DISTRIBUTED SYSTEMS

Assignment 3

**Remote Procedure Call**

Sensor Monitoring System and Real-Time Notification

Student: Ștefan-Andrei CHICHIȘAN

Group: 30441

Course teacher: Cristina POP

Laboratory teacher: Oana MARIN

Table of contents:

1. Proposed task
2. Conceptual architecture of the distributed system
3. UML deployment diagram
4. Build and execution considerations
5. **Proposed task**

The second assignment of the distributed systems course is to design and implement a component for Assignment 1 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). 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.

1. **Conceptual architecture of the online platform**

Backend

The backend is written in Java programming language using Spring Boot framework. The architecture used in the development is a 3-tier REST service (Controller – Service -Repository). The architecture is well suited for web applications in general having the components well spread and organized.

All the packages implement entities, DTOs, mappers, services, controllers and repositories for only one main component. For example, the package *device* has a repository interface that implements the repository for users, a controller class that has all the request methods, a service class that connects the requests to the repository. It also has DTOs and mapper (transforms the DTO to model/entity and vice-versa). There is a thumb rule that models/entities shouldn’t be sent or received through the controller, that’s what DTOs (Data Transfer Object) are for, to send the data between the frontend and backend using JSON type information.

Diagram

Description automatically generated

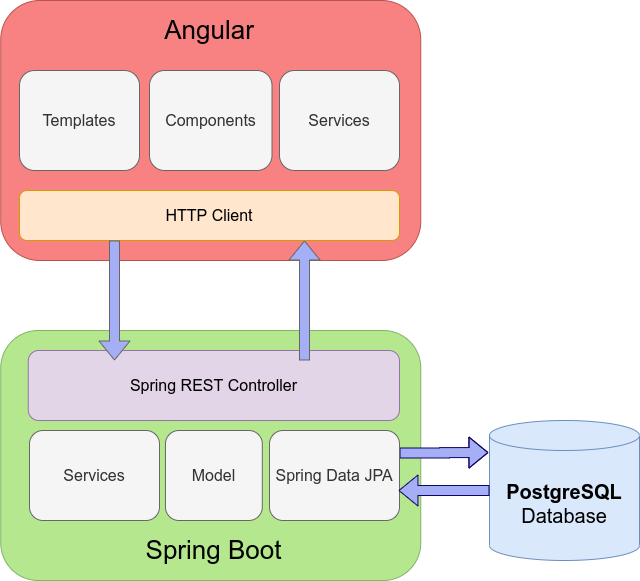
Frontend

The front part of the application was designed and implemented using Angular framework, TypeScript, HTML and CSS (SCSS). The application can be only accessed by the administrator, the clients being redirect to a page error.

There are a few main pages and dialog boxes displayed. For the administrator privileges: users page, devices page and measurement page and for each of them dialog boxes for creating/editing/deleting. For the client page, as I said above, he will be redirected to an error page.

Diagram

Description automatically generated



*Application architecture schema*

gRPC, gRPC-Web and EnvoyProxy

gRPC is a high-performance, open-source framework for building remote procedure call (RPC) APIs. It uses the Protocol Buffers data serialization format and supports a variety of programming languages. gRPC allows for bi-directional streaming and flow control, making it well-suited for real-time applications.

