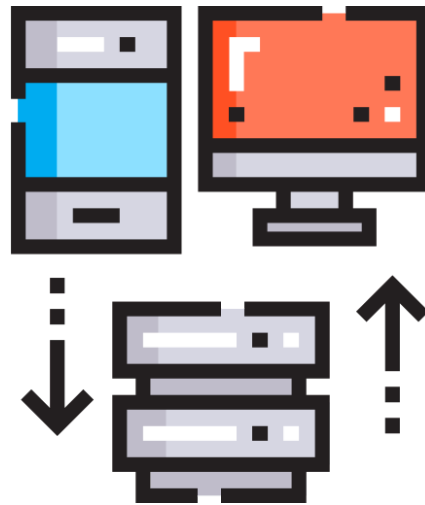


Innovative Smart Systems Project



Thales IoT Predictive Maintenance

DJEBAR	Loïc
FERRANDI	Emmanuel
LAURENS	Pierre
TISSOT	Evan

2019 - 2020

Tutor : Mr. BOYER

Acknowledgments

We would like to express our appreciation to all those who supported us for this project. A special gratitude we give to the crucial role of Thales Alenia Space staff members : Mr. THOMAS, Mr. DOS SANTOS, Mr. CRUZ, Mr. ROUSSEAU, Mr. PAUMEL. They were easily reachable, they took time to answer to our multiple questions and they helped us to manage properly the project.

Furthermore we would like also to acknowledge with much appreciation our tutor at INSA Mr. BOYER who gave us some technical advices, Mr. DI MERCURIO who order the components we needed.

We thank Mr. SCANLAN for their well-informed advices on English documentations and for helping us for the final presentation of our project.

Abstract

The need for corporations to monitor their equipment has led them to take an interest in connected objects. Companies such as Thales Alenia Space have to be able to anticipate the need for maintenance of their equipment in order to save time and money. Our goal was therefore to build an IoT system gathering real time metrics about the internal components of a test bench. The collected data will then be stored and displayed on a dashboard for Thales's engineers to analyze it. We separated our architecture into several parts, each encapsulated in a Docker container. A Python script collects temperature and current consumption of the components. A SSD drive hosts a NoSQL database that stores the retrieved data. A dashboard coded in React displays the meaningful data and allows the configuration of the system. Finally, an API using the Python Flask framework makes the link between all the parts. The result is a working prototype, based on a Raspberry Pi, that supports our architecture. However we were unable to implement some features such as relay monitoring. Further work should concentrate on improving the predictiveness of the system using machine learning algorithms and improve sensor modularity to collect more diverse data.

Keyword: IoT, Satellite, Test benches, MongoDB, Python, Docker, Predictive Maintenance, Dashboard, React.

Outline

Innovative Smart Systems Project	1
Thales IoT Predictive Maintenance	1
Acknowledgments	2
Abstract	3
Outline	4
Introduction	6
Project objectives	7
Context and issues to solve	7
Specifications	7
Organisation	8
Time management	8
Resources management	10
Global architecture of the system	11
Main parts of the system	13
Data Acquisition	13
Sensor for RF relay	13
Temperature and Current sensor	16
Software data acquisition	17
Application Programming Interface (API)	19
Purpose	19
Technical choices	19
Implemented services	19
Dashboard	22
Requirements	22
Technical choices	22
Configuration interface	22
Data visualization interface	23
Database	25
Technical choices	25
Database architecture	25
Deployment	26
Objective	26
Technical choices	26

Implementation	26
Remaining work and further objectives	28
Cloud	28
Predictive maintenance using IA	28
Feedback	30
Code source	31
Used dependencies	31
Dashboard	31
API	31
Python collect data script	31
Download	31
Conclusion	32
Appendices	33
Abstracts	33
Codes	37
Arduino	37

Introduction

The aim of this report is to sum up the approaches and the results related to our project which was conducted during our final year in the ISS speciality. The subject we worked on was proposed by Thales and is about IoT. In an industry context, IoT is at the heart of the fourth industrial revolution (known as Industry 4.0). It consists of having machines and sensors that are connected to a system that can help in managing efficiently resources.

In this report, we discuss about the project management aspect in a first part and we deeply detail the technical points in a second part.

Project objectives

Context and issues to solve

Thales Alenia Space company has many radio frequency test benches that test RF equipments in order to improve RF satellite communications. Those test benches are stored in racks in the company and are likely to be moved all over the world.

Due to intensive use, the components inside the racks are often breaking. It is critical for Thales's Team to have a way to predict those break, in order to anticipate them, and replace the component as fast as possible.

This can be done by collecting data from the components thanks to sensors and by monitoring them.

We can then, using a dashboard, analyze the data and predict maintenance needs. This way we can save time and money. We can also understand more complex failures than parts breaking down thanks to an history.

Specifications

As for all projects, we were provided a list of specifications that the final prototype has to satisfy in an ideal case. Those **61** specifications were first submitted to us, in order to know what was feasible. The specifications are accessible [here](#), and they are mainly divided in three parts :

- **System requirements (34):**

Those requirements are concerning type of energy used, type of connectivity, environmental interaction and hardware used.

- **Dashboard requirements (15):**

Those ones are about the functionalities that the dashboard will have to provide, the way to display the useful metrics and the modification of the system configuration.

- **Integration requirements (12):**

Those requirements are about the integration of the all system inside the racks, size of the parts, non alteration of the RF environment of the rack or electrical protection.

Organisation

Our project was complex, and to be efficient we had to use our management skills. There was mainly two sides to handle: time and resources.

Time management

This project lasted approximately 3 months, this is not much taking into account that we also had numerous courses in parallel. Thus, time management is a key factor for a successful project.

There was two kinds of time management for this project.

- The time management of our meetings and feedbacks with Thales's engineers.
- Our own internal time management of the project and technical aspects.

We received guidance from Thales Alenia Space since they provided us with a schedule :

Design review:

- Proposal of solution(s): End of October

Architectural and design brief: Mid-November

Presentation of the material model: Mid-December

Presentation of the software model:

- Data base : Beginning of December
- Dashboard : Mid-December
- Configuration and modularity of the HMI: Mid-December

System Validation Review:

- Presentation of the validation procedure: Mid-January

Acceptance review: End-January

- test file (plan, procedure, report)
- REX (feedback - costs - summary)

This organization of the different phases of the project has been provided as an indication, however we have tried to stick to it as much as possible. We communicated with Thales as often as possible to ensure a good progress of the project.

We were sending a mail to them every wednesday about our advancement, technical questions or documents they requested and made some face-to-face sessions at Thales headquarters for more significant matters.

Using the schedule given by the engineers, we used time management tools to subdivide the tasks in time and meet the objectives of each phase. We chose to use an **Agile method** because this is a flexible way to manage our project and redefine our priorities at each sprints taking into account the new needs of the client. We decided to use **Trello** to implement this method because we were already familiar with it, it's free and very easy to use.

