Online Energy Utility Platform

Bărăgan Andrei Iustin

2022

Cuprins

[Introduction 2](#_Toc119487268)

[Domain 2](#_Toc119487269)

[Scope 2](#_Toc119487270)

[Theoretical Aspects 3](#_Toc119487271)

[Design 4](#_Toc119487272)

[Functional Requirements 4](#_Toc119487273)

[Non functional Requirements 4](#_Toc119487274)

[Implementation 5](#_Toc119487275)

[Conceptual Architecture of the online Platform 5](#_Toc119487276)

[Database Diagram 6](#_Toc119487277)

[The database contains 3 tables: 6](#_Toc119487278)

[ User Table where user are stored. The user contains an username a password which is encrypted and an user\_type which can be an administrator or a client. 6](#_Toc119487279)

[ Device which containts just the id to distinguish between devices and the id of the user to whom it belongs 6](#_Toc119487280)

[ Measurements Table store all mesurements made by a device at each hour 6](#_Toc119487281)

[UML Deployment diagram 7](#_Toc119487282)

[Conclusions 8](#_Toc119487283)

[Bibliografie 9](#_Toc119487284)

# Introduction

## An online platform should be designed and implemented to manage users, their associated smart

## energy metering devices, and the monitored data from each device. The system can be accessed

## by two types of users after a login process: administrator (manager), and clients. The administrator

## can perform CRUD (Create-Read-Update-Delete) operations on user accounts (defined by ID,

## name, role: admin/client), registered smart energy metering devices (defined by ID, description,

## address, maximum hourly energy consumption), and on the mapping of users to devices (each user

## can own one or more smart devices in different locations). After the mapping is done, for each

## device the energy consumption is stored on hourly basis as tuples of the form <timestamp, energy

## consumption> in the database.

Implement a component for online platform application based on a message broker middleware that gathers data from the smart metering devices, pre-processes the data to compute the hourly energy consumption and stores it in the database. A Smart Metering Device Simulator module will be the Message Producer. It will simulate a sensor by reading energy data from a file (sensor.csv - one value at every 10 minutes) and sends data in the form < timestamp, device\_id, measurement\_value > to the Message Broker (i.e., the queue). The timestamp is taken from the local clock, the measurement\_value is read from the file and represents the energy measured in kWh, and the device\_id is unique to each instance of the Smart Metering Device Simulator and corresponds to the device\_id of a user from the database (as defined in Assignment 1). The sensor simulator should be developed as a standalone application (i.e., desktop application). The file sensor.csv can be downloaded from https://dsrl.eu/courses/sd/materials/sensor.csv. The measurements are sent to the queue using the following JSON format:

{ “timestamp": 1570654800000, “device\_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2” “measurement\_value”: 0.1, }

A Message Consumer application will pre-process the data to compute the total hourly energy consumption and stores it in the database. If the computed total hourly energy consumption exceeds the smart device maximum value (as defined in Assignment 1) it notifies asynchronously the user on his/her web interface.

Develop a chat system to offer support for the clients of the energy platform if they have questions related with their energy consumption. The chat system should allow communication between the clients and the administrator of the system.

## Domain

Distributed systems are now used in a variety of environments and at different levels within business organizations, both in offices and factories. As the number of devices attached to these systems grows, and as the systems become internetworked together, management tools become increasingly essential to retain control of the complete distributed system. A distributed systems management model is presented which is suitable for implementation in the real world.

## Scope

The scope of the project is to learn how to make a CI/CD deployment using Docker and to get deepen in knowledge of using React and Spring.

# Theoretical Aspects

React is a free and open-source front-end JavaScript library[3] for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

Docker is a set of platform as a service (PaaS) product that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc.

A web application (or web app) is application software that is accessed using a web browser. Web applications are delivered on the World Wide Web to users with an active network connection.

A server is a computer program or device that provides a service to another computer program and its user, also known as the client. In a data center, the physical computer that a server program runs on is also frequently referred to as a server. That machine might be a dedicated server or it might be used for other purposes.

Message-oriented middleware (MOM) is software or hardware infrastructure supporting sending and receiving messages between distributed systems. MOM allows application modules to be distributed over heterogeneous platforms and reduces the complexity of developing applications that span multiple operating systems and network protocols. The middleware creates a distributed communications layer that insulates the application developer from the details of the various operating systems and network interfaces. APIs that extend across diverse platforms and networks are typically provided by MOM.

## Key difference betwwen Queue and Topic

Consumer options

* Queue: It is possible to have multiple receivers, but each message will be received only by one of those receivers.
* Topic: There are multiple receivers (subscribers) to receive the messages, each message copy can be sent to any number of subscribers associated with the Topic.

Message Filtering

* Queue: Queue does not essentially require any filters as the messages are being received by only one user.
* Topic: Each message that is sent over a Topic can have a collection of properties. Properties are used when a custom filter per subscription is applied.

# Design

## Functional Requirements

* Users log in. Users are redirected to the page corresponding to their role.
* Administrator/Manager Role:
  + CRUD operations on users and devices.
  + Create user-device mappings.
* User/Client Role
  + Can view on his/her page all the associated devices.
  + Can view the daily energy consumption for each of his/her associated devices as line charts or bar charts per day (OX- hours; OY- energy value [kWh] for that hour). The day should be selected from a calendar.
* The users corresponding to one role will not be able to enter the pages corresponding to the other role (e.g., by log-in and then copy-paste the admin URL to the browser).
* The message broker allows Smart Metering Device Simulator to act as messages producer and send data tuples in a JSON format
* The message consumer component of the system processes each message and notifies asynchronously using WebSockets the client application.
* The client application displays a chat box where clients can type messages.
* The message is sent asynchronously to the administrator, that receives the message together with the client identifier, being able to start a chat with the client.
* Messages can be sent back and forth between the client and the administrator during chat session.
* The administrator can chat with multiple clients at once.
* A notification is displayed for the user when the other user reads the message.
* A notification is displayed for the user (e.g., typing) while the user from the other end of communication types its message

## Non functional Requirements

Security: use authentication to restrict users to access the administrator pages (cookies, session, etc.)

