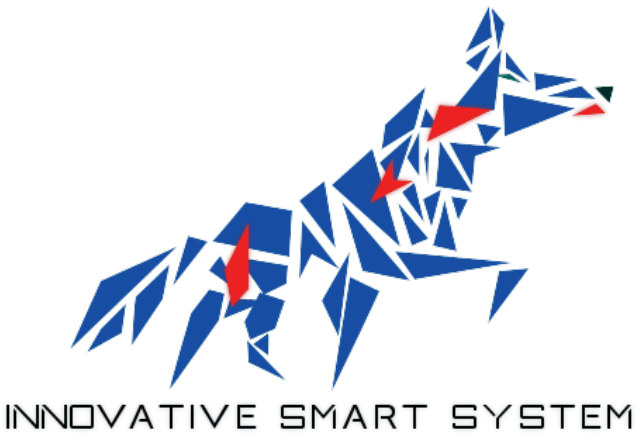


INSA Toulouse - 5ISS

Innovative Project

GEOμBEACON

Students :
Benjamin Abonneau
Kenza Bouzergan
Lamiaa Housni
Stig Griebenow

Teacher :
Thierry Monteil

January 31, 2024

Contents

1 Introduction

This project report documents the creation of a prototype of a floating satellite beacon as part of the Innovative Project course at INSA. The goal was to build a minimum viable product (MVP) for the company ACTIA. Throughout this report, we detail our step-by-step approach, technical details, and practical experiments involved in creating this prototype. This document describes our process from initial ideas to the final prototype. We share the rationale behind our design choices, technical specifications, and the tests we conducted to ensure the prototype's effectiveness and reliability.

1.1 ACTIA

ACTIA GROUP is a group specialized in the manufacture of electronic components for the automotive, telecommunications and energy sectors. The group has twenty-four sites in sixteen countries. Its headquarters are located in Toulouse. We worked under the Aerospace Division. Our project manager was the CTO Aerospace named Jordi Munoz-Pons.

1.2 Project Background

Our project was born out of a gap in the market - a lack of cost effective floating satellite beacons. ACTIA recognized this gap and the critical need for affordable solutions, especially for applications where budget constraints are a challenge for customers. One example is the small-scale fishing sector, where individuals, such as a Thai fisherman managing his nets, yearn for a tracking system but are constrained by the cost of existing options. In essence, our project solves the problem of an untapped demand for low-cost floating satellite beacons in a variety of applications. This problem led us to the following solution.

1.3 Solution

Our solution consists of four components (as seen in figure 1), the beacon itself, an application for configuring the beacon and viewing the data, the Kinéis satellite service, and a cloud architecture for storing and managing the collected data. The detailed requirements are described in the following chapter 2.

1.4 Kinéis Service

Kinéis is based in Toulouse, France, and was officially launched in early fall 2018 alongside a larger company, CLS. The organization deals with telecommunications services, with the aim of providing communication for Internet of Things (IoT) products, using a multi-satellite connection from space to connect devices on Earth. The Kinéis solution is a perfect fit for our product, not only because it is located on the same site as ACTIA in Toulouse, but also because it is a perfect technological fit. This is explained further in chapter 3.

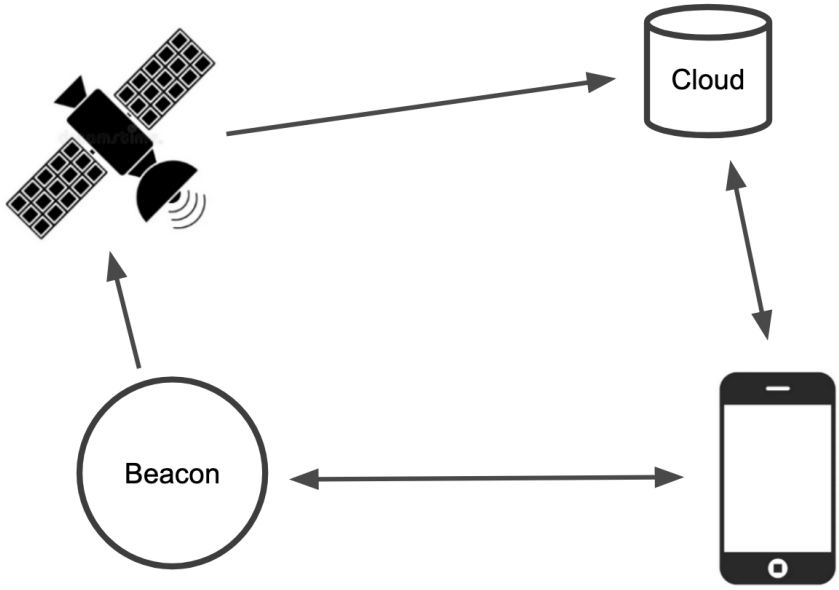


Figure 1: A high level model of our proposed solution.

2 Requirements

In the following, we first describe the initial requirements of ACTIA and then the subset of requirements that we want to implement in our MVP. As the project progressed, we realized that the scope required by ACTIA was not feasible, so we significantly reduced the requirements. Those reduced requirements are described in chapter 2.2.

2.1 Requirements From ACTIA

The initial ACTIA requirements can be found in Appendix A.1. There are 32 requirements that apply to the beacon, the application, and the cloud. In the table, they are described in the "Reqt text" column and the "Rationale" column describes how the requirements are verified. These requirements are summarized in this chapter.

2.1.1 Beacon

In summary, ACTIA's requirements for the beacon included a floating prototype that would transmit sensor data via Kineis every 30 minutes, would run for over a year without recharging, and could be recharged by induction. The whole beacon had to fit in one hand, be waterproof and cost less than 50€.

2.1.2 Cloud

The requirements for the cloud are as follows. The cloud should store the data and locations collected by Kineis. It should also provide user management and authentication. This means that each user can only access the data and locations of his own beacons. The technologies used in the cloud should be agnostic, i.e. platform-independent.

2.1.3 App

The requirements for the application can be summarized as follows. The app should display the (user-specific) data from the cloud. Each user should be able to see their beacons, locate them, and analyze the collected data. In addition, it should be possible to "program" the beacon via the app (via BLE). This means making settings for how often and when data is sent, or when, for example, a warning should be sent. In addition, the app should be attractively designed and not platform specific (i.e. it should run on both Android and IOS).

2.1.4 Common requirements

There are also some common requirements. All parts of the system and communication should be cybersecure and CE compliant. It should also be designed for mass production.

2.2 Requirements for our MVP

As mentioned above, during the course of the project we realized that not all of the requirements could be implemented in the short time available, so we decided to reduce the number of requirements and implement only the most important ones. These changes are discussed below.

2.2.1 Beacon

For the beacon, we decided to ignore the ability to swim, the water resistance, and the battery (and charging it). This helped us focus on the important things (sending data via Kineis and Bluetooth). We did take size into account when choosing parts. However, the beacon was not implemented in the desired size because we were working with evaluation boards to save the effort of printing a custom PCB.

2.2.2 App

In the app, we decided to implement only a simple display of the cloud data. In addition, the implementation of the "programming" is limited to a button that prompts the beacon to send or not send data.

2.2.3 Cloud

We decide not to implement user authentication. Apart from that, we decided to implement all requirements.

2.2.4 Common requirements

None of our parts have security mechanisms implemented. However, we decided to devote a separate chapter to the topic of security in this report (see chapter 3.7). We completely ignored the topic of CE certification.

3 Design

In this chapter, we describe and discuss the design of our solution. We discuss how and why we chose this design.

3.1 Choice of components

In this section, we will explore the components selected for each function and delve into the reasons behind our choices.

1. Microcontroller:

   We went for the STM32WL55 chipset and its evaluation board NUCLEO-WL55JC2 because it already has the Kinéis service built-in. Thanks to Kinéis' software stack, ST's existing STM32WL microcontrollers became compatible with the Kinéis system, reducing development cycles for new Kinéis-ready hardware developments.

   Also, we chose the STM32WL55JC2 microcontroller because it has strong processing power, built-in wireless communication options (LoRa, GFSK, BPSK), and efficient low-power modes. Its variety of peripherals and good support from the STM32 ecosystem make it a practical choice for our application.

2. BLE connectivity:

   The NUCLEO-WL55JC2 doesn't have built-in BLE, so we had to look for a ble chipset that we could integrate to the board. We went with the BlueNRG-2N expansion board because it's from STM32 Microelectronics, making it easy to integrate with our existing setup. Plus, it's a simple "plug-in" option, so we don't need to worry about soldering or complicated wiring.

3. Sensor:

   For our application, we opted for a temperature sensor due to its straightforward functionality. The chosen sensor is the DHT11, which serves as both a temperature and humidity sensor.

