2024

IoT-Lab: Final Report

VAN OLFFEN VICTOR, SAHIB OUSSAMA, DUFOUR ARTHUR, JARDINET LOUIS

Table of Contents

1. Introduction

An efficient waste management system plays a major role in our modern urban living. To address this problem, our team developed a prototype as part of our IoT Lab course, given at ECAM in 1st year of master.

The goal of our prototype is to automate waste management in urban areas, using IoT sensors to remotely monitor the fill level and weight of bins. We aim to optimize waste collection, prevent overflow and improve the efficiency of collection services. The device should be powered by a battery and directly installed inside the bins.

Our prototype will use the LoRaWAN technology in its architecture with an Adafruit Feather M0 as well as sensors like ultrasonic. Since it is used to know the fill level of a bin, we called this project "**Garbage Filling Notifier**". It will communicate with The Things Network server using the Internet. Then, a user can use an application that will show details on the filling level of each bin.

This report will outline every step of the prototype's development.

2. Architecture

The architecture is represented on the Figure 1. It shows how the developer is sending the directives to the endpoint which then transmits the sensor's data to the TTN server using the Internet and LoRaWAN. It also shows the main components of the endpoint.

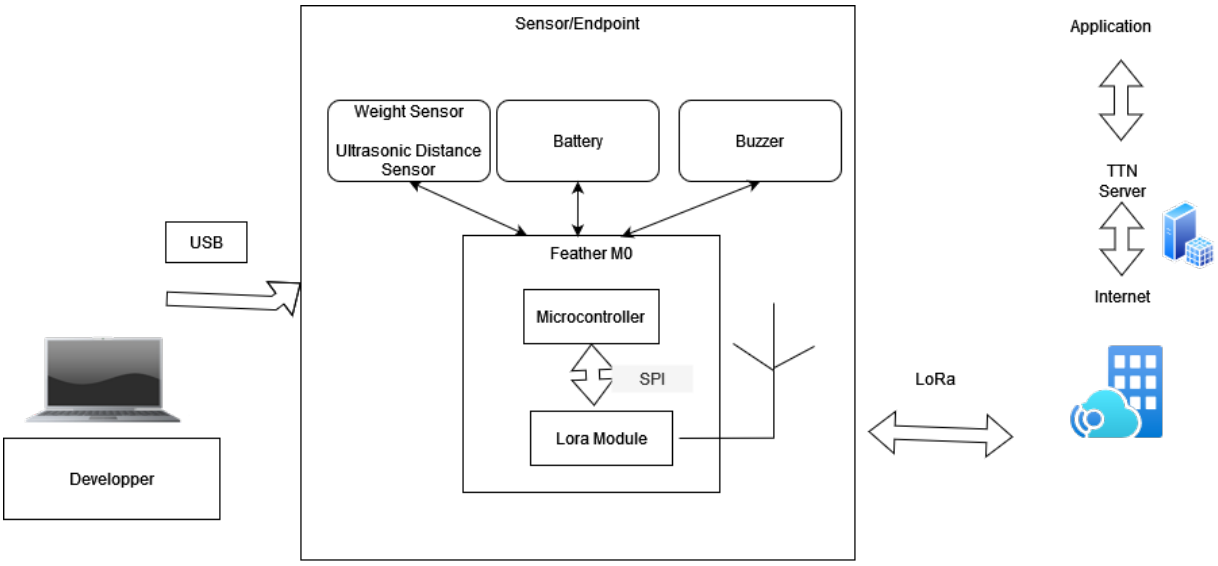

Figure 1: Garbage Filling Notifier project architecture

3. Bill of Materials

Once we established our project's architecture, we need to assemble it. Here is the Bill of Materials (BOM) of this project. All the components we need are already available at ECAM.

The ultrasound distance sensor will allow us to measure the bins' filling level depending on the cumulated trash's height. We will measure the distance between the top and the bottom of the bin. As the distance gets smaller, we can calculate in percentage, from 0 to 100%, how much volume of the bin is filled.

The Load cell is our weight sensor. We will use it to acknowledge the bin's weight so that we know when the maximum weight is reached [1]. This will also be shown in percentages on the application. It is useful to monitor both variables as a filled trash bin with low weight could cause the trash to overflow and a bin with only heavy objects could tear the trash bag apart.

Once the bin's maximum weight or volume is reached, the buzzer will get activated.

Component	**Quantity**	**Cost**
Adafruit feather M0 microcontroller	1	19,15€
HC-SR04 - Ultrasound distance sensor	1	3,9€
Custom Load cell	1	/
HX711 - Signal Amplifier for the load cell	1	2 €
Buzzer	1	6€
Battery 3.3V	1	12,90€
Breadboard	1	3,50€
Wires Breadboard	10	5€

4. Electronic Schematic

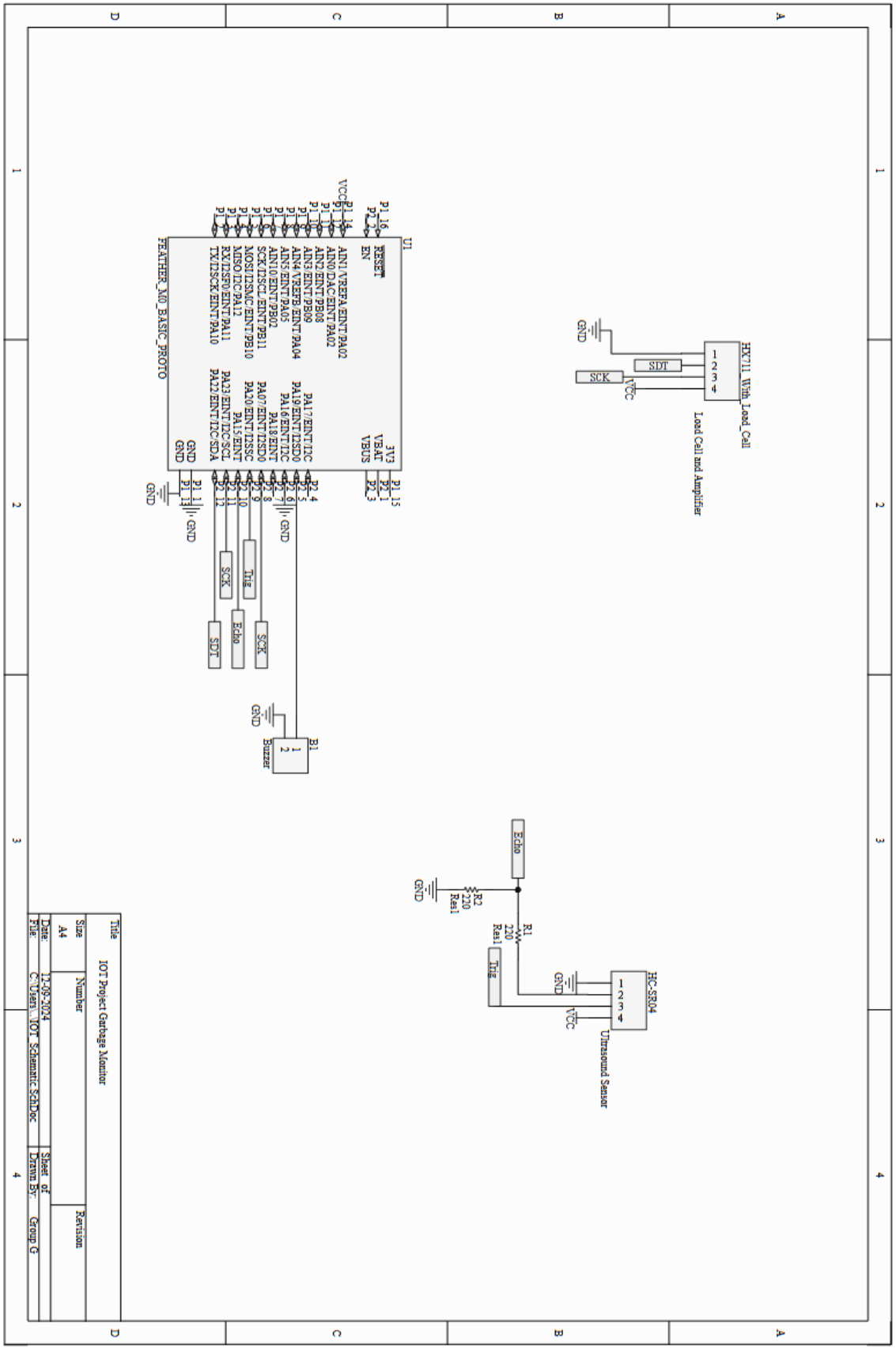

Figure 2: Electronic schematic of our prototype

5. The Things Network Configuration

The first step before sending data is to configure a TTN server [2]. To do so, we go and connect to the TTN's console webpage. Then, we create an application called "**Garbage.io**".

