DISTRIBUTED SYSTEMS

Assignment 2

Asynchronous Communication

Sensor Monitoring System and Real-Time Notification

Chișluca Miruna

30243

Project 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_time* and *end_time* represent 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 7 hours
- 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*, configured as a message producer and send every second one monitored sample data to the queue defined.

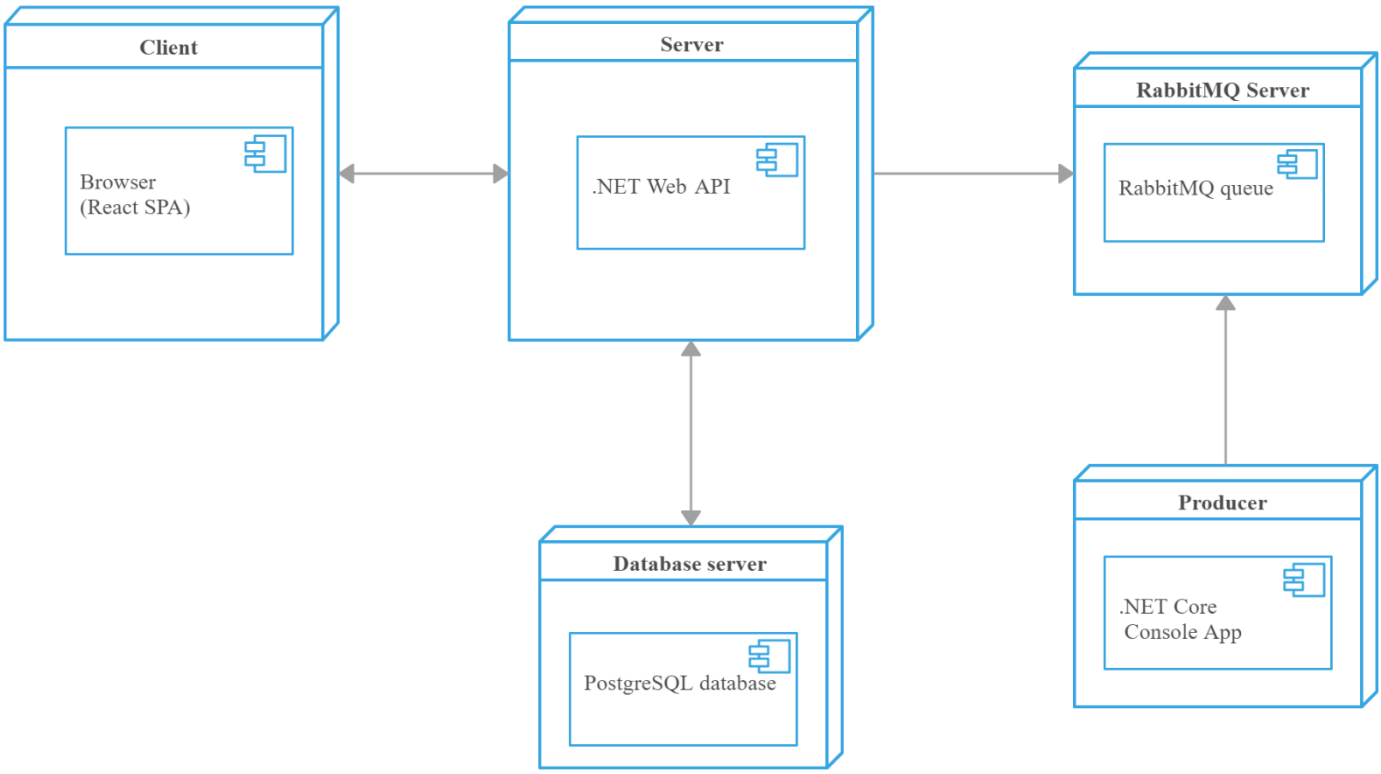
1. Conceptual architecture of the distributed system

The system implemented uses the same backend and frontend applications presented in the last assignment. RabbitMQ is a message broker. The two applications that communicate are called the producer and the consumer. The producer application, in my case, is a .NET Core console application that reads the activity.txt file, creates a list of Activity objects and sends each second one object from that list to the Rabbit message queue, meaning that it publishes messages to the queue. The consumer application is the backend itself. A service class called ActivityService connects to the same queue that the producer uses to publish messages and consumes those messages. While consuming one message, the application inserts the activity in the database and verifies whether the activity violates the corresponding rule for it. If it does, the frontend application is notified.

For notifying the frontend application about the rule violations I have used WebSockets. .NET has a library called SignalR that uses WebSockets for adding real-time functionality to an application. With the use of SIgnalR I have implemented a hub that has a method for sending a message to the client application and waits for the client to receive the message. This hub is injected into the Startup class and has an endpoint "/hubs/activity" which enables the communication between the API and the client application. This hub is used in the ActivityService, and called when one of the rules is violated with the appropriate message.

The frontend application uses a connection to connect to the endpoint defined in the API and receives the notifications from the API in a JSON format containing the patient's name and a message that corresponds to the activity that has violated the rule. The message is shown as a notification in the bottom right corner of the screen using React's snackbar.

2. UML Deployment diagram



3. READ ME

The application can be found here: https://medplatformapp.herokuapp.com/

Caregiver account:

- Username: caregiver1
- Password: abcd123