4. Antenna:

   For our demonstrator, due to time constraints, we retained the integrated antenna with the Nucleo board, specifically the ANT-SS450-510 from LPRS company. Nevertheless, we conducted a comparison among antennas to determine the most suitable option for our application. Finally, we chose the antenna ANT-418-HETH/HESM antenna from Linx Technologies for these reasons :

   - Size: The compact size of the antenna is suitable for applications with space constraints. Its dimensions (diameter of 8.89 mm and height of 38.1 mm) make it versatile for integration into our devic without occupying excessive space.
   - Price: Cost-effectiveness is a significant factor in component selection. The ANT-418-HETH/HESM antenna, priced at 1€, offers a balance between performance and affordability, making it a cost-efficient choice for the application.
   - Frequency Range: The specified frequency range (393–433MHz) aligns with our desired frequency 401 MHz.

- VSWR at 400MHz: The Voltage Standing Wave Ratio (VSWR) at 400MHz is a measure of how efficiently the antenna transmits power without reflecting it back. A VSWR of 1.4/1 indicates good impedance matching and efficient power transfer at the specified frequency.
- Satellite Reception:

  Kinéis rigorously tested the ANT-418-HETH/HESM antenna for satellite reception. The antenna, mounted on the Kinéis building's roof on a wooden support, underwent a series of evaluations. Connected to a common reference transmitter with 1W output power, LD-A2 modulation, and 60-second periodic transmissions, the test measured the reception of maximum-length messages (31 Bytes).

  The evaluation, lasting 8 to 10 days for each orientation, assessed the antenna's performance in various conditions. By calculating the received messages against expected messages during satellite visibility, the antenna scored 3/5, indicating a good performance level.

  This detailed testing, conducted by Kinéis, reaffirms the antenna's reliability in receiving satellite messages, bolstering our confidence in its suitability for our application.

5. Battery: For our project, we initially planned to use the STWLC86 wireless power transmitter and the STWBC68 wireless power receiver to make an inductive battery as their features were perfect for our needs.

   This battery is very powerful, it can provide up to 5W of output power, which is a lot for a device like our beacon. It's designed to work really well with wireless charging systems, and it follows the Qi standard.

   One of the advantages about the STWLC68 is that it's very flexible with its voltage. It can adjust anywhere between 3.6V to 20V, and you can fine-tune it in small steps. This meant we could have tailored the power supply to match exactly what our beacon needed, making it super efficient. Efficiency was an important requirement for us as we wanted our beacon to work for a long time without needing to recharge it.

   The STWLC68 is also designed with safety in mind. There are features included that protects it from overheating and other problems, which is necessary for a product like ours as the last thing you want is your battery causing issues.

   But, even with all these great features, we ended up not using these components in our project due to timing constraints. Ordering, testing, and making sure these batteries worked perfectly with our beacon would have taken a significant amount of time, and we had strict deadlines to meet for our project.

3.2 Hardware

We opted for a NUCLEO-WL55JC2 evaluation board, as it offers the Kinéis service already integrated, as well as an antenna. Although we didn't use it, this card also offers the LoRa service, which could prove useful for future work. On this board, we clipped a BlueNRG-2N card giving us access to Bluetooth Low Energy (BLE), which is necessary for communication with the mobile application. A DHT11 temperature and humidity

sensor has also been added to the board, enabling us to take temperature readings (as shown in figure 2)

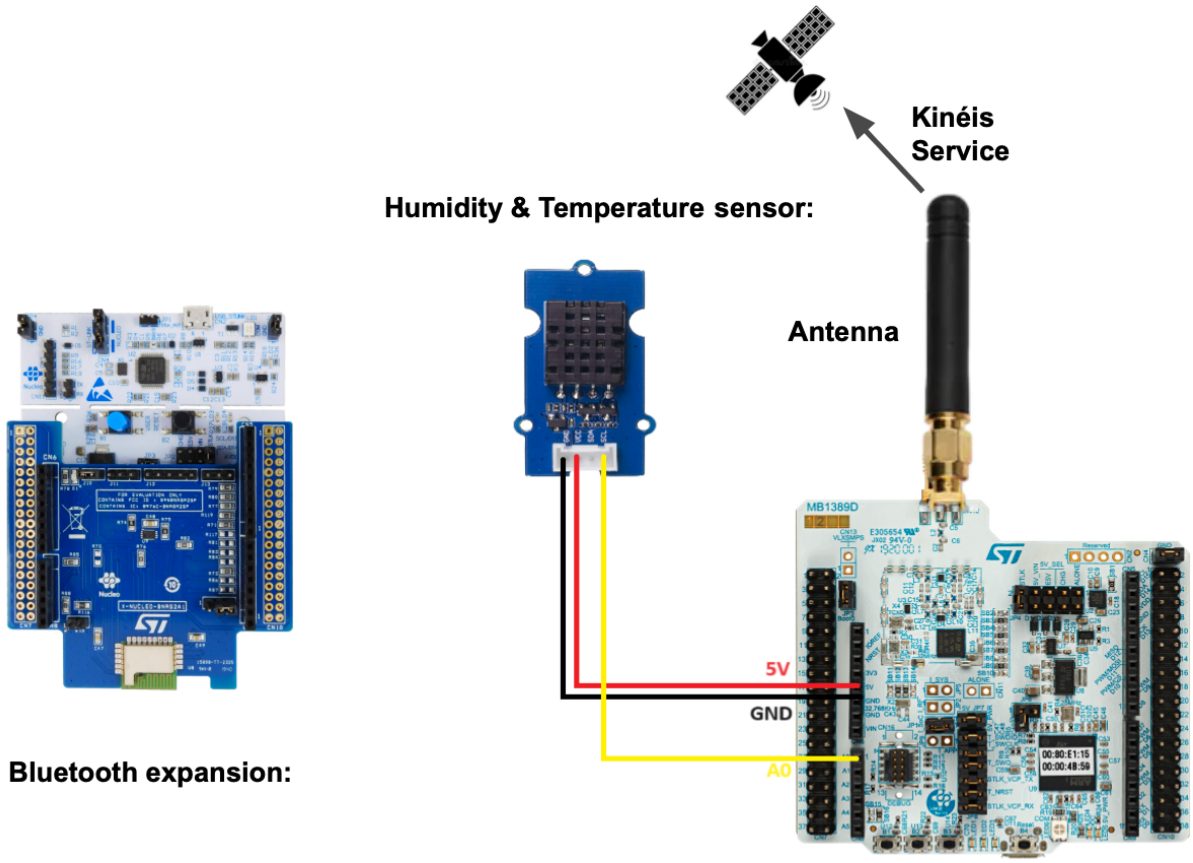


Figure 2: Hardware wiring diagram.

3.3 Cloud

The first step in designing the architecture was to decide on an architecture paradigm. Then we determined the technologies to be used to implement that paradigm. Once these decisions are made, the actual architecture can be defined. These decisions are discussed in the following chapter.

3.3.1 Microservices

We decided to implement a microservices architecture on the cloud side. This type of architecture has become a standard for larger backend architectures. The advantages are clear: Microservices can be deployed and scaled independently. This allows you to scale specific services that require more resources without having to scale the entire application. This makes it easier to efficiently handle changing workloads. In addition, microservices enable a more flexible, agile development process and align well with the principles of CD/DevOps practices. They allow for frequent and automated deployments, making it easier to deliver updates and enhancements to the application.

3.3.2 Technologies

Our three decision criteria for cloud technologies were Scalability, Agnosticism, and Popularity. Since we will be dealing with large amounts of data in the future (thousands of beacons), the technologies should allow for easy scalability. One of ACTIA's requirements was that the cloud be developed in an agnostic (i.e., platform-independent) manner. This reduces dependency on a single cloud provider. The popularity of a framework is also important to ensure long-term support for a technology and to attract skilled staff. We mainly decided to use two technologies in the cloud. Firstly, Springboot to implement the microservices and Redis as a database. One technology that we did not plan to use, but which turned out to be useful in retrospect, is mqtt (more on this in Chapter 5.9).

Springboot As a web server technology, Springboot scores very high on all three criteria. It is incredibly scalable. It can be hosted by any major cloud provider (Amazon, Google, Microsoft) with a few clicks. And it is one of the most popular Java frameworks used by many companies. Java is one of the most popular programming languages.

Redis Like Springboot, Redis scores very high on all three criteria. Redis is one of the fastest and most efficient databases available. Since we don't need to model complex relationships between data, a NoSQL database like this is a good choice. Also, like Springboot, Redis is open source and can be hosted by any major cloud provider.