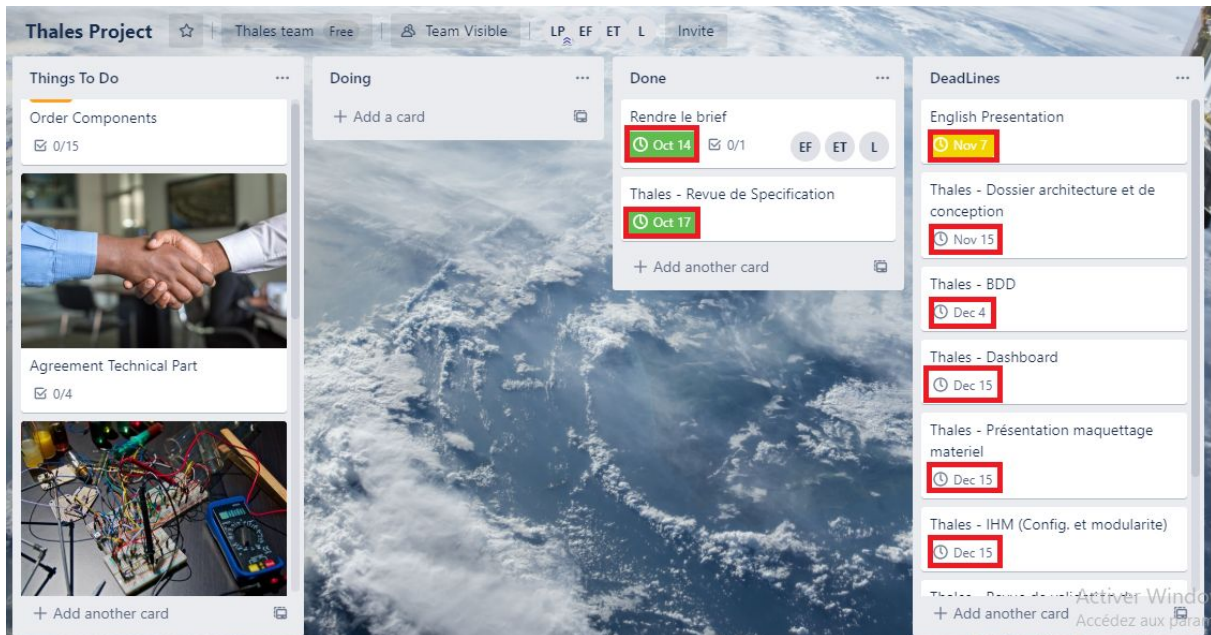


Figure 1 : Time Management using Trello

As we can see, there are four categories: Things To Do, Doing, Done, DeadLines. The *DeadLines* column contains all the known significant dates for this 3 months project. The *Things To Do* column is updated every week with new objectives and at the end of the week all the objectives have to be in the *Done* column.

We also made a project timeline to have an idea of the workload, but since it is a static document and we are using an Agile method, we often updated it to recalculate each time the new estimated workload.

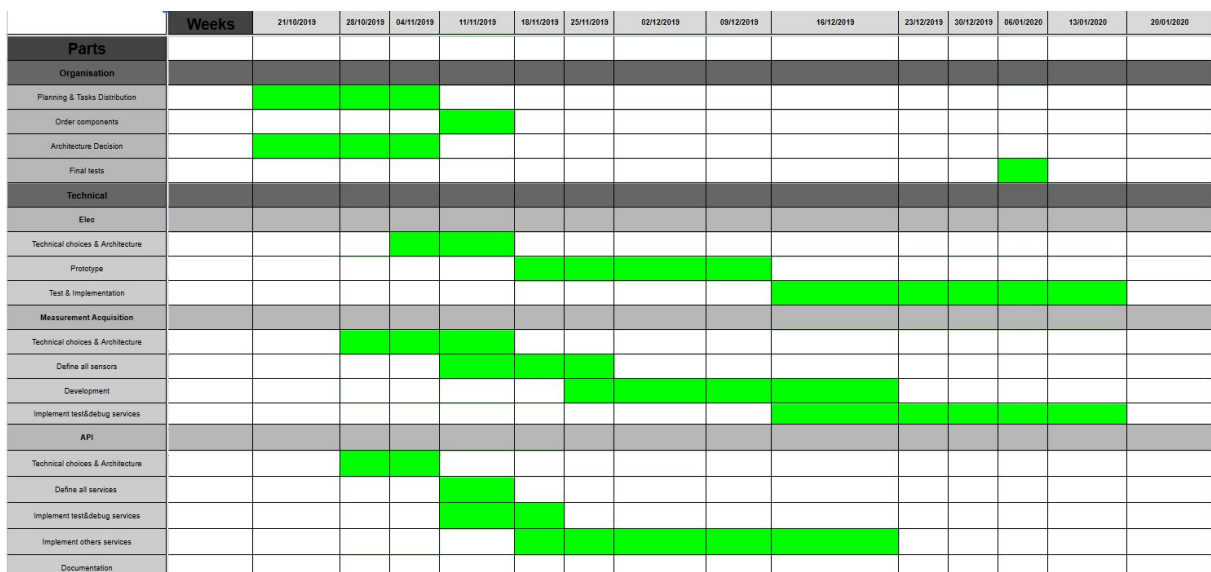


Figure 2 : Project TimeLine for workload estimation

Resources management

This kind of management is also very significant in project management. We chose to consider the skills of the members of the group as resources and allocated them to different tasks.

	API	Dashboard	Cloud/Deployment	Sensors
Loïc Djebbar	X		X	
Emmanuel Ferrandi	X			X
Pierre Laurens	X	X		
Evan Tissot	X	X		

Figure 3 : Skill matrix

With this organization we were able to assign specific tasks to members on the group using **Trello**.

To store our code and work on the project at the same time we used a private repository on **Github** and we used **Google Drive** for all the presentations, documents, and reviews we had to prepare for INSA or Thales.

Global architecture of the system

The architecture of our system has been chosen to be as modular as possible. We divided the project down into a set of major functionalities to be more efficient when working on it and to reduce the impact of a part failure. This choice was also motivated by our method of deployment that we explain in **Deployment** part.

Each part can be developed separately, so that each member of the group can work on a part, but they all have to be compatible with the API which is at the center of our system. The API allows the exchange of information between the different parts and is also useful for testing.

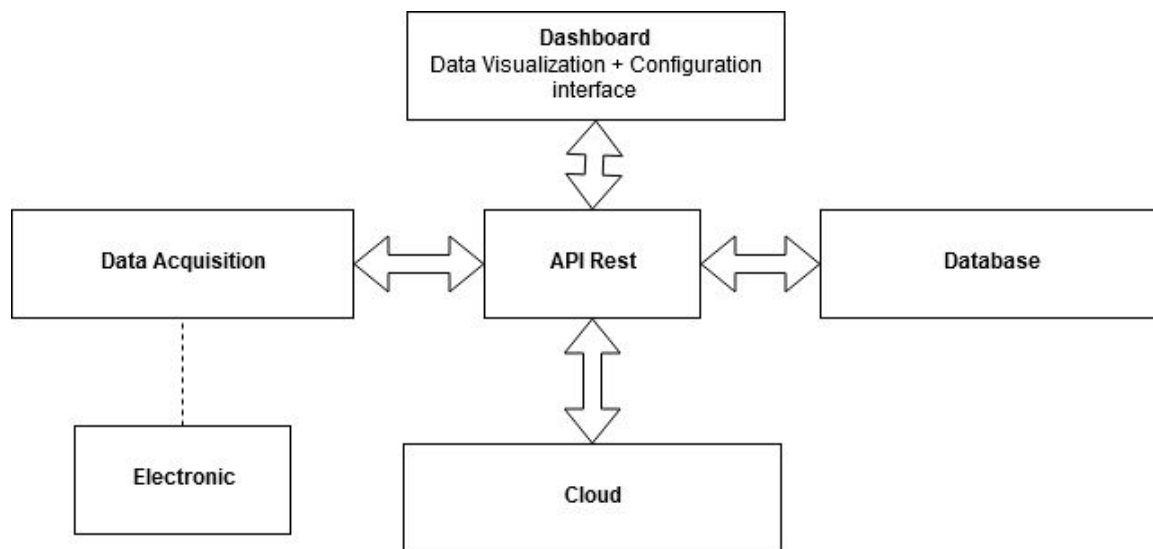


Figure 4 : Global system architecture

There are five parts in this architecture :

- The **API**: At the center of the system, it allows all the parts to exchange information and communicate together. This is a bridge that interconnects different technologies so that the entire system works.
- The **Database**: This is the part that will store all the data needed for the system.
- The **Cloud**: This entity will centralize all the data coming from the different racks in one server in order to have a general back up and a global approach from a monitoring point of view.
- The **Data Acquisition** part: This part is tasked with the data collection of all usable metrics for the components. This is one of the most difficult part because there are electrical and physical parameters to take into account and not just coding.

Concerning the physical implementation, we used a raspberry Pi to embed the several software we developed. A PoE shield is connected to it in order to power the system with an ethernet cable. The sensors are linked to the raspberry using the GPIO. The database stores its data on a 120Gb SSD.

