Technical Report
Project specifications

GeaniHouse

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 19 de Janeiro de 2020
Students:	93015: Daniel Gomes 93430: Mário Silva 93446: Leandro Silva 93302: Bruno Bastos
Project abstract:	GeaniHouse is an application that allows controlling and configuring Smart Homes.

Index

1. Introduction

This document intents to describe all the work done In the context of the assignment for the Introduction to Software Engineering Course, whose main objectives consisted in :

- Developing a product specification, from the usage scenarios to the technical design
- Proposing, justifying, and implementing a software architecture, based on enterprise frameworks
- Applying collaborative work practices, both in code development and agile project management.

2. Product concept

Vision statement

In the last few years, the popularity of Smart Homes has been increasing due to the multifaceted ways it can improve our houses and our lives. As it is a topic that interests all of us, we looked forward to making this the theme of our assignment. Therefore, we decided to develop **GeaniHouse**: an application that allows users to configure devices of their smart home divisions in a simple way, monitor their houses, and analyse data from sensors like temperature and humidity.

Personas

João

João is a young man with a lot of interest in new technologies. During the last few years, he started wanting to have a smart house where he could program and configure its devices according to his liking. As he works remotely he has a pretty flexible schedule and spends a great amount of time inside his house.

António

António is a middle aged man that works during the night and needs to automate some electronic processes for his family during the day.

Célia

Célia is an elderly lady that wants to take the advantages that recent technologies have to offer. Since she has a lot of difficulty and really little external help, it would be great if some processes in her house were automated to help her day-to-day life.

User Stories

- Any user who wants to use the system must register first and then log in to authenticate.
- Any registered user wants to add devices to be able to control them through the app.
- Any registered user wants to add homes and divisions to be able to control them using the app.
- As António, I want to set up the water cylinder to warm the water at any time he wants for my family.
- As António, I want to give permission to my 12 years old daughter so that she can configure her room devices.
- As António, I want to share access to the smart home app with my wife so that if she needs anything she can also check it out.
- As João, I want to be able to turn on the air conditioner automatically around 18:00, during half an hour, in order to heat the division.
- As João, I want to maintain a temperature of 18°C-24°C throughout the day, to be more comfortable.
- João wants to always be aware of how the application is performing on the paired devices, always receiving notifications of changes in their state.
- Célia, as she doesn't know much about technology, wants the whole process of learning the application to be easy, so she needs an option that makes the settings optimized for her needs automatically.

3. Architecture notebook

Key requirements and constraints

There are some key requirements and system constraints that have a significant bearing on the architecture:

All private information of the users must be protected, by encrypting the data stored in the database. The access to the platform must be done using a login system (user identification and password control).

The database system must ensure that all sensor data is stored, all configurations that any user made are saved, and must be always available to perform the operations that our application requires.

The system must be always available to the users of it, 24/7, to answer their needs.

The sensors installation and the connection to the remote devices are not supported by the application.