The next step is to register an end device that will be linked to our application. The device will be our Adafruit.

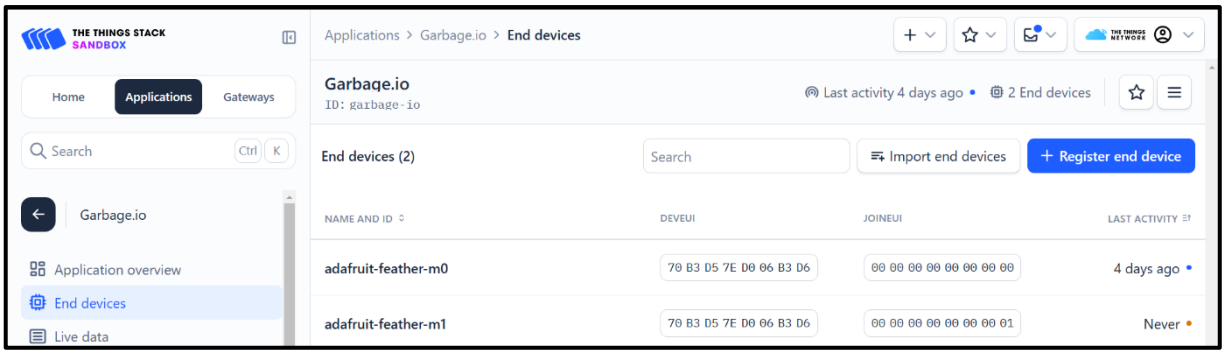

Figure 3: Garbage.io application's end devices

5.1 TTN Connection

Once we have configured a TTN server and created our application, we need to register our physical end device to the network to be able to send messages. We will use an **Over-The-Air-Activation (OTAA)** to carry out this procedure [3].

The OTAA is the most secure activation method. During the join procedure with the network, the end devices establish a connection and receive a dynamically assigned device address and negotiate security keys.

Before the activation, we need to store the AppEUI, DevEUI (both in little-endian) and AppKEY (big-endian) in our end device. We will retrieve this information by going on:

TTN Website → Applications → Garbage.io (our application's name) → End devices → Activation Information (Figure 4)

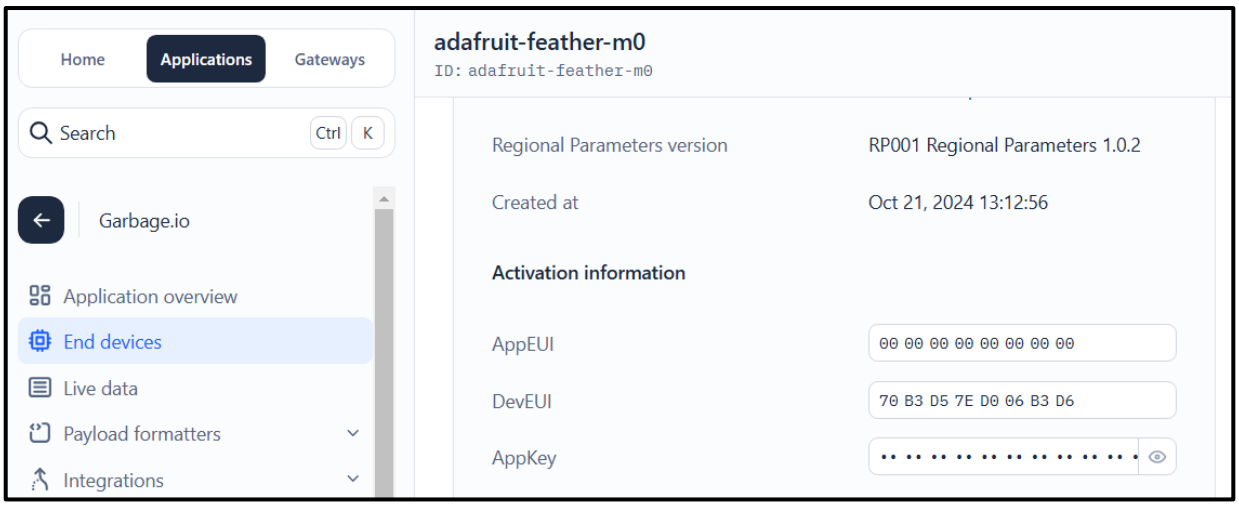

Figure 4: Our application's activation information

Then, we store this information into our end device using an Arduino and we are finally able to send data to the TTN server by supplying our microcontroller using a computer or a battery.

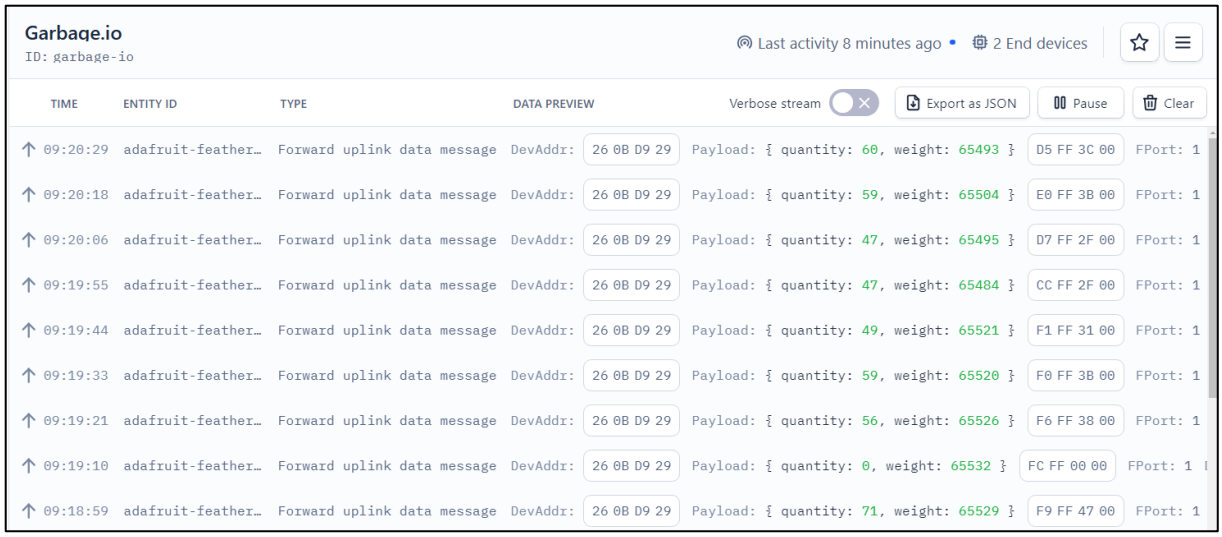

Figure 5: Adafruit's Live data payloads

We can see on our Adafruit's Live data, showed on the Figure 5, that we are able to communicate with the TTN server and send dummy payloads.

Since we have a weight sensor and a supersonic sensor, we can use their data to determine the bin's quantity and its weight. This information will be sent via the payloads to the TTN server.

We can format our payloads using the "Payload formatters" in the end device configuration of our application. The formatter code is shown on the Figure 6. This code specifies which information is received from the payload and will allow better data extraction for the next section.