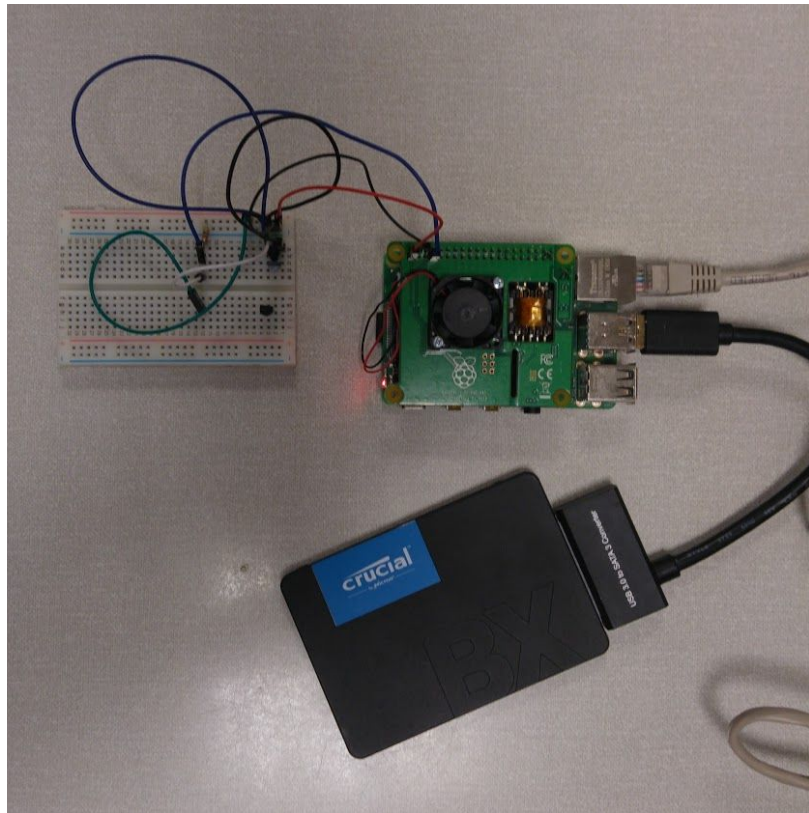


Fig 5 : Physical implementation

Main parts of the system

Data Acquisition

Concerning the Electrical Part, Thales Alenia Space asked us to sense parameters in their racks that can be useful to improve their devices management. Components have a high sensibility to high temperature and current peak. This is why we have to provide a system that can capture and save data in order to see what can be dangerous for the system and replace the broken component as soon as possible. Moreover, the initial purpose was to count the number of switch on radio frequency relay. This component has a certain amount of mechanical channel changes (about one million) and it can break after (or before) this threshold is overcome. No one knows where the problem is so it takes time to find the problem and solve it. This is why, our system can provide information about the rack and help technicians to understand the source of the problem.

Sensor for RF relay

To retrieve the channel changes, we decided to use an Arduino which provides a large number of analog and digital IO. In order to see if we can use this method and to avoid damage on the Arduino we have to ensure that the voltage and the current are not too high. We determined using a multimeter that the voltage was about 5V and the current about 400 mA. These values were measured on a Radiall R574 and when we look at the technical specifications about others RF relays used in the racks, the intensity and voltage can be higher. To face this issue, we think about using optocouplers that can separate one circuit into two distinct circuits. It hence isolates the Arduino from overloads. The component chosen is 4N35.

We designed a PCB (Printed Circuit Board) which are connected between the relay and the cable. Thanks to this, Thales team has to put this component easily without thinking about connections between Arduino and optocouplers. Furthermore, in order to satisfy all relay types, we had to make a setup before using it. The connection between relay and optocouplers has to be done with hand wiring cable between. With all these specificities, we realised the PCB, as you can see on the figure 5.

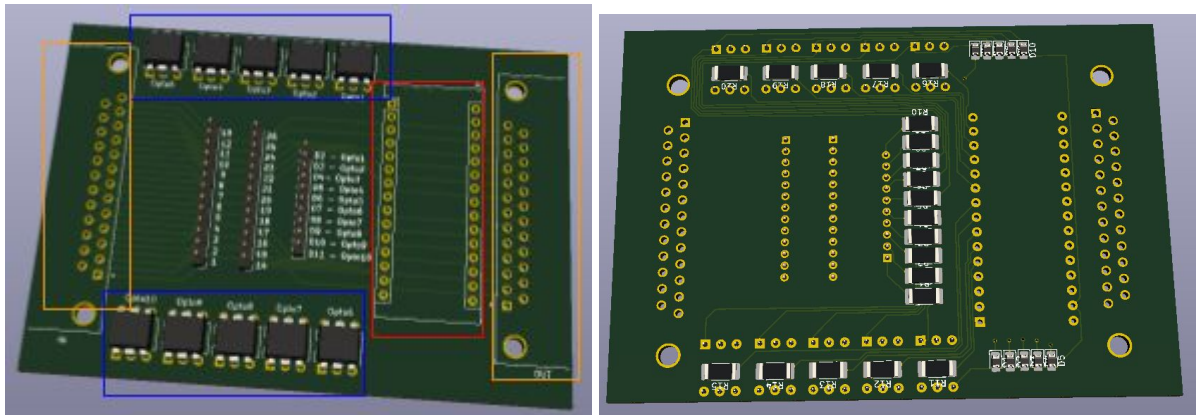


Fig 6 : (a) Front (b) Back View PCB from Kicad

You can see on the figure 6, the Arduino position in red, optocouplers in blue and in orange connectors for relays and command cables. This schematic view rendered in Kicad is very useful to see where can we put components.

We also have an another view showing all wired connection more difficult to read when you show all cables, as you can see on the figure 7.

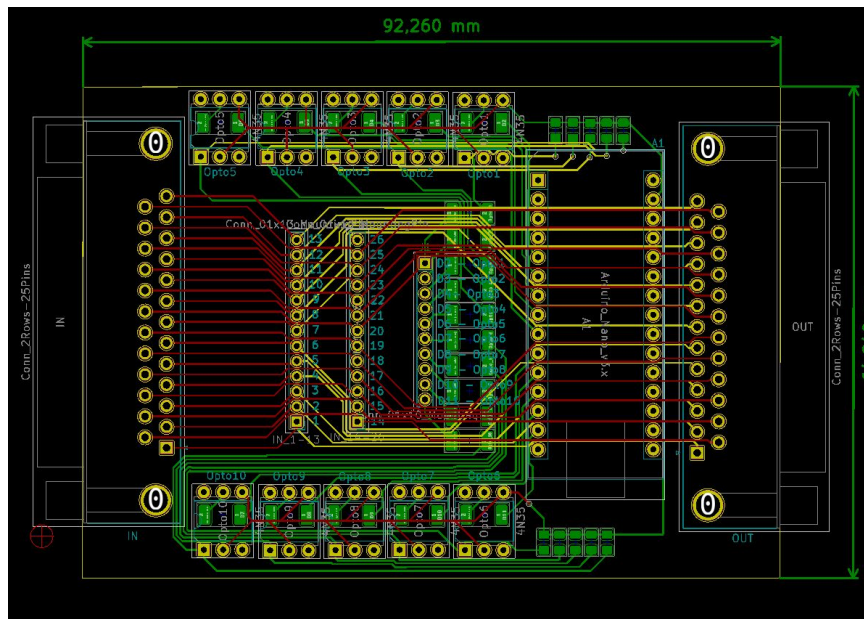


Fig 7 : Wired view of our PCB

Before doing this, we have to connect on an another view all components and set-up them with footprint. We can choose two different way to place it on a PCB :

- SMT: Surface Mount Technology which means that it has not hole into the PCB
- THT: Through-Hole Technology which means that components have leads or pins

At the end, the schematic view for this PCB looks like the figure 8.

