Distributed Systems

Project Documentation

*Student: Francesca Dițulescu*

**Requirements for the laboratory**

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.

**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 for building user interfaces based on UI components. It is maintained by Meta 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 platforms 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.

**RabbitMQ** is an open-source [message-broker](https://en.wikipedia.org/wiki/Message_broker) software (sometimes called [message-oriented middleware](https://en.wikipedia.org/wiki/Message-oriented_middleware)) that originally implemented the [Advanced Message Queuing Protocol](https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol) (AMQP) and has since been extended with a [plug-in architecture](https://en.wikipedia.org/wiki/Plug-in_(computing)) to support [Streaming Text Oriented Messaging Protocol](https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol) (STOMP), [MQ Telemetry Transport](https://en.wikipedia.org/wiki/MQ_Telemetry_Transport) (MQTT), and other protocols.

Written in [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language)), the RabbitMQ server is built on the [Open Telecom Platform](https://en.wikipedia.org/wiki/Open_Telecom_Platform) framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages. The source code is released under the [Mozilla Public License](https://en.wikipedia.org/wiki/Mozilla_Public_License).

**gRPC** (gRPC Remote Procedure Calls[[2]](https://en.wikipedia.org/wiki/GRPC#cite_note-2)) is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) open source high performance [Remote Procedure Call (RPC)](https://en.wikipedia.org/wiki/Remote_procedure_call) framework. gRPC was initially created by Google, which has used a single general-purpose RPC infrastructure called Stubby to connect the large number of microservices running within and across its data centers for over a decade[[when?](https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Dates_and_numbers#Chronological_items)]. In March 2015, Google decided to build the next version of Stubby and make it open source. The result was gRPC, which is now used in many organizations aside from Google to power use cases from microservices to the “last mile” of computing (mobile, web, and Internet of Things). It uses [HTTP/2](https://en.wikipedia.org/wiki/HTTP/2) for transport, [Protocol Buffers](https://en.wikipedia.org/wiki/Protocol_Buffers) as the [interface description language](https://en.wikipedia.org/wiki/Interface_description_language), and provides features such as authentication, bidirectional streaming and [flow control](https://en.wikipedia.org/wiki/Flow_control_(data)), blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. Most common usage scenarios include connecting services in a [microservices](https://en.wikipedia.org/wiki/Microservices) style architecture, or connecting mobile device clients to backend services.

**Design**

**Functional Requirements**

* 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
* 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, who 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

**Nonfunctional Requirements**

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

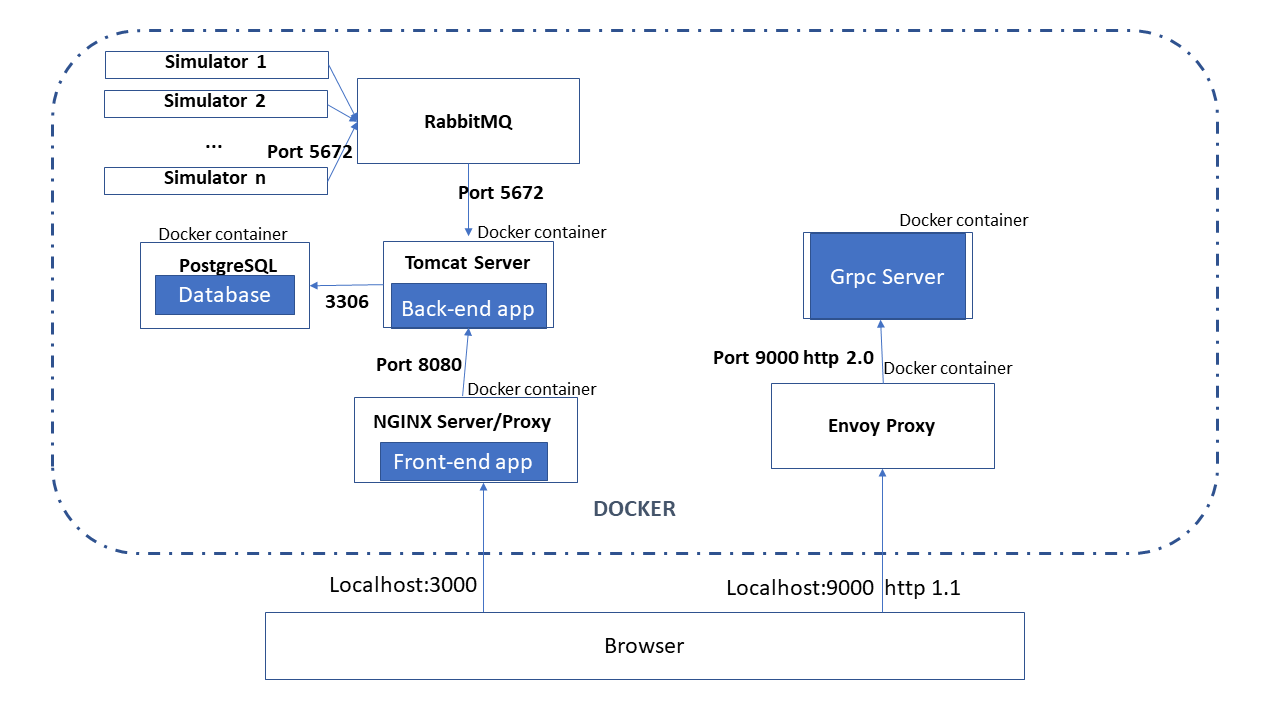
**DIAGRAMAS**

1. DATABASE DIAGRAM

O imagine care conține masă

Descriere generată automat

1. UML DEPLOYMENT DIAGRAM



1. CONCEPTUAL ARCHITECTURE

