MINISTRY OF NATIONAL EDUCATION
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

Distributed Systems

Assignment 2

Online Medication Platform

Student: Horatiu-Stefan Spaniol

Group: 30442

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

Table of contents

Requirement Analysis

Assignment Specification Requirements

The patient has deployed a set of sensors in their house to monitor their activity and automatically detect if they have problems and alert the caregivers or the doctors. The sensors send data as tuples (patient_id, start_time, end_time, activity_label), where start_timeand end_timerepresent the date and time when each activity has started and ended while the activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming.Two consecutive activities are different.Implement a system based on a message broker middleware that gathers data from the sensors and pre-processes them before storing them in the database to the corresponding patient. If the queue consumer application that preprocesses the data detects an anomalous activity according to the following set of rules, it notifies asynchronously the caregiver application that a patient has problems:

- R1: Sleep period longer than 7hours
- R2: The leaving activity (outdoor) is longer than 5 hours
- R3: Period spent in bathroom is longer than 30 minutes

A sensor simulator should be developed as a standalone application to read the sensor monitored activities from the file activity.txt,configuredas a message producer andsend every second one monitored sample datato the queue defined.The file activity.txtcan be downloaded from http://coned.utcluj.ro/~salomie/DS_Lic/3_Lab_Project/Assignment_2/. The activities are sent to the queue using the following JSON format:

```
{
        “patient_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”,
        “activity” :  “Sleeping”,
        "start" : 1570654800000,
        "end" : 1570654860000
}
```

Functional Requirements

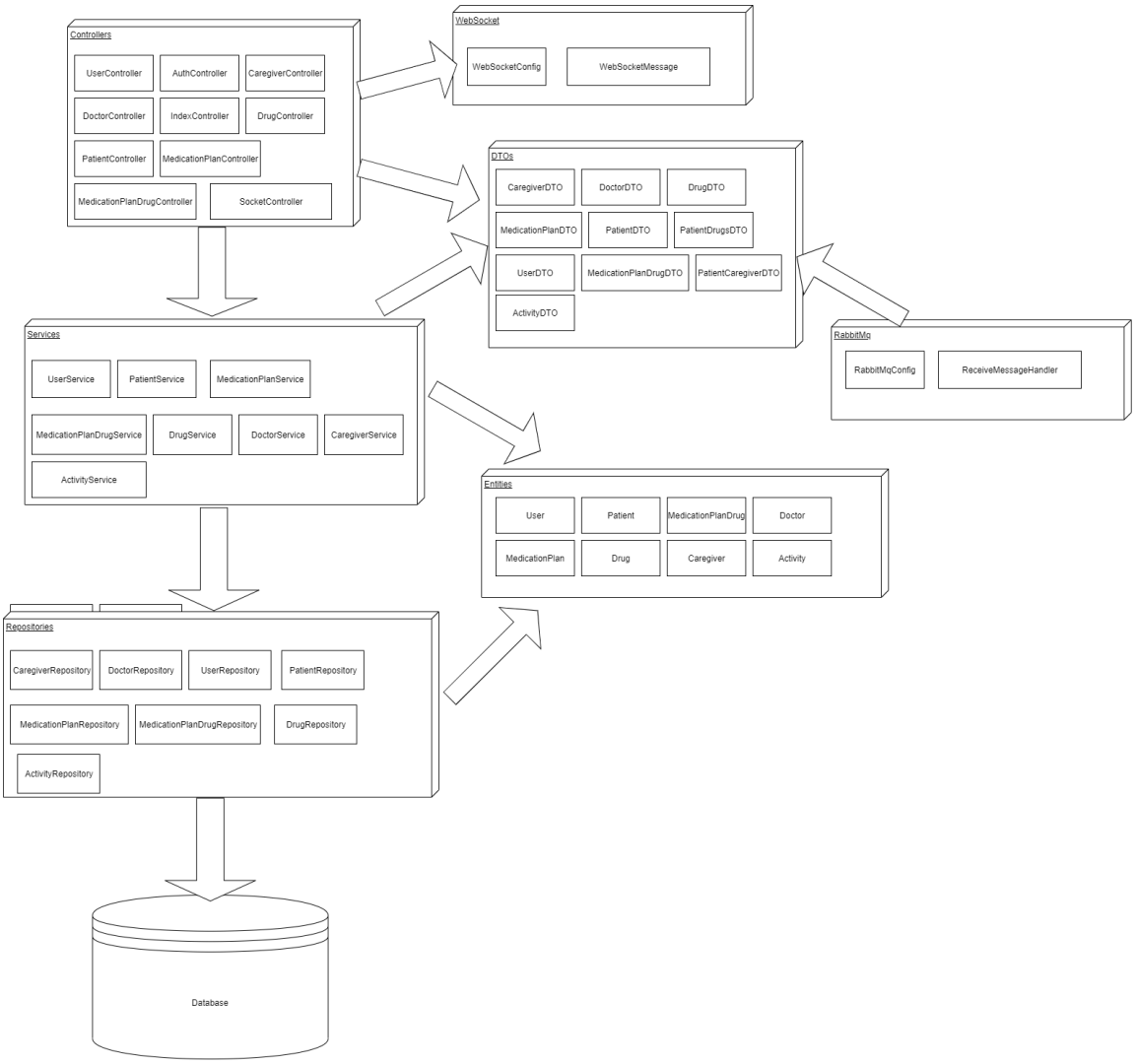
- The message-orientedmiddleware allowsthe sensor system tosend data tuples (patient_id, start_time, end_time, activity_label) in a JSON format
- The message consumer component of the system processes each message,applies the 3rules defined (R1-R3) and notifies asynchronously using WebSockets the caregiver application