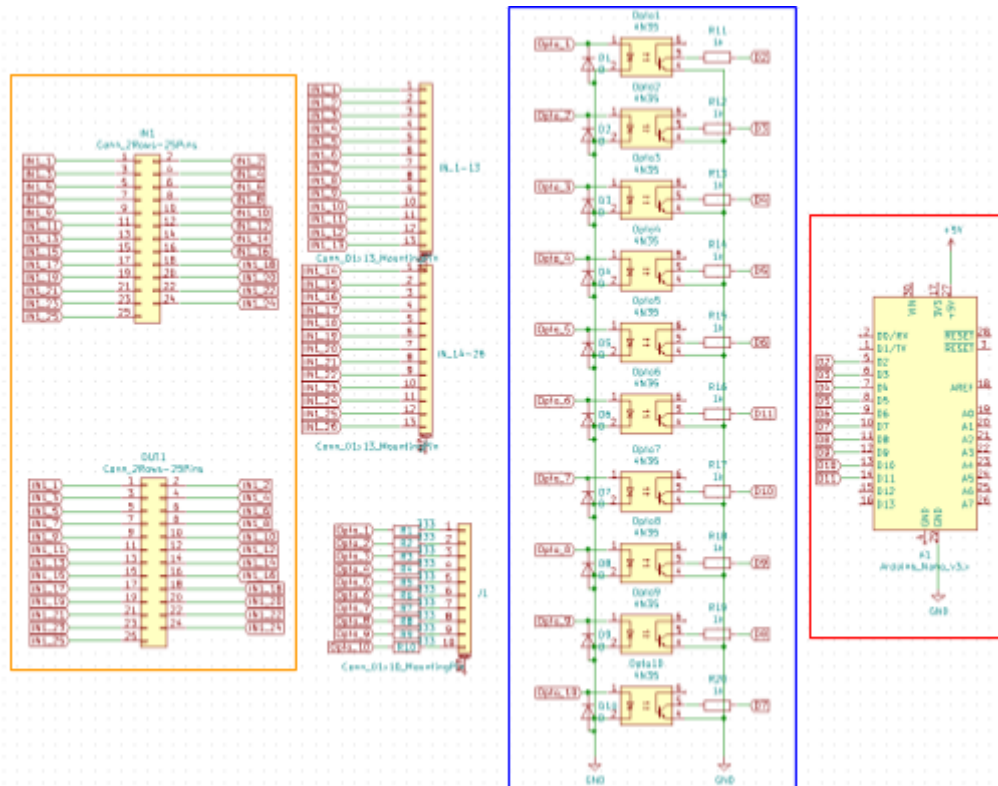


Fig 8 : Schematic view on Kicad

Sadly, this work was not correct due to the digital reading, we was trying to read voltage on digital pin with an Arduino and that is not possible. After research and talking, we have to use the analog pins of the Arduino but only 5 analog pins are accessible. Hence, we have to implement a multiplexer to multiply the port number.

To solve the solution of 10 channels, we chose a multiplexer which totally suits to our application thanks to the voltage and the current allowed. It works with 3-4 pins of command and one pin of reading as you can see on this figure 9.

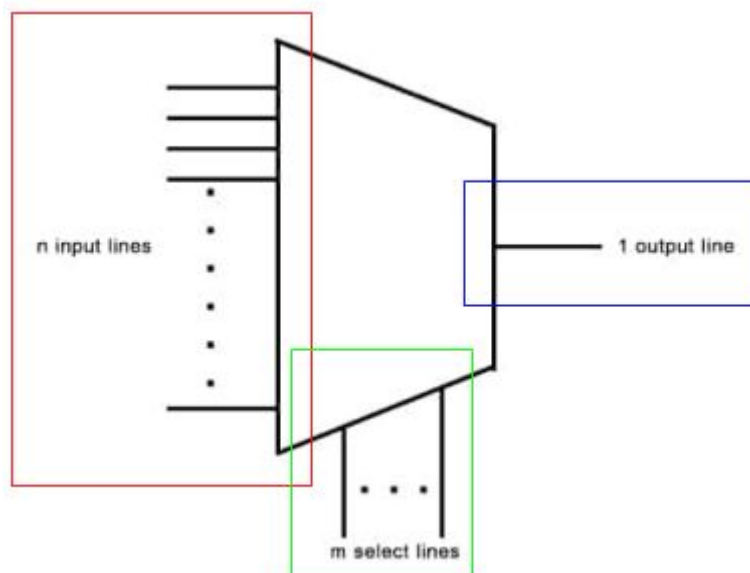


Fig 9 : Multiplexer schema

We choose the 'input line' to read thanks to "Select Lines" in green and we read the value on the 'Output line' in blue with an Arduino. So this component multiply the analog input but we have to deal with the code. You can find the Arduino code with a multiplexer in the Annexe part, code 1.

At the end, the electronic part worked well. It reads over and over all channels and when a change is detected a request is sent to the Raspberry Pi where a Python script is running. This program is linked to the API. The global architecture of this part in order to have a summary of this data acquisition for the RF relay, is visible on the figure 10.

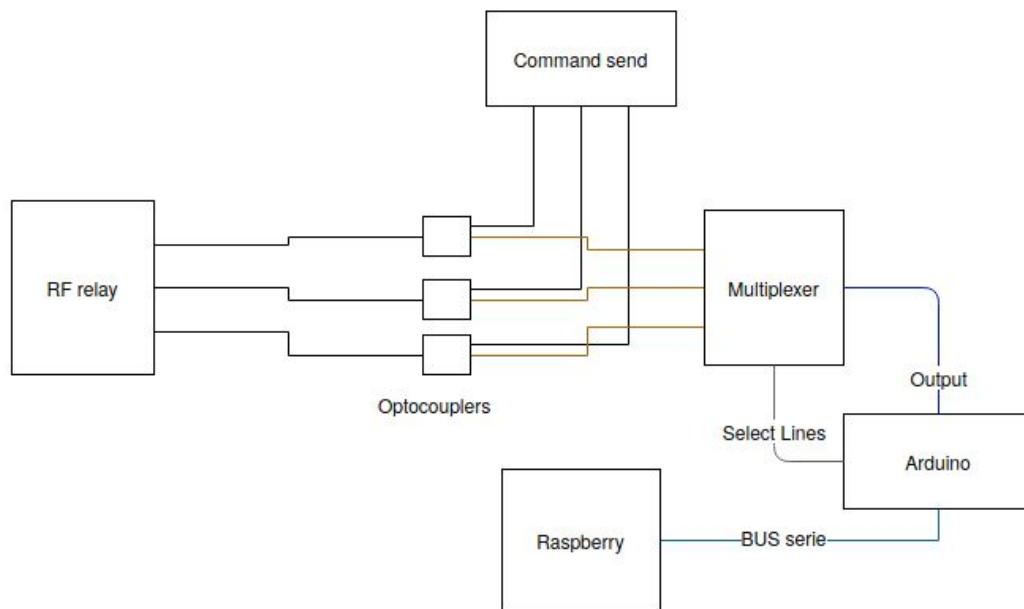


Fig 10: Schema general for RF relay

Temperature and Current sensor

Concerning temperature and current sensors, we were facing technical choices. The bus which controls the sensors have to be easy to deploy and modular. We first thought about I2C bus, but the drawback of this bus is that each type of sensor (e.g. temperature sensor) have the same address. And so we are not able to identify individual sensors.

We shifted our attention towards another type of bus which is called 1-Wire. Each device implementing this technology has an hard coded unique address. It allows us to uniquely identify each sensor. The DS18B20 temperature sensor works with 1-Wire and satisfies the requirements (sensing range, easily implementable in a rack). That is why we chose to use this sensor in our system.



Fig 11 : DS18B20 sensor

Unfortunately, we did not find a current sensor that operates with the 1-Wire bus. We planned to use the ACS712 current sensor which is an analog sensor. Since the raspberry Pi do not have analog inputs, we have to add an ADC (Analog to Digital Converter). We chose the ADS 1115 ADC which work with I2C. On each ADC, up to 4 current sensors can be connected. Due to a lack of time, we were not able to implement the current sensors.

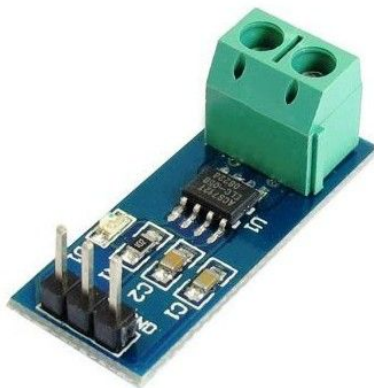


Fig 12: ACS712 sensor



Fig 13: ADS 1115 ADC

Software data acquisition

The temperature measurements are retrieved using a python script. The main steps of this script are the followings:

The configuration of the system is retrieved by making a request to the API. This configuration contains a list of devices and sensors (for the temperature sensors, it is the 1-Wire address).

