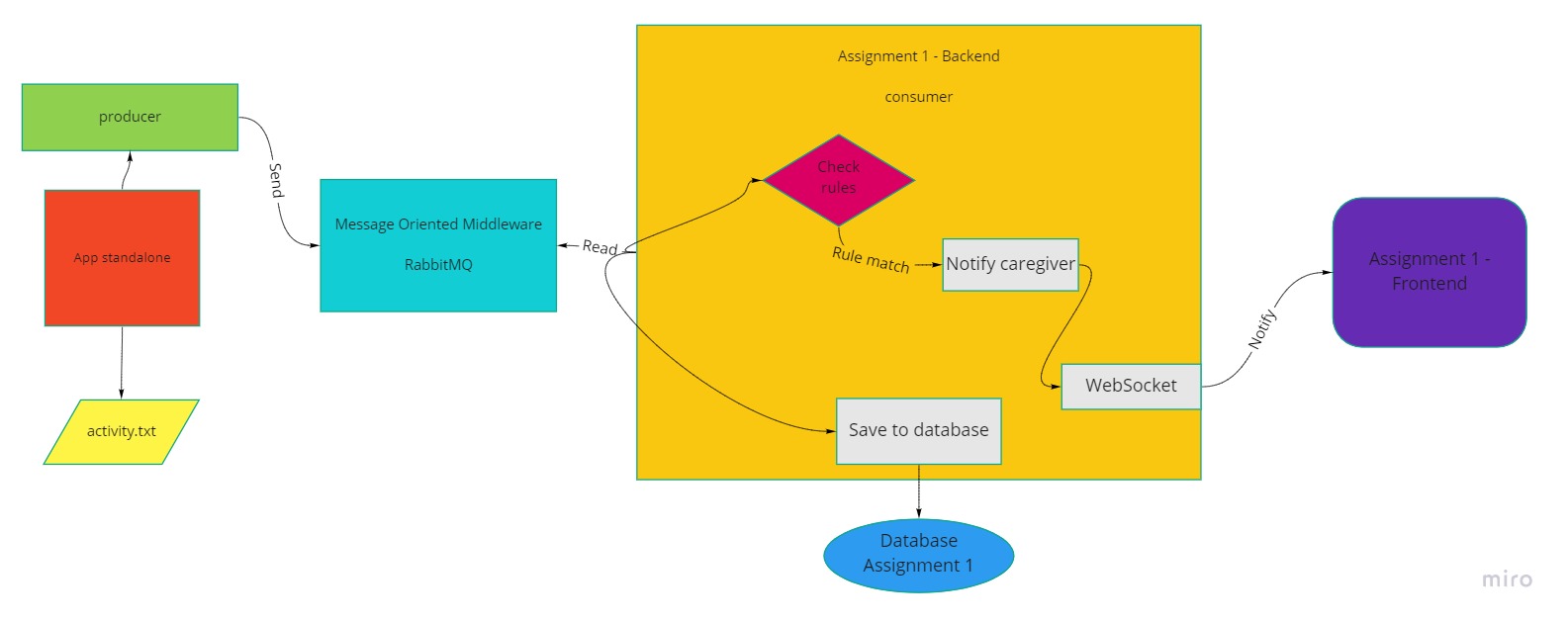
**Sensor Monitoring System and Real-Time Notification**

-Distributed systems-

**Name**: Péntek Tamás

**Group**: 30642



**Figure 1. Conceptual description of the distributed system**

This documentation presents a solution for a sensor monitoring system and for a system that sends notifications in real-time. A conceptual description of this complex system can be found on the Figure 1.

This system gathers data from the sensors, which is stored in a file with name activity.txt, preprocesses it and after that this data is stored in a database. The preprocessing is done by consumer application and if detects an anomalous activity according to a set of predefined rules, it notifies asynchronously the caregiver’s application. This system has 4 main parts: producer standalone app, message-oriented middleware, consumer backend and frontend.

The standalone app acts like a producer of the message-oriented middleware, which means that this application sends data to the middleware. This app reads data from activity.txt file, converts it to JSON format and sends these JSON messages to the queue, which is created on the middleware. The middleware is implemented using RabbitMQ, which is a message-broker software that implements the Advanced Message Queuing Protocol (AMQP) and is written in Erlang programming language. On this middleware is declared a queue which contains the message sent by producer application.

The consumer backend was implemented for Assignment 1, now this application is extended to support the consumer role of the message-oriented middleware. This application reads the JSON message from the queue, converts it to object and saves to the database. Another role of this application is to check that the incoming JSON data breaks the predefined rule. If breaks the rule, then an asynchronous notification is sent to the frontend using WebSocket. WebSocket is a computer communication protocol that provides full-duplex communication channels over a single TCP connection.

To send a notification we use STOMP (Streaming Text Oriented Messaging Protocol) which is a subprotocol operating on the top of the WebSocket protocol. After a caregiver logs in, the connection between frontend and backend is created using WebSocket.

