DISTRIBUTED SYSTEMS

Assignment 3

Remote Procedure Call

(RPC)

Smart Home Appliance

Andrei Dorofte

Group 30641

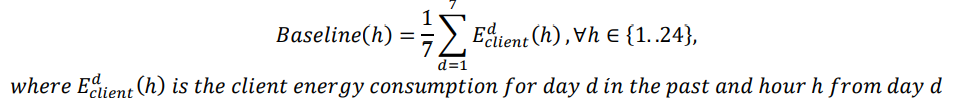
1. **Conceptual architecture of the distributed system**

**RPC**

In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remotely.

For the current assignment, the RPC is used in order to compute

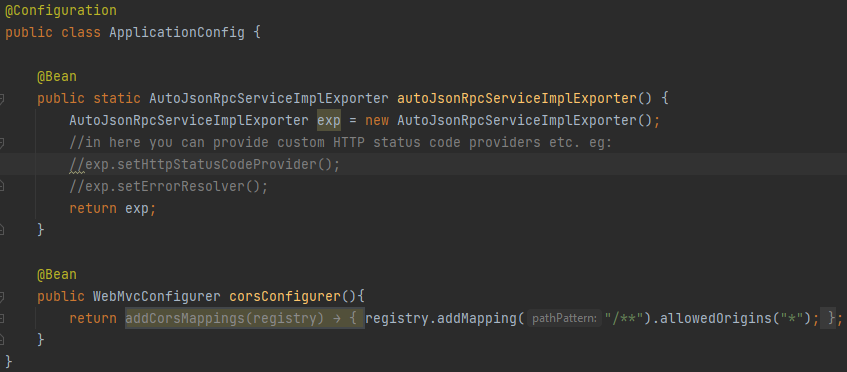
* the client’s hourly historical energy consumption over d days in the past, where d is entered by the user. In this way, the data is returned to the frontend and displayed in a chart, as an evolution line.
* gets the averaged energy consumption for the client over the past week (e.g. client baseline) and returns the data to the frontend, in order to be displayed in a chart, as an evolution line.



* allows the selection of a program with a duration in hours (select a duration D of a program) and returns the value to be displayed by the frontend

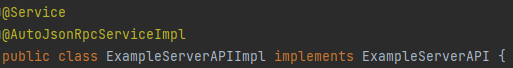
**Backend**

I implemented the RPC handler using JSON-RPC dependency for Java, in order to compute the necessary data for this assignment. This meant that I had to add the dependency to pom.xml and create a configuration class for it:

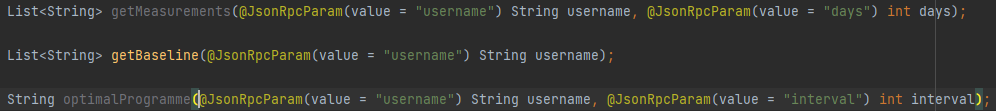


This class contains 2 Beans, one for configuring the dependency to auto detect the interfaces that are annotated with @JsonRpcService and their implementations, and a cors configurer, although I added the endpoint for RPC in the security configuration of Spring Security and added „.permitAll()”:





Then, I proceeded with creating the methods that would handle the calls for the specific requirements:



The getMeasurements method handles the first requirement, and gets the measurements for all the client’s devices.

If there is more than one measurement per hour, then the algorithm takes only the last value of every hour, as the measurements of every hour are put in a HashMap<Date, List<Double>>, then sends it to the frontend application as a json array.

The getBaseline method initialises a HashMap<Integer, List<Double>>, as a template for every hour of the day, and the measurements from every device of the client for that hour. In the end, all the measurements of the list are summed and divided by 7, in order to get the arithmetic mean, then sends it to the frontend application as a json array.

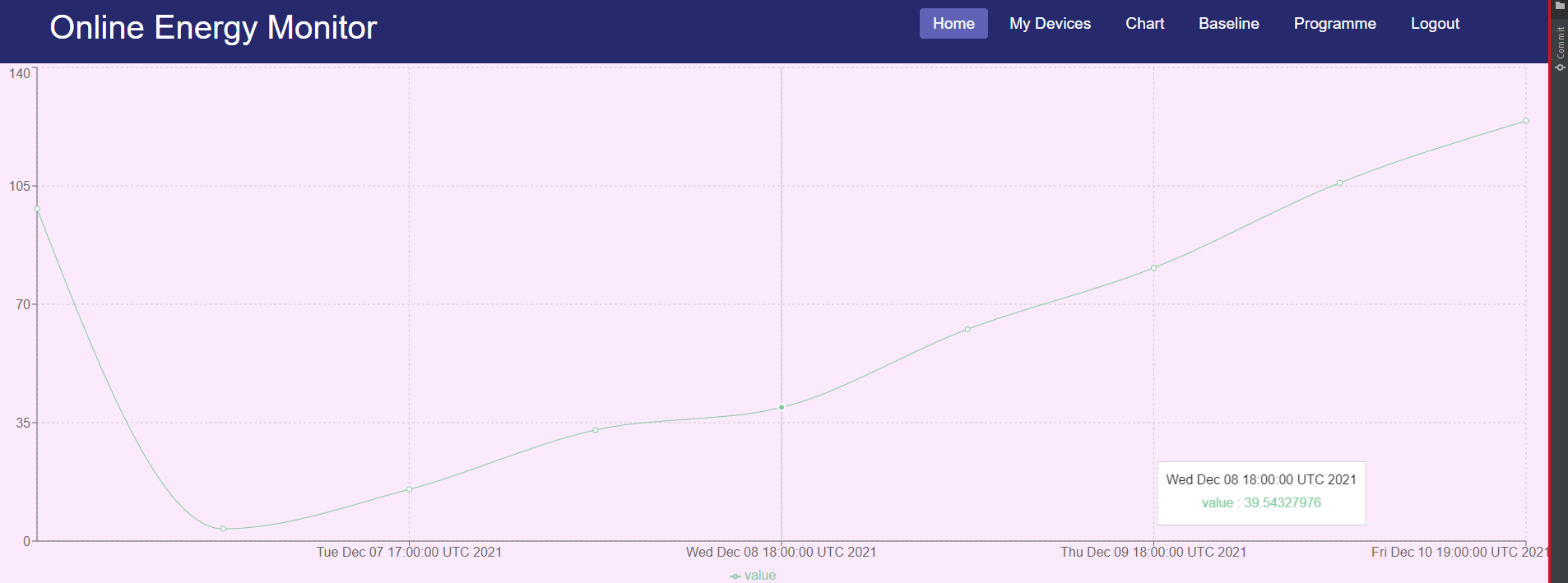
The optimal programme method gets the baseline result from the previous method and loops through the hours of the day in order to find an interval with the lowest consumption, then sends it to the frontend application as a string.