For each sensor, a function is scheduled using the APScheduler library. This function is periodically called (based on the refresh period which was configured for this sensor). The purpose of this function is to retrieve the measurement, to add the timestamp and to store it in a global list.

Another job is periodically scheduled and which is responsible for sending the content of the global list to the database through an API call.

Application Programming Interface (API)

Purpose

The purpose of the API is to provide a modular and easy way to interconnect the different components of the system. The API give access to many kind of services (e.g. add content to the database, retrieve a measurement from a sensor...). The list of implemented services is described in the part below. Thanks to the modularity of the API, a service can easily be added to extend the capability of our system.

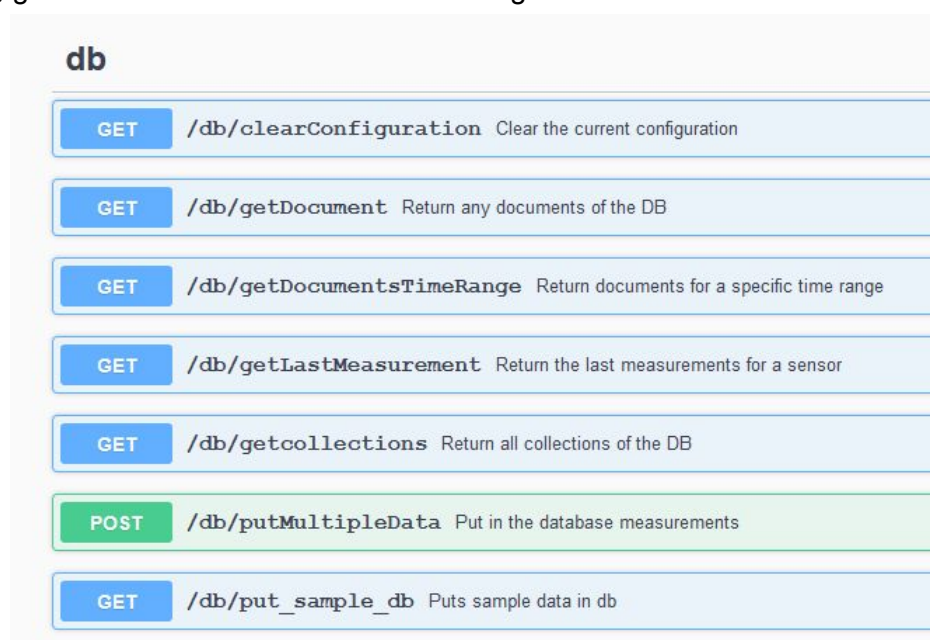
Technical choices

We chose a REST API since it is one of the most used architecture. The implementation of the API was realised using Python language and more precisely with Flask framework. This is a lightweight Python framework that allows developer to define and deploy services. A large community is involved in the project and hence the framework can be extended with many plugins and extensions. We decided to use a plugin that is called Connexion. Connexion allows us to write an OpenAPI specification, then maps the endpoints to our Python functions. It also gives access to a graphical interface where services can be tested and a nice documentation is displayed.

Implemented services

The implemented services are grouped by what the subject they are related to. Below are shown screenshots of the OpenAPI interface.

The db services are used to interact with the database in order to retrieve data or to store data. We have implemented useful services such as get the last measurement for a give collection, get the measurements over a date range...



db	
GET	/db/clearConfiguration Clear the current configuration
GET	/db/getDocument Return any documents of the DB
GET	/db/getDocumentsTimeRange Return documents for a specific time range
GET	/db/getLastMeasurement Return the last measurements for a sensor
GET	/db/getcollections Return all collections of the DB
POST	/db/putMultipleData Put in the database measurements
GET	/db/put_sample_db Puts sample data in db

Figure 14: Implemented services for the database

A very interesting feature of the OpenAPI user interface is that each services are described. On the figure below, we expanded the description of the service `putMultipleData`. It helps people to understand that in order to use correctly this service, a POST request has to be sent with a body which contains a list of object with the following fields : `sensorI2CAddress`, `sensorID`, `timestamp` and `value`.



Figure 15: Description of a service

The system services allow us to retrieve information concerning the system such as the uptime or the remaining free space on the storage.

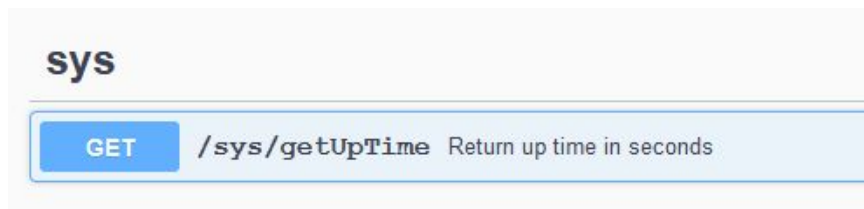


Figure 16: Implemented service for system management

A test category has been developed in order to test and debug our system.

Tests		
POST	/test/createTestConfiguration	Create a test configuration
GET	/test/generateRandomData	Generate random Number
GET	/test/getTimeMeasurements	Get measurements for a specified date range
GET	/test/populateTestMeasurements	Populate the dB and the collection MeasurementsTest with random data

Figure 17: Implemented tests for debug

Dashboard

Requirements

A user interface is required in order to have access to the measurements. It has to be user friendly, easy to use and displays all relevant informations in an efficient way.

Thales Alenia Space's team have expressed expectations concerning the user interface that are written in the specification paper and we need to fulfill them.

The dashboard is composed of two main parts : a data visualization section in which measurements are displayed in near real time and a configuration section which helps in adding sensors as well as giving a name to the system.

Technical choices

Concerning the technical aspect, we decided to use React language to develop the dashboard. React is a JavaScript library for building user interfaces and dashboards. It is a very efficient language since when a change occurs it reloads only the component that needs to be refreshed and not the entire page. React is also a very popular language and take advantage of a great and active community. Hence, many plugins are freely and openly available.

We used Material UI framework which provides components that helped to design the user interface. This framework is based on Material Design which is a design language that Google developed.

Configuration interface

The configuration interface allows people to configure the system. Parameters (e.g. the name of the rack, the uptime or the IP) can be retrieved in this section.

Configuration du système

Nom du tiroir	myRack
Uptime (en heures)	8.28
Adresse IP Locale	



Figure 18: Configuration of the system parameters

It is also in this section that components can be added to the system. Two tables are displayed :

- A table where devices are listed (amplifier or switches)
- A table where sensors are listed (temperature, current, switches count)

For the both, content can be added as well as deleted. When a device or a sensor is added, several parameters are asked (name, limit values, addresses...)

Composants

Q Search X			
Actions	Nom	Type	ID
	myAmp	Amplificateur	Ampli1
	DeuxiemeAmpli	Amplificateur	Ampli2

Ajouter un composant



Amplificateur ▼

Nom du composant

VALIDER

Figure 19 : List of devices

Capteurs

Q Search X			
Actions	Nom	Type	Composant
> 	FirstTemp	Temperature	myAmp
> 	qsd	Temperature	DeuxiemeAmpli

Ajouter un capteur

Type de composant ▼

Nom du composant ▼

VALIDER

Figure 20: List of sensors

When the configuration is applied, a request is sent to the API and the configuration is written in the database.

Data visualization interface

This is the section that is immediately displayed when we browse the dashboard. It displays in an efficient way the measurements. According to the made configuration, each configured device is represented as a tile in which the last measurement is displayed. The value is coloured depending on if it is within the functional range or not.