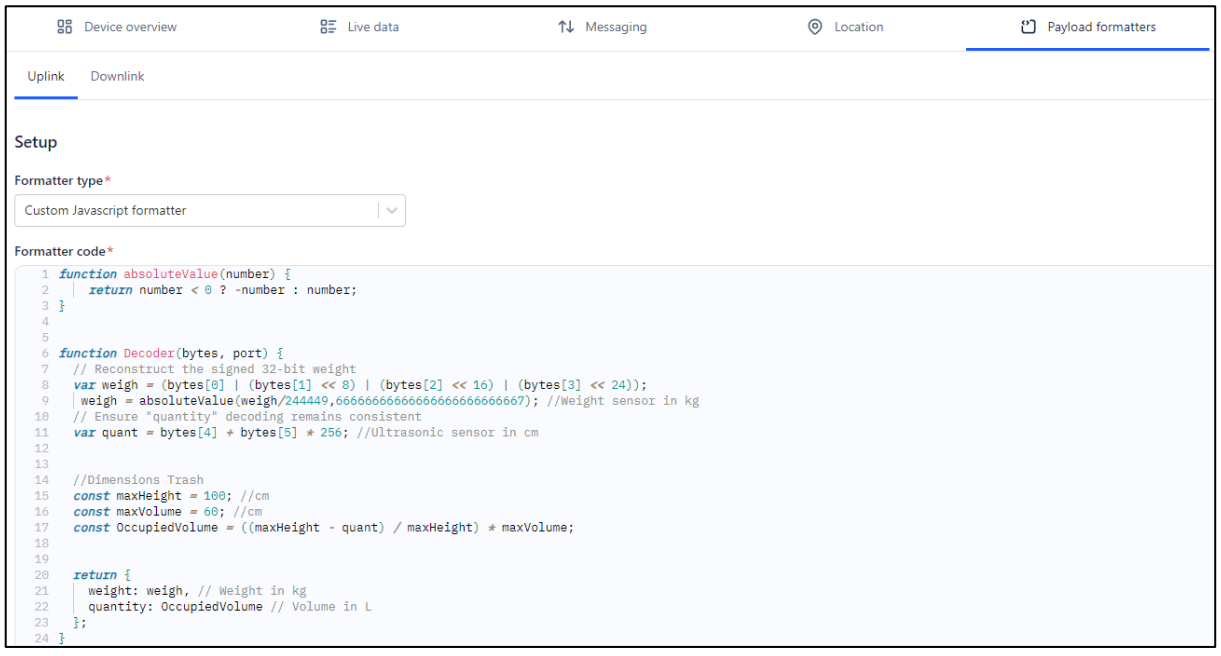

Figure 6: Application's payload formatter code

NB: We can note that in the payload formatter, a conversion formula is used to calculate the occupied volume in the trash bin. It is based on the height measured by the ultrasonic sensor to determine the proportion of the bin filled, which is then multiplied by the maximum volume: $Occupied\ Volume\ (\%) = \frac{\text{Total Height} - \textbf{Measured Height}}{\text{Total Height}} * Maximum\ Volume$
In our case, we consider the trash bin to have a maximum height of 100 cm and a maximum volume of 60 L.

Finally, the device needs **its location set in TTN (**Figure 7**).** This is done by configuring the **latitude** and **longitude**, which are the exact GPS coordinates of the device. These coordinates will be used to:

∗Show the bin's position on the map
∗Calculate the best route for collecting selected bins

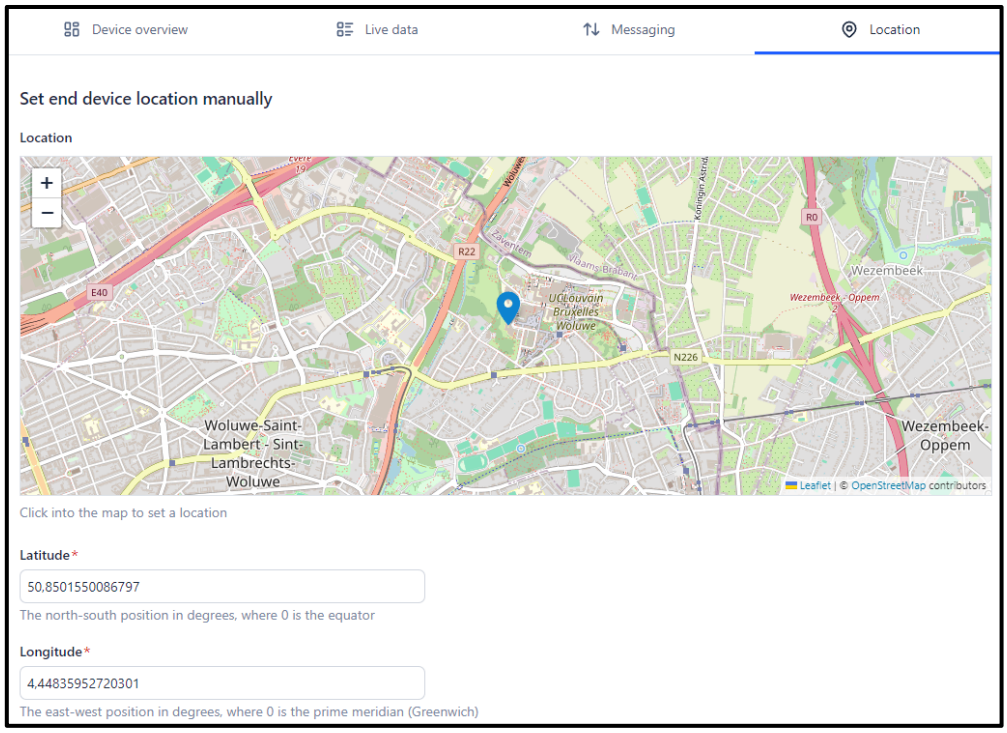

Figure 7 : Manual Configuration of a device location in TTN

5.2 TTN Data Extraction

Once we have the sensor data, we extract this data from TTN, process it in real-time, and display it on a web interface powered **by a Node.js server**. Whenever TTN sends new data from a device, the server listens to these messages, which include details like the device **ID**, **weight**, **quantity**, and **location** (latitude and longitude).

The data is then stored **in a local JSON file ("trashs.json")**, ensuring that all tracked device information is saved and continuously updated. If a device's data already exists, the record is updated; otherwise, a new entry is created.
Still on the topic of JSON files, but not related to data extraction from TTN, as we will see later, trajectory routes are also saved in a separate JSON file ("trashsroutes.json").

Using Socket.IO, the server broadcasts this data **in real-time** to any connected clients, allowing immediate access for users or other systems.

6. Software

The Web Interface is designed to optimize waste management for an enterprise called **Garbage Enterprise**. The goal of this application is to enhance the efficiency of trash collection operations by leveraging IoT technology and real-time data.

6.1 Description of pages

6.1.1 Main Menu

The main menu serves as the entry point for the application *(*Figure 8*)*.
Users can choose between 2 profiles:

1. **Trash Manager**:
   Responsible for selecting trash bins to be emptied and calculating the most optimized route for the Trash Collector.
2. **Trash Collector**:
   The operational team responsible for emptying trash bins as directed by the Trash Manager.

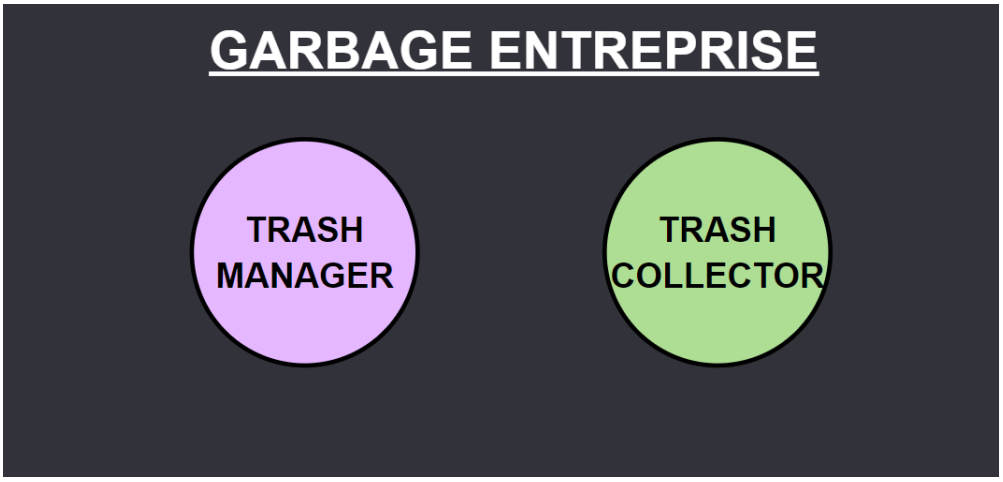

Figure 8: Web interface main menu

6.1.2 Trash Manager

When selecting the Trash Manager profile, the application prompts the user to enter **a password** for secure access *(*Figure 9*)*. An optional feature allows him to mask or unmask his password.