Implementation technologies

- Use the following technologies: RabbitMQ, WebSockets.

Project Conceptual Architectural

The project conceptual architecture followed was the one provided in the requirements of the assignment 1. The architecture for the assignment 2 is just an extension from the previous assignment.



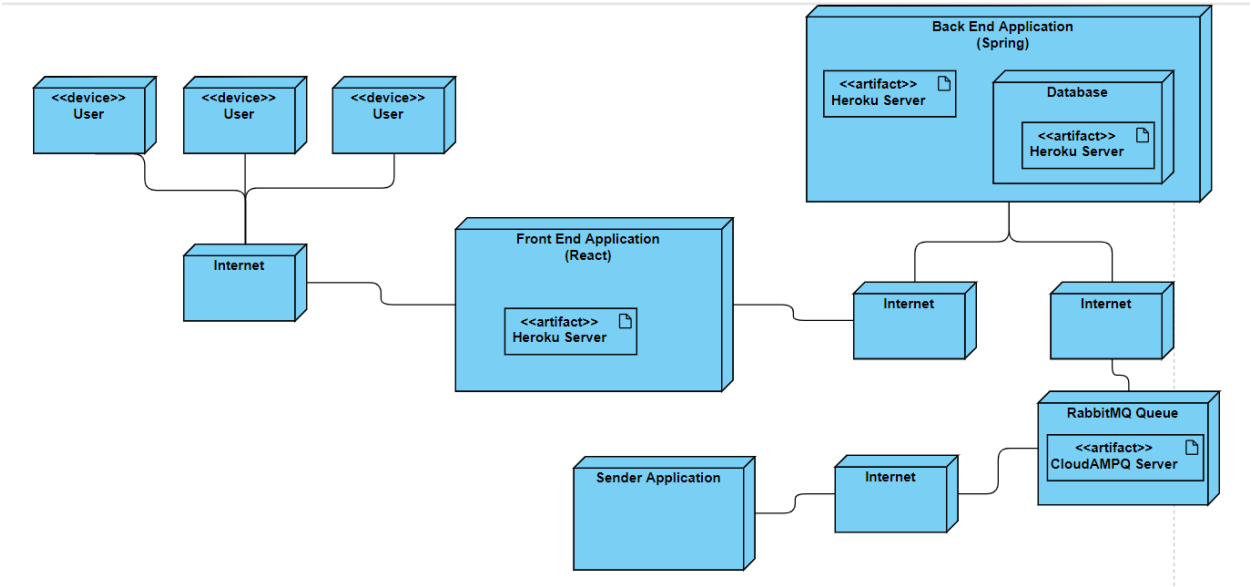
As we can see in the figure above, the architecture is very straight-forward.

