TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

SOLUTION DESCRIPTION

Energy Management System

Assignment 2

Distributed Systems 2023

Student: **Radu-Augustin Vele**

Group: **30442**

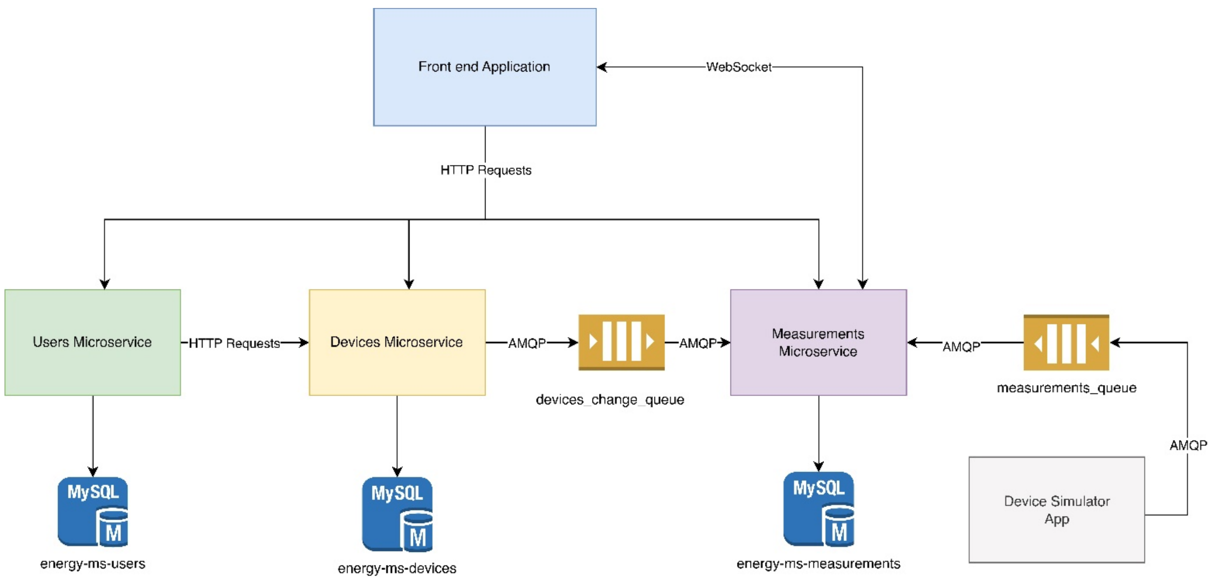
Table of Contents

1. Project objectives

In this phase of the system, the goal is to provide additional functionality on top of the existing capabilities. As explained previously, the energy management system needs to bring an interface between the user and all its devices with the focus on the devices energy consumption. This brings the need for a way to gather data from many devices, aggregate it, and present it to the user. Moreover, this data needs to be persistent so that the user can query it for custom periods of time.

To cover this need, the system will be enriched with another microservice, named "Measurements Microservice" with the responsibility of consuming the data sent wirelessly by the devices, processing it, and saving it into a database. It will also provide access to the persistent data. This microservice will be connected to the rest of the services and will be accessible from the front-end application. Moreover, the users, devices, and measurements data will be consistent across the system.

2. Conceptual architecture



Besides the existing components, the added ones are:

- Measurements microservice – built using Spring Boot
- Device Simulator Application – built using Flutter.

According to the system requirements, the messages produced by the device simulator are published to a queue (using protocol AMQP) and are consumed by the Measurements microservice (which listens to the queue).

Asynchronous communication is also employed for ensuring data consistency between the devices and monitoring microservice. This is done through the devices change queue. Every time a change is reported to a device (creation, deletion, editing, changing of ownership), the new device details are sent through the queue and consumed by the monitoring microservice, which updates its database.

2.1. Components description

Measurements Microservice

The microservice is organized in a layered architecture and offers functionality related to devices and measurements.

The core functionality is the one related to the *measurements*. The acquisition of new measurements is done using the Measurement listener class that is configured to listen to the measurements queue (using the Spring **AMQP** Rabbit Listener annotation). When a message is received, an object of the expected format is created

(Measurement Message DTO) and the method for processing new measurements (in the Measurement Service) is invoked. This processing involves creating a new measurement object and saving it to the database. Moreover, the past hour consumption is computed based on the database entries of the device and added as information in the database row corresponding to the new measurement.

Note: the past hour consumption includes the sum of all measured value in the past hour (including the latest measurement that arrived).

