profiluri (`UserProfile`);
- rețete (`Prescription`, `PrescriptionItem`);
- farmacii (`Pharmacy`);
- stocuri de medicamente (`InventoryItem`);
- eventual loguri de audit.

Microserviciile `user-profile-service`, `prescription-service` și `inventory-service` folosesc un ORM (ex. SQLAlchemy) pentru accesul la DB.

7) **keycloak** (open-source)

Server de autentificare și autorizare (SSO):

- oferă login SSO bazat pe OAuth2 / OpenID Connect;
- gestionează utilizatorii și rolurile (Admin / Doctor / Farmacist);
- emite token-uri JWT folosite de `gateway-service` și de celelalte servicii backend.

8) **redis-cache** (open-source)

Serviciu de caching distribuit:

- folosit în principal de `inventory-service` pentru interogările frecvente de stoc;
- reduce numărul de interogări către PostgreSQL pentru citiri intensive;
- permite construirea unui cache distribuit între replicile `inventory-service`.

9) **prometheus** (open-source)

Sistem de colectare de metrici:

- colectează metrici de la `gateway-service`, `prescription-service`, `inventory-service`, `user-profile-service`;
- interoghează periodic endpoint-uri de tip `/metrics` expuse de microservicii.

10) **grafana** (open-source)

Interfață de monitorizare:

- oferă dashboard-uri pentru metricile colectate de Prometheus;
- permite vizualizarea stării și performanței serviciilor (trafic, erori, latență).

3.2 Rețele și interconectare

Vor fi definite cel puțin două rețele Docker overlay, pentru separarea logică a componentelor:

- **frontend_net**:
  - conectează `frontend-web` cu `gateway-service` și `keycloak`;
  - `frontend-web` vede doar `gateway-service` (și Keycloak pentru autentificare).
- **backend_net**:
  - conectează `gateway-service` cu `user-profile-service`, `prescription-service`, `inventory-service`, `postgres-db`, `redis-cache`, `prometheus`;

– `postgres-db`, `redis-cache`, `prometheus` nu sunt expuse direct în exterior.

Comunicarea se face prin numele de serviciu (DNS Docker), injectate în aplicații prin *environment variables* (nu se folosesc adrese `localhost` hardcodate).

3.3 Stack Docker Swarm și replicare

Întreaga soluție va fi livrată ca stack Docker Swarm, definit într-un fișier `stack.yml`, care descrie serviciile, rețelele și volumele.

Componenta `inventory-service` va fi configurată cu mai multe replici (`deploy.replicas` > 1), toate replicile folosind aceeași bază de date PostgreSQL și același Redis.

Rolul replicării este strâns legat de modul de caching distribuit:

- interogările de stoc (ex. "are farmacia X suficient din medicamentul Y pentru această rețetă?") pot fi foarte frecvente;
- Docker Swarm distribuie cererile între replicile `inventory-service`, permițând scalarea operațiilor de citire;
- fiecare instanță consultă mai întâi cache-ul din Redis; dacă există o intrare pentru cheia construită din (farmacie, medicament), răspunsul este returnat direct din cache.

4 Module și funcționalități

Conform cerințelor, proiectul expune cel puțin 5 module, din care 3 sunt comune (autentificare, profil utilizator, bază de date), iar 2 sunt module avansate.

4.1 Modul de autentificare (SSO)

Tehnologii: Keycloak, OAuth2 / OpenID Connect, JWT.

Funcționalități:

- login SSO prin Keycloak:
  - frontend-ul redirecționează utilizatorul către Keycloak;
  - după autentificare, aplicația primește un *access token* și (opțional) un *ID token*;
- `gateway-service` validează token-urile JWT la fiecare request;
- endpoint-urile backend sunt protejate și accesibile doar cu token valid și rol corespunzător.

4.2 Modul de profil utilizator și managementul rolurilor

Roluri principale:

- **ADMIN**: poate gestiona utilizatori, farmacii, poate vizualiza metrici;
- **DOCTOR**: poate crea rețete, poate vizualiza rețele proprii;
- **FARMACIST**: poate vedea rețete pentru farmacia sa, poate elibera sau respinge rețete.