3.3.3 Architecture

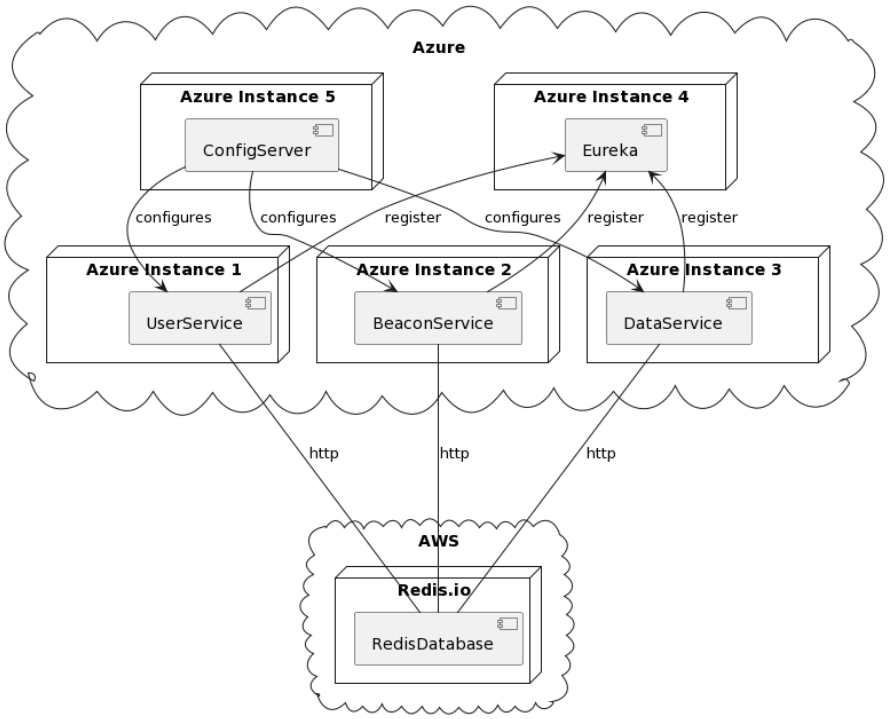


Figure 3: The Architecture of the Cloud.

Figure 3 illustrates our proposed architecture. It consists of 3 microservices as well as a configuration service and a discovery service. The three microservices store their data in

a Redis database. Each microservice can communicate with the database to modify, add, or delete data in the form of an API. The first microservice (BeaconService) manages beacons, allowing you to add, delete, or simply view the beacons owned by the customer. The second microservice (UserService) does the same thing, but with clients, allowing them to connect to the application. Finally, the DataService manages the data collected by the beacons and their locations.

3.4 Kinéis

Kinéis has collaborated with ST to facilitate widespread access to satellite IoT connectivity, employing the STM32WL55 chip. The STM32WL55, a wireless microcontroller unit (MCU), incorporates a sub-Giga RF transceiver, enabling the integration of both application code and the Kinéis stack on a single component.

Utilizing the Kinéis service, we implemented a system to transmit sensor data to satellites every half an hour. Leveraging the Cloud architecture we designed, we can efficiently retrieve sensor data, GPS localization, and timestamps. This information is then stored in our Redis database.

The figure 4 illustrates the communication between the Beacon and the Kinéis satellites.

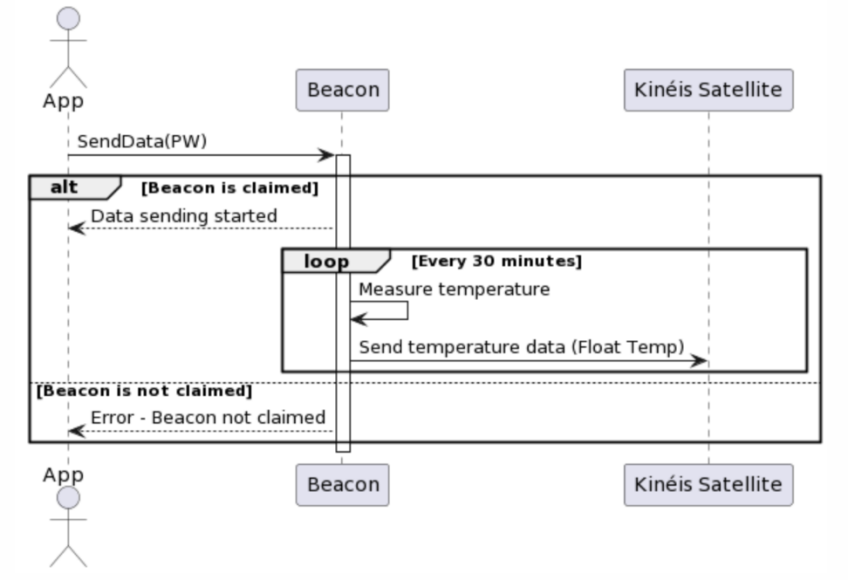


Figure 4: Sequence diagram illustrating the communication between the Beacon and the Kinéis Satellites.

3.5 Embedded Software

In our beacon project, we faced the challenge of enabling Bluetooth Low Energy connectivity. Our choice of microcontroller, the NUCLEO-WL55JC2, did not have built-in BLE capabilities, which led us to explore external solutions. We chose the BlueNRG-2N, an expansion board from STM, known for its reliable BLE performance.

The design process involved using software tools like STM32CubeMX and STM32CubeIDE. STM32CubeMX was instrumental in the initial configuration, helping us map out the re-

quired peripherals and middleware for BLE functionality. STM32CubeIDE provided an integrated development environment to write, debug, and compile our code, ensuring a smooth workflow from design to implementation.

This sequence diagram illustrates the communication flow between the beacon and the app. The process begins with the end-user connecting to the beacon via BLE. Once connected, the user can claim the beacon by requesting a unique ID, which is generated and sent back alongside a password created by the cloud service. This two-step verification ensures that the beacon is securely paired with the user's app.

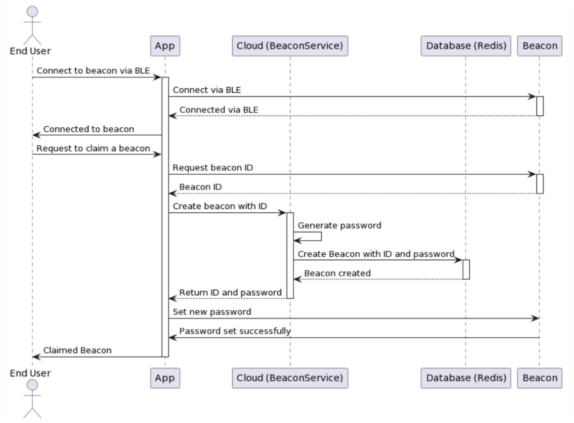


Figure 5: Process of connecting to the beacon via BLE

Throughout the design process, we ensured that the Bluetooth communication between the beacon and the app was not only functional but also secure, efficient, and user-friendly.

3.6 App

3.6.1 Technology choice

Before we started designing the application, we had to think about the technology we could use to meet our needs, particularly in terms of compatibility between different platforms. We hesitated between Cordova and Flutter, but in the end we opted for the latter, which offers native performance, which is not the case with Cordova. Moreover, Cordova uses Web technologies, whereas our application may not have access to the Internet when a fisherman is at sea. So Flutter seemed the most promising technology for our needs.

3.6.2 Flutter

Flutter is a software development kit for developing mobile applications. It's based on the Dart programming language, and you can use Android Studio, Visual Studio Code or Xcode to develop your application. We chose to develop our application using Visual Studio Code because we're comfortable with this software, it's easy to use and available on a Windows computer (which is not the case with Xcode, for example).

3.6.3 Interfaces

To meet the need for each customer to be able to see and locate the beacons they own, we chose to design a sign up and sign in interface (interface 1). When a customer logs on to

their account, they will be able to access the different beacons they own (interface 2) and the map showing their location (interface 4). The user must also be able to register a new beacon if it is not already registered on the application: this is represented in interface 5. This interface must use Bluetooth communication in order to be able to communicate with the Beacon and exchange information during registration. Once registration is complete, the user must be able to configure the beacon and tell it when to start transmitting data. This is done using various buttons on interface 3. The last interface is used to view the data from the user's various beacons after they have been processed by the Cloud.

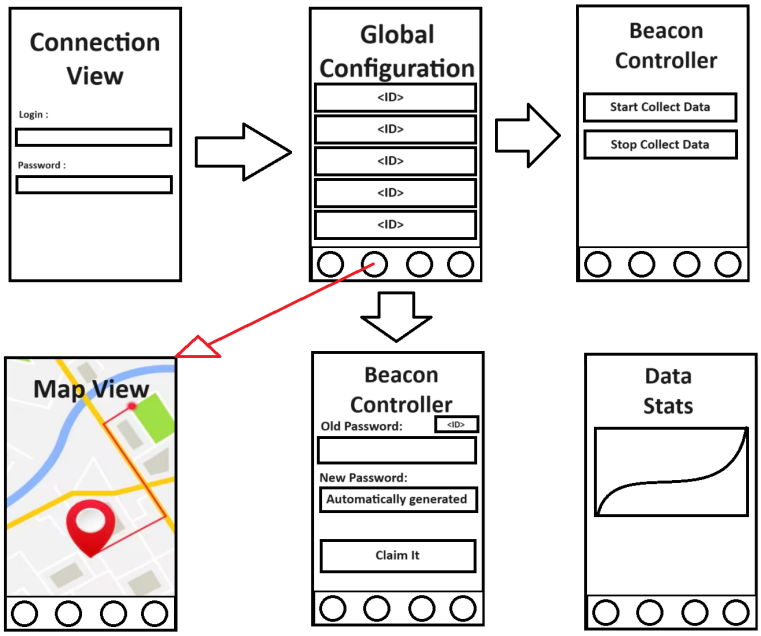


Figure 6: Design diagram for the various application interfaces.