# Implementation

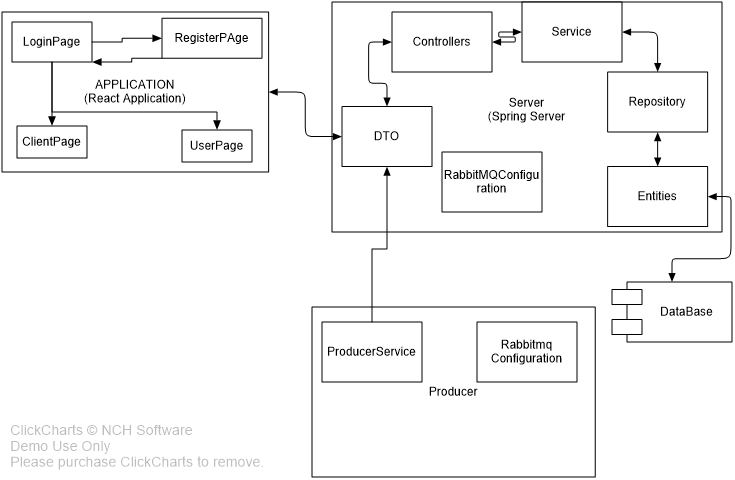
The solution is divided in 3 parts.

1. Backend where is implemented the server part which is responsible of communication with database, data encryption and security system. The server is implemented by using Spring Framework in IntelliJ IDE. Server is also responsible for all http request and manage websocket streaming.
2. Frontend where is implemented the application which is responsible of data display and which allows the user to manage the data easily. The application is implemented by using React technology and VISUAL STUDIO CODE IDE.
3. Producer where is implemented the simulator of devices which reads data from csv file and transmits them through RabbitMQ. The producer is implemented by using Spring Framework in IntelliJ IDE.

## Conceptual Architecture of the online Platform

The architecture is also divided in 3:

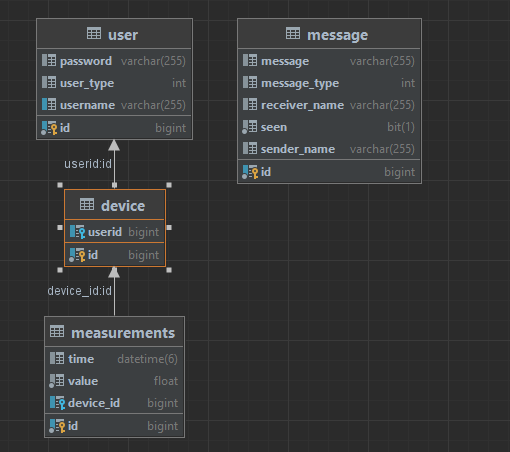
1. In Front-end graph you can see each visual Module and how you can navigate between them. For example from Login Page you can navigate to Client Page, to Administrator Page and to Register but from Register you can navigate only to Login Page. Also Client Page contains a ClientNavBar module a Device Screen and a Consumption Screen.
2. In Back-end you can see which components communicate with each. Controllers communicate with Services, services with Repositories and Repositories with Database.
3. In producer we have just RabbitmqConfig which makes the configuration for rabbitmq and Producer Service which transmit data.



Database Diagram

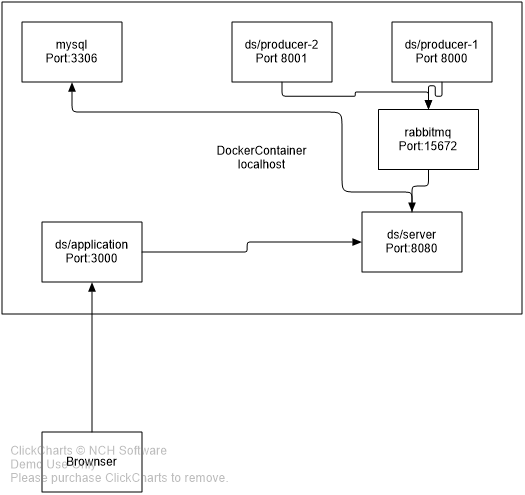
The database contains 4 tables:

* User Table where user are stored. The user contains an username a password which is encrypted and an user\_type which can be an administrator or a client.
* Device which contains just the id to distinguish between devices and the id of the user to whom it belongs
* Measurements Table store all mesurements made by a device at each hour
* Message which contains the text message, type of message (MESSAGE,SEEN, or TYPING), receiver name, if message was seen by receiver and sender name



## UML Deployment diagram

Deployment diagrams are used to visualize the hardware processors/ nodes/ devices of a system, the links of communication between them and the placement of software files on that hardware.



# Conclusions

In conclusion the project succeeds to package a server and an web application and their dependencies into a virtual container that can run on any Linux, Windows, or macOS computer

# Bibliografie

[**https://en.wikipedia.org/wiki/React\_(JavaScript\_library)**](https://en.wikipedia.org/wiki/React_(JavaScript_library))

[**https://en.wikipedia.org/wiki/Spring\_Framework**](https://en.wikipedia.org/wiki/Spring_Framework)

[**https://en.wikipedia.org/wiki/Docker\_(software)**](https://en.wikipedia.org/wiki/Docker_(software))

END OF DOCUMENT