If the last hour consumption exceeds the maximum value for that device, a notification message is sent to the user who owns the device (if the user is logged in in that moment). This is done using **Web Sockets**, that are included into the spring boot application with the aid of spring web dependency. The initial configuration includes creating a web socket handler instance (class created in the project) and registering it in the Web Socket Registry at the endpoint "/websocket" and allowing all origins (for the current implementation).

For notifying the logged in clients we need to keep track of all the sessions opened at a time. Therefore, the handler overrides the "afterConnectionEstablished" method of the Text Web Socket Handler and every time a new connection is open a new entry is added to a concurrent hash map containing the user id (given as parameter when the socket request is issued by the client) and the corresponding Web Socket Session object. This data structure was chosen as it provides thread safety for put operations, that is useful for the case multiple clients attempt to open a web socket at the same time.

Therefore, keeping track of all sessions by user id allows the Measurement service to send the notification to the owner of the device that exceeded its maximum hourly consumption.

The measurements side also exposes a **REST endpoint** allowing the caller to retrieve the sum of measured energy for all the devices belonging to a user for each hour in a chosen day. The caller needs to specify the timestamp (milliseconds since the 1st of January 1970) corresponding to the beginning of the day (e.g., 1st of December, 00:00).

The functionality related to *devices* has been added to ensure consistency across the system. For this, all changes (create, edit, delete) related to devices (that are processed and persisted in the devices microservice) are propagated to the measurements microservice which also persists a database containing the id, owner, and maximum hourly consumption of the devices. The propagation is done using a queue (once again Spring AMQP supports the implementation). The device change listener retrieves the message corresponding to a device change and mirrors the received fields in the devices database table.

Note: when a device is deleted, the measurements corresponding to it are kept in the database. In a further enhancement of the system, we can consider allowing the users to delete the history of their measurements.

Devices Microservice

The addition of the new measurements microservice included the need to update the devices microservice. The change involves connecting to a queue (through AMQP) and sending the modified device data. This operation is performed when a device is added, removed or the fields maximum energy consumption or owner id are updated.

Device Simulator App

This is a desktop app meant to simulate the workings of a device. It provides a basic mechanism to send some measurement data to a queue without being aware of the location of the consumer of that data.

The user is supposed to input the device name and optionally a line offset in the csv file containing values to be sent as measurements. After that, if the inputs are valid, the user can start the simulation and all the sent messages are output on the screen.

From the design point of view, the simulator is built using three components.

- The main component (defining the widget tree, specific to flutter)
- The csv reader
- The amqp writer

When the simulation is turned on, a function starts being executed periodically (according to the specified number of seconds in the function call). On each execution a line is read from the csv file containing a double, then a JSON containing the device id, the double value as measurement, and the timestamp is created and sent to the queue. The simulation stop event halts the periodic execution.

Front end application

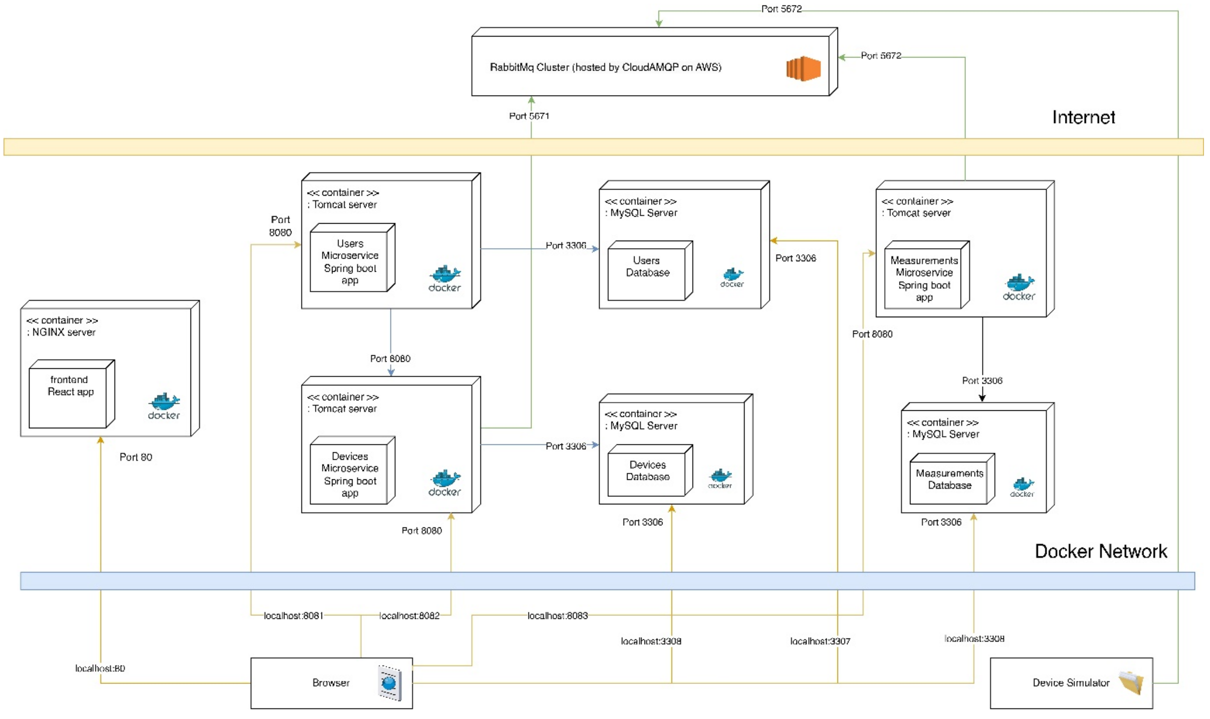
The front-end application needs to provide a way to access the newly added functionality. For that, the user home page was added two new main components: the notifications section, and the measurements chart section.