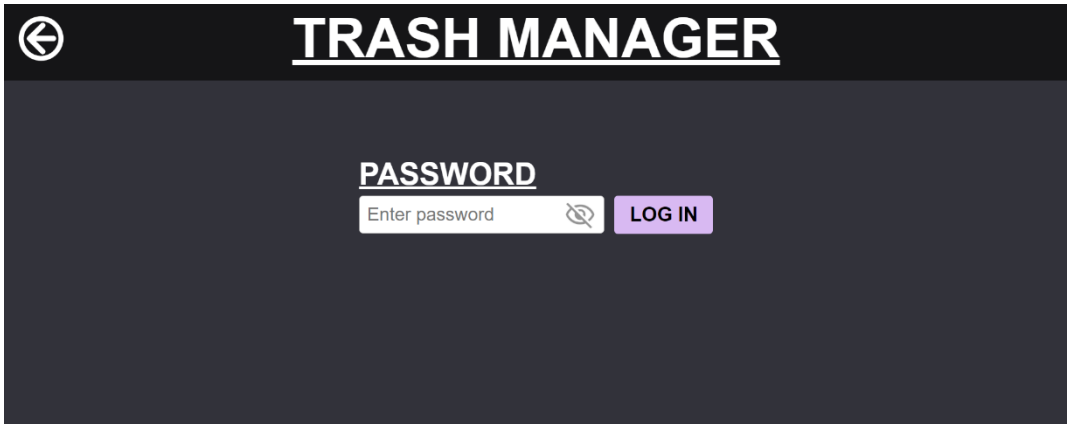

Figure 9: Web interface - Trash Manager login

If the password is incorrect, **a red error message** is displayed to notify them (Figure 10)

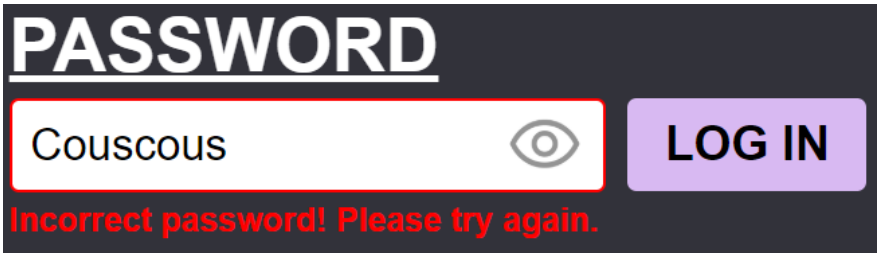

Figure 10: Web interface – Wrong password for the Trash Manager's login

Once the correct password is entered, the Trash Manager can log in and gain access to their dedicated dashboard. The Trash Manager's password is: **1234**.

The dashboard provides the Trash Manager with **a comprehensive overview of all trash bins managed by the company (**Figure 11**).**

Key components include:

1)Trash List Table:

- **ID Trash**: Unique identifier for each trash bin
- **Filling Level**: Shows the percentage fill level of each trash bin, based on weight and ultrasonic sensor data.
- **Weight (kg)**: The current weight detected by the weight sensor.
- **Quantity (L)**: The trash bin's current volume in liters, converted from ultrasonic sensor data.

→The filling levels are categorized as follows:

Filling Level (%)	Weight Range (kg)	Quantity Range (L)
0%	Weight < 2.5	Quantity < 15
25%	2.5 ≤ Weight < 5	15 ≤ Quantity < 30
50%	5 ≤ Weight < 7.5	30 ≤ Quantity < 45
75%	7.5 ≤ Weight < 8.5	45 ≤ Quantity < 54
100%	Weight ≥ 8.5	Quantity ≥ 54

For each row, the conditions for weight and quantity are connected by **"AND/OR"**, meaning either condition can be met individually or both together to determine the filling level.

2)Selection Options:

- **Select All**: Automatically selects all trash bins in the list.
- **Select Filled Trash**: Selects only bins filled to 100%.
- **Selected Trash Info**: Displays the number of selected trash bins versus the total.
- **Pathway Calculation**: Button to calculate the optimized route for the selected bins.

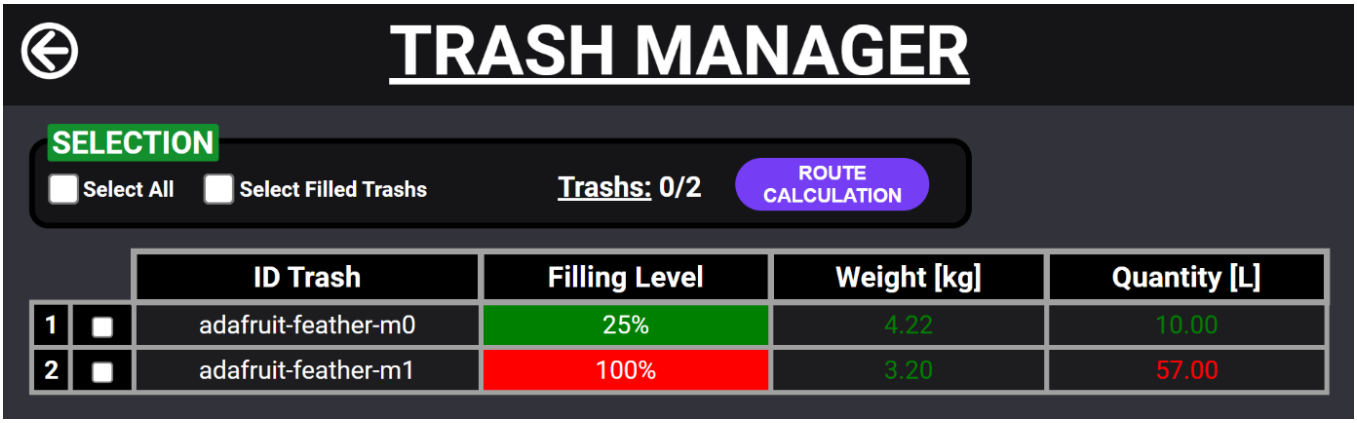

Figure 11: Trash Manager - Dashboard

NB: It is important to note that in the TTN project, the prototype corresponds to the trash bin "adafruit-feather-m0". The rest of the list corresponds to devices created on TTN where sensor data is simulated. Moreover, the server is implemented in such a way that, with a single line of code, a new TTN device can be easily added.

After selecting bins and clicking "Pathway Calculation," the application displays at the bottom of the page (Figure 12):

1) **Map**: An optimized route map generated using Google Maps API. The route is calculated based on the GPS coordinates (latitude and longitude) of the selected bins.
2) **Route Summary Table**: A table including the starting point, stopping points (trash bins), and the return itinerary to the starting point.
3) **SEND Button**: Sends the route information, map, and trash bin details to the Trash Collector, along with the current date.