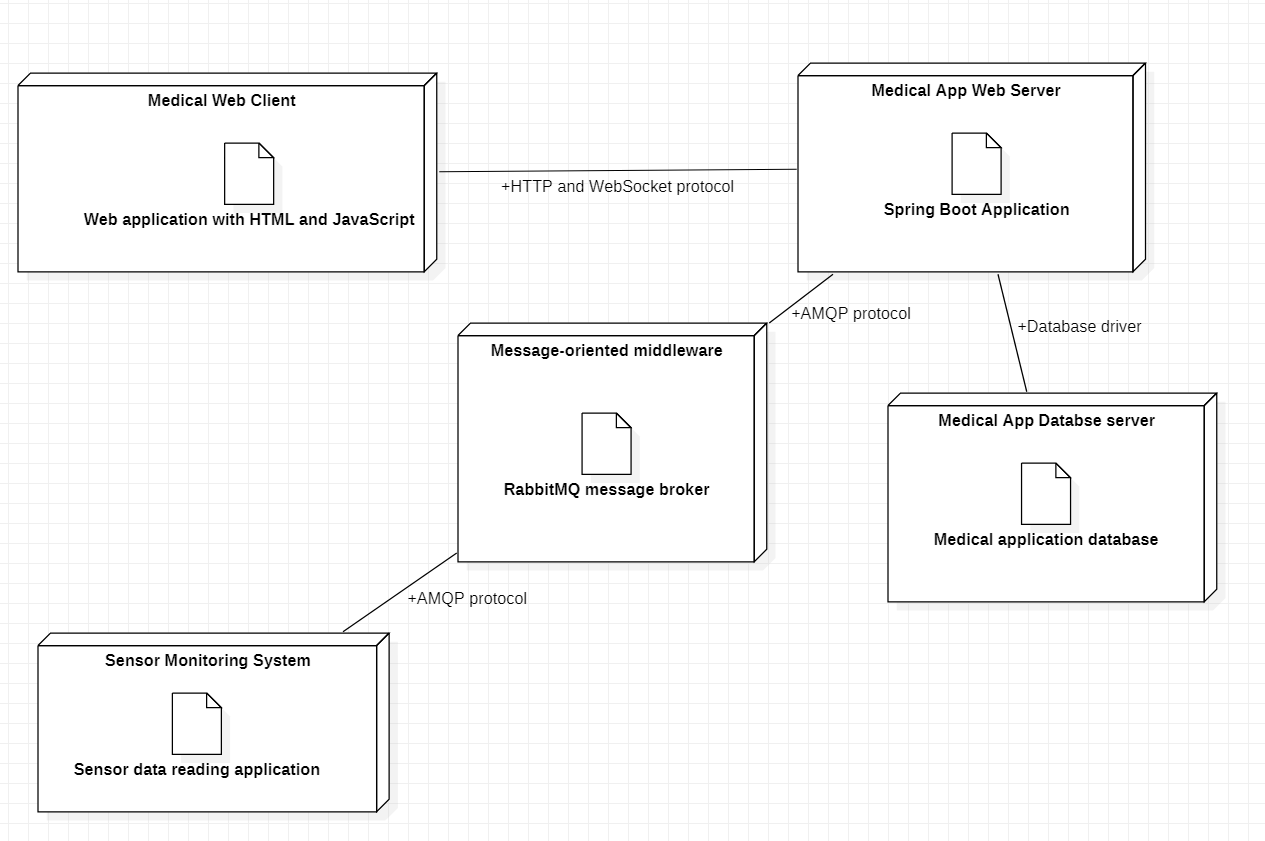
The standalone application sends in every second a JSON data to the queue, the backend application reads it and if detects an anomalous activity according to the set of the predefined rules, sends a notification to caregiver’s page. On this page appears a notification on the right up corner with a message that contains the name of the patient and the reason of the notification.

On the next diagram is presented the Deployment diagram of the distributed system. The first part of this diagram is the relational database server, on which are stored the activities read from activity.txt and details about patients and caregivers. In this case the database server is Postgres, which is an open source database server and it has an add-on for Heroku Cloud, so it is useful when the application is planned to be deployed on cloud. The second part is represented by the web server of the system, which is implemented using Spring Boot. The communication between web server application and database is done by database driver, which offers an API to simplify the communication between these two parts.

The web server communicates with fronted via HTTP and WebSocket protocol. HTTP protocol is used to load data on frontend and to save data from frontend to backend. These operations are done by GET and POST methods. The WebSocket is used to notify the caregiver’s page if something is wrong with his/her patient. The web server application gets the activity data of the patient from the message-oriented middleware using AMQP protocol.

The RabbitMQ message broker represents a middleware between web server application and sensor data reading application. On both sides uses AMQP protocol to read data from producer application and to send data to consumer application. RabbitMQ defines a message queue on which is stored the data sent by the sensor data reading application. The sensor monitoring system reads sensor data from acivity.txt file, where is stored the data from the activity tracking sensors.

If an anomalous activity is recorded by the sensors, a notification arrives to the web client, which is implemented using React. On the web client a caregiver is logged in and if the sensor monitoring system sends data to the web server and if appears an anomalous activity, on the page of the caregiver a notification will appear.



**Figure 2. UML Deployment diagram**

The standalone sensor monitoring application is written in Java and is built using maven and the RabbitMQ message broker is written in Erlang. The web server and web client are the same applications that were presented for Assignment 1. This distributed system is deployed on Heroku cloud using Docker. To run, to test the application is much better using cloud than localhost, because in this way the application can be accessed from everywhere, without location restrictions.

To access the page of a caregiver you have to log in as doctor and to create some patients and a caregiver. There is no possibility to create a doctor via user interface, so it can be done by inserting direct in the database or by using a tool like Postman, where we can make a POST method with a JSON, which contains details about doctor account (for example HOST\_URL/doctor/insert with a body: {"id":"45774962-e6f7-41f6-b940-72ef63fa1943","account":{"userName":"dr","password": "1234", "accountType":"DOCTOR"}}). After a doctor is created, we simply log in as doctor and create some patients and a caregiver, to which we add the created patients. Then we have to log out from doctor’s page and to log in to the caregiver’s page. If the standalone sensor monitoring application is started and if something is wrong with the patient’s activity, a notification will appear on the caregiver’s page. The activity.txt file contains activities of the same patient and the patient ID can be changed in the standalone application and in this way, we can assign the activity.txt file with any patient.