3.7 Security

For our project, we used the EBIOS Risk Manager methodology for managing and assessing information on security risks. Developed by ANSSI, the French National Agency for the Security of Information Systems, EBIOS is a framework designed for a structured approach to cybersecurity. It helps with understanding the specific security needs, identifying threats and vulnerabilities, and formulating security objectives. The methodology involves several key steps, including establishing the context of the project, identifying potential risks to digital assets, analyzing the likelihood and impact of these risks, developing strategies for risk mitigation, and implementing a plan for monitoring the effectiveness of these strategies over time. Using this approach was crucial in ensuring the security of our system. Though time constraints prevented the implementation of cyber attack countermeasures, we thoroughly conceptualized them.

Context Establishment In the context of our project, focusing on the use-case where its goal is to help fishermen track their nets, we evaluated the environment in which the beacon would operate. This included understanding the typical usage scenarios, the technical infrastructure (from beacon to server to user interface), and the potential vulnerabilities in such a setup. Our target users were fishermen who required a reliable and secure system for tracking their nets in open waters.

Risk Identification We identified several risks across various aspects and connections. Physical risks included the beacon being damaged by harsh marine elements or being stolen. Cyber risks involved unauthorized access to the data transmitted by the beacon and potential data breaches, while operational risks encompassed the possibility of the beacon or user interface malfunctioning, leading to inaccurate or complete loss of tracking. Specifically, for connections, we considered C1 (Beacon to Kinéis) and C2 (Kinéis to Cloud) to be secure, as they are managed by Kinéis. The same security assumption applied to C3 (Beacon to App), as the BLE connection is made in a safe environment. However, connections between the Cloud and the App required careful examination for potential vulnerabilities, particularly in terms of data privacy and integrity. The security of personal devices used to access the app was outside our control, introducing additional challenges.

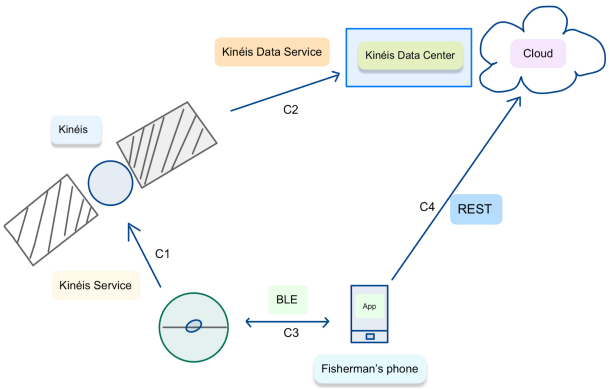


Figure 7: Connections between the Beacon, the App, Kinéis and the Cloud

Risk Analysis and Evaluation Our analysis prioritized risks based on their likelihood and potential impact. For example, data breaches and device malfunction were considered high-priority risks due to their potential to compromise the system's integrity and reliability. The impact analysis included the implications of these risks on the fishermen's operations, trust in our system, and associated financial costs.

Risk Treatment To mitigate these risks, we envisioned on several strategies. These included using encrypted communication protocols to secure data transmission, implementing robust authentication processes to secure access to the server and app, and designing the beacon with durable materials to withstand marine conditions. We also planned for regular software updates to address any newfound vulnerabilities.

Action Plan The action plan will involve rigorous testing and regular security audits to ensure the effectiveness of our security measures. Real-time monitoring of the beacon's performance will be set up to quickly detect and respond to any functional anomalies. Furthermore, educating the fishermen on secure usage practices also seemed essential to ensure the overall security of the system.

3.8 C1

Since Kinéis manages this connection, we considered it secure for our application.

3.9 C2

To secure C2, we would implement user authentication using Spring Security. This is a Spring Boot library that provides out-of-the-box authentication methods. We also need to configure https. To do this, we will need to purchase an SSL certificate for each service. A second attack vector is the communication between the services. This communication must also be encrypted using ssl. The communication with redis is already encrypted over ssl because redis.io provides this natively.

3.10 C3

For improving the security of connection C3, which is the Bluetooth connection between the beacon and the app, we are contemplating the integration of robust encryption and secure pairing protocols. Future developments include implementing AES encryption to shield the data flow and enforcing a one-time mutual authentication system. This authentication will ensure that only authorized users can claim a beacon. Once a beacon is claimed, it will be locked to that user, preventing subsequent connections from other devices. This measure will not only enhance security but also preserve the integrity of the user-beacon relationship, making the system more resilient against unauthorized access attempts.

3.11 C4

The connection between the mobile application issuing POST requests and the Cloud that orchestrates microservices needs to be secured in order to guarantee optimum security. Otherwise, Man-in-the-Middle attacks or injections could be launched.

To ensure that this connection is secure, it is essential to introduce an encrypted communication protocol such as Transport Layer Security (TLS). The best way to do this is to use HTTPS, which already ensures confidentiality, integrity and authentication of the server.

It is also possible to add a robust authentication system such as the use of access tokens with OAuth 2.0, which guarantees exclusive access to resources.

In this way, the connection between the application and the Cloud becomes secure because the connection is encrypted and there is an authentication and access control system.

4 Implementation

In this chapter wedescribe if and how we managed to meet the requirements.

4.1 Hardware

The development environment that was used is STM32CubeIDE that is a comprehensive and user-friendly IDE that incorporates various tools necessary for embedded system development, such as code editor, compiler, debugger, and graphical configuration tools.

4.1.1 Temperature Sensor

DHT11 digital temperature and humidity sensor is a composite Sensor that contains a calibrated digital signal output of the temperature and humidity. In our project, we will be using it only as a temperature sensor.

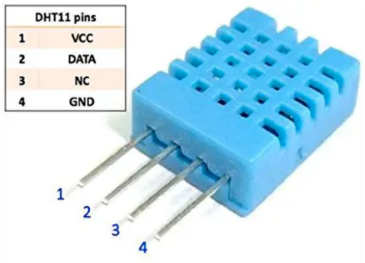


Figure 8: DHT11 sensor.

.
The pins used in out nucléo-board are :

- DATA (SIG) : pin PB2 (GPIOB,PIN2)
- VCC : 5V or 3V3
- GND : GND
- NC : not connected

Now, let's see the principle of operation of the DHT11 sensor.

- Initialization :

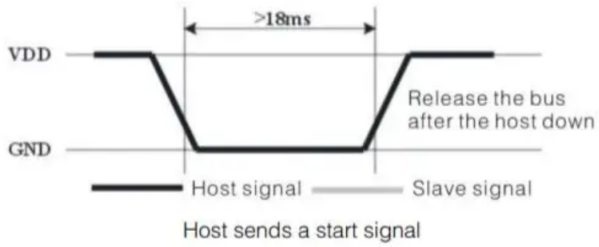


Figure 9: DHT11 host signal.

.

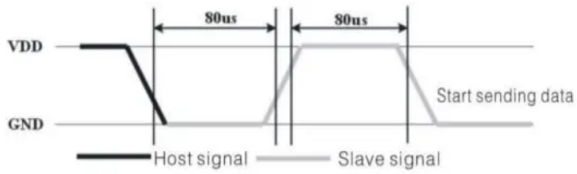


Figure 10: DHT11 slave signal.

. As shown above, in order to initialize the sensor, we have to :

1. Pull the data line LOW for around 18 ms.
2. After this, the DHT11 will pull the line LOW for 80 µs and then HIGH for 80 µs.
3. Once this is done, the sensor will be initialized and start transmitting.

To initialize the sensor, the steps are then as follows:

1. Set the pin (data) as output
2. Pull the pin low for 18ms
3. Release the pin by setting it as input

- Response : In order to indicate it's presence, after receiving the start signal, DHT11 will send a response signal. To do so, it will pull the data line low for 80 us, and than high for another 80 us. To read this response, we will do the following

  1. Wait for 40 us
  2. Read the pin, it must be low at this point
  3. wait for 80 us
  4. Read the pin, this time it should be HIGH

  If the above conditions are satisfied, that means the sensor is present, and we can proceed with reading the data.