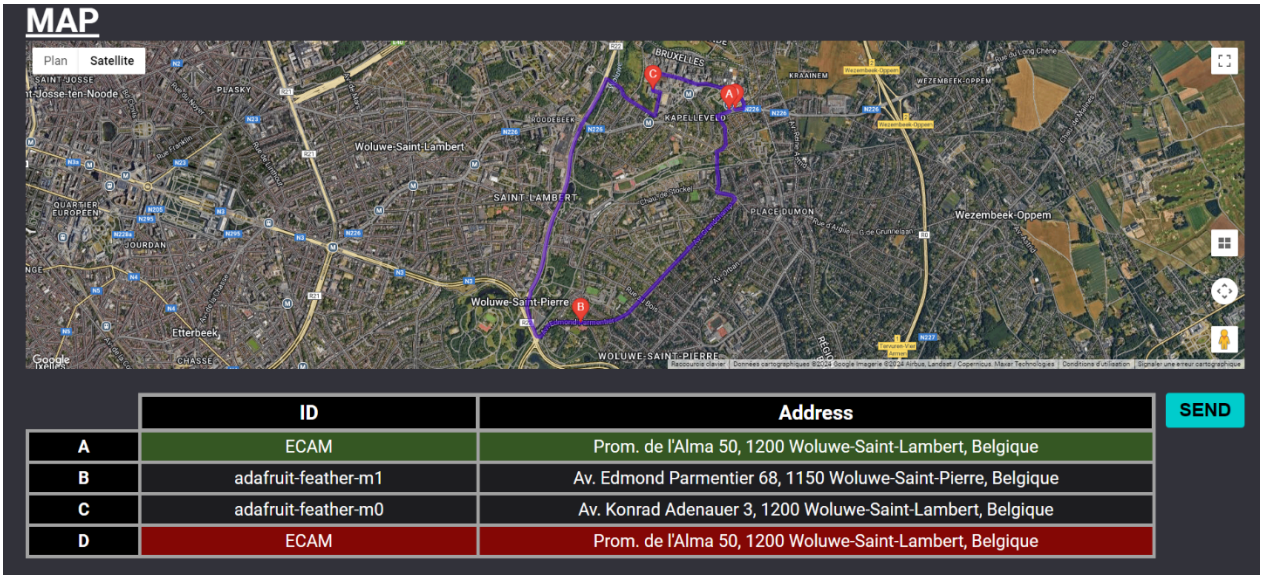

Figure 12: Trash Manager – Pathway Calculation

Once the Trash Manager presses the SEND button, a Popup appears to confirm that the route has been sent to the Trash Collector (Figure 13).

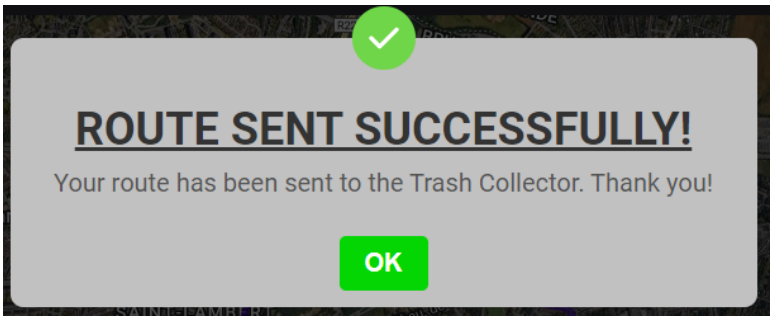

Figure 13: Trash Manager – Confirmation Popup

6.1.3 Trash Collector

When selecting the Trash Collector profile, the user is directed to their dedicated dashboard (Figure 14). This page is tailored to provide all the necessary information for completing the waste collection tasks.

The Trash Collector dashboard displays:

1) **Date**: The current date for clarity and synchronization.
2) **Route Map and Table**: Contains the optimized route and details sent by the Trash Manager. The Trash Collector uses this information to empty the specified bins.

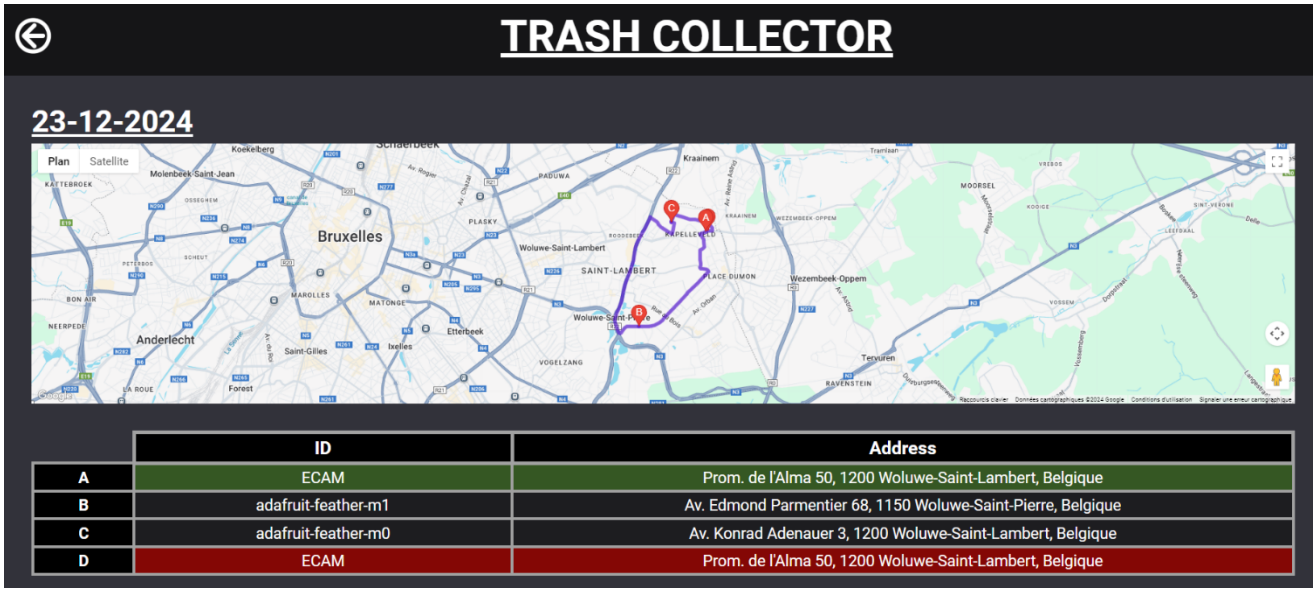

Figure 14: Trash Collector - Dashboard

If there is no route for the day yet, it will be indicated to them (Figure 15).

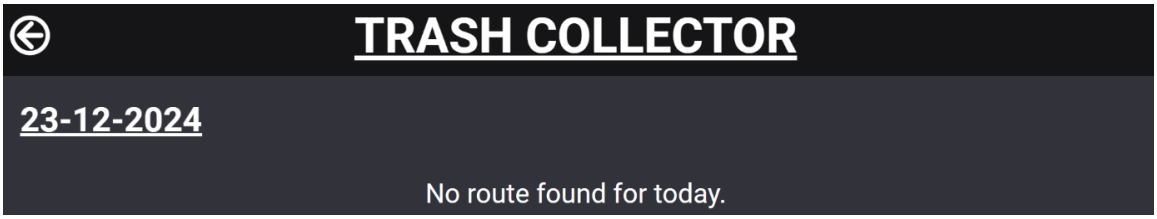

Figure 15: Trash Collector – No route

6.2 Activity Diagrams

To better understand the workflows of both the **Trash Manager** and the **Trash Collector**, the following activity diagrams are presented (Figure 16 and 17). These diagrams illustrate **the step-by-step processes and interactions involved in their respective tasks.** Together, these diagrams provide a comprehensive view of the operational flow within the system.

