i

**SAFEJOURNEY CASE**

A Senior Project Report

Presented in Partial Fulfillment of the Requirements for the Degree Bachelor of Engineering in the

Faculty of Engineering of Notre Dame University

By:

Charbel Ishak: 20197051

Jouna Harb: 20197029

**Notre Dame University-Louaize**

Fall 2023

Advised by:

Dr. Chady El-Moucary

**Department of Electrical, Computer, and Communication Engineering**

Committee Member Advised by:

Dr. Abdallah Kassem

Mr. Walid Zakhem Dr. Chady El-Moucary Department of Electrical, Computer, and

Communication Engineering

# Abstract

This report presents the proposed "SafeJourney Case" project, aimed at resolving the prevalent issue of luggage loss and theft at airports through the implementation of advanced sensors including ultrasonic sensor, photoresistor sensor and load cell sensor in addition to a smart tag along with an intuitive mobile application. The project integrates a diverse array of technical disciplines, including electronics, software development, and user interface design.

With a primary focus on ensuring a worry-free travel experience for passengers, the project seeks to offer a robust solution that mitigates the risks associated with luggage security during air travel. The report elucidates the project's key objectives, essential prerequisites, inherent design limitations, and a comprehensive risk analysis.

Furthermore, it delineates the strategic project plan, encompassing crucial project activities, a meticulously crafted Gantt chart, and a comprehensive cost estimation for the proposed implementation. We collaborated on this project, bringing together our expertise in product development and technology integration to realize the vision of a secure and stress-free journey for every air traveler.

# Acknowledgements

We would like to express our deep gratitude to all those who have contributed to the successful realization of our "SafeJourney Suitcase" project. Our sincere thanks go to our project supervisor, Dr.Chady El-Moucary, whose unwavering support, invaluable guidance, and insightful feedback were instrumental in navigating the complexities of the project.

Additionally, we extend our heartfelt appreciation to Mr. Walid Zakhem for his continuous assistance and provision of essential hardware tools and solutions, which were pivotal in the project's development and implementation.

Special thanks are due to our parents, whose unwavering support and encouragement provided the necessary motivation and strength to excel and deliver the best possible outcomes in this journey.

Finally, yet importantly, we express our gratitude to each other for our collaborative efforts, dedication, and persistence, which were fundamental in overcoming challenges and achieving the objectives of this innovative project.

**Table of Contents**

[Abstract 3](#_Toc153907867)

[Acknowledgements 4](#_Toc153907868)

[LIST OF FIGURES 7](#_Toc153907869)

[LIST OF TABLES 10](#_Toc153907870)

[Chapter 1 – Introduction 11](#_Toc153907871)

[Chapter 2 - Standards, Constraints, and Requirements Specification 13](#_Toc153907872)

[Chapter 3 - Functional Decomposition 14](#_Toc153907873)

[Chapter 4-Project Management (Gantt chart, WBS, Cost Analysis) 19](#_Toc153907874)

[Chapter 5- System Failure Modes and Effects Analysis (FMEA) 23](#_Toc153907875)

[Chapter 6- Hardware Implementation 25](#_Toc153907876)

[6.1 - Load cell sensor YZC-1B series | 40kg 26](#_Toc153907877)

[6.2 - Load cell amplifier HX711 28](#_Toc153907878)

[6.3 - Ultrasonic sensor HC-SR04 30](#_Toc153907879)

[6.4 - Light Dependent Photoresistors GL5506 33](#_Toc153907880)

[6.5 - ESP 32 WROOM DEVKIT V1 36](#_Toc153907881)

[6.6 - Samsung smart tag 39](#_Toc153907882)

[6.7 - Connecting everything to ESP32 and building the prototype 42](#_Toc153907883)

[Chapter 7- Software application 54](#_Toc153907884)

[7.1 - Using serial connection 54](#_Toc153907885)

[7.2 - Using the BLE method 58](#_Toc153907886)

[7.3 - Pubspec.yaml 64](#_Toc153907887)

[7.4 - Bluecall.dart 65](#_Toc153907888)

[7.5 - Noti.dart 71](#_Toc153907889)

[7.6 - AndroidManifest.xml 72](#_Toc153907890)

[7.7 - main.dart 73](#_Toc153907891)