- DATA Transmission:

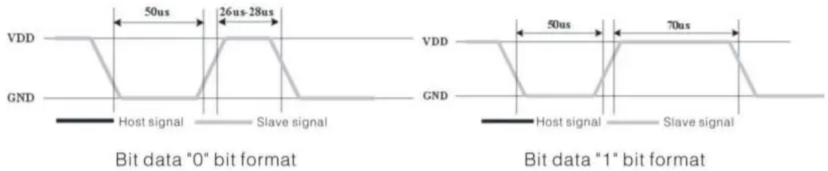


Figure 11: DHT11 data transmission.

.

Now DHT11 will send 40 bits of data. Each bit's transmission begins with low-voltage-level that last 50 us, the following high-voltage-level signal's length decides whether the bit is "1" or "0".

- If the length of high-voltage-level is around 26-28 us, the bit is “0”
- And if the length is around 70 us, than the bit is “1”

The 40 bits sent by DHT11 are as follows :

DATA = 8 bit integral RH data + 8 bit decimal RH data + 8 bit integral T data+8 bit decimal T data + 8 bit checksum

If the data transmission is right, check-sum should be the last 8 bit of “8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data”

Finally, these are the steps to READ DATA from the sensor

1. Wait for the pin to go high
2. Wait for 40us. This is because the length of “0” bit is 26-28us, and if the pin is high after 40us, it indicates that the bit is “1”
3. write the respective values to the variable

Below, some insight into the code:

- Initialization :

```
void DHT11_Start (void)
{
    Set_Pin_Output (DHT11_PORT, DHT11_PIN);  // set the pin as output
    HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);   // pull the pin low
    delay (18000);   // wait for 18ms
    Set_Pin_Input(DHT11_PORT, DHT11_PIN);    // set as input
}
```

Figure 12: DHT11 initialization function.

.

- Response :

```
uint8_t Check_Response (void)
{
    uint8_t Response = 0;
    delay (40);
    if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
    {
        delay (80);
        if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
        else Response = -1;
    }
    while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));   // wait for the pin to go lo

    return Response;
}
```

Figure 13: DHT11 response function.

.

- Response :

```
uint8_t DHT11_Read (void)
{
    uint8_t i,j;
    for (j=0;j<8;j++)
    {
        while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));   // wait for the pin to
        delay (40);   // wait for 40 us
        if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
        {
            i&= ~(1<<(7-j));   // write 0
        }
        else i|= (1<<(7-j));  // if the pin is high, write 1
        while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));  // wait for the pin to go
    }
    return i;
}
```

Figure 14: DHT11 read data function.

.

4.2 Cloud

In this section, we will explain how we implemented the design described in chapter xxx regarding the cloud component.

4.2.1 Database

As discussed in chapter 3.3.2, we decided to use Redis as our database. In our deployment, we deployed the database via Redis.io, which offers a free tier. We deployed it with the Redis Stack package and in version 6.2.6. We also installed the advanced capabilites. Figure 15 shows our full configuration.

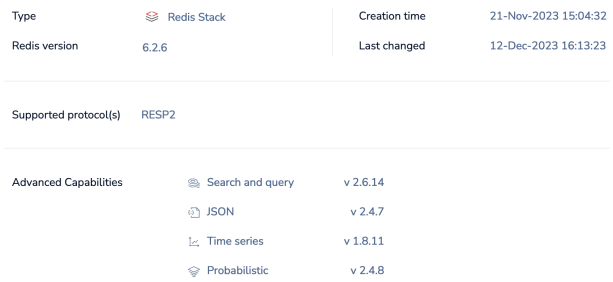


Figure 15: The configuration of the Redis Database on Redis.io

4.2.2 All Micro Services

As discussed in chapter 3.3.2 we decided to use Spring Boot (Version 3.1.5) for all microservices as well as Java 17. There are also some dependencies that we used for all services.

- spring-boot-starter-web
- spring-cloud
- netflix-eureka-client
- jedis
- maven-plugin

More details can be found in the respective implementation of the services (pom.xml).

4.2.3 User Service

We have implemented a standard Spring Boot Rest API. The following APIs are implemented.

- **getAllUsers()** Returns all users
- **getUserByEmail(String email)** Returns a user record for a given email.
- **addUser(User user)** Adds a user to the database.

4.2.4 Beacon Service

We have implemented a standard Spring Boot Rest API. The following APIs are implemented.

- **createBeacon(int beaconId** Adds a beacon to the database.
- **getBeacon(int beaconId)** Returns a beacon record for a given id.
- **getAllBeacons()** Returns all beacons
- **getAllBeaconsByUser(String userId)** Returns all beacon records for a given user.
- **deleteBeacon(int beaconId)** Deletes a beacon record for a given id.

4.2.5 Data Service

We have implemented a standard Spring Boot Rest API. The following APIs have been implemented. As discussed in chapter xxx, the post APIs are temporary and should be switched to mqtt as it is the best practice.

- **saveData(DeviceData deviceData)** Is used by the Kineis service to post the data sent by the beacon.
- **getDataPoints(int beaconid, long startDate, long endDate)** Returns all data records for a given beacon in a given time interval (dates are in Unix format).
- **saveLocation(KineisLocation kineisLocation)** Is used by the Kineis service to post the location of the beacon calculated by Kineis.
- **getLocations(int beaconid, long startDate, long endDate)** Returns all location records for a given beacon in a given time interval (dates are in Unix format).

4.2.6 Kineis Service Portal

In figure 16 and 17 you can see the configuration inside the Kineis Service Portal. This configuration is important. An incorrectly configured service portal will not be able to send data to the DataService. For the send data we decided to use the "Processed Type Expert" format because the processing of Kineis includes error correction. For the location data we decided to use the "Location Type Kineis format V2". This is the latest standard and best practice.

Name	Active	Device Group	Distribution Format	Reference Format	Distribution Endpoint	Delivery Protocol
Data Service Data	☑	ALL	DATA	Processed type Expert	DATA	HTTP
Data Service Location	☑	ALL	LOCATION	Location type Kineis format V2	LOCATION	HTTP

Figure 16: The configuration of the two Distributions in the Kineis Service Portal.

Device Address ↑	Device ID	Device Name	Tags	Device Groups	Distributions	App. Codec
4B56E4C	240020			All Devices	Data Service Data, Data Service Location	KIMV1

Figure 17: The configuration of the Devices in the Kineis Service Portal.

4.2.7 Deployment of the Services

The Spring Boot services we have deployed in Azure. We used Azure Spring Boot to easily deploy the services. For each service, we created an application. We also configured Github actions that automatically deploy the services to their respective Azure Spring Applications. We used Azure because of the free tier for students which offers 100 Dollar free. In figure 18 you can see the money spent on the cloud instance, we stopped the instance sometimes to save money, but if we would run it full time it would cost approximately 150 Dollars per month.

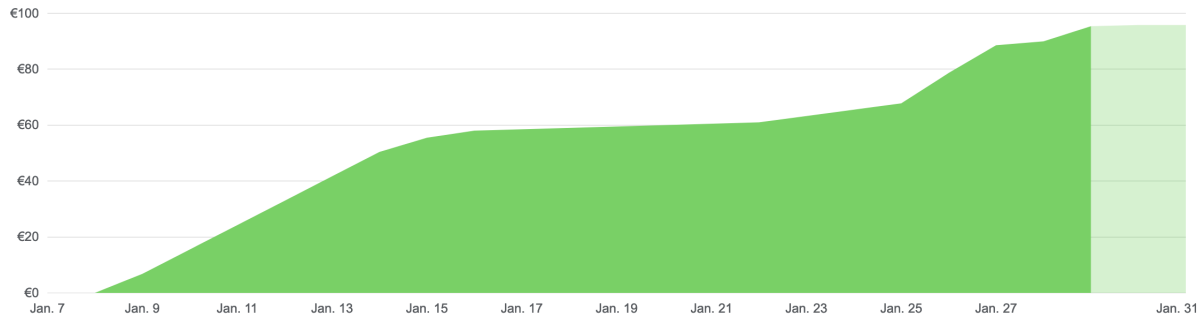


Figure 18: The cost of the deployed Azure Services.

4.2.8 Testing

For the final test of the cloud, we placed the beacon outside overnight and let it send dummy data every 10 minutes. During the test, the data was successfully received by Kineis satellites and also successfully posted to the cloud and stored in the redis database. The other APIs were tested via Postman and also worked fine.

4.3 Kinéis

To benefit from the Kinéis services we had to follow The SmartSat program, this program includes :

- Kinéis Stack : Full documentation to help implement Kinéis connectivity on the components.

- Unlimited connectivity : The SmartSat program gives unlimited access to satellite connectivity for 3 months: location, data collection, alerts...
- Technical support : A ticketing tool is available to contact the kinéis teams as well as a privileged access to the knowledge base for all the necessary information.
- Web interface : To visualize the data, Kinéis provides its customers with a web interface.

