

**SISTEME DISTRIBUITE**

**Tema 2**

**Documentație**

Asynchronous Communication

Sensor Monitoring System and Real-Time Notification

Pop Claudia Dorina

Grupa: 30241

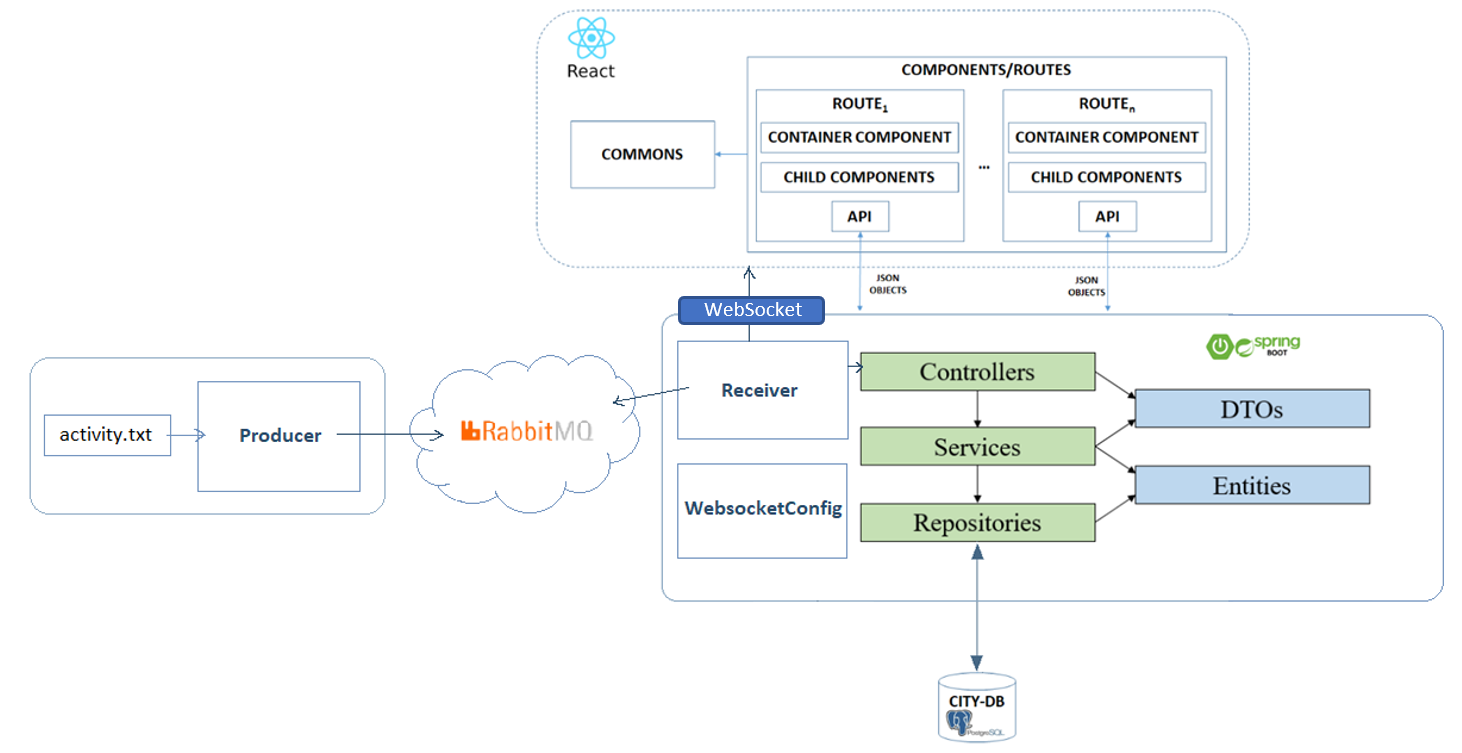
Cuprins

[1. Arhitectura conceptuală a sistemului distribuit 3](#_Toc56673660)

[2. Diagrama UML de Deployment 4](#_Toc56673661)

[3. Build and execution considerations 5](#_Toc56673662)

# Arhitectura conceptuală a sistemului distribuit



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

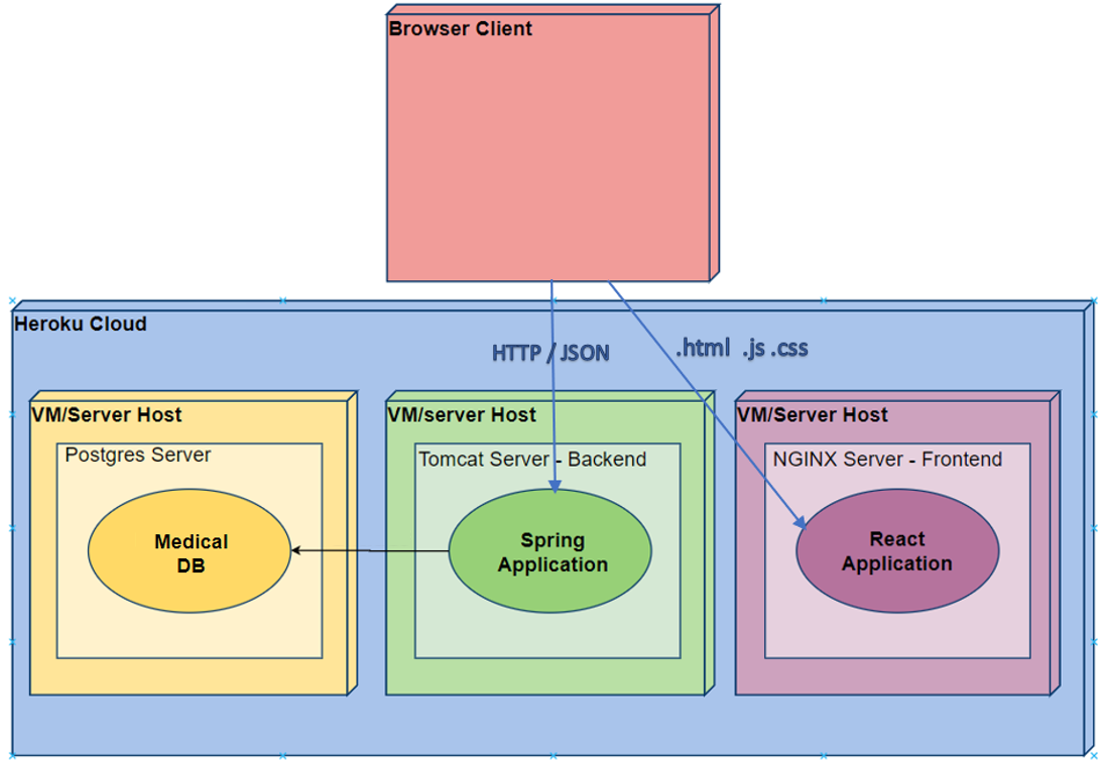
Din arhitectura aplicației React reies principalele componente. Directorul Commons conține fișiere ce se utilizează global în aplicație. Fiecare pagină diferită ce va fi vizibilă în browser este reprezentată de un container care conține mai multe componente. Componentele API sunt cele care trimit cererile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

