Distributed Systems

2020-2021

Assignment 2

Indirect Communication Using Queues and Web sockets

Online Medication Platform

Student:Pop Cristian

Grupă: 30244

1. Requirements

Pacientul a instalat un set de senzori în casă pentru a-și monitoriza activitatea și detectează automat dacă are probleme și alertează îngrijitorii sau medicii. Senzorii trimit date ca tuple (pacient\_id, start\_time, end\_time, activity\_label), unde start\_time și end\_time reprezintă data și ora la care fiecare activitate a început și s-a încheiat, în timp ce, eticheta “activity” reprezintă tipul de activitate desfășurată de persoană: Plimbare, Toaletă, Duș, Dormit, Mic dejun, Prânz, Cină, Gustare, Timp liber / TV. Două activități consecutive sunt diferite. Implementați un sistem bazat pe un broker de mesaje middleware care colectează date de la senzori și le pre-procesează înainte de a le stoca în baza de date la pacientul corespunzător. Dacă aplicația Consumer care preprocesează datele, detectează o activitate anormală conform următorul set de reguli, notifică asincron îngrijiritorul pacientului respectiv.

Probleme:

➢ R1: Perioada de somn mai mare de 7 ore;

➢ R2: Activitatea de ieșire (în aer liber) este mai mare de 5 ore;

➢ R3: Perioada petrecută în baie depășește 30 de minute.

Simulatorul de senzori ar trebui dezvoltat ca o aplicație independentă pentru a citi activitățile monitorizate din fișierul activity.txt, și configurat ca un producător de mesaje care trimite în fiecare secundă la coada definită, datele preluate. Activitățile trebuie trimise în coadă, folosind următorul format JSON:

{ “patient\_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”,

“activity” : “Sleeping”,

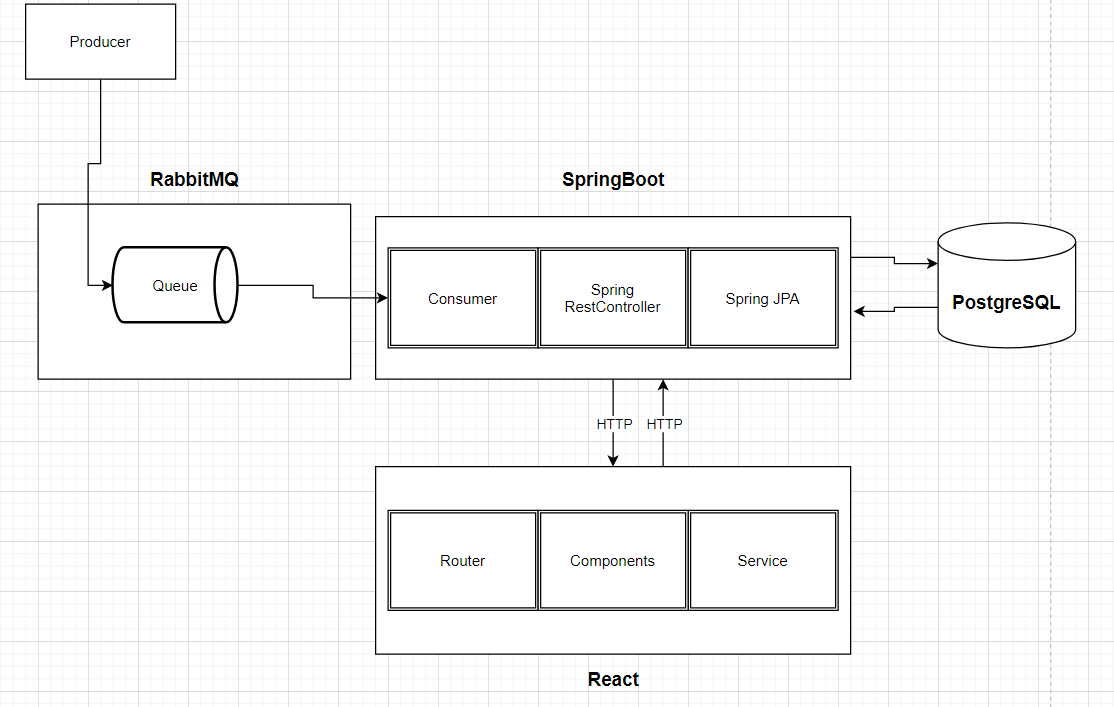
"start" : 1570654800000,

"end" : 1570654860000 }

1. Conceptual Architecture of the Distributed System

Aplicația concepută este de tip Client-Server, legătura dintre Front-End și Back-End realizându-se printr-un API. Pentru implementarea părții de Back-End am folosit limbajul Java. Totodată pentru conexiunea la baza de date PostgreSQL, am folosit framework-ul Java Spring și Hibernate, iar pentru partea de Front-End am folosit React care este o librărie pentru Javascript. Fiind o aplicație de tip Client-Server, Client-ul trimite request-uri către Server, iar acesta îi trimite în schimb reply-uri. Pentru transmiterea activităților, s-a folosit broker-ul pentru mesaje, RabbitMQ, care acceptă și transmite mesaje. Acest proces necesită trei elemente esențiale și anume:

* Producer – acesta are rolul de a trimite mesajele.
* Queue – aceasta este asemănătoare cu o cutie poștala. Aici sunt stocate mesajele trimise de către Producer și tot de aici, Consumer-ul încearcă sa preia datele.
* Consumer – așteaptă sa primească mesaje.



Pentru transmiterea mesajelor, producerul citește fișierul cu activități, procesează fiecare linie, transformându-le în obiecte JSON, iar apoi acesta le pune in coadă. Pe partea de Back-End, s-a implementat Consumer-ul care cu ajutorul adnotării @RabbitListener marchează ca o metodă să fie ținta unui listener Rabbit pentru mesaje. În cadrul acestei adnotări este specificată numele cozii. Fiecare mesaj primit este verificat prin cele trei condițiii specificate anterior, în cazul în care se detectează o activitate anormală aceasta este stocată în baza de date. Acest assignment presupunea și înstiințarea îngrijitorilor și a doctorilor prin intermediul Web Socket-urilor, dar această funcționalitate nu a mai fost implementată.

1. UML Deployment diagram

