

Name: Nitu Cristian-Mihai

Group: 30241

Subject: Distributed systems

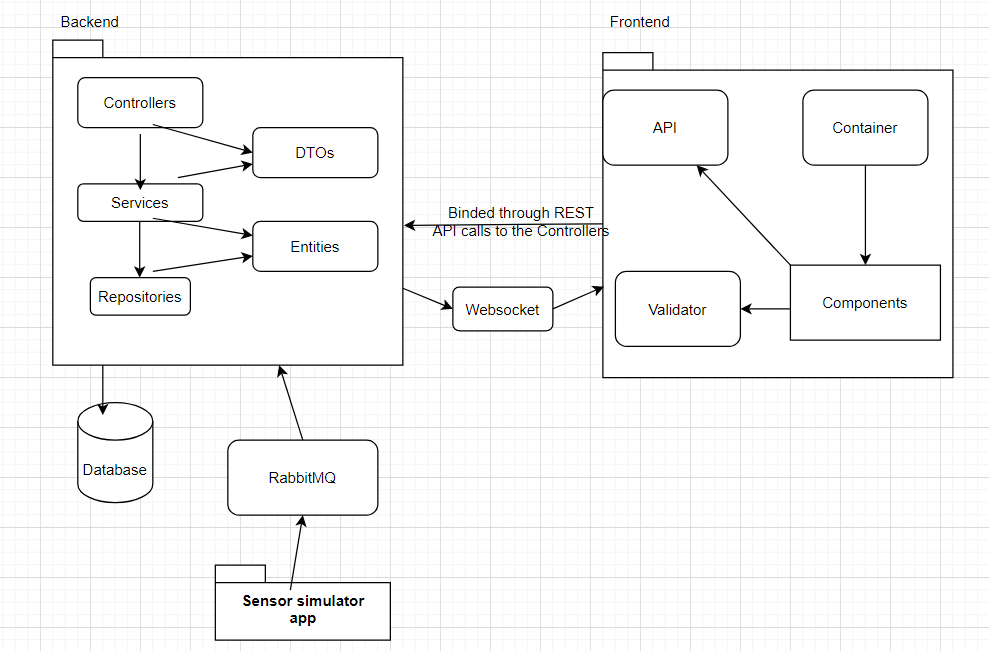
Assigmnent\_2: Medical Web Platform

**1.System description**

The scope of the distributed system is to manage pacients, caregivers and medications. The system can be accessed by three types of users after a login process: doctor, patient and caregiver. The doctor can perform CRUD operations of pacient accounts, caregiver accounts and on the medication list available in the system. Furthermore, the doctor can create a medication plan for a patient, consisting of a list of medication and intake intervals needed to be taken daily, and the period of the treatment. The patients can view their accounts and their medication plans. The caregivers can view their associated patients and the corresponding medication plans.

Furthermore, a set of sensor monitorize a patient, providing informations about the everyday activity. The informations contains the name of the activity and the period (start and end of the activity). There are some rules applied to the patient (such as the sleep period which cannot be longer than 7 hours). If any rule is violated, the patient’s caregiver is notified about that in real time, through a websocket.