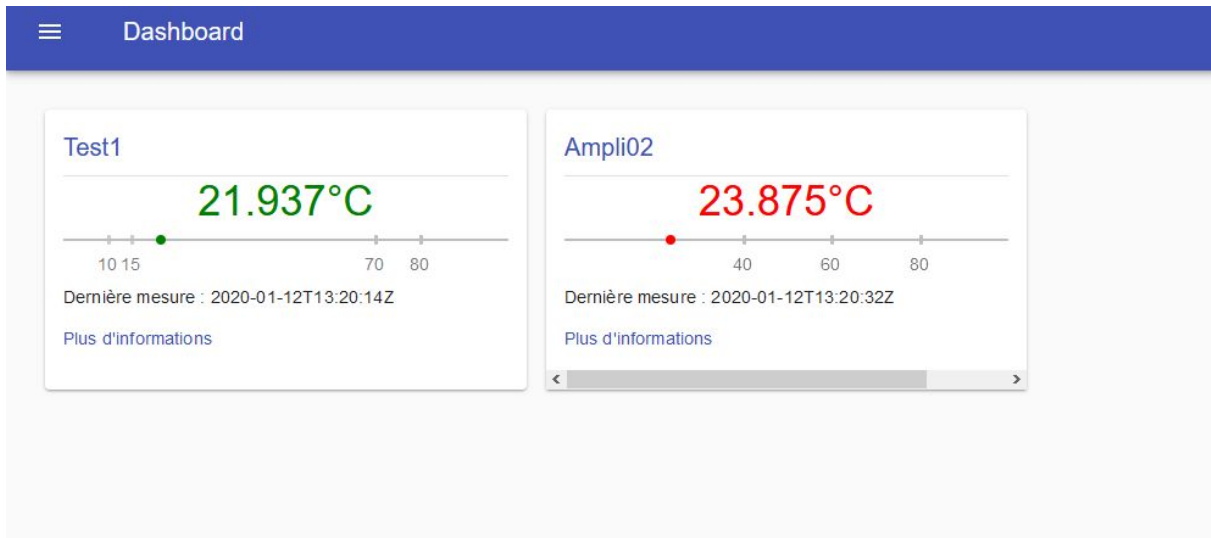


Figure 21: Main page of the Dashboard

We can get more informations by clicking on the button “Plus d’informations”. Then a chart is displayed showing all the previous measurements. The range is configurable using inputs as well as dragging an area over the chart.

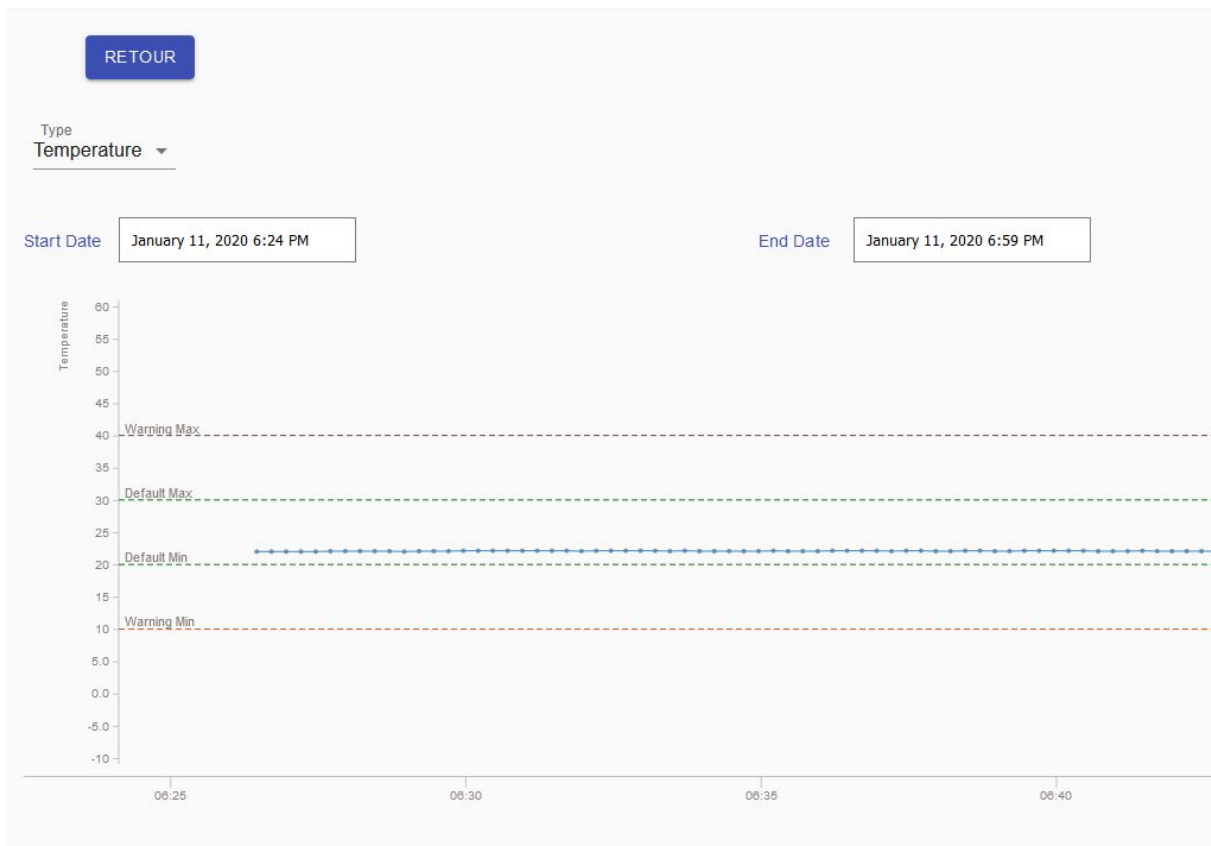


Figure 22: Line chart of a sensor

Database

Data have to be stored locally on the system. Hence a database is needed. The data will be stored on a SSD drive plugged to the Raspberry Pi card so that they can be easily accessible and protected in case of issues with the card.

Technical choices

We decided to use a NoSQL database since it offers many benefits such as a flexible data model (various type of data can be stored and the structure can evolved dynamically) and high performances.

The implementation was made using mongoDB which is one of the most popular noSQL database. There is a large community and hence many plugins are available. The pyMongo tool allows us to link easily the database to the API.

Database architecture

In a noSQL architecture, database are organised into several collections. Each collection contains many documents.

Our architecture is organized as presented in the figure below. There is a configuration collection which contains only one JSON document. This document describes the configuration of the system (name of the system, list of devices and sensors and their respective configurations). The others collections are related to the sensors : each sensor is linked to a collection in which it stores its measurements. A measurement is a JSON document containing a value and a timestamp.

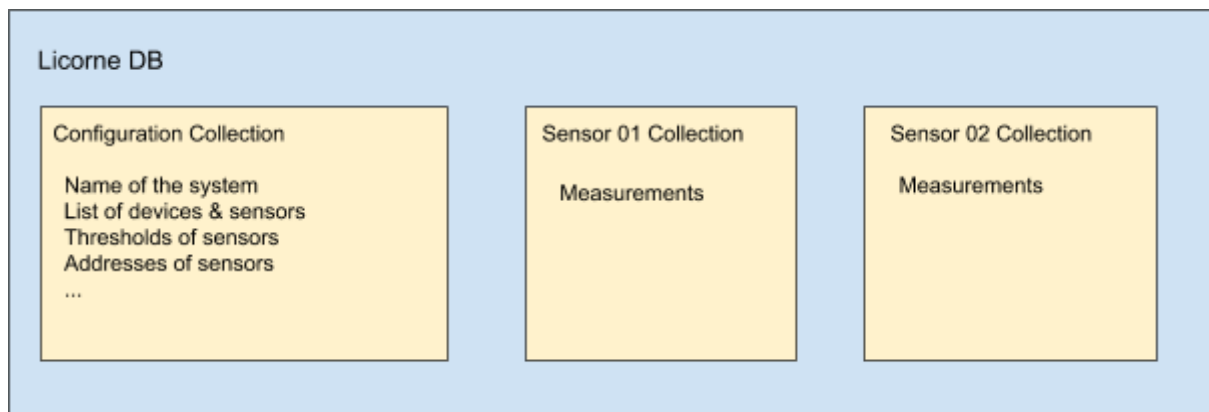


Figure 23: Database's Organisation

Deployment

Objective

Since Thales's engineers are not familiar with the IoT environment which is quite new, one of the main requirement was to make an easy-to-use system. Therefore the system should be easy to use and to replicate.

