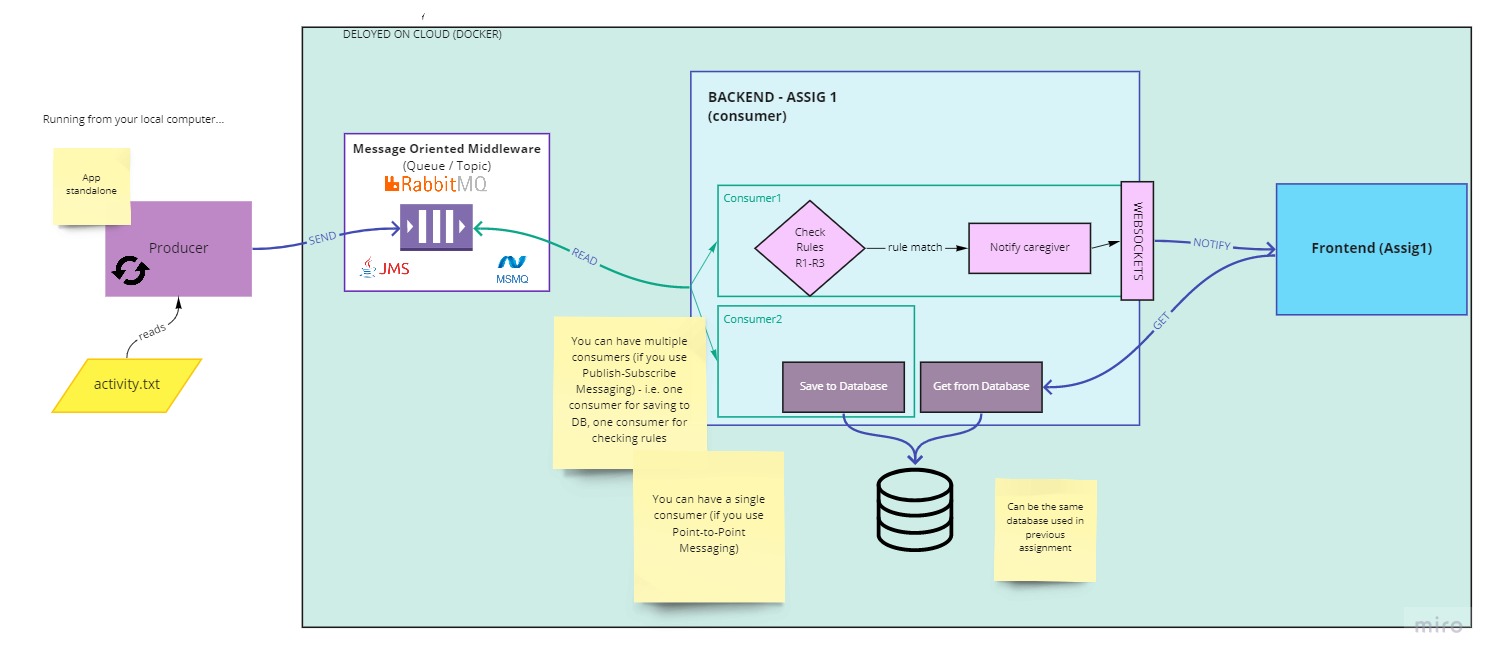
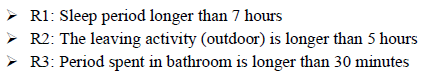
**Conceptual architecture of the distributed system**

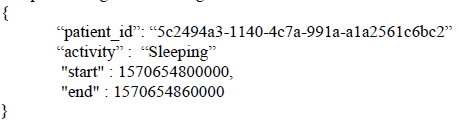
The application is designed as a distributed system, playing the role of an extension for the previously assigned task, where an Online Medication Platform was built for Doctors, Patients and Caregivers, where doctors could create medication plans for patients with existing or custom medications, caregivers could see their assigned patients and patients could see details about themselves and their medical record. As an extension to the previous application, which will be referred in this case as the Consumer, the current application, referred to as the Producer, will extract the activities from an .txt (text) file and send them to a Queue, after being parsed in a JSON format as described in the requirements. Then, we need to run the Consumer application, which will take the messages (i.e. activities) from the queue and display the anomalies based on the rules R1-R3 mentioned in the assignment requirements.