**2.Conceptual architecture of the distributed system**

****

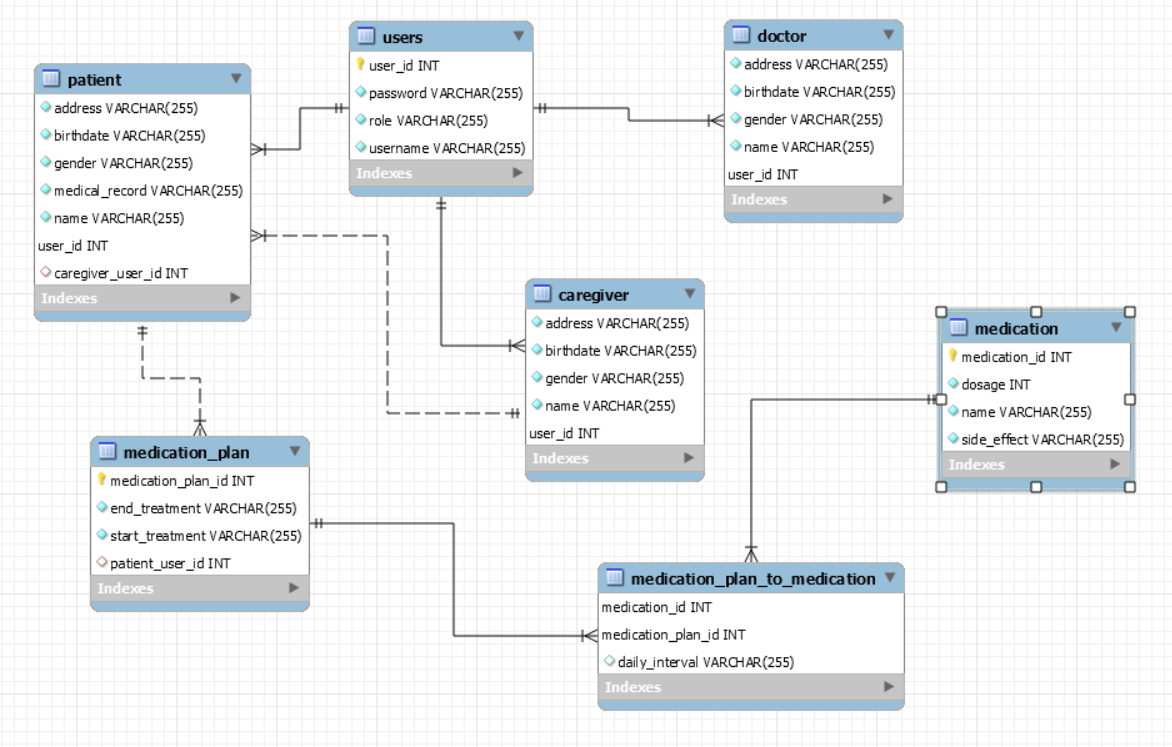
In the image depicted above, we can observe the conceptual architecture of the distributed system. It has a front-end part, a back-end part and a database used for storage and persistence. The front-end is a React application and is composed of multiple containers (a container for every page of our Web App), each container is has multiple components used for different tasks. The components have data-input validators and use different Rest APIs where are defined functions which make calls to our back-end Spring Boot application. Thus, when a user try to delete a patient from our application for example, that button makes a calls to the patient REST controller.

The Back-end is a Spring Boot application which makes Rest APIs calls for manipulating the Database CRUD operations. There is a controller corresponding to each entity mapped by a database table. The data exchanged between the back-end and the front-end applications is encapsulated in DTOs. The DTOs are then taken by the services, the layer where the transformations between Data Transfer Objects and entities takes place.The repositories have direct access to the database used.

The persistence space is a relational Postgres database.