The first thing we had to do is getting the credentials and register our board, to do so we followed the procedure as explained in their website. After navigating through some paperwork, including document signing we managed to get the credentials and eventually access the knowledge base. Then, we received a code application to test the connection to the satellites. The modulation used in the code package is VLD-A4 modulation for Low Power short messages service(3-Bytes messages). This modulation is compatible with the ARGOS-4 (Azelle (CS) and OceanSat-3 (OC)) as well as ARGOS-Neo (ANGELS (A1))

At the beginning, we faced some difficulties to visualize the data sent in the kinéis services portal. In fact, we had to be in an open space with a clear view to the sky and transmit during the time passes of the satellites. Following this logic, it was hard to transmit at the right time so after some discussions with the Kinéis hotline and our professor we decided to leave the device outdoors for at least 24h.

After that, we managed to receive the transmitted data and visualize them in the kinéis services portal (as shown in figure 19)

Figure 19: Kinéis services portal.

.

When we checked the satellite time passes against our received data, we found that Argosweb wasn't always accurate. Some satellites didn't pick up our messages as expected. This inconsistency posed a challenge in aligning our data transmission with the actual satellite passes.

4.4 Embedded Software

Configuration and Initial Setup: Our initial configuration was done using STM32CubeMX, where we mapped the pinout and peripherals for the NUCLEO-WL55JC2 microcontroller, ensuring the inclusion of the BlueNRG-2N module for BLE functionality.

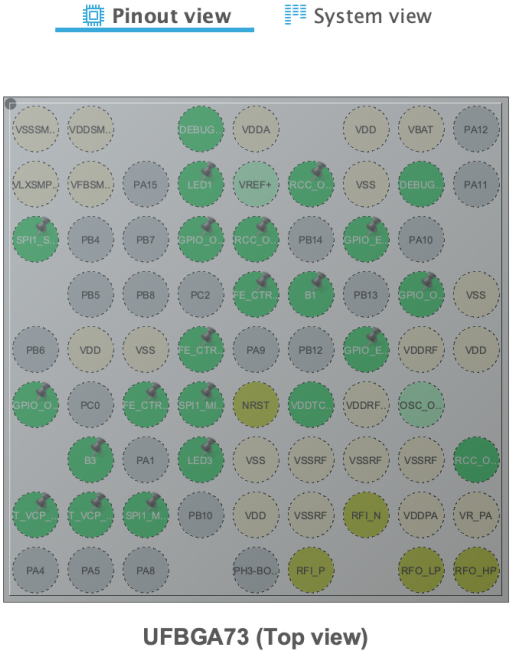
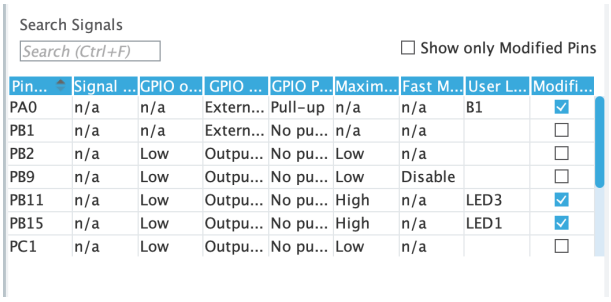


Figure 20: Screenshot of the STM32CubeMX pinout view showing the selected pins and peripherals.



Pin...	Signal ...	GPIO o...	GPIO ...	GPIO P...	Maxim...	Fast M...	User L...	Modifi...
PA0	n/a	n/a	Extern...	Pull-up	n/a	n/a	B1	☑
PB1	n/a	n/a	Extern...	No pu...	n/a	n/a		☐
PB2	n/a	Low	Outpu...	No pu...	Low	n/a		☐
PB9	n/a	Low	Outpu...	No pu...	Low	Disable		☐
PB11	n/a	Low	Outpu...	No pu...	High	n/a	LED3	☑
PB15	n/a	Low	Outpu...	No pu...	High	n/a	LED1	☑
PC1	n/a	Low	Outpu...	No pu...	Low	n/a		☐

Figure 21: Screenshot of the STM32CubeMX showing the configuration of the pins.

Clock Configuration: We meticulously configured the clock settings to ensure optimal power management and system performance. The clock configuration was critical to the beacon's operation, balancing power consumption with the required processing speed.

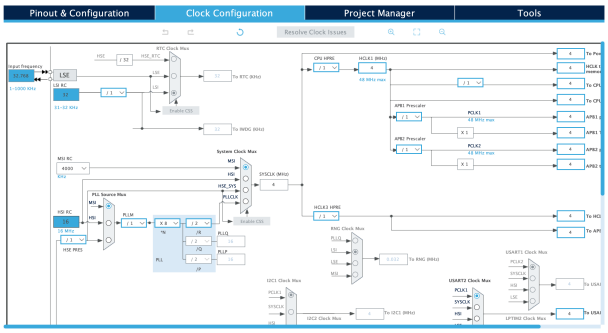
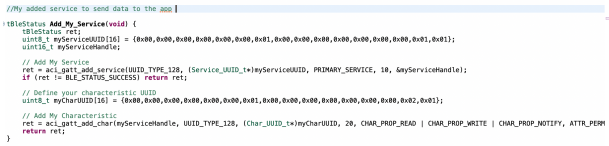


Figure 22: Screenshot of the clock configuration in STM32CubeMX, detailing the clock sources and frequencies set for the system.

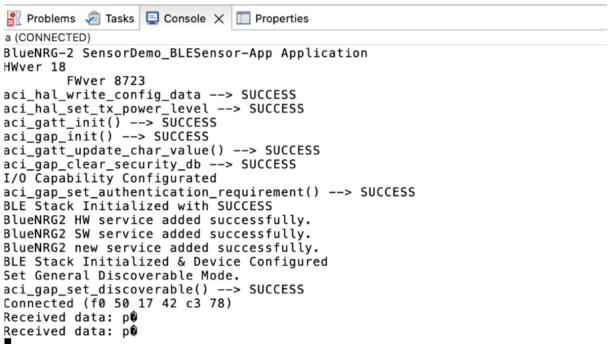
Software Development and Custom Service Integration: Transitioning to STM32CubeIDE, we focused on software development. We removed unnecessary default services and introduced our custom service for data transmission between the beacon and the app. This step was crucial in tailoring the beacon's functionality to our project needs.



```
//My added service to send data to the app

tBleStatus Add_My_Service(void) {
    tBleStatus ret;
    uint8_t myServiceUUID[16] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01};
    uint16_t myServiceHandle;

    // Add My Service
    ret = aci_gatt_add_service(UUID_TYPE_128, (Service_UUID_t*)myServiceUUID, PRIMARY_SERVICE, 10, &myServiceHandle);
    if (ret != BLE_STATUS_SUCCESS) return ret;

    // Define your characteristic UUID
    uint8_t myCharUUID[16] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,0x01};

    // Add My Characteristic
    ret = aci_gatt_add_char(myServiceHandle, UUID_TYPE_128, (Char_UUID_t*)myCharUUID, 20, CHAR_PROP_READ | CHAR_PROP_WRITE | CHAR_PROP_NOTIFY, ATTR_PERM
    return ret;
}
```

Figure 23: Code snippet from STM32CubeIDE showing the function AddMyService() and its associated UUIDs.

Testing and Validation: In STM32CubeIDE, rigorous testing was conducted. We debugged the BLE communication, validated the custom service functionality, and confirmed successful data transmission to the app.



```
Problems  Tasks  Console  Properties
a (CONNECTED)
BlueNRG-2 SensorDemo_BLESensor-App Application
HWver 18
        FWver 8723
aci_hal_write_config_data --> SUCCESS
aci_hal_set_tx_power_level --> SUCCESS
aci_gatt_init() --> SUCCESS
aci_gap_init() --> SUCCESS
aci_gatt_update_char_value() --> SUCCESS
aci_gap_clear_security_db --> SUCCESS
I/O Capability Configurated
aci_gap_set_authentication_requirement() --> SUCCESS
BLE Stack Initialized with SUCCESS
BlueNRG2 HW service added successfully.
BlueNRG2 SW service added successfully.
BlueNRG2 new service added successfully.
BLE Stack Initialized & Device Configured
Set General Discoverable Mode.
aci_gap_set_discoverable() --> SUCCESS
Connected (f0 50 17 42 c3 78)
Received data: p0
Received data: p0
```

Figure 24: Code snippet showing a test output from the beacon, with the successful connexion and the received data printed for verification.

4.5 APP

We're now going to talk about the different interfaces implemented in the application.

4.5.1 Login interface

This interface allows users to log in to their account. The login interface consists of two fields and a button. The two fields correspond to the email address and password, while the button is used to establish the connection once the fields have been filled in. When the connection is established, the asynchronous login() function is called, retrieving the contents of the two fields and making a POST request across the Internet to query the UserService microservice on the Cloud. The service then returns a response and the login() function issues an error if the connection information is incorrect. Otherwise, if they are correct, it sends the user back to the Beacons interface.

Two other buttons are also available on this interface (the forgotten password button and the register button), but they are not implemented and nothing happens when they are pressed.