[Chapter 10- References 93](#_Toc153907892)

# LIST OF FIGURES

[Figure 3.1: functional decomposition level 0 15](#_Toc153137349)

[Figure 3.2: functional decomposition level 1 16](#_Toc153137350)

[Figure 4.1: Gantt chart 20](#_Toc153137351)

[Figure 6.1.1: Load cell sensor | 40kg 27](#_Toc153137352)

[Figure 6.1.2: load cell installation size (mm) 27](#_Toc153137353)

[Figure 6.2.1: load cell amplifier HX711 29](#_Toc153137354)

[Figure 6.2.2: HX711installation size 29](#_Toc153137355)

[Figure 6.3.1: Ultrasonic HC-SR04 sensor 32](#_Toc153137356)

[Figure 6.3.2: HC-SR04 installation size (mm) 32](#_Toc153137357)

[Figure 6.4.1: LDR voltage divider 34](#_Toc153137358)

[Figure 6.4.2: Light dependent resistor GL5506 35](#_Toc153137359)

[Figure 6.4.3: GL5506 installation size 35](#_Toc153137360)

[Figure 6.5.1: Esp 32 Wroom devkit v1 37](#_Toc153137361)

[Figure 6.5.2: esp32 installation size 37](#_Toc153137362)

[Figure 6.5.3: ESP32 Pinout 38](#_Toc153137363)

[Figure 6.6.1: Samsung SmartTag 41](#_Toc153137364)

[Figure 6.6.2: Samsung smart tag installation size 41](https://d.docs.live.net/86d7ee94331802bd/Desktop/johnny/SENIOR%202/FINAL/SENIOR%20REPORT.docx#_Toc153137365)

[Figure 6.7.1: calibrating HX711 pt1 43](#_Toc153137366)

[Figure 6.7.2: calibrating HX711 pt2 43](#_Toc153137367)

[Figure 6.7.3: The load cell Arduino code 44](#_Toc153137368)

[Figure 6.7.4: Ultrasonic sensor Arduino code 45](#_Toc153137369)

[Figure 6.7.5:Photoresistor Arduino code 46](#_Toc153137370)

[Figure 6.7.6: the full circuit 46](#_Toc153137371)

[Figure 6.7.7: testing out all the connections together pt1 47](#_Toc153137372)

[Figure 6.7.8: testing out all the connections pt2 48](#_Toc153137373)

[Figure 6.7.9: case closed 49](#_Toc153137374)

[Figure 6.7.10: results from case closed 50](#_Toc153137375)

[Figure 6.7.11: case opened 51](#_Toc153137376)

[Figure 6.7.12: results of case opened 52](#_Toc153137377)

[Figure 6.7.13: testing out a 1kg weight on the load cell 53](#_Toc153137378)

[Figure 7.1.1: serial IDE code pt1 54](#_Toc153137379)

[Figure 7.1.2: serial IDE code pt2 56](#_Toc153137380)

[Figure 7.2.1: BLE IDE code pt1 58](#_Toc153137381)

[Figure 7.2.2: BLE IDE code pt2 59](#_Toc153137382)

[Figure 7.2.3: BLE IDE code pt3 60](#_Toc153137383)

[Figure 7.2.4: BLE IDE code pt4 61](#_Toc153137384)

[Figure 7.2.5: BLE IDE code part5 61](#_Toc153137385)

[Figure 7.2.6: BLE IDE code pt6 63](#_Toc153137386)

[Figure 7.3.1: pubspec code 64](#_Toc153137387)

[Figure 7.4.1: Bluecall code pt1 65](#_Toc153137388)

[Figure 7.4.2: Bluecall code pt2 65](#_Toc153137389)

[Figure 7.4.3: Bluecall code pt3 66](#_Toc153137390)

[Figure 7.4.4: Bluecall code pt4 67](#_Toc153137391)

[Figure 7.4.5: Bluecall code pt5 68](#_Toc153137392)

[Figure 7.4.6: Bluecall code pt6 69](#_Toc153137393)

[Figure 7.4.7: Bluecall code pt7 70](#_Toc153137394)

[Figure 7.5.1: Noti code 71](#_Toc153137395)

[Figure 7.6.1: AndroidManifest code pt1 72](#_Toc153137396)