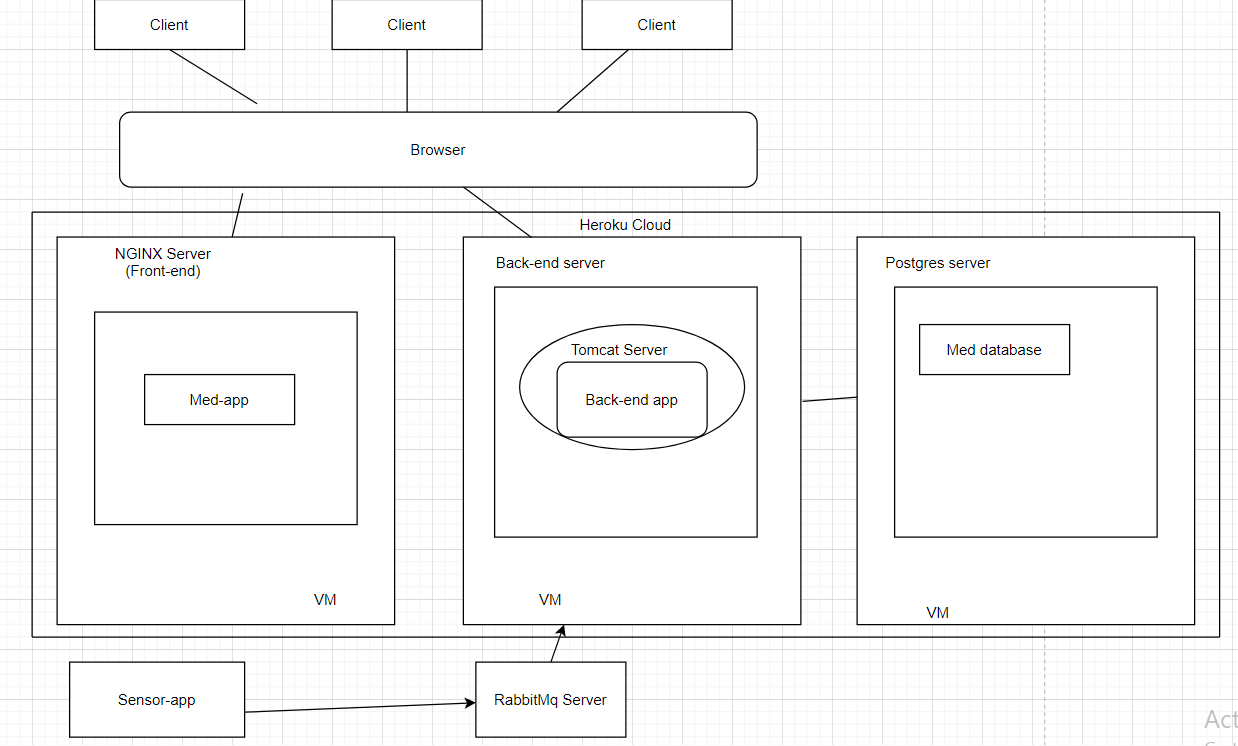
Additionally, we have a simulation app which gets information about the patient activity from a file, preprocess it, and save the activities in the database The information are sent to the back-end app through a Message-Broker Middleware (that is, RabbitMQ). In the back-end, there are some rules verified on every activity sent (one per second), and, if any of the rules is violated, the caregiver from the front-end is notified through a web-socket.

**3.Database**

****

In the picture above, we remark the database used in the development process of the application. As a user can be of 3 types, I considered as the best practice the joined table technique, where, as we can notice, the tables of the 3 types of users have as a primary key the same primary key as in the user table. This approach brings us a lots of benefits, as we don’t have to stock foreign keys to the other tables.The patient table, however, has a foreign key to caregiver table, as there is a one-to-many relationship between these 2 tables. The medication\_plan and medication tables are in many-to-many relationship which has as a common field the daily\_interval when the medications have to be taken. So that, I decided to create another table for this association.

**4.UML Deployment Diagram**

****

The client can access out web application using a browser. The deploy took place in Heroku Cloud for both applications (front-end and back-end). A client has access to the front-end application. This one is found in a NGINX Server. The npm server is necessary as it the front-end was being done in React. The front-end app communicate with the back-end app. For the back-end, a Tomcat Server was necessary. We built the database used in Postgres, so the Postgres server is another component of the UML Deployment Diagram.

**5.Build and execution consideration**

As we all know, the medical platform was deployed using the Heroku clouding platform. The first thing to mention are the URL’s for accessing the servers.

Front-end app: <https://nitu-cristian-medical-frontend.herokuapp.com>

Back-end app: <https://nitu-cristian-medical-backend.herokuapp.com>

Since it was deployed on heroku, there is no need to run the server locally.

For the possibility of seeing real-time notifications, we need to start locally the sensor app which sends infos to a queue.