*Fig 1. The conceptual architecture of the distributed system*

**

*Fig 2. The three rules denoting anomalies in patient behavior*



*Fig 3. The JSON format of the activity structure*

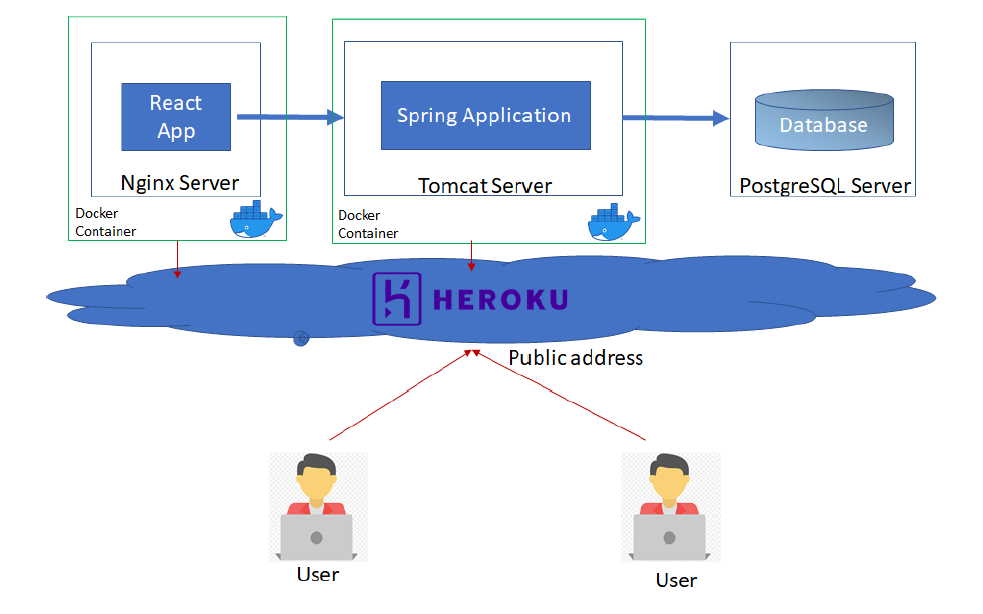
**Database design**

Data inside the database is necessary in order to keep track of information regarding patients, caregivers and doctors, as well as medications, side effects and medication plans designed for each user individually. Also, login information needs to be stored inside the database in order for the different users of the application to be able to retrieve contents about themselves or about other people or objects they are allowed to see or use. In the following diagram we have 11 tables, keeping data about patients and their medication plans, medication that may or may not be part of medication plans, caregivers that can take care of patients by viewing information about them and doctors that can manage data about caregivers, patients, medications and medication plans. In addition to the database from the previous assignment, we added 2 more tables, namely Activity and Activity\_Of, in the same manner we included last time the tables Caregiver and Care\_Of, to simulate a Many-to-Many relationship between the 2 table entities (Caregiver with Patient, last time and Patient with Activity, this time).

A screenshot of a cell phone

Description automatically generated

**UML deployment diagram**

****

**Build and execution considerations**

When running for the first time the application, the user needs to have an IDE installed, that can compile and run Java Spring related tasks, a database server in order for the data to be stored, Apache Tomcat and an IDE with React development support. In my implementation, I chose to use IntelliJ IDEA for the backend development, storing the data in a PostgreSQL database and using Visual Studio Code for the frontend development part using React components. For the application to be successfully executed, the project needs to have a Maven related structure, because of the connections and dependencies between IDE and database. The backend (executed from IntelliJ IDEA) needs to run first by executing the **SHIFT + F10** command. After the server has started, the frontend part of the application can be run, by opening a terminal inside Visual Studio Code and executing the command **npm start**. If everything is successful, a new tab inside your default browser will be opened with the link [*http://localhost:3000*](http://localhost:3000). By clicking the Login button in the top right corner of the recently opened webpage, the user will be redirected to the login page located under link [*http://localhost:3000/login*](http://localhost:3000/login)where he will be prompted to enter his login credentials, afterwards being redirected to a profile page, where they can view information about themselves. Depending on their role, they may be able to view details about themselves, meaning medication plans, if they are patients, the list of patients they care of ([*http://localhost:3000/patient*](http://localhost:3000/patient)), if they are caregivers or the list of caregivers they manage ([*http://localhost:3000/caregiver*](http://localhost:3000/caregiver)), together will all the information previously presented, if they are doctors. If the user wants to access certain screens, like patients of caregiver or medication plan of a patient, he needs to test it from Postman, since these pages have not been finished. If a user will try to enter a webpage he is restricted to access, he will be prompted to an error page, saying that the page he is looking for does not exist. Also, the user will stay logged in only for a certain time period. The application can be stopped, by executing the **CTRL + C** in the Visual Studio Code terminal, for stopping the backend and by executing the **CTRL+ F2** inside IntelliJ IDEA. In addition to the considerations from last time, the first application which needs to be run is called SensorSimulator, such that the messages will be sent to the Queue. For the Queue related part of the project, I used RabbitMQ which was later deployed to CloudAMQP platform. In addition to the last assignment, when a caregiver will login and view his profile, a notification will pop up related to the anomalies of the patient it takes care of. These can also be viewed in the Console logs of the browser. After that, the messages will be consumed and inserted into the database.

**Bibliography**

[1] <https://www.jmfurlott.com/handling-user-session-react-context/>

[2] <https://codesandbox.io/s/389o3qyoq5?file=/src/index.tsx>

[3] <https://www.baeldung.com/jpa-many-to-many>

[4] <https://gitlab.com/ds_20201/spring-demo>

[5] <https://bezkoder.com/react-jwt-auth/>

[6] <https://gitlab.com/ds_20201/react-demo>

[7] <https://www.geeksforgeeks.org/how-to-create-a-rest-api-using-java-spring-boot/>

[8] <https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/>

[9] <https://bezkoder.com/spring-boot-jwt-mysql-spring-security-architecture/>

[10] <https://www.javainuse.com/spring/boot-jwt>

[11] <https://www.rabbitmq.com/getstarted.html>

[12] <https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html>

[13] <https://www.baeldung.com/rabbitmq>