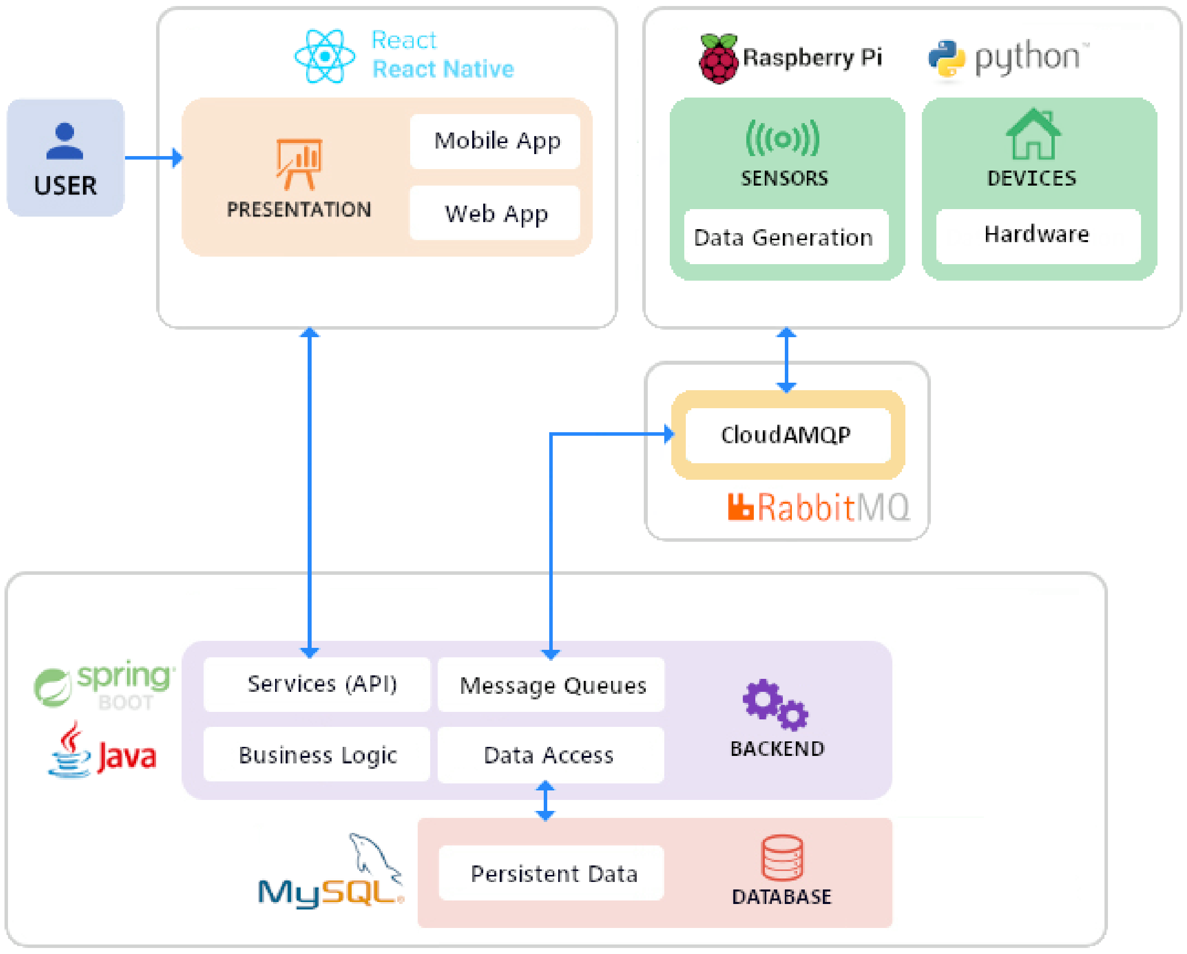
According to the type of sensor and its difficulty of handling and implementation, and since none of us has great experience with Raspberry Pis, the data may or may not be simulated.

Architectural view

→ Discuss architecture planned for the software solution.

→ include a diagram

In the following image, it is shown the Architectural View for the software solution, in which we will explain with more detail, every part of it.



Data generation/sensing

We are using a temperature sensor, and for other things like humidity and light, all data will be simulated. To run the scripts to simulate data,send the data streams to our

Backend, and for the use of sensors, it is being used a Raspberry Pi. Besides this, python is being used to communicate with the server.

Frontend

For the mobile app, in the beginning, we were thinking of using React native since it can be used for cross-platform development for both Android and IOS but since we didn't have enough time, we decided to use Ionic which builds a progressive web app with the code that we wrote for the webapp. The web app uses React since it's one of the most used frontend frameworks and we wanted to know it better by practicing.

Backend

Database:

The database management system in use is MySQL. In the beginning we were thinking about a NoSQL database like MongoDB but since we don't have much experience with this technology we opt for a relational database system. MySQL might have some scalability problems, however, for our project it is more than enough.

Data Access:

To access the data we decided to use the JPA, Java Persistence API, since we got to know it from the Practical Classes and really useful to do SQL Queries.