The controllers receive information as requests or DTOs; if needed, they transform these requests into DTOs to be sent to the services, which in turn transform the DTOs into entities to be sent to the repositories that access the

database. Services also sometimes receive information directly from the controllers as requests, without transforming these requests into DTOs beforehand. Repositories return entities to the layer above, Services, which transform the entities into DTOs or into some other custom response to be sent back to the Controllers.

Extended in this architecture we can see the RabbitMq component which is the Consumer part of the assignment, integrated in the backend of the previous assignment. This component consumes messages from the RabbitMq queue and adds them to the database accordingly. Also the WebSocket component, used by the SocketController to connect to the frontend and send messages asynchronously.

Project Deployment Diagram



The deployment diagram shown above presents the physical relation between the applications that are deployed on nodes.

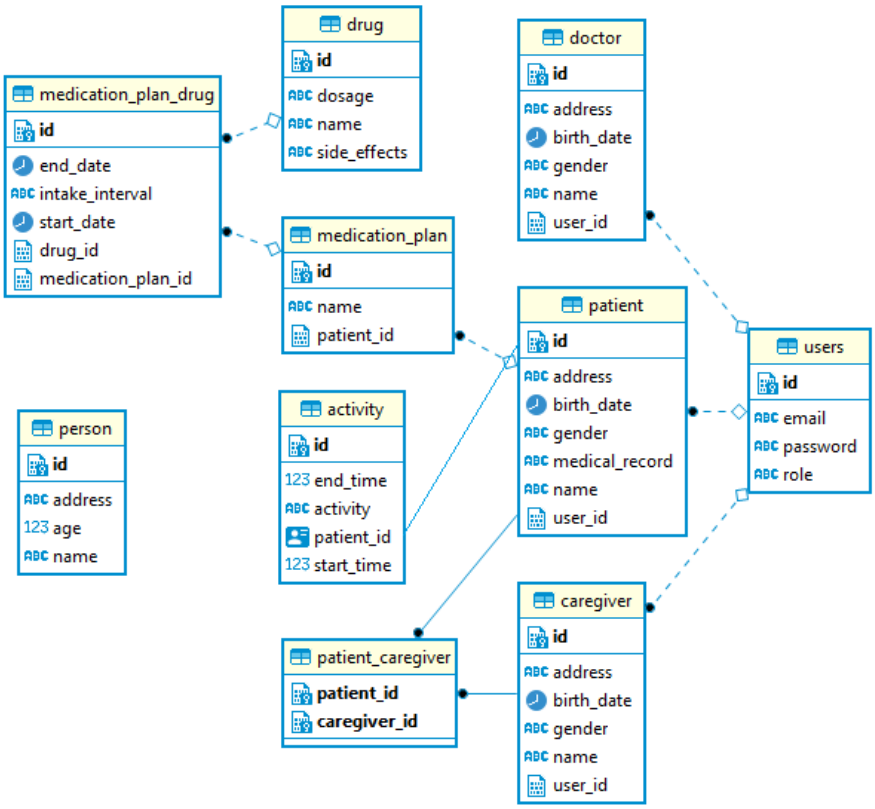
The user node symbolizes the fact that the user can use any device to access the application(i.e. computer, mobile phone, tablet); however, this device would have to communicate through the Internet node with the Front end application, which is deployed on a Heroku server. This application in turn, communicates through the Internet with the Back end application that runs on another Heroku server. The back end application also has a Heroku PostgreSQL Database, which was added prior to deploying the Backend application on that respective Heroku server.