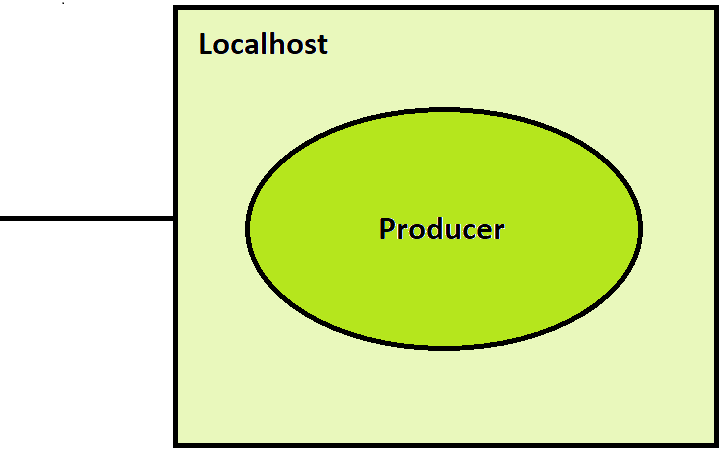
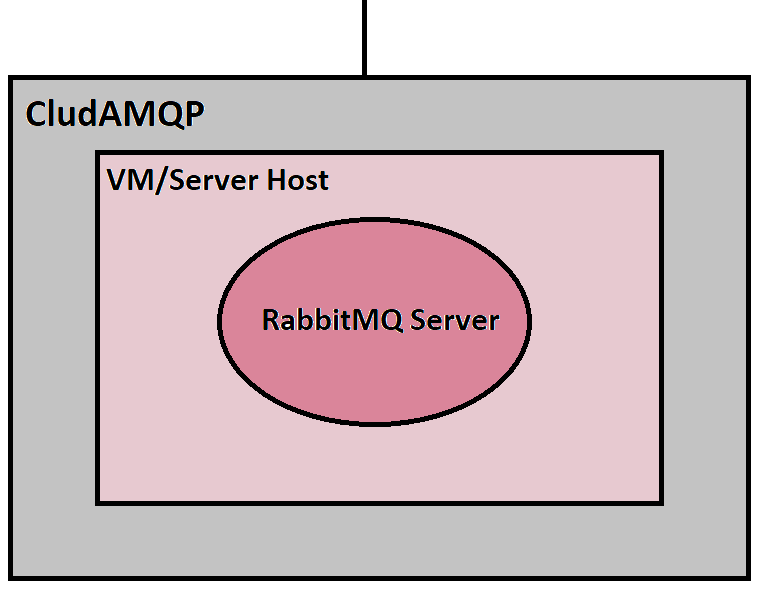
Partea de backend are o arhitectură bazată pe layere. Controllerele sunt componentele capabile să primească cererile de tip HTTP REST de la partea de frontend a sistemului distribuit și să trimită răspunsuri. Acestea comunică in mod direct cu Serviciilefolosind obiecte de tip Data Transfer. Aceste obiecte 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.

Modulul Producer are rolul de a simula citirea și transmiterea unor date de la niște senzori care monitorizează activitatea pacienților. Aici se citesc informațiile din fișier și se trimite câte un set de date la fiecare secundă. Aceste date ajung într-o coadă și sunt preluate de aici de către Receiver(Consumer). Consumerul are rolul de a verifica datele primite și de a le stoca în baza de date. Dacă apar anomalii în datele primite de la un pacient, atunci un client Java notifică îngrijitorul responsabil de pacient prin intermediul unor websocket-uri.

# 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. Separat, pe un alt cloud se află serverul RabbitMQ care comunica cu Backend-ul.





**Figura 3.** Diagrama UML de Deployment

# Build and execution considerations

Trebuie să ruleze în paralel aplicația de Backend, aplicația de Frontend și modulul care citește date de la senzor. În același timp serverul RabbitMQ trebuie să fie activ pentru a putea media transmiterea datelor dintre Producer și Consumer.

Credențiale pentru testarea aplicației:

|  |  |  |
| --- | --- | --- |
| Utilizator | Nume utilizator | Parolă |
| Doctor | Doctor | doctor |
| Patient | Molnar Paul | patient |
| Caregiver | Rus Cristiana | caregiver |
| Caregiver | Mureșan Călin | caregiver |