Service layer (API) + Processing & Bizz Logic

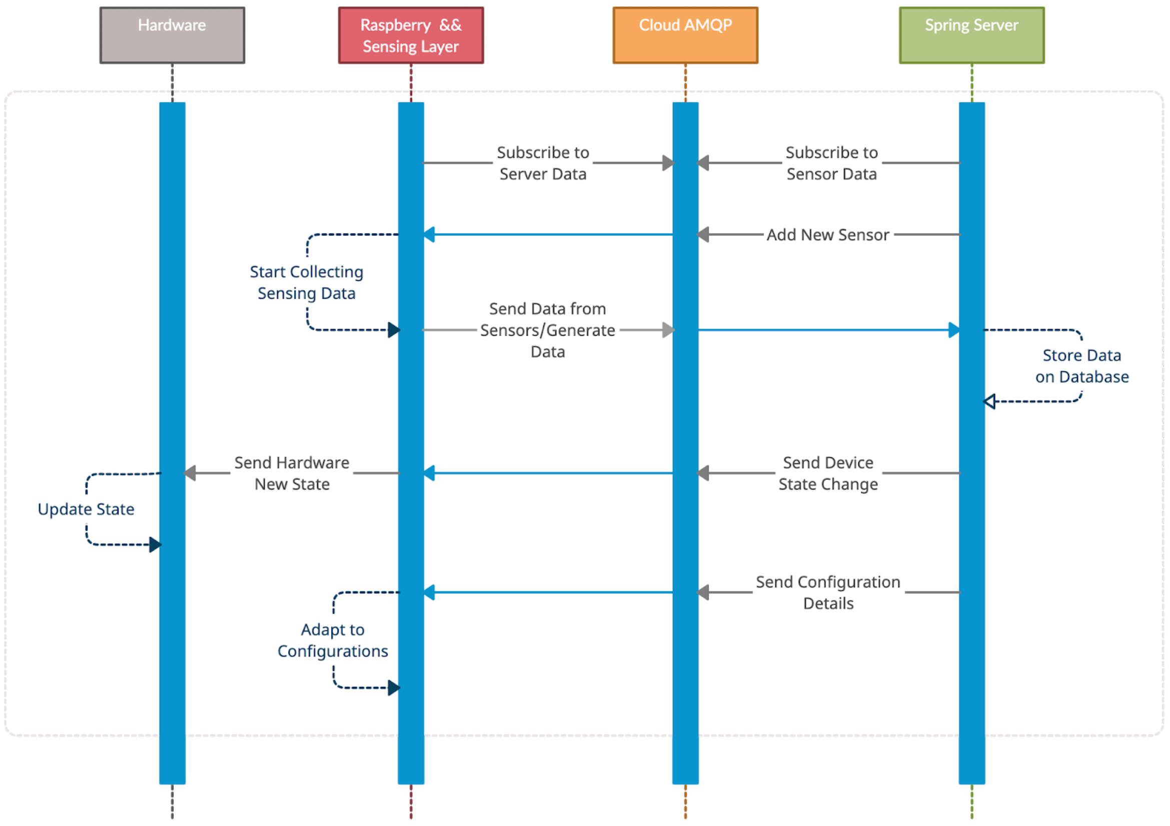
For the backend of our System, since we got to know the Spring framework from the practical classes of this course and it was highly recommended by the Teacher, we will use Spring.

Message Queues:

RabbitMQ is going to receive the messages from the sensors and pass them to the backend. Our choice comes from the fact that RabbitMQ is one of the most used message brokers which makes it easy to find information about it and is easy to integrate in spring-boot. In order to make testing and integration on the project much easier, we decided to use the Cloud Version of the RabbitMQ (CloudAMQP).

Module interactions

Our Frontend, the part of the system used by the Client, starts by communicating with our Backend, by using the REST API services offered by it, SpringBoot. The SpringBoot accesses the Persistence Storage, MySQL, in order to retrieve the data needed to answer the Request.



In the previous sequence diagram, it is shown some of the interactions in our system.When needed, SpringBoot communicates with the Cloud Message Broker (CloudAMQP) publishing a message to it. The Cybersensing layer, as it is subscribed to the topic that SpringBoot published the messaged, will receive it, and, according to the Method of the message, it will do the correct operation and send message(s) back to the Server, using the Broker once again. A good example of this kind of interaction is when a Client adds a Sensor: the CyberSensing Layer will start generating data to be sent in messages to the Server, that consequently saves them in the Database. Another kind of interaction that occurs is when a configuration is added by the Client: The Details of the configuration are sent towards the Sensing Layer where it will produce values according to those configurations.

There are a few details in our implementation of the interaction of the Server with the Sensing Layer:

- The messages sent by the Sensing Layer, use the **MQTT** protocol since it should be used in lightweight Devices, with **QoS**(Quality of Service) 0, at most Once, which means we rather prefer not receiving the message than receiving twice, in order to give a better accuracy of the values to be shown in the Frontend.
- The messages sent by the Server use the **AMQP** Protocol, because it gives more reliability, security and much more features. However it has a greater overhead than MQTT. As the messages sent by the Server cannot be lost, we think that the use of this protocol was the best choice on the Server-side.
- RabbitMQ, supports both protocols, which makes the implementation much easier.

The Messages sent by Spring to the Raspberry Pi, will have the following *JSON* Format:

{“method”: method , “type” : type, “value”: value, “id”: id}

The method identifies the operation to be executed :

- ADDSENSOR/REMOVESENSOR, that would be sent everytime we removed or added a new Sensor a Division.
- DEVICE, message sent to change the state of the hardware
- START_CONF/END_CONF, message sent to start or end Configurations

The type represents the type of the Device or Sensor: Temperature, Humidity, Electronic, Luminosity.

The value would be useful in Configuration Messages in order to let the Sensing Layer know what is the desired value for the Client.

The ID will identify according to the type of Message, the sensor or the device.