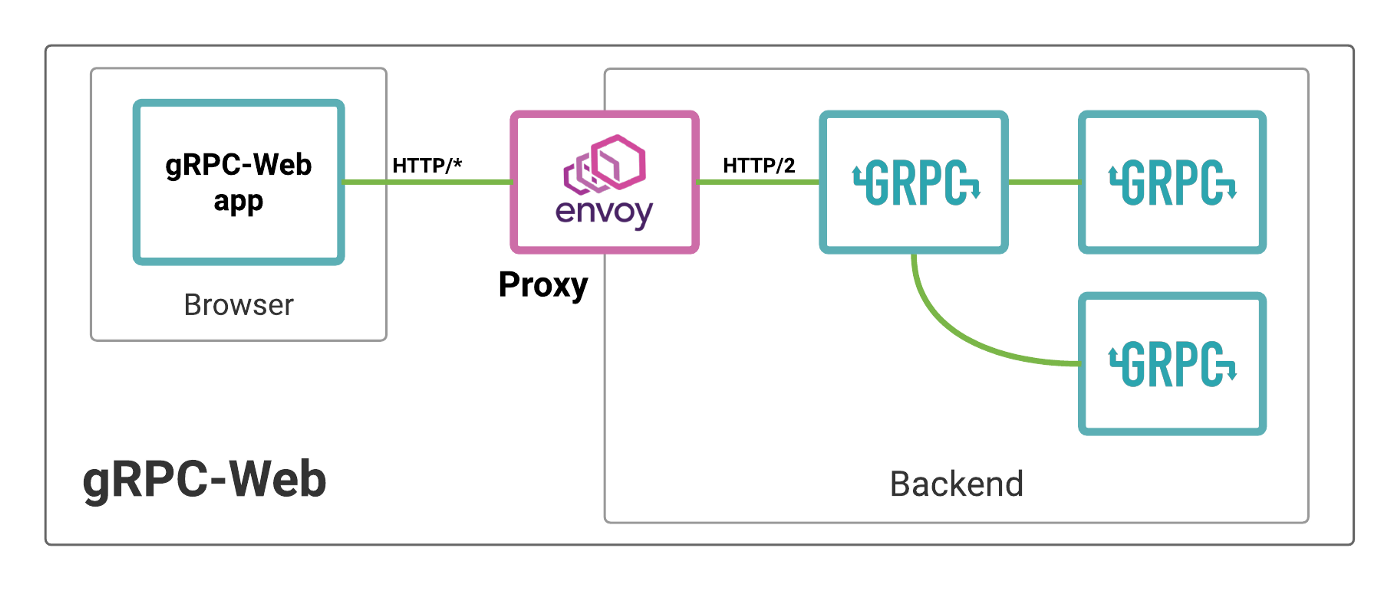
gRPC-Web is a JavaScript client library that enables web applications to communicate with gRPC services. It allows for using gRPC APIs in web browsers and other JavaScript environments without the need for a gRPC-specific proxy. gRPC-Web supports unary and server streaming requests, but not yet bi-directional streaming.

EnvoyProxy is a high-performance, open-source proxy for microservices. It can be used as a gRPC proxy, providing features such as load balancing, service discovery, and security for gRPC services. EnvoyProxy can also be used to handle gRPC-Web traffic and translate it to gRPC calls to the backend service.

Diagram

Description automatically generated

*Example of communication of gRPC Server-Client*



*Communication of gRPC and EnvoyProxy*

1. **UML deployment diagram**

Graphical user interface

Description automatically generated

1. **Build and execution considerations**

Localhost:

* Create database in PostgreSQL using Hibernate
* Set database credentials in the *application.properties*
* Create the host variable in Angular project with the API of the backend
* Start running RabbitMQ Docker container by running this command
  + *docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.11-management*
* Run the backend on the port configured for the apache-tomcat server
* Start frontend by using *ng serve* which uses webpack dev server
* Run the Java desktop application and enter the device id already existing in the database
* Check the endpoints with the Postman API
  + Here is my own collection : <https://www.getpostman.com/collections/a14a5fcf4238c4fccd57>

Docker:

* Create Dockerfile for backend and frontend
* Create docker-compose.yml for backend
* Create envoy.yaml for EnvoyProxy
* Create envoy.Dockerfile for EnvoyProxy and run it
* (Not necessary) Create docker-compose.yml for frontend

Alternative, run command *docker run [...]*

* Create the database for the docker container *psql-dev*
* Run the docker-compose.yml for backend to start the server
* Start docker container for frontend
* Run the Java desktop application and enter the device id already existing in the database