[Figure 7.6.2: AndroidManifest code pt2 72](#_Toc153137397)

[Figure 7.7.1: main code pt1 73](#_Toc153137398)

[Figure 7.7.2: main code pt2 74](#_Toc153137399)

[Figure 7.7.3: main code pt3 74](#_Toc153137400)

[Figure 7.7.4: main code pt4 75](#_Toc153137401)

[Figure 7.7.5: main code pt5 76](#_Toc153137402)

[Figure 7.7.6: main code pt6 77](#_Toc153137403)

[Figure 7.7.7: main code pt7 78](#_Toc153137404)

[Figure 7.7.8: main code pt8 78](#_Toc153137405)

[Figure 7.7.9: main code pt9 79](#_Toc153137406)

[Figure 7.7.10: main code pt10 80](#_Toc153137407)

[Figure 7.7.11: main code pt11 80](#_Toc153137408)

[Figure 7.7.12: main code pt12 80](#_Toc153137409)

[Figure 7.7.13: main code pt13 81](#_Toc153137410)

[Figure 7.7.14: main code pt14 82](#_Toc153137411)

[Figure 7.7.15: main code pt15 82](#_Toc153137412)

[Figure 7.7.16: main code pt16 83](#_Toc153137413)

[Figure 7.7.17: main code pt17 83](#_Toc153137414)

[Figure 7.7.18: main code pt18 84](#_Toc153137415)

[Figure 7.7.19: main code pt19 85](#_Toc153137416)

[Figure 7.7.20: upon opening the application. 86](#_Toc153137417)

[Figure 7.7.21: the drawer 87](#_Toc153137418)

[Figure 7.7.22: initial state of the closed empty suitcase 88](#_Toc153137419)

[Figure 7.7.23: seeing the initial state of the ultrasonic sensor 89](#_Toc153137420)

[Figure 7.7.24: LDR when case is opened 90](#_Toc153137421)

[Figure 7.7.25: the smart tag page in the app 91](#_Toc153137422)

# LIST OF TABLES

[Table 3.1: Table showing inputs, outputs, and functionality of the SafeJourney case 15](#_bookmark8)

[Table 3.2: Table showing inputs, outputs, and functionality of the ultrasonic sensor 16](#_bookmark8)

[Table 3.3: Table showing inputs, outputs, and functionality of the Light Dependent Resistor 17](#_bookmark8)

[Table 3.4: Table showing inputs, outputs, and functionality of the Load cell sensor 17](#_bookmark8)

[Table 3.5: Table showing inputs, outputs, and functionality of the HX711 17](#_bookmark8)

[Table 3.6: Table showing inputs, outputs, and functionality of the ESP32 18](#_bookmark8)

[Table 3.7: Table showing inputs, outputs, and functionality of the Samsung smart tag 17](#_bookmark8)

[Table 4.1: Project Cost 20](#_bookmark8)

[Table 4.2: WBS 21](#_bookmark8)

[Table 5.1: Table showing sensitivity, occurrence and detection ratings as well as the RPN for each risk above 24](#_bookmark8)

# Chapter 1 – Introduction

In today's age of frequent air travel, the issue of luggage mishandling and theft continues to plague millions of passengers, adding an unwelcome layer of stress and uncertainty to their journeys. The statistics are telling: according to SITA's latest Baggage Insight Report in May 2023, approximately 26 million pieces of luggage were lost, delayed, or damaged in 2022 alone, translating to nearly eight bags per 1,000 passengers. Issues during transfers between flights were found to be the leading cause, accounting for a significant portion of mishandled baggage [1].

SITA, the multinational information technology company responsible for managing IT systems for 90% of airlines, sheds light on the scale of the problem, revealing that in the first quarter of 2022, major US airlines reported over 684,000 instances of lost and mishandled bags [1]. Furthermore, the Bureau of Transportation Statistics recorded that carriers handled a staggering 393 million bags in 2021, of which over 2 million were reported lost [2].

Airport travel, despite its pivotal role in modern life, continues to be marred by persistent luggage-related issues. The ever-present threat of luggage theft, both during transit and at baggage carousels, remains a significant concern for travelers. Reports of stolen valuables amounting to a staggering $1.2 billion in 2018 underscore the urgent need for effective solutions to safeguard passengers' belongings [3].

