Universidade do Porto

Faculdade de Engenharia

FEUP

# IoT Live Monitor

Final Report

Software Systems Architecture
Master in Informatics and Computing Engineering

**Authors - Group 5:**
Hélder Antunes - up201406163@fe.up.pt
Inês Proença - up201404228@fe.up.pt
Marcelo Ferreira - up201405323@fe.up.pt
Renato Abreu - up201403377@fe.up.pt

June 15, 2018

# Contents

# 1 Project Overview

The aim of this project is to manage a network of IoT devices and the communications established between them.

Hence, this tool gives an overview of a mesh of devices. Per example, it could be quite useful for someone who owns a smart house since it enables them to perceive the whole network on a simple dynamic graph. A network tree is presented, which animates the messages sent from publishers to subscribers.

Beyond an analysis of the devices, and the communication that is happening between them, a client can manage the network by adding new devices. Regarding each device, it's possible to control the topics that each one is subscribing, publish a message, and analyze the messages consumed and sent.

Although managing the devices is important, one of the key features of this tool is the possibility to have a higher degree of control of the communications established between devices. Each message, as explained in more detail in the next sections, is published to a broker, that is responsible to redirect the messages to different queues, and those queues dispatch the messages to the consumers.

In our client, we've added the management of queues, with the visualization of messages in a factory-like view, using animations to display the arrival of a new message to the queue following by being dispatched to the consumers. Each message can be examined, modified or simply deleted, by simply clicking on them.

On this report, the main problems that we've faced during the implementation are specified in detail, explaining the patterns/approaches used to solve them, as well as its advantages and disadvantages. Likewise, the main features also are identified together with the difficulties that arose from developing them.

# 2 Architectural Styles

## 2.1 Problem 1

### Problem

On an Internet of Things network, there are a set of devices that provide information and others that receive information about particular topics (consumers). So the problem is: How to correctly notify the consumers that there is information of interest?

### Solution

To solve this problem, the pattern used was publish-subscribe. This pattern is commonly used on IoT systems and enables a clear communication between services.

Internally, it uses the Broker pattern to broadcast the messages accurately to all the subscribers. As the figure 1 shows, the publisher pushes the message to the broker who, in turn, transfers the message to those that subscribed to the respective topic.

The publisher doesn't need to know who is using the information that it is broadcasting, and the subscribers don't need to know who the message comes from.

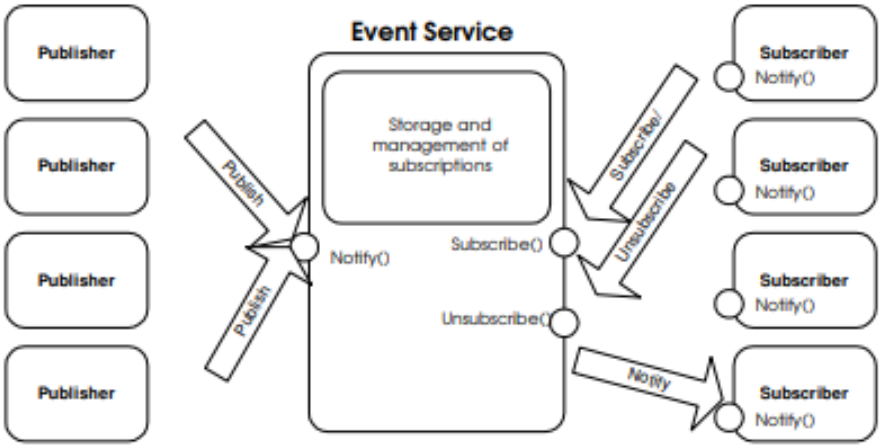


Figure 1: Publish Subscribe system [1]

The strength of this architectural style lies in:

- The publisher does not needs to know who the subscribers are (Decoupling).
- Allows a high number of clients to use it with only one server.
- Messages don't require a specific format.

However, it also comes with some disadvantages [2]:

- The middleman(broker) might not notify the system of message delivery status, so there is no way to know of failed or successful deliveries. Tighter coupling is needed to guarantee this.
- Publishers have no knowledge of the status of the subscriber and vice versa. How can you be sure everything is alright on the other end?
- As the number of subscribers and publishers increase, the increasing number of messages being exchanged leads to instabilities in this architecture; it buckles under load.

- The need for a middleman/broker, message specification and participant rules adds some more complexity to the system.

## 2.2 Problem 2

### Problem

One of the requirements of this project is the control of messages. A user must be able to see the content of a message, update the content or delete the message.

### Solution

We simulate the flow of communication, by creating two brokers (two exchanges in rabbitMQ nomenclature):

- The first broker is a proxy broker that listen to all messages from publishers and afterward sends them to the client application.
- The second broker is the real broker that dispatches the messages to the original consumers.