Technical choices

In order for our system to be easily deployable and replicable, we chose to put each part in a docker container. We have four main parts, so logically, we have four docker containers that needs to be launched when the raspberry starts. The containerization technology allows a great flexibility since we can provide Thales's engineer with the images of each part already configured to work properly and they only have to run it.

Implementation

As explained before, we built four custom docker images for this project :

- API: We built a container to host the API using a Python image (it's an optimized Ubuntu image with Python installed on it) as a base. On this image, we copied the files and the Python scripts needed for the API to run and installed the necessary libraries. When this image is started, it will execute a Python command to launch the API.
- MongoDB NoSQL Database: We explained previously that we chose with Thales's engineers to use a NoSQL model for the database. What we didn't plan for, was that since the Raspberry Pi system architecture is unique, we had trouble finding a compatible image. We lost considerable time, but an image running MongoDB compatible was found. A shared file between the SSD and the container was created so that all the data can be stored inside the SSD.
- Dashboard: The React language we chose for the dashboard allows the building of the code into a small archive to reduce and optimize its size and performances. Once the dashboard was completed, we built it, and put the small files on a very light nginx image (web server).
- Data Collection: This container is also using a Python image to run a python script that can access the USB ports and the GPIO pins of the Raspberry. This container has to be run in "privilege" mode to access the hardware of the card.

Each image had to be built on the Raspberry to be compatible with its architecture.

Once the images of each parts are loaded on the machine, we use the **docker-compose** container manager to deploy all the containers with a single command using a configuration file. The container manager will then autonomously manage the network connectivity between the containers and with Thales's network, reboot the containers in case of failure or anything related to them according to the specification written in the configuration file.

A BASH script is used to run the docker command that will start the containers when the Raspberry Pi starts.

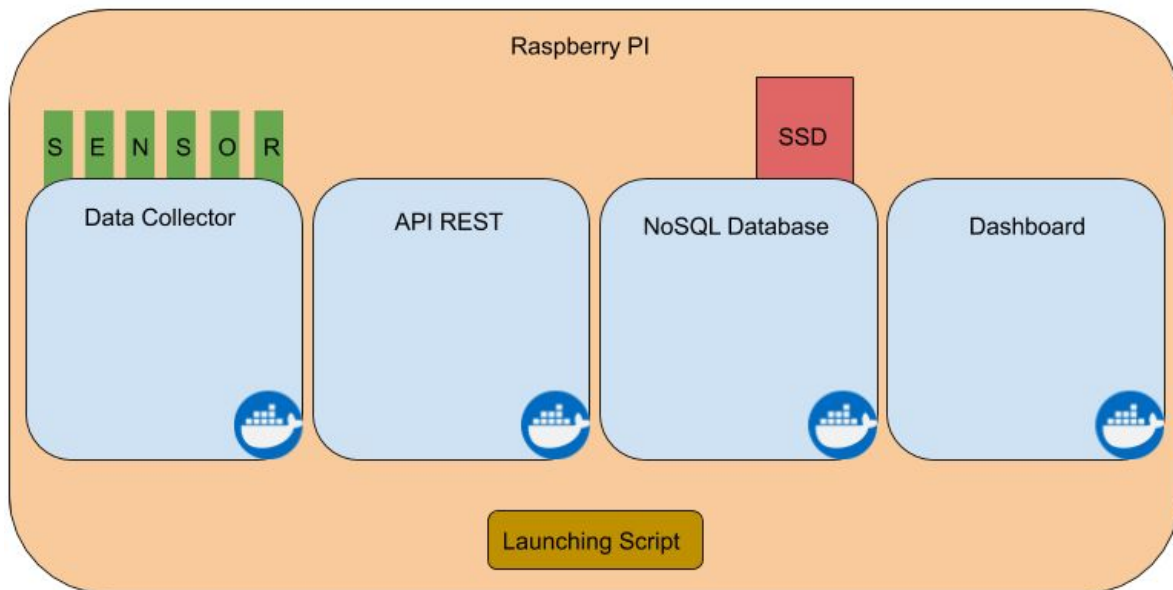


Figure 24: Docker Architecture on the Raspberry Pi

Remaining work and further objectives

Cloud

One of the initial goals of this project was also to be able to display all the data collected from the different racks on a single centralized interface. This dashboard should have been running on a server in Thales's network and accessible to Thales's engineers.

The idea was to improve our initial API code so that it can handle the communication with multiple racks and modify the menus of the dashboard to be able to consult the data from each rack. Those improved version would also have been containerized so that once they are ready we just have to deploy the three images on the server. The general database would have been periodically update thanks to request made by the General API on each racks.

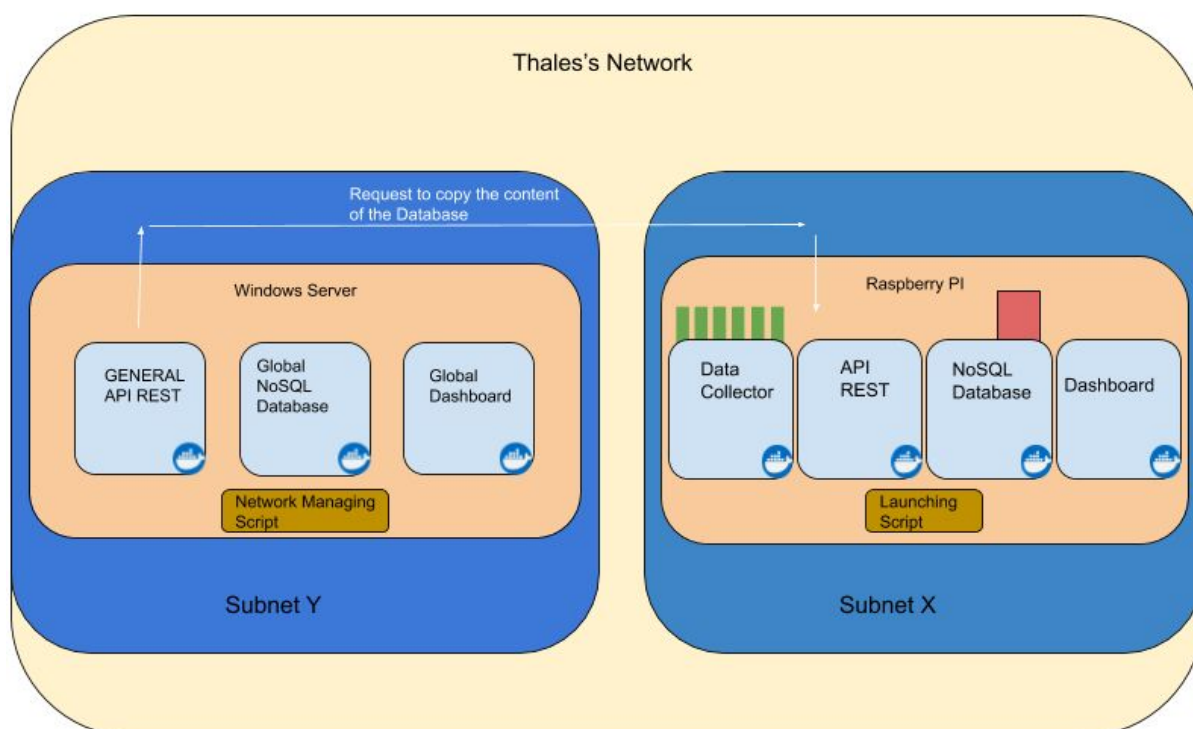


Figure 25: Theoretical representation of the network architecture

Unfortunately we didn't have the time to realize that part, Thales's engineer and the team decided that our efforts should be focused on the local part of the system.

Predictive maintenance using IA

This part is the natural next-step for this project. It consists of developing machine learning algorithms that could analyse the collected data.

Based on the large amount of data, forecasts can be realised in order to predict potential breakdowns. Neural networks can detect the actual trend and determine if a breakdown will occur.