Configurations*:*

There are 2 types of configurations, one for divisions and the other for devices. Division configurations allow the user to choose the acceptable interval of values that the user wants for a given type. For example, if a user wants to have the temperature always between 10-20 degrees, he would use a division configuration.

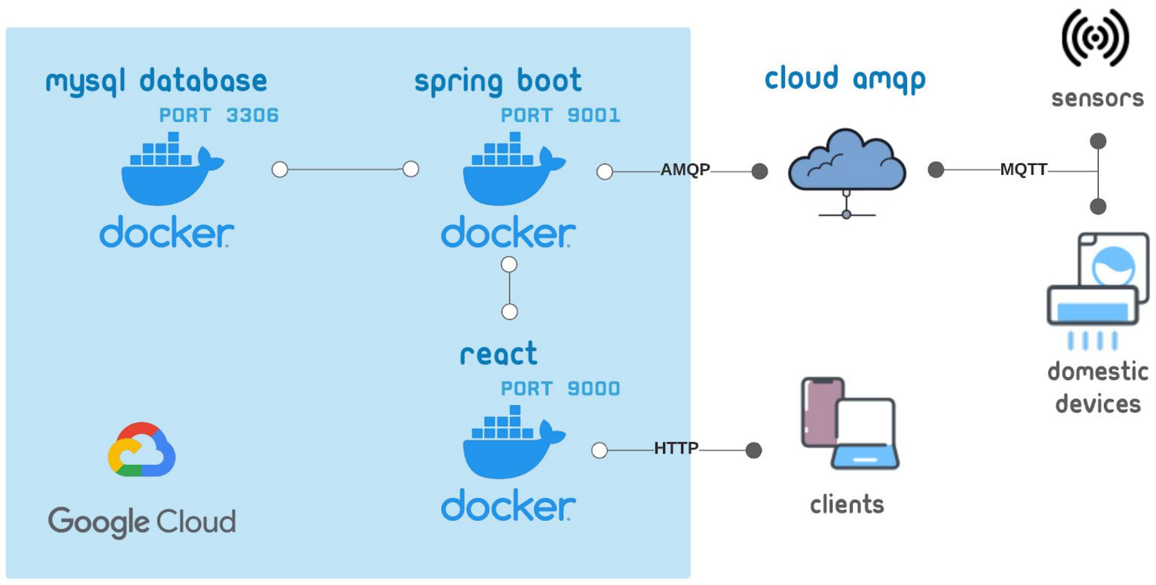
Devices can be configured to turn On/Off at a given time. If the device type is not electronic, the user can choose at which value the device is turned On. For that we used a scheduler in spring that is activated at the time of the configuration.

Notifications:

In order to create Notifications and Alerts to the end-user, whenever there’s action that should be notified, for example, really low and really high sensor values, we save them in a specific table of the Database. Later on, the user may click on a specific button in the Frontend, that will trigger a GET request to obtain the latest notifications. Unfortunately, we don't make these requests asynchronously, but it would be a great feature four our System.

DevOps - Deployment e CI/CD:

Initially we thought about doing the deployment, in the virtual machines, offered by the University. However, inside the University network, a great part of the network ports are blocked, and therefore, the connection with the Cloud Message Broker could never be made. Therefore, we tried to find solutions to our problem, and, as a solution we decided to use a Virtual Machine of the Google Cloud Service, since it was free and not very hard to implement. In order to make the deployment possible in a simple way, we used Docker, a powerful technology used in our practical classes that encapsulates all the code and libraries necessary to run a service. Therefore, there are three containers being used: one for the MySQL database, another for the SpringBoot Server and for the React Application. These three containers are used in a Docker-Compose container, in order to run all services at the same time.



Another concept that we applied to our work was Continuous Integration, a topic that was taught in the theoretical part of the IES course. Therefore, we used **CircleCI** that would try to run our Docker containers after each push of a branch and check if it ocurred an compilation error, which was important to know if we could merge the Pull Requests from each branch, or not. This process definitely improved our dynamic as a team.

4. References and resources

CloudAMQP

Spring integration with CloudAQMP

React

React + Spring + MySQL - Docker

6 Conclusion

With this work, we manage to achieve experience working with the Agile Methodology, a greater interaction in group projects, and the ability to work with many different technologies and frameworks like React, and Spring which initially were difficult to work with, but, as time passed, we got to understand it much better.

All in all, we think we completed our objectives with this project, which was very challenging, but totally worth the time invested into it since it helped us improve as a team to work together better and also increase our individuals capabilities.

As future work, we would definitely improve the user feedback in the frontend, add permissions to the divisions of any house (the only incomplete User Story), and refine the sensing Layer as it is not Fault-Tolerant to our Application.