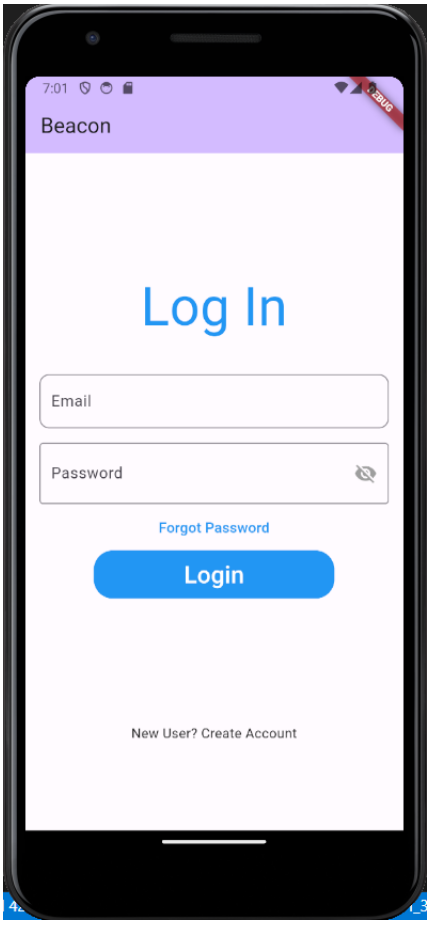


Figure 25: Representation of the connection interface

.

4.5.2 Beacons interface

This interface allows the user to view the beacons he owns. This is represented in the form of a list formed using the builder() function of the ListView class. Each item displays the beacon ID and password. The beacons are loaded using the fetchBeacons() function, which executes a POST request to query the BeaconService microservice on the Cloud.

This request takes the user ID as a parameter. The request returns a list of the user's beacons in JSON format, which is then converted into a list in the application.

If a beacon is not present in the application, you can click on the + button at the top right of the interface, which redirects you to the Bluetooth interface.

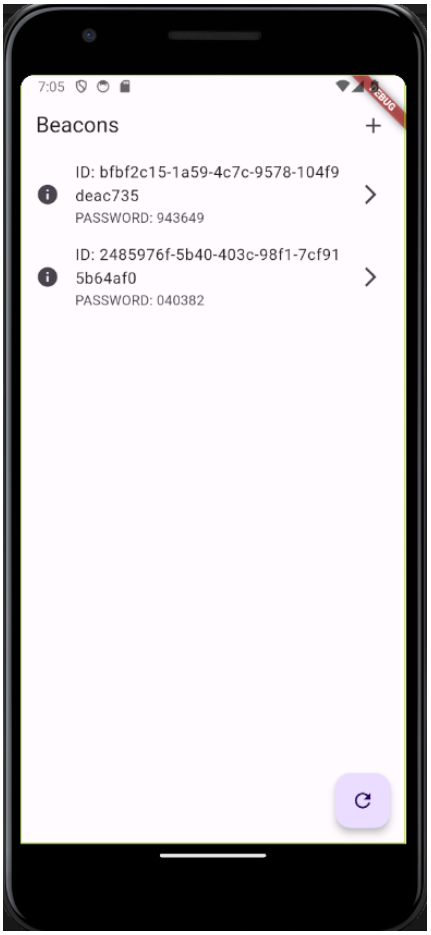


Figure 26: Representation of the beacon interface

.

4.5.3 Bluetooth interface

This interface provides access to all the beacons present in the environment and close to the user. Users can add the Beacon they own simply by grabbing the device. Like the Beacons interface, the Bluetooth interface takes the form of a list. Each item has the name of the device, its MAC address and a button that changes state depending on whether you are connected to the device or not. If you are not connected, the "connect" button appears; if you are already connected, the "open" button appears. When you click on the "open" button, another interface appears for managing the device. This includes the device's Bluetooth services. The service we're interested in here is MyService, which sends the "StartCollectData" command to our beacon to tell it to start collecting data.

Other commands such as "StopCollectData" have not yet been implemented.

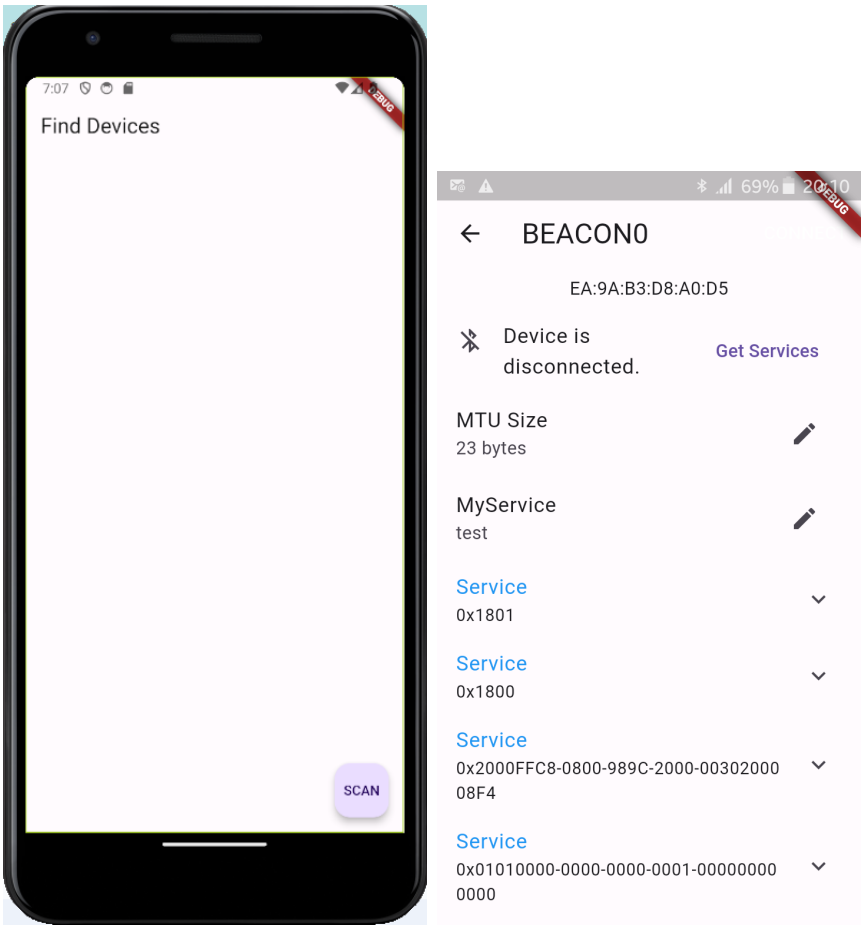


Figure 27: Bluetooth connection and features interface representation

.

4.5.4 Other interfaces

The interfaces for signing up, viewing beacons on a map or viewing beacon statistics have not been implemented.

4.6 Security

We did not implement security because of time constraints. Our design is explained in chapter 3.

4.7 Energy Consumption

Due to time constraints preventing the integration of all components simultaneously, we were unable to measure the collective current consumption. However, here are the individual current consumption values for each component, initially in run mode and subsequently in sleep mode : :

Component	Voltage Scaling	Idd Typ (25°)	Idd Max (25°)
stm32wl55cc	Range 1	5.55 mA	7.40 mA
	SMPS Range 1	3.40 mA	-
bluenrg-2n	-	1.9 mA	-
DHT11	-	0.5 mA	1.1 mA

Table 1: Power Consumption Characteristics in Run mode

Component	Voltage Scaling	Idd Typ (25°)	Idd Max (25°)
stm32wl55cc	Range 1	1.70 mA	2.20 mA
	SMPS Range 1	1.35 mA	-
bluenrg-2n	-	0.9 mA	-

Table 2: Power Consumption Characteristics in sleep mode

5 Future Work

In future iterations of our beacon project, several key developments were envisioned.

5.1 BMS and inductive charging

First, the integration of an inductive battery was planned, but time constraints prevented its implementation. We aimed to explore and test suitable inductive charging technologies for this purpose. This will significantly improve the device's energy efficiency and user experience.

5.2 Additional Sensors

Additionally, we also plan to expand the beacon's sensory capabilities beyond temperature measurement. Integrating additional sensors like humidity, light, and motion detectors will transform the beacon into a versatile tool for environmental monitoring, applicable in various IoT scenarios.

5.3 Antenna

A crucial technical improvement involved redesigning the beacon's antenna. Swapping the current stick antenna for a wired antenna could greatly improve the beacon's data transmission and reception. A wired antenna, unlike a stick antenna, doesn't rely on being upright for optimal performance, offering greater flexibility and efficiency in data communication. We already did a selection of the best wired antennas available but we couldn't find the time to order one and use it for our project.

5.4 Housing

Another important aspect was developing a robust, waterproof, and shock-resistant housing for the beacon. This development would ensure the device's durability in varied environmental conditions. Prototyping and testing different materials and designs were part of this plan.