Outliers detection algorithm could be developed in order to detect abnormal values efficiently and in near-real time. A such method can be implemented with a moving window average technique or with a neural network.

Feedback

This project was a valuable experience since we worked directly with professionals. In a way, they were our customers and we discovered the process of a project (expression of the requirements, reviews, documents to be returned). Moreover Thales Alenia Space is a company with a great renown and we had the opportunity to visit their site.

We are proud of what we have realised but we are frustrated since we did not have time to complete everything. During the specification review with Thales Alenia Space team, we probably underestimated the technical challenges and the short period of time we had for this project.

Code source

Used dependencies

Our project used some external softwares. It was mandatory to work with dependencies that have a non-contaminating licenses because otherwise we have to share in an open way what we have done and Thales did not want this. Below are listed the integrity of the dependencies with their respective licenses.

MIT, BSD and Apache are non contaminating licenses

Dashboard

React : MIT

Material-ui : MIT

@material-ui/icon : MIT

axios : MIT

ringjs : MIT

material-table : MIT

pondjs : BSD-3-Clause-LBNL

react-datepicker : MIT

react-timeseries-charts : BSD-3-Clause-LBNL

API

Flask : BSD

Flask-cors : MIT

Connexion (OpenAPI) : Apache License

Pymongo : Apache License

Flask-pymongo : BSD 2-Clause "Simplified" License

Uptime : BSD

Python collect data script

APScheduler : MIT

W1thermsensor : MIT

Download

Source code is downloadable using the link below :

<https://drive.google.com/drive/folders/1IGWWlrzj4pORZCbTPxZzes387ptkyk7X>

Conclusion

This project was one of our first collaboration with a company such as Thales which was, in this case, our client. During these first steps in the industrial field we discovered the different stages of the realization of a project in a large group.

Furthermore, we thought that this experience is well integrated within the ISS formation since it gave us the opportunity to have a practical approach on different subjects we studied during the cursus (service oriented architecture, containerization, project management...).

Appendices

Abstracts

From Pierre Laurens - IoT System for THALES Alenia Space HR Test Bench

To avoid a maximum of malfunctions in the High Radio Frequency communication system of their satellites, Thales Alenia Space has designed a test bench. This test bench reproduces the behavior of the Radio Frequency system and can detect errors in advance. Our objective was to build an IoT system to monitor in real time the above equipment, to detect anomalies, to log the history of the events raised on these devices and to predict maintenance for optimizing operation of the test bench system. To reach these objectives we set up various sensors: sensors for measuring temperature of the amplifiers, sensors for monitoring electrical current on the amplifier input, sensors for capturing the number of relay switches. Thanks to the installation of a Raspberry Pi and an Arduino card we developed a Python software for acquiring data from these sensors and for storing them in the MongoDB database. We designed a React API to set up Dashboards from data stored in the Database. The majority of the features of the software have been implemented but we were unable, however, to successfully design the Relay sensor due to a lack of electrical skills. Accordingly we demonstrate the feasibility of implementing an IoT system to monitor a satellite test bench. While we delivered Thales Alenia Space a working prototype, this prototype has only partial functionality. Future work will concentrate on the construction of the Relay sensor.

Keywords: *Thales Alenia Space, Tests bench, Raspberry Pi, Arduino, React, Python, Dashboard, MongoDB, Sensors, Amplifier, Relay, High Radio Frequency, IoT, Electrical.*

Thales Alenia Space had an issue with their test bench products when a component is broken. But the Thales team can't find where the problem can come from. They had to test all components one by one in order to see the component to fix. It was a waste of time and money. Instead changing or testing each material inside the rack, this was our objective to provide an application which can calculate and see what can be wrong or if there is potential damage on it. This is why, our team built an application using several parts:

- The collected data is ensured by current and temperature sensors which are collected by Python programs*
- An API using Python which is the head of the architecture, providing communication between all parts*
- A Dashboard where all information can be set up and viewed based on React language*
- A Database developed with NoSQL to store data*

After few months of work, we succeeded to put all parts together and displayed data sensors on our dashboard. We used a Raspberry Pi and Arduino board to develop this Proof of Concept. To improve our solution, we can provide a predictive maintenance with the previous collecting data in order to optimize the component replacement.

Keywords: *Rack for RF, Test bench, predictive maintenance, Thales Alenia Space, Raspberry Pi, Arduino, Python, React, I2C / OneWire, PCB*

From Evan Tissot- IoT System for THALES Alenia Space HR Test Bench

Thales Alenia Space's engineers are facing issues with their radio-frequency test benches since some of their parts might break down. This is problematic because it leads to unusable test benches during the repair time. It costs time and money.

We were asked to build an IoT system in order to monitor in real time the state of the several internal components. The system has to detect and report incidents (e.g. abnormal temperature). Collected data have to be stored and will help Thales teams to understand the origin of the potential breakdowns.

This type of system can be extended to several industrial use cases. In this paper, we suggest an architecture and we present its implementation.

We designed an architecture that is composed of several parts :

- Python scripts that retrieve temperature and current consumption of amplifiers and number of switches*
- A noSQL database that stores measurements*
- A dashboard built with React language that displays live measurements of the system and provides access to the past measurements. A configuration interfaces allows the staff to add/remove and configure sensors*
- An API developed in Python with the Flask framework that allow communication between the parts*

We managed to build a working prototype on a Raspberry Pi that can be used as a proof of concept.

The natural next step is to use collected data in combination with machine learning algorithms to detect more accurately anomalies. Many features could be added to improve the system such as notifications (SMS – emails) and implementation of plug and play sensors.

Keywords : *IoT – Predictive Maintenance – Test benches – Dashboard – Measurements – Monitoring – MongoDB – Python*

From Loïc Djebbar - IoT System for THALES Alenia Space HR Test Bench

The need for corporations to monitor their equipment has led them to take an interest in connected objects. Companies such as Thales Alenia Space have to be able to anticipate the need for maintenance of their equipment in order to save time and money. Our goal was therefore to build an IoT system gathering real time metrics about the internal components of a test bench. The collected data will then be stored and displayed on a dashboard for Thales's engineers to analyze it. We separated our architecture into several parts, each encapsulated in a Docker container. A Python script collects temperature and current consumption of the components. A SSD drive hosts a NoSQL database that stores the retrieved data. A dashboard coded in React displays the meaningful data and allows the configuration of the system. Finally, an API using the Python Flask framework makes the link between all the parts. The result is a working prototype, based on a Raspberry Pi, that supports our architecture. However we were unable to implement some features such as relay monitoring. Further work should concentrate on improving the predictiveness of the system using machine learning algorithms and improve sensor modularity to collect more diverse data.

Keyword: IoT, Satellite, Test benches, MongoDB, Python, Docker, Predictive Maintenance, Dashboard, React.

Codes

Arduino

```
int valuePin;
int memoPosition = 0;
#define MUX_CH_COUNT 10 // Reduce this number if you use less channels
#define PIN_D_MUX_S0 8 // bit 7 of PORTB
#define PIN_D_MUX_S1 9 // bit 6 of PORTB
#define PIN_D_MUX_S2 10 // bit 5 of PORTB
#define PIN_D_MUX_S3 11 // bit 4 of PORTB
#define PIN_A_MUX_SIG 3 // This pin will read the input from the mux.
void setup() {
  pinMode(PIN_D_MUX_S0, OUTPUT);
  pinMode(PIN_D_MUX_S1, OUTPUT);
  pinMode(PIN_D_MUX_S2, OUTPUT);
  pinMode(PIN_D_MUX_S3, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Read all channels
  for (byte i=0; i<MUX_CH_COUNT; i++) {
    PORTB = (PORTB & B11110000) | i;
    short val = analogRead(PIN_A_MUX_SIG);
    // Print the values...
    Serial.print(i);
    Serial.print(": ");
    Serial.print(val);
    Serial.println(" | ");
    if(val > 500 && memoPosition != i){ //by default to have positive
Voltage value
      Serial.print("Position : ");
      Serial.println(i);
      memoPosition=i;
    }
  }
}
```

Code 1: Code Relay with Multiplexer