Airport travel has become an indispensable aspect of contemporary life, fostering global mobility and connectivity on an unprecedented scale. Nonetheless, within the bustling confines of airport terminals, a persistent dilemma continues to haunt travelers: the perennial issue of luggage-related complications. Luggage theft, pilfered items from bags after check-in, and the all-too-common mix-up of suitcases at baggage carousels have evolved into a considerable source of inconvenience, frustration, and anxiety for travelers worldwide. According to a news article by CNN, cameras caught airport workers stealing from luggage in Miami International airport [4]. These challenges not only disrupt the smooth flow of journeys but also lead to significant financial losses and emotional distress, leaving a lasting impact on passengers' travel experiences.

Despite the stringent security measures in place, luggage theft remains a persistent problem, especially within airport premises. Opportunistic thieves often target momentarily unattended bags or those left unguarded on the baggage carousels, exploiting the chaotic ambiance and the distraction of fellow travelers. This not only undermines the sense of security passengers should feel but also exposes their valuable possessions to considerable risk. Incidents of stolen passports, electronics, jewelry, and other irreplaceable personal effects continue to surface, underscoring the urgent need for innovative solutions to bolster airport security measures.

Furthermore, even after passengers successfully check in their luggage, the threat of items being stolen during transit looms large. The intricate journey from check-in to the final destination encompasses several touchpoints where bags can be tampered with, allowing valuable items to vanish discreetly. Compounded with the frenzy of reclaiming luggage at the carousel, instances of travelers inadvertently mistaking one suitcase for another further compound the issue, leading to unintended theft or misplacement of belongings. Not to forget that many suitcases look exactly the same which leads to a lot of travelers picking up the wrong suitcase of the carousel.

To tackle these persistent challenges head-on, the "SafeJourney Case" project proposes a comprehensive approach leveraging advanced smart sensors and a user-friendly mobile application. By prioritizing real-time monitoring, security alerts, and precise location tracking, the project endeavors to curtail the incidence of theft, mitigate the risk of valuable items being stolen from bags, and ensure that travelers never again face the predicament of mistakenly grabbing the wrong suitcase. The SafeJourney Suitcase represents a revolutionary stride toward ensuring a secure and hassle-free travel experience, promising to restore peace of mind and confidence for travelers embarking on their journeys.

This smart suitcase was developed and designed using some sensors (ultrasonic, light, weight sensors) all connected to an ESP32 allowing it to send all the data needed to a well-designed and easy to use mobile application. This helps the traveler to keep updated with any minor inconvenience regarding their suitcase. In addition, a Samsung smart tag is provided in the suitcase in order to track your case wherever it is without the need of Wi-Fi.

# Chapter 2 - Standards, Constraints, and Requirements Specification

In the pursuit of developing a comprehensive solution to address the persistent issue of luggage security and tracking, a set of stringent standards, constraints, and requirements must be met to ensure the efficient and reliable performance of the proposed SafeJourney Suitcase system. The paramount objective is to significantly increase the security of luggage theft pertaining to unattended bags, while simultaneously providing travelers with an enhanced travel experience. The system design specifications mandate that the solution allow for seamless tracking of luggage, ensuring that travelers can devote their attention to the enjoyment of their journey without the constant concern of misplaced or stolen belongings.

Integral to the system's functionality are the implementation of real-time alerts to promptly notify users of potential threats to the security of their suitcase if it was ever opened. Furthermore, the system is engineered to enable swift and precise location tracking, guaranteeing that in the event of loss or misplacement, users can locate the suitcase. The incorporation of a weight tracking mechanism with an approximate 0.1 Kg error rate serves to assist travelers in avoiding the inconvenience of over packing, facilitating adherence to airline luggage regulations and minimizing additional costs associated with excess baggage.

To fortify the system's security measures, the project stipulates the integration of mechanisms for immediate notifications in the event of unauthorized access to the suitcase. Additionally, a proactive alert system has been engineered to notify users in the case of opening the case alarming them about a potential theft. Of crucial importance is the seamless tracking of the suitcase's location without relying on having conventional Wi-Fi connected to the suitcase, highlighting the system's capability to operate effectively in