This flow of communication is shown on the figure 2.

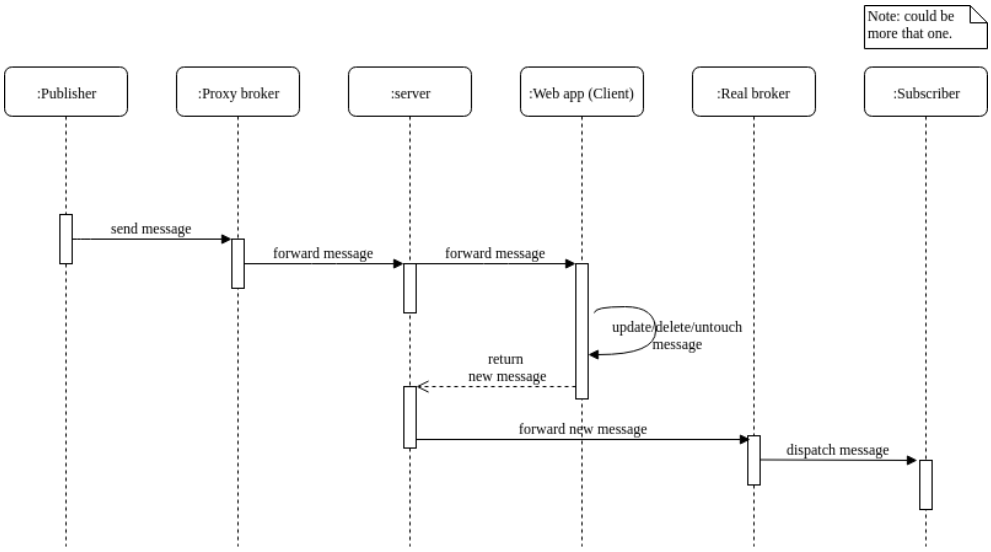


Figure 2: Sequence diagram of the message flow.

The use of proxy broker has the following main advantage:

- more secure because there isn't a direct access to the real broker.

Nonetheless, it also has disadvantages such as:

- complex implementation
- a further step in communication and the consequent loss of efficiency.

## 2.3 Problem 3

### Problem

One of the main features of this project to easily show the message flow between devices. This feature bears an issue, how to correctly animate the network tree or a "package" going through a queue when a message is exchanged?

### Solution

An event-driven architecture was the answer to this difficulty. As the name implies, it triggers some action when an event occurs.

Since our project is mainly based on a simulation of all devices and its communications, the event-driven architecture solves most of the problems that derive from this context.

When a message is published, a new event is triggered. The same occurs when a message is consumed. The action triggered on each event depends completely on the current state of the application and its settings.

By developing an architecture based on this philosophy, we were able to overcome several problems of interconnection between the different parts of the system, maintaining at the same time all the components synchronized and up to date.

Like every other, this pattern has some advantages as well as some disadvantages, such as:

Advantages:

- Well suited for loosely coupled structures.
- It is versatile and facilitates great responsiveness because they are normalized to unpredictable, nonlinear and asynchronous environments.

Disadvantages:

- Low security.
- Increased complexity.

## 2.4 Problem 4

### Problem

This tool features two strategies for delivering messages. In one, the delivery is automatic after a specified amount of time. In the other, the delivery occurs when the user manually clicks in a button.

Furthermore, if the client is not currently viewing the queues, the messages have to be instantly delivered, to not lose pertinent information.

So, how to change the strategy used in delivering the messages, accordingly to the situation?

## Solution

As the problem states, changing strategy on delivering the messages is required, and for that, the design pattern Strategy is used. As such, the strategy is exchanged in an orderly manner.

The correct action for a given instance is chosen basically taking into account the current state of the client's application. By using Vuex, the application state is maintained and altered in a cohesive way. With this library, the necessary data is accessed to decide the next steps that are to be taken.

Below are the advantages and disadvantages of using this pattern.

Advantages:

- Being able to switch strategies in run-time.
- Enables the client to choose the required algorithm without using a *switch* or an *if-else*.

Disadvantages:

- The application must be aware of all the strategies to select the right one for the right situation.

# 3 Logical Architecture

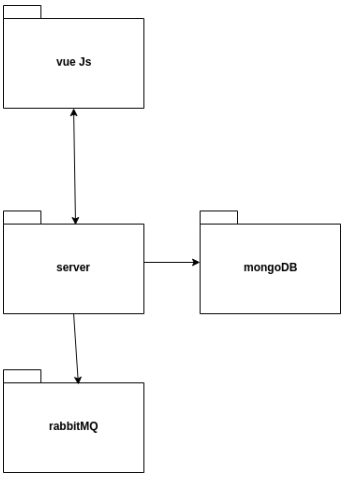


Figure 3: Package Diagram

The system is divided into four main packages (figure 3). The Vue package is view layer, responsible to present information to the user and communicate with the server application.