5.5 BLE

The Bluetooth component is set for several enhancements. Future work would aim to fortify data security through advanced encryption methods, ensuring safer transmission between the beacon and the application. We'll also target energy efficiency, potentially leveraging Bluetooth 5 technologies to minimize power usage.

5.6 Security

A significant area for future development is the establishment of strong security protocols. These protocols will safeguard data transmission between the beacon, the cloud, and the application, ensuring user data privacy and security. This aspect will involve an extensive understanding and application of cyber-security principles specific to IoT devices.

5.7 User authentication and SSL

As explained in chapter 4.6, we have not implemented any security. In the future, the security mechanisms described in chapter 3.8 should be implemented.

5.8 Further Tests

Furthermore, overall testing and validation of these new features and improvements will be crucial. This process will cover both technical performance assessments and real-world durability tests, ensuring that the beacon is ready for diverse applications.

These future developments aim to enhance the beacon's functionality, ensuring its utility, adaptability, and security. Each proposed improvement will significantly contribute to the evolution of the beacon, making it a more versatile and reliable tool in the field of IoT.

5.9 mqtt

For posting data, Kineis recommends (in their documentation) using mqtt as the protocol. As discussed in chapter 4.2.6, we used http to post the data. A future improvement might be to use mqtt instead of http.

6 Conclusion

As we reach the end of our beacon project, it's been an incredible journey of growth and development. We managed to create a functional prototype, capable of transmitting temperature data and establishing Bluetooth and Satellite connections. Despite the limitations we faced, like time constraints that led us to prioritize certain features, we achieved a significant milestone in delivering a working prototype.

Throughout this project, we've gained not only technical expertise in areas like IoT, embedded systems, and cloud integration but also practical skills in navigating real-world challenges. We learned the importance of teamwork, effective communication, and the ability to adapt under pressure. These experiences have been invaluable, enhancing our problem-solving capabilities and our understanding of complex technology systems.

If we would've had more time, we would've planned to refine our beacon by adding more robust security measures and optimizing its energy efficiency. We also aim to enhance the hardware to make it more durable and adaptable to various environments.

Reflecting on this project, it's clear that it has been a crucial part of our educational journey, providing us with a solid foundation of skills and knowledge. These will be indispensable as we move forward in our careers. The project has equipped us to face future challenges with confidence.

A Appendix :

A.1 Requirements

Category	Reqt nr	Reqt text	Rationale	Product
Interfaces	REQ-010	APP shall communicate with CLOUD through any available internet communication (e.g. WIFI, xG...)		app
Interfaces	REQ-020	BEACON shall transmit and receive data to APP (via BLE or Induction)	Configuration data, ID data, update data	app; beacon
memory	REQ-030	BEACON shall include a safe UPDATE System (Integrity check in FLASH before updating, rolling updates...) and so a FIRMWARE memory space and dedicated data memory space (parameters...). Memory corruption has to be managed (e.g. check data after writing in FLASH, writing cycles to be analyzed to avoid damages). Current data to be transmitted (e.g. measurements) should be stored in RAM (writing cycles constraints).		app; beacon
Function	REQ-040	BEACON SW shall be configurable through APP (i.e. fields, BEACON ID). Several level of access should be available (super administrator ; administrator ; user). Each ID shall be unique. APP shall adress dedicated BEACON memory fields thanks to an HMI (examples : sensors measurement conversion, unique ID, parameters for ephemeris, periodic wake-up time, measurements periods, message size...). In order to keep message consistency with CLOUD, APP shall transmit configuration to CLOUD and additional costs, delay for storage has to be computed by CLOUD BEFORE updating BEACON.	Set up should be easily loadable.	app; beacon cloud
Function	REQ-050	BEACON shall provide time and date information with data transmitted to CLOUD (through SAT NETWORK) and should be set up through APP	For measurement datation	app; beacon; cloud

Category	Reqt nr	Reqt text	Rationale	Product
Function	REQ-060	APP Application shall be compatible with ANDROID, IOS, LINUX or WINDOWS	APP has to be hybrid (web site with custom access)	app; cloud
Function	REQ-070	APP Shall include several screens: - map view - global configuration view - beacon configuration view - data list and computation (e.G. average, graphics ...) - to be detailed ACCess and rights have to be set before coding in order to ensure modularity and adaptability Note : data shall be partitioned according to customers (don't share all data to all customers)	HMI has to be detailed according to possible use cases. Ux/UI Approach have to be considered.	app; cloud
Geolocalization	REQ-080	Each BEACON shall be geolocalized in APP/CLOUD	BEACON associated with unique ID	app; cloud
Function	REQ-090	BEACON shall provide ACK to APP or CLOUD when updated or if parameters are updated	to ensure correct reception	app; beacon; cloud
Function	REQ-100	All products shall be designed in order to be easily customized and modified	Design has to be performed in order to ensure modularity. Documentation is important to ensure continuity of project	app; beacon; cloud
Function	REQ-110	All products shall be designed in order to be cybersecure-compliant	e.G. ISO 27001	app; beacon; cloud

Category	Reqt nr	Reqt text	Rationale	Product
Function	REQ-120	APP main functions shall be stored on CLOUD (<=>website) and linked to CLOUD data. APP shall provide views, HMI, data processing. The platform part (on computer or smartphone, tablet...) shall allow to view data from CLOUD and to keep the last update recorded. The APP BEACON configuration tool shall execute locally (BLE) and this configuration shall be stored on CLOUD. APP shall be update to view selected BEACON current data (e.g.measurements) through BLE (ID selection and dedicated page).		app; beacon; cloud
Cost	REQ-130	BEACON cost should be inferior to 50€ for 100 parts. RC and manufacturing operations has to be optimized.	BOM cost to be estimated	app; beacon; cloud
Regulations	REQ-140	Products shall be CE compliant	Analysis only	app; beacon; cloud
Form factor	REQ-150	BEACON shall fit in one hand (electronics, antenna, board, Battery, housing).	For demonstrator, estimation will be sufficient. Technology has to be chosen and tested	beacon
Weight	REQ-160	BEACON shall weight less than 100grams, should weight less than 50g	For demonstrator, estimation will be sufficient.	beacon
Architecture	REQ-170	BEACON shall be based on a RTOS (TBC)	RTOS to be chosen and designed to ease evolutions	beacon
Spares	REQ-180	BEACON shall include Spares IO for further evolutions (ie. Sensors, data buses...)		beacon
Function	REQ-190	BEACON shall include sensors	Sensors choices to be analyzed against environmental constraints.	beacon

Category	Reqt nr	Reqt text	Rationale	Product
Architecture	REQ-200	BEACON shall be based on a STM32WL (TBC) and a KINEIS chipset (KIM-1 or KIM2 if available)	can be discussed	beacon
Life Duration	REQ-210	BEACON life duration shall be at least 5 years.		beacon
Environment	REQ-220	BEACON shall be waterproof (e.g.mechanics) and include protection against ground and sea environment (dust, thermal aspects, corrosion, sun).	Plastic hermetic enclosure, wire or printed internal antenna for example. Sensors choices to be analyzed. For demonstrator, estimation and materials choice may be sufficient	beacon
Performance	REQ-230	BEACON shall float with antenna pointing towards sky in order to ensure communication with satellite.	Weight allocation with battery and antenna to ensure correct communication from the sea. Mechanics should allow an easy positioning on gorund.	beacon
Power	REQ-240	BEACON consumption shall be minimzed to ensure battery optimization and at least 1 year autonomy.	Use of standby mode and low power consumption strategy, wake up mode on ephemeris.	beacon
Power	REQ-250	BEACON shall not include external connectors.	Use of standby mode and low power consumption strategy, wake up mode on ephemeris.	beacon

Category	Reqt nr	Reqt text	Rationale	Product
Power	REQ-260	BEACON shall be compliant with Qi induction charge standard	to avoid connectors and to be charged through Qi CHARGER.	beacon
Environment	REQ-270	BEACON shall work in a -[30 ; +60°C] environment	Components shall be selected to ensure derating	beacon
Environment	REQ-280	BEACON shall be storable in a -[40 ; +85°C] environment	Components shall be selected to ensure derating	beacon
Interfaces	REQ-290	BEACON shall transmit and receive (TBC) data to CLOUD through KINEIS satellite network	For demonstrator, simulation may be sufficient. ARGOS network TBC	beacon; cloud
Interfaces	REQ-300	BEACON shall include LORA connection option (between BEACONS)	for MESH network ability. Power consumption to be estimated, tests should be performed.	beacon; cloud
Function	REQ-310	CLOUD shall be based on a data center stored on clouds (like AWS...) and ideally should be independent from platform ("agnostic").		cloud
Function	REQ-320	CLOUD shall store data and manage access rights, payments, service access. CLOUD has to allow to process payments (data service, ordering beacons) in a secured way.		cloud