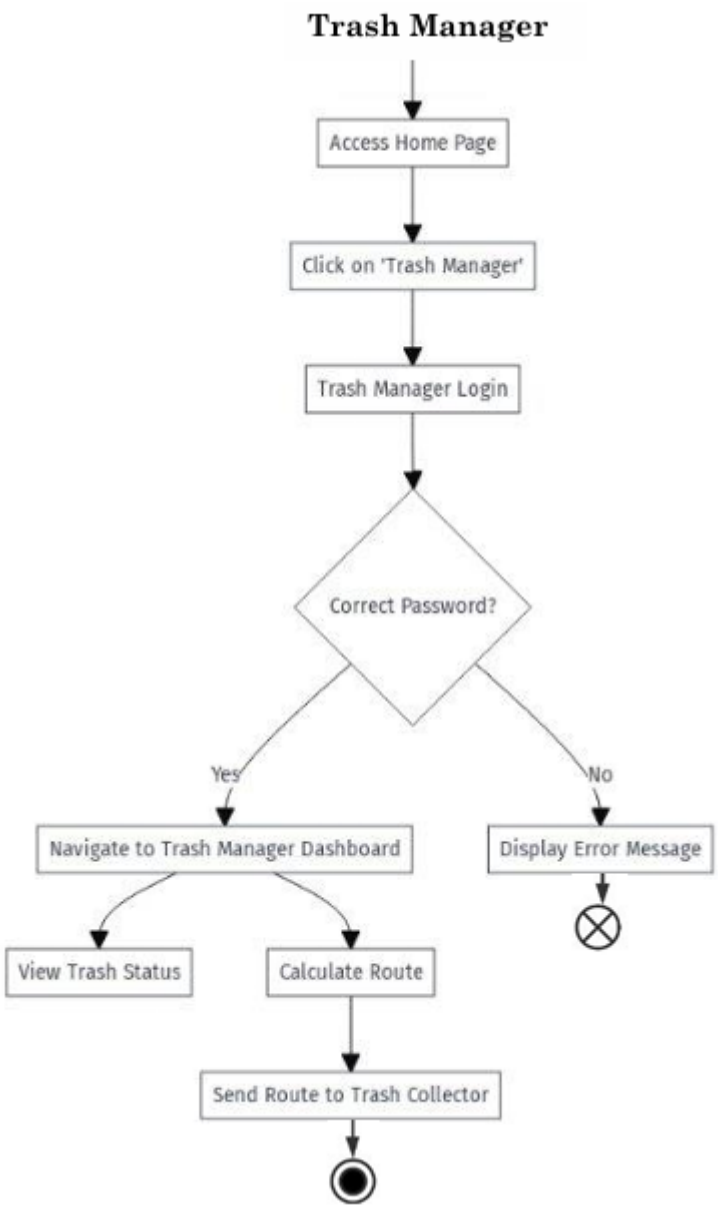

Figure 16: Activity Diagram - Trash Manager

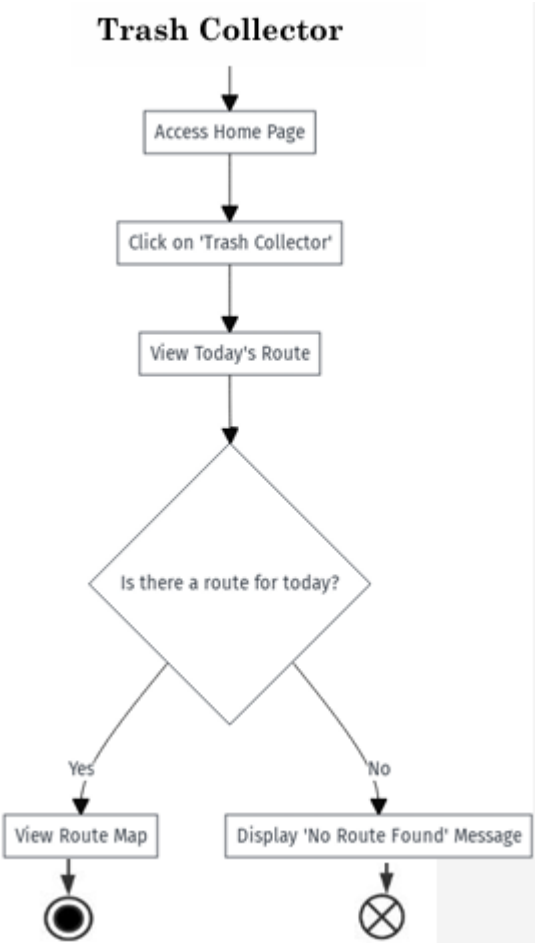

Figure 17: Activity Diagram - Trash Collector

6.3 Sequence Diagram

To provide an overview, the sequence diagram of the web interface is shown below (Figure 18). It helps to understand **the overall structure** and **the main features accessible to the user.**

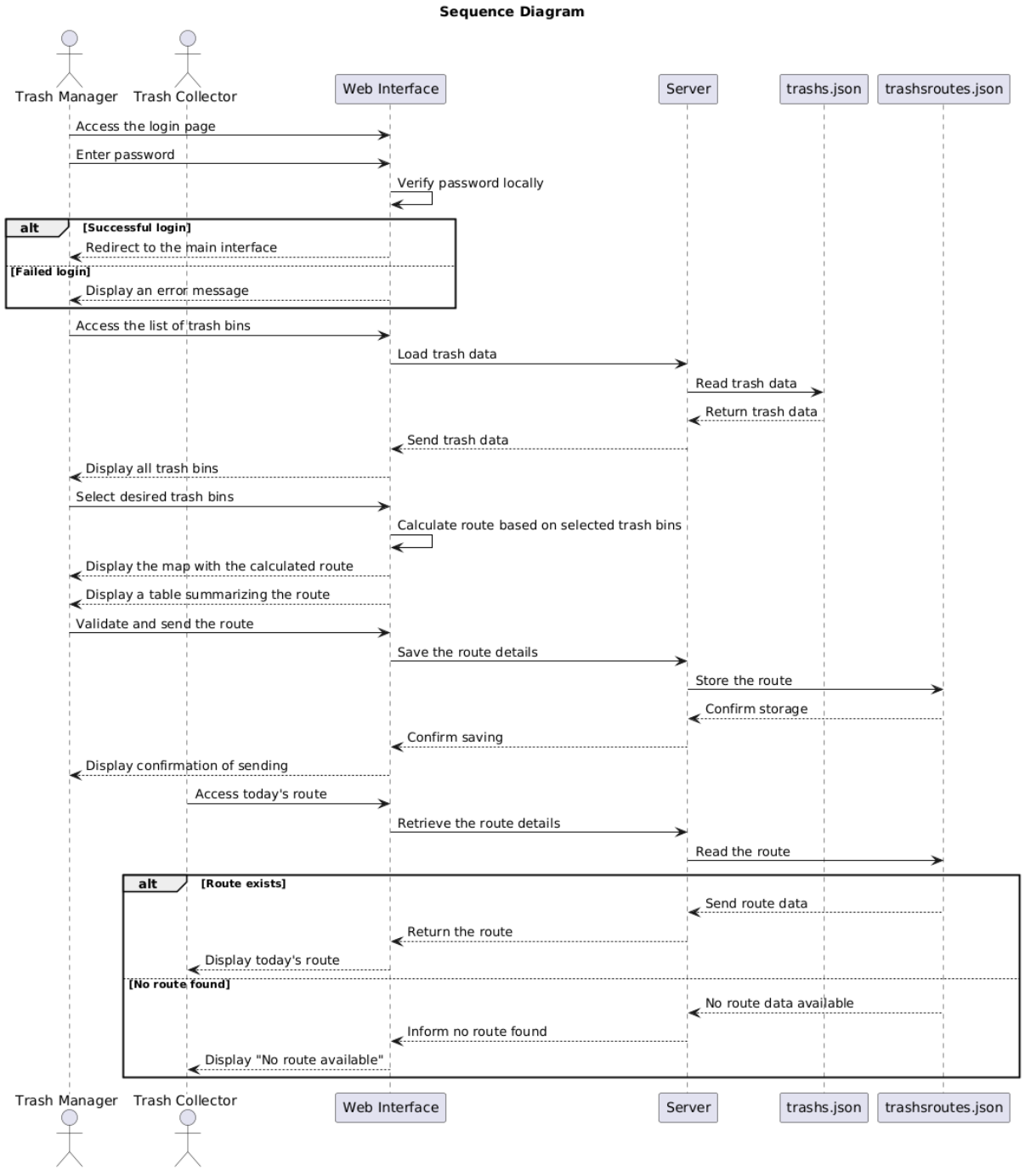

Figure 18: Sequence Diagram

6.4 Github Link

Here is the GitHub link to the project, where you can find **the .ino code of the prototype** and **the web interface**, along with detailed explanations about its configuration and the libraries used **in the README**:
https://github.com/OussamaSahib/4MA-TrashECAM-IOTLABO

6.5 Software architecture

Below is a simplified representation of the data flow within our system, illustrating the interactions between the **Frontend**, **Backend**, and **The Things Network (TTN)** (Figure 19). Each step highlights how data moves and transforms, ensuring seamless functionality across all components.