The Back End Application communicates with the RabbitMQ Queue hosted on a CloudAMPQ Server, and consumes the messages the queue receives from the Sender application, which is not deployed. The Sender application is ran locally and receives the sensor data via a POST HTTP request. This is made for demonstration purposes, as the Sender application would ideally work asynchronously, sending messages to the RabbitMQ Queue as soon as it receives each piece of data from the sensor(assuming the sensor would be able to transmit the measured

data as soon as it is measured, one measurement at a time instead of a big piece of data containing a lot of measurements, like in our case).

Database Design

The database design process went through 5 versions, and the one below is the final version.



The “users” table contains 3 string fields that hold an email, password and the role of the user. This role is either a patient, caregiver or doctor.

The “doctor” table has account information of the doctor, and a one-to-one relation to the “users” table, indicating that a doctor has a user account associated to it and that user account belongs to just that one doctor.

The same principle applies to the “patient” and “caregiver” tables, which contain personal account information alongside the foreign key to the “users” table. However, the “patient” table and “caregiver” table are also in a many-to-many relationship, which can be seen illustrated by the “patient_caregiver” table. This depicts the fact that a

patient can have many caregivers associated to them, and a caregiver can also have many patients associated to them.(i.e. Patient P1 has associated Caregivers C1, C2 and Patient P2 has associated Caregivers C2, C3, which leads that Caregiver C2 has associated Patients P1 and P2).

The “patient” table also has a one-to-many relationship with the “medication_plan” table, as one patient can have many medication plans, but a medication plan can only belong to one patient. This medication plan contains the name of the specified medication plan, and a list of drugs that need to be taken(and more information about the way this intake should be done) by the patient associated with the medication plan.

The drugs table is a standalone table that contains drugs, with relevant information such as dosage and side effects.

The table “medicationplan_drug” is another many-to-many relationship between the “medicationplan” table and the “drugs” table, as it contains a foreign key towards the mentioned drug and a foreign key to the mentioned medication plan. It also contains extra information about intake intervals and the period of the intake. This table was added because intake intervals is not the same for different people even though the drug is the same.

The “activity” table was added in the assignment 2, to store each sensor data measurement. Each activity holds a reference to a patient, the name of the activity, the start and end time expressed in milliseconds from 01/01/1970 and an id for identification.

Build and Execution Considerations

In order to build and execute this project locally, there are some prerequisites. The first one of these, is have a Java JDK installed. The project was built with Java 11, so at least 11 is necessary to correctly build and execute the project. The second prerequisite would be the pgAdmin application for creating and connecting to a PostgreSQL database locally and to also host this database server. The third prerequisite would be a web browser, be it Microsoft Edge, Google Chrome, Mozilla Firefox or any other browser. The fourth prerequisite would be the Intellij IDEA application, which has the Java JDK mentioned beforehand associated with it.

On top of the prerequisites mentioned above, for running assignment 2, Docker Desktop(if ran on Windows) is needed in order to have a RabbitMQ Server running in a Docker container. However, this can be skipped by using an already deployed CloudAMPQ RabbitMQ Server (which, in my opinion, should be done as creating a new server should take less than 5 minutes).

In order to deploy the application, Docker, Gitlab’s Continuous Integration/Continuous Development pipelines and the Heroku cloud were used, and the application can be found on the following links:

Backend: https://ds2020-spaniolhoratiu-1.herokuapp.com

Frontend: https://ds2020-spaniolhoratiu-1-front.herokuapp.com

Bibliography

https://gitlab.com/ds_20201/react-demo

https://gitlab.com/ds_20201/spring-demo

https://bezkoder.com/spring-boot-react-jwt-auth

https://en.wikipedia.org/wiki/Deployment_diagram

https://docs.spring.io/spring-amqp/reference/html/

https://www.youtube.com/watch?v=WzO6_4jeliM