The server establishes the connection and communication with external services like MongoDB and RabbitMQ, acting as the central entity.

RabbitMQ is the message broker used to establish and handle the communication made between devices.

MongoDB is a NoSQL storage service used to persist information that is inherent to the IoT devices

# 4 Technological Architecture

The web application was made using Vue.js with ElementUI, due to is adaptability and user-usability.

The server is developed in Node.js, with Express.js, to quickly develop an API service with external integrations. The server communicates with both the web application, RabbitMQ, and MongoDB.

## 4.1 Problem 1

The first problem was to find out the best service that we could use as a message broker, and offered the necessary features with the possibility to be easily extended. Among several options, there are Mosquitto, RabbitMQ, Apache Kafka, etc.

After an extensive study of the aforementioned options, we decided to choose RabbitMQ, due to the fact that is a "mature" service, with excellent documentation and online support, already used by several users. In addition, it contains a lot of extra features that could turn up useful for our development. Besides, there are tested libraries build with NodeJS, which eased the integration with our application.

### 4.2 Problem 2

The client needs to communicate with the server in a bidirectional way, to keep the data visualization up to date.

We could use RabbitMQ for that purpose, but the service utilities do not quite lie in this kind of utilization (browser directed). Afterward, we considered other options like Socket.IO.

Socket.IO by allowing real-time bidirectional event-based communication appeared as the best tool to use so that we could develop efficiently our application's features. Another reason to choose Socket.IO is the ease of implementation.

### 4.3 Problem 3

Since it is possible to manage the network, there was the persist in the components of the network as well as the connections between them. We choose to use mongoose as a wrapper of our MongoDB database, but other NoSQL or even a SQL database could be used.

### 4.4 Problem 4

The presentation layer needs to draw a network of devices and animate the network with circles transitions and text representing the content of messages. We consider using multiple libraries implemented on top of D3.js with Vue support. Unfortunately, all libraries found doesn't support the required animations. We ended up using D3.js itself and found out that it is an excellent library with extreme flexibility and immense online support.

Regarding the animations used on each queue page, we recurred to the powerful capabilities of CSS transitions and animations. Nevertheless, we also integrated VueJS transitions which made the final result even smother.

## 5 Physical Architecture

The server machine needs to have the Node.js installed (version 8.x carbon). The web app runs in runs in any browser with JavaScript installed. See the figure 4 to see the connections between the components.

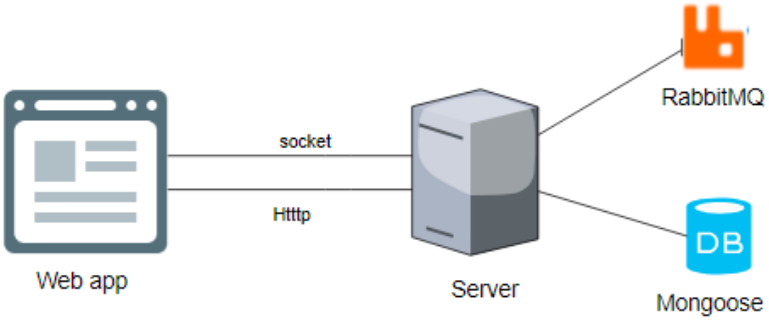


Figure 4: Deployment Diagram

# 6 Execution

In order to all the group work on the same environment, it was used docker. With docker the dependencies are easily processed and it makes sure that the project is reproduced the exact same way on every machine that runs it.

To execute the project just have to do the following:

1. Go to the project folder;
2. On the console, input "*docker-compose up –build* "
3. To access the client side application, access *localhost:3000*
4. To access the server side application, access *localhost:8080/api/*
5. To access the RabbitMQ management, access *localhost:15672*

# 7 Future Improvements

Like in all projects there is always something that could be improved and this project is no exception. So one future improvement would be to, connect a real device (subscriber) into the system, as well as a real publisher.

Since the goal of this project is only to create a platform to manage a network of IoT devices, all the devices are simulated as well as the messages sent between them.

Our improvement would be to test the application in a real environment with real devices. As such, it would be necessary to allow the connection to the

network for external devices, whereas the client service, should, in real-time, analyze the devices introduced and the messages published and consumed. We believe that the main change lies in the exposure of the server application to the devices. The communication, in theory, would be more or less the same.

If time was not an issue and if we wanted to broaden our horizons, the next step would be to create a minimalist application for the mobile. Through the mobile application, it would be possible to access and control the IoT communications in real time.

# References

[1] P Th Eugster. The Many Faces of Publish / Subscribe. pages 1–17.

[2] AbdulFattah Popoola. Design patterns: Pubsub explained.