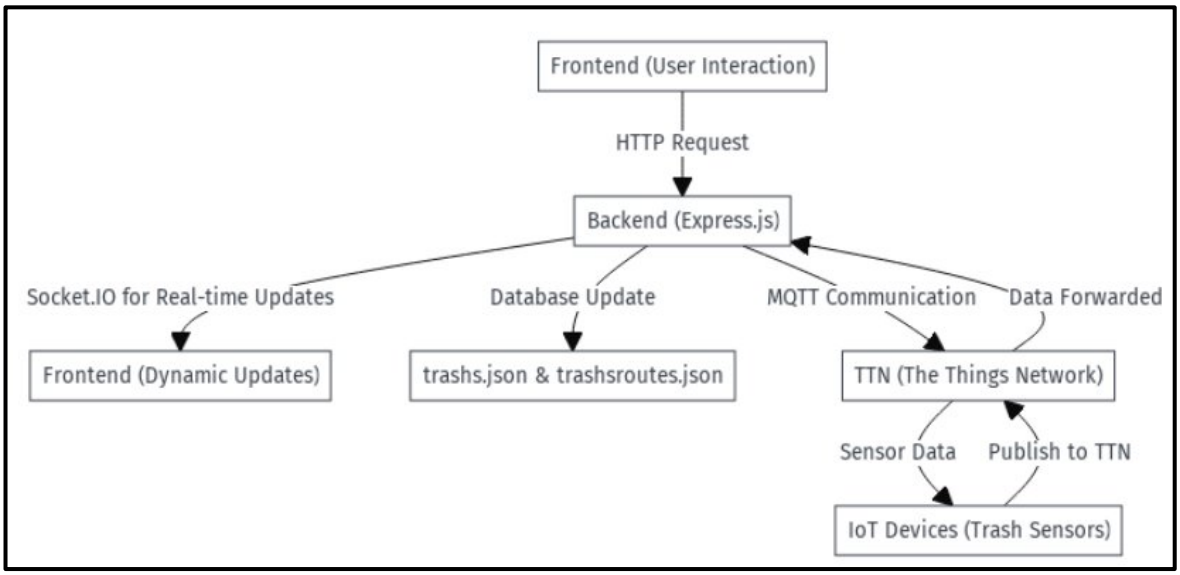

Figure 19: Software architecture

➢**Frontend → Backend :**

- Users interact with the application via the Frontend by sending HTTP requests to the server (view trash data, request a route).
- The Backend processes these requests and provides real-time updates using Socket.IO.

➢**Backend ↔ TTN**

- The **Backend** communicates with **TTN** using the **MQTT protocol** to send and receive data from IoT devices.

➢**TTN ↔ IoT Devices**

- IoT sensors installed in trash bins monitor attributes such as weight, fill level, and location.
- These sensors publish their data to **TTN**, which forwards it to the **Backend** for further processing and storage.

➢**Backend → Database**

The server saves and retrieves trash and route data in two JSON files:

*__trashs.json:__ Stores the status and data of all trash bins from TTN.

*__trashsroutes.json:__ Keeps route details for trash collection, after the Trash Manager has pressed the SEND button.

7. Performance Objectives

Target Performance	Quantified objective	Description
Scope of the device	15km	The device must be at a maximum distance of 15 km from the central unit.
Sampling frequency	144 transmissions/day	The device must transmit weight and distance data every 10 minutes. We assume that someone in charge of the garbage, a janitor for example, would be interested to take care of it quickly.
Data flow	19584 bits/day	13 bytes for the overhead[1] + 2 bytes for each metric, 144 times a day.
Current Consumption	Active mode : 3 mA Sleep mode : 5 µA	Total power consumption in active mode around 3 mA for µcontroller [2] and respectively 1.6 [3] and 15 mA [4] in active mode for the HX711 and HC-SR04 sensors.
Precision of the sensors	Relative error: Weight : 4,5% Quantity : 5,5%	Since our objective is to detect when a threshold is exceeded, high precision is not required. The relative weight error and distance represent around 200 g and 3,3 cm maximum capacity.

8. Prototype Tests

Once we have assembled our prototype and determined our objectives, we can start the tests protocol. There are multiple tests that can be done on our prototype. For example, we can measure the sensors' accuracy since they are the main components to measure a bin's filling level.

8.1 Tests protocol

8.1.1 Ultrasonic sensor:

- **Sensor**: Ultrasonic
- **General objective**: Measure accurately a distance
- **Constraints on specific conditions**:
    1. The sensor must operate within 10cm to 60cm
    2. The surface must be flat and stable

[1] Data recovered from https://eadalabs.com/lorawan-overhead/
[2] Data gathered from https://docs.rs-online.com/47c2/0900766b81534352.pdf , page 942
[3] See https://www.tinytronics.nl/en/sensors/weight-pressure-force/load-cells/load-cell-amplifier-hx711
[4] Data available on https://makersportal.com/shop/hc-sr04-ultrasonic-distance-sensor

- **Input**: Known distances (10cm, 20cm, 40cm, 60cm) between the sensor and our target
- **Output**: Real-time sensor's reading in centimeters
- **Test procedure**:
    1. Place the sensor on a flat stable surface, connected to a microcontroller with data logging.
    2. Position a target in front of the sensor at a predefined distance from the sensor.
    3. Compare the sensor's logged values with the actual distances.

- **Results & Interpretation**:

Test case	**Distance (cm)**	**Surface Type**	**Sensor Reading (cm)**	**Error (cm)**
1	10	Flat Table	10	0
2	20	Flat Table	20	0
3	40	Flat Table	39	1
4	60	Flat Table	58	2

- **Interpretations**: With these results, we can tell that the ultrasonic sensor is not completely accurate. There is a margin of error that is increasing with further distances. Since we want the sensor to be accurate when the
- **Actions post-test**: No action needed as the accuracy is acceptable for our application

8.1.2 Weight sensor:

- **Sensor**: Weight sensor
- **General objective**: Measure weight accurately
- **Constraints on specific conditions**:
    1. The sensor must operate within 0 to 8.5Kg
- **Input**: Known weights (1Kg, 2Kg, 3Kg) placed on the sensor
- **Output**: Real-time sensor's reading in kilograms
- **Test procedure**:
    1. Place the sensor on a flat stable surface, connected to a microcontroller with data logging.
    2. Position a calibrated weight on top of the sensor.
    3. Compare the sensor's logged values with the actual weights.
- **Results**:

Test case	**Weight (kg)**	**Surface Type**	**Sensor Reading (kg)**	**Error (kg)**
1	1	Flat Table	0.9734	0.0266
2	2	Flat Table	1.8826	0.1174
3	3	Flat Table	2.8646	0.1354

- **Interpretations**: With these results, we can tell that the weight sensor's errors increase as there is more weight.
- **Actions post-test**: Division Factor modified.
- **Results post-actions**:

Test case	**Weight (kg)**	**Surface Type**	**Sensor Reading (kg)**	**Error (kg)**

1	1	Flat Table	0.9527	0.0473
2	2	Flat Table	1.9767	0.0233
3	3	Flat Table	2.9772	0.0228

- **Interpretations**: With a better division factor, we get better results as the weight increases, but in return, the small weights get more error. This is a better result because what matters the most is when the weight is close to the maximum.

8.1.3 Power consumption

- **Sensor**: Yocto-Watt circuit
- **General objective**: Supply power to all the components and be efficient.
- **Constraints on specific conditions**:
  1. The power consumption must average 19.6mA in active mode and 5µA in sleep mode.
  2. Data must be sent to the TTN server during the test to see sensor activity
- **Input**: Known Tension (5V) and time
- **Output**: Power consumed over time in mW
- **Test procedure**:
  1. Plug the microcontroller to the Yocto Watt module
  2. Plug the module to a pc to get both power supply and measurements.
  3. Run the YoctoWatt.exe
- **Results & Interpretation**:

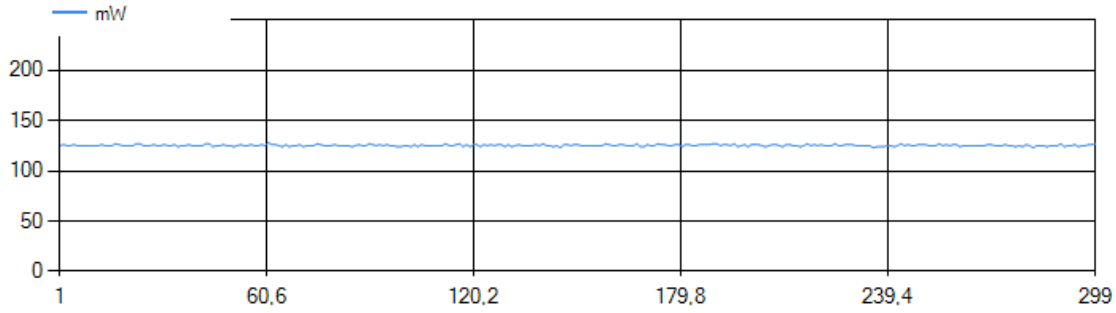

Figure 20 - Power measured in [mW] over a period of time

- **Interpretations**: Due to a lack of time, the test was not performed in great conditions as we didn't measure the communications properly. The average power measured in active mode is 125,294mW, so 24,66mA.
- **Actions post-test**: This isn't the average current expected. It would be necessary to make this test again in proper conditions and then chose other more power efficient sensors.