Funcționalități modul:

- la primul login, `user-profile-service` primește meta-date din token (username, email, rol) și creează un profil intern;
- profilul conține:
  - ID utilizator;
  - rol intern (ADMIN / DOCTOR / FARMACIST);
  - pentru farmacist: farmacia asociată;
  - pentru doctor: specializare.
- rolurile sunt folosite pentru:
  - restricționarea accesului la anumite endpoint-uri;
  - ascunderea în UI a butoanelor/funcțiilor nepermise unui rol.

4.3 Modul bază de date

Tehnologii: PostgreSQL, ORM (ex. SQLAlchemy).

Entități principale:

- `UserProfile` (profil utilizator);
- `Pharmacy` (farmacie);
- `Prescription` și `PrescriptionItem` (rețete și linii de rețetă);
- `InventoryItem` (stocuri de medicamente per farmacie).

Integrare DB:

- se folosește un layer de repository (ex. `PrescriptionRepository`, `InventoryRepository`);
- entitățile ORM sunt mapate la tabelele PostgreSQL;
- tranzacțiile sunt folosite pentru operații critice (eliberare rețetă + actualizare stoc).

4.4 Modul avansat 1: Caching distribuit pentru stocuri (Redis)

Tehnologii: Redis, Python, `inventory-service`.

Scop:

- reducerea timpilor de răspuns și a încărcării pe PostgreSQL pentru interogări frecvente de stoc.

Mecanism:

- la interogări de tip "care este stocul medicamentului X în farmacia Y?":
  - `inventory-service` verifică mai întâi în Redis;
  - dacă există o intrare pentru cheia (Y, X) și nu a expirat, răspunsul este returnat din cache;
  - altfel, datele sunt citite din PostgreSQL, iar rezultatul este scris în Redis cu un TTL.
- la actualizarea stocurilor (ex. eliberare rețetă sau recepție stoc nou):
  - `inventory-service` actualizează PostgreSQL;
  - invalidează sau actualizează intrările relevante din Redis pentru a menține consistența.

Cele mai multe interogări de stoc sunt deservite de replicile `inventory-service`, care folosesc același Redis ca sistem de caching distribuit.

4.5 Modul avansat 2: Monitorizare și observabilitate (Prometheus + Grafana)

Tehnologii: Prometheus, Grafana.

Scop:

- monitorizarea în timp real a comportamentului aplicației și a microserviciilor.

Descriere:

- microserviciile expun un endpoint `/metrics` (ex. prin Prometheus client pentru Python);
- Prometheus colectează metrici precum:
  - număr de request-uri per serviciu;
  - latență medie;
  - număr de erori (4xx / 5xx);
  - număr de interogări de stoc și rata de cache hit/miss.
- Grafana afișează dashboard-uri cu:
  - status servicii (up/down);
  - grafice de trafic și latență;
  - grafice de activitate (rețete create / eliberate, interogări de stoc).

5 Conformitatea cu regulile generale

1. **Autentificare și autorizare**
   - autentificare SSO prin Keycloak (serviciu dedicat);
   - protocol OAuth2 / OpenID Connect;
   - `gateway-service` validează token-urile JWT și protejează endpoint-urile backend;
   - microserviciile interne se bazează pe informațiile de rol din token pentru autorizare.

2. **Managementul rolurilor**
   - roluri principale: ADMIN, DOCTOR, FARMACIST;
   - Keycloak furnizează meta-date despre utilizator (inclusiv roluri);
   - `user-profile-service` creează și menține profiluri interne pe baza acestor meta-date;
   - accesul la endpoint-uri și acțiuni în UI este controlat de roluri.

3. **Baza de date și ORM**
   - bază de date: PostgreSQL;
   - integrare prin ORM (ex. SQLAlchemy) și un layer de repository;
   - entitățile din DB (UserProfile, Pharmacy, Prescription, InventoryItem) sunt mapate la structuri interne (modele/DTO-uri) folosite în logică de business.

4. **Livrarea proiectului**
   - proiectul este livrat ca stack Docker Swarm, cu servicii pentru: `frontend-web`, `gateway-service`, `user-profile-service`, `prescription-service`, `inventory-service`, `postgres-db`, `keycloak`, `redis-cache`, `prometheus`, `grafana`;

- stack-ul pornește aplicația completă și poate include date demonstrative inițiale în DB.