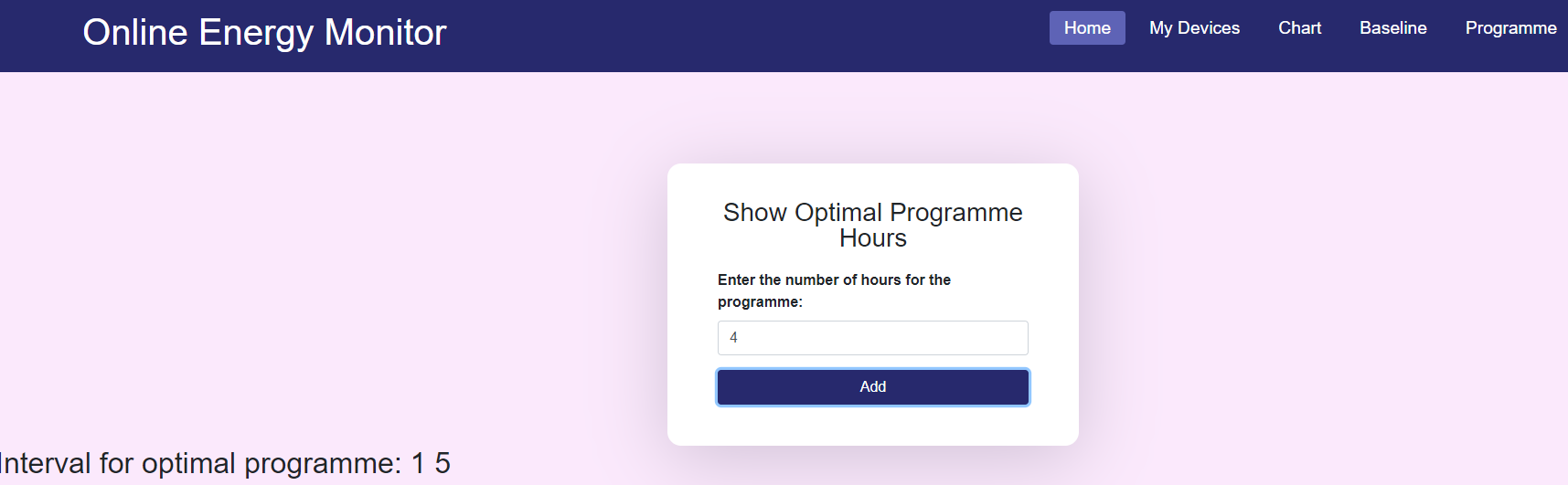
**Frontend**

On the frontend application I created 3 separate pages: Chart, Baseline, Programme. Each page calls the backend through RPC, using axios. I defined custom headers for each call:

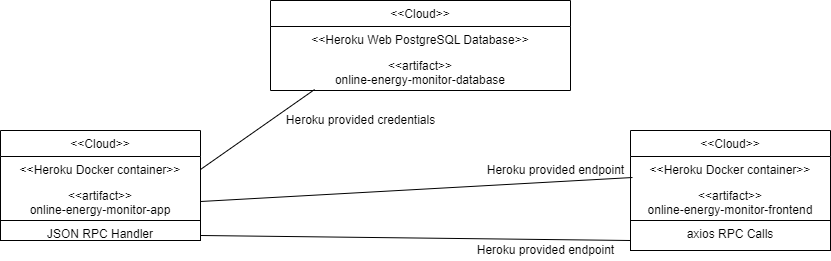
Text

Description automatically generated





1. **UML Deployment diagram**

****

Here is the new deployment diagram. The user accesses the link provided by Heroku, from the deployment of the frontend application. The deployed frontend app then calls the endpoints of the deployed backend application, over the Heroku provided endpoint. Then, the Heroku database of the backend container is used to store all the objects used. On the newly created pages, axios makes the calls to the backend, using the custom headers defined by me, and displays the data in charts, or as a string.

The deployment is donw using Heroku’s docker containers, in which the deployment was done using the GitLab Pipeline workers. I had to create a Dockerfile for both the frontend and the backend, and also add some files needed for configuring the Nginx server (nginx.conf), along with some variables in each GitLab repository.

The backend application still uses the Heroku PostgreSQL database, whose credentials are stored in the backend’s Dockerfile.