9. Areas for improvement

9.1 Database type Json vs Mysql

During this project, we worked with a minimal amount of data as we had only one working prototype. Should we upscale this project, it could prove to be useful to use more robust database types, for example MySQL.

9.2 Universal and robust design

Some improvements could be made on durability and robustness: the load cell has a lot of bending once the load is applied at the end of the plastic piece, the cables connected to the load cell had to be resoldered to the part a few times and the cables connected to the microcontroller can disconnect if the prototype moves around too much.

The garbage notifier could benefit from a more universal design that could be compatible with every type of bin in a city.

9.3 Lower Energy consumption

The sleep mode should be enabled between the communications which is a major improvement from our current design regarding current consumption. For now, the sensors are constantly in active mode, which is a net loss of energy. If a lower energy consumption is still needed once sleep mode is enabled, using less energy- greedy components could play a role too.

9.4 Upscaling possibilities

Should we ever manufacture this project at the scale of monitoring dozens of trash cans at a time, a printed circuit board should be necessary.

10. Conclusion

The main goal here is: simplicity makes things more reliable. Our “Garbage Filling Notifier” prototype is based on a simple and yet effective architecture.

The components are easy to assemble, low-priced and it doesn’t hurt that all of them are already available at ECAM.

The setup of the microcontroller, ultrasonic and weight sensors allow us to have a quick and practical measure to ensure fast, reliable tracking. Including the LoRaWAN protocol allows us to communicate on a long-range and achieve real-time monitoring of trash bins.

We collect data from the endpoint to the TTN server using their website to format our payloads and ensure a better data extraction for the Web Interface. Three types of information are sent: the weight, the volume and the location of the bin.

The web interface is designed to optimize waste management for a company. The Trash Manager profile lists the different bins available and prompts their filling level based on the weight and the volume. If the bin is maxed out, the manager can generate an optimized route and send it to the Tash Collector. The Trash Collector will use all the previous information to summarize it and optimize the collection of the bins.

Performance-wise, the system is designed for energy efficiency, relatively precise sensor readings within tolerable error margins, data transmission every 10 minutes and a wide scope of 15 km.

This Project highlights the potential of IoT to enhance the efficiency of public services. It can help the technician with a reasonable bin weight and the environment by preventing the bin from overflowing.

Further improvements and additional features could be researched. For example:

- a led indicator on the bin for the filling level
- a phone app with notifications in real-time if a bin is filled
- optimizing the energy consumption
- using a more robust database type
- using a more “universal” and reliable design, …

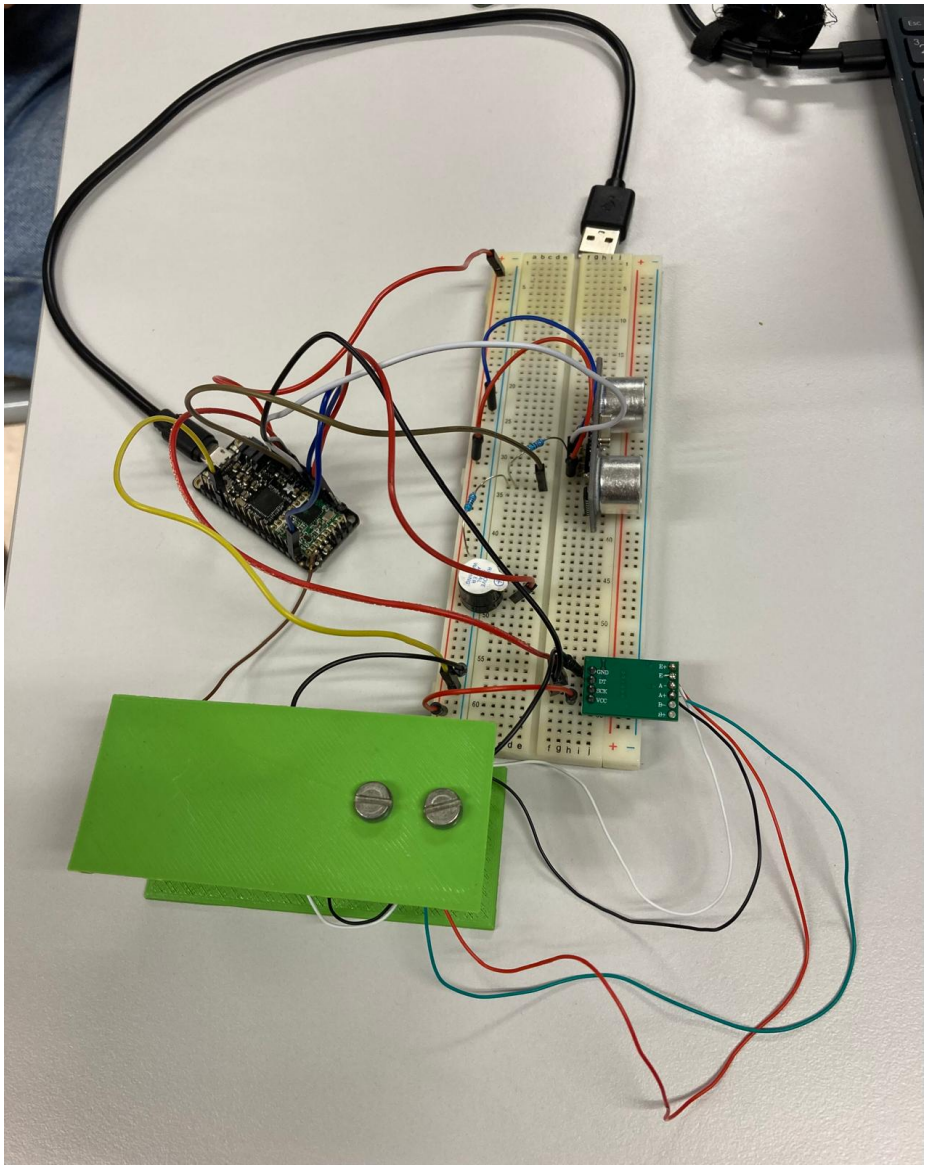

Figure 21: Final prototype design

Bibliography

[1] S. Santos, "Arduino with Load Cell and HX711 Amplifier (Digital Scale)," [Online]. Available: https://randomnerdtutorials.com/arduino-load-cell-hx711/. [Accessed 23 December 2024].

[2] Q. Delhaye, "IT4L-IOT," 18 October 2024. [Online]. Available: https://github.com/parastuffs/IT4L-IOT/wiki. [Accessed 21 October 2024].

[3] "End Device Activation," [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/. [Accessed 21 October 24].

[4] R. J., 3 October 2024. [Online]. Available: https://eadalabs.com/lorawan-overhead/. [Accessed 30 October 2024].

[5] Atmel, "Atmel SAM D21E / SAM D21G / SAM D21J".

[6] "Load Cell Amplifier-HX711," [Online]. Available: https://www.tinytronics.nl/en/sensors/weight-pressure-force/load-cells/load-cell-amplifier-hx711. [Accessed 30 October 2024].

[7] "HC-SR04 Ultrasonic Distance Sensor," [Online]. Available: https://makersportal.com/shop/hc-sr04-ultrasonic-distance-sensor. [Accessed 30 October 2024].