5. **Microservicii și containere**
   - pentru fiecare componentă proprie există Dockerfile:
     - Dockerfile pentru `gateway-service`;
     - Dockerfile pentru `frontend-web`;
     - Dockerfile pentru `user-profile-service`;
     - Dockerfile pentru `prescription-service`;
     - Dockerfile pentru `inventory-service`.
   - PostgreSQL, Keycloak, Redis, Prometheus și Grafana folosesc imagini oficiale open-source.

6. **Structura proiectului**
   - cel puțin 5 componente: `gateway-service`, `frontend-web`, `user-profile-service`, `prescription-servic` `inventory-service`, `postgres-db`, `keycloak`, `redis-cache`, `prometheus`, `grafana`;
   - cel puțin 2 componente proprii: `gateway-service`, `frontend-web`, `user-profile-service`, `prescription-service`, `inventory-service`.

7. **Stack Docker Swarm**
   - toate componentele sunt definite ca servicii în fișierul `stack.yml`;
   - lansarea se face cu `docker stack deploy`, folosind modul `replicated` pentru `inventory-service`;
   - configurațiile de rețele și volume sunt definite în același fișier.

8. **Interconectarea componentelor**
   - serviciile comunică prin numele DNS Docker (ex. `postgres-db`, `redis-cache`, `keycloak`);
   - adresele sunt injectate în aplicații prin *environment variables*;
   - nu se folosesc adrese hardcodate de tip `localhost` în codul microserviciilor.

9. **Rețele și securitate**
   - rețele separate (`frontend_net`, `backend_net`) limitează vizibilitatea serviciilor;
   - `frontend-web` vede doar `gateway-service` și Keycloak;
   - `gateway-service` vede doar serviciile backend necesare (`user-profile-service`, `prescription-service`, `inventory-service`, `postgres-db`, `redis-cache`, `prometheus`);
   - `postgres-db`, `redis-cache`, `prometheus`, `grafana` nu sunt accesibile direct din exterior (decât Grafana eventual pe un port protejat).

10. **Replicare și testare**
    - componenta proprie `inventory-service` este replicată la mai multe instanțe;
    - această componentă interacționează cu:
      - modulul de caching distribuit (`redis-cache`);
      - baza de date PostgreSQL pentru operații de citire/scriere pe inventar.
    - vor fi definite teste (unitare / de integrare) pentru:

- logica de business legată de verificarea și actualizarea stocurilor;
- accesul la baza de date prin repository-urile `InventoryRepository`;
- comportamentul API-urilor principale din `inventory-service` și `prescription-service`.

6 Fluxuri principale

6.1 Emiterea unei rețete

1. Doctorul accesează `frontend-web` și este redirecționat către Keycloak pentru autentificare.
2. După autentificare, frontend-ul primește token-ul și îl transmite către `gateway-service`.
3. Doctorul completează datele rețetei (pacient, medicamente, doze, farmacie) și trimite formularul.
4. `gateway-service` validează token-ul și trimite request-ul către `prescription-service`.
5. `prescription-service` validează rolul (Doctor), salvează rețeta în PostgreSQL și întoarce un răspuns către frontend.
6. Frontend-ul afișează confirmarea și permite doctorului să consulte ulterior rețeta și statusul acesteia.

6.2 Verificarea stocului și eliberarea unei rețete

1. Farmacistul se autentifică prin SSO și accesează lista de rețete din `frontend-web`.
2. `frontend-web` cere lista rețetelor prin `gateway-service` către `prescription-service`.
3. La deschiderea unei rețete specifice, `frontend-web` cere detalii de stoc pentru medicamentele din rețetă de la `inventory-service` (prin `gateway-service`).
4. `inventory-service` verifică mai întâi în `redis-cache` dacă există informații actuale despre stoc:
   - dacă există date valide în cache, răspunsul este furnizat direct din Redis;
   - dacă nu, `inventory-service` citește stocurile din PostgreSQL, calculează disponibilitatea și salvează rezultatul în Redis cu un TTL.
5. Dacă stocul este suficient, farmacistul poate marca rețeta drept "eliberată":
   - `prescription-service` actualizează statusul rețetei în baza de date;
   - `inventory-service` actualizează cantitățile din stoc și invalidează/actualizează intrările relevante din Redis.
6. Frontend-ul afișează statusul actualizat al rețetei pentru farmacist și, eventual, pentru doctor/pacient.