For the notifications, a custom hook ("useWebSocket") allows the notifications component to open a web socket (or access an already opened socket) and send or receive messages. Whenever a new message is recorded it changes the "lastMessage" variable and the data is appended to an array of notifications visible to the user. The notifications offer details about the device whose measured energy consumption exceeds the threshold and the values recorded. Every time the page containing the notification box refreshes or closes, the web socket is closed and then reopened on mount.

For the consumption chart, the user can pick a day from a calendar and the hourly consumption is displayed in a bar chart.

3. UML Deployment Diagram

The system is deployed using Docker containers, which facilitates their inclusion in a network of interconnected Docker containers. At this stage, the Docker containers are executed locally and are made accessible through specific local host ports.



The device simulator is run on a local machine. For real use case, the software in the embedded devices needs to be able to send messages over AMQP (or MQTT for a different queue configuration) and will use its own device id.

The RabbitMQ queues are hosted on a single instance deployed on AWS. The hosting services are provided in the free plan of CloudAMQP and currently the system does not exceed the free tier limits (for connections, queues, messages). Alternatively, the queues could be deployed on docker containers (especially once the system is available to an unlimited number of users).

The newly added microservice is instantiated into a separate container, the same goes for its corresponding database. In the compose file, we need to add the RabbitMQ authentication details, the address of the deployed queues, and the address, port, and database name for the energy-ms-measurements database.

4. Readme – build and execution considerations.

Spring + React Setup

For instructions related to running the microservices locally, and dockerization check out the assignment 1 notes. The prerequisites remain the same.

Make sure to fetch the code from the correct branch on GitLab.

RabbitMQ

If you want to use CloudAMQP you need to create a new account and create an instance (Lemur type ensures the free tier is chosen).

For more details follow: https://www.cloudamqp.com/docs/index.html.

Notes:

- If spring security is enabled in the project, you must use the port 5671 (that supports communication over TLS)

Flutter App

You can choose to directly run the exe (option that supports running multiple instances at the same time). The executable is not included in the repository, so to run the program and generate an executable you must have flutter installed on your machine.

Follow the installation instructions available here: https://docs.flutter.dev/get-started/install.

If you need to develop for another system instead of windows, then you can generate a new flutter program, and include the source code from the repository there.

Notes:

- It seems that the AMQP communication does not work over the eduroam network.
- The system has only been tested for desktop development on windows, some of the packages may not be supported for other platforms.
- In case the AMQP writer does not work, there is a python script included in the repository that can be called from the flutter app in order to send a message to the queue.

5. Useful Links

Spring and RabbitMQ

- https://medium.com/javarevisited/getting-started-with-rabbitmq-in-spring-boot-6323b9179247

Spring Web Sockets

- https://medium.com/@parthiban.rajalingam/introduction-to-web-sockets-using-spring-boot-and-angular-b11e7363f051

Flutter

- https://pub.dev/packages/dart_amqp
- https://pub.dev/packages/path_provider

React Web Sockets:

- https://www.npmjs.com/package/react-use-websocket

React Charts:

- https://recharts.org/en-US/examples/SimpleBarChart

Cloud AMQP (hosting RabbitMQ)

- https://www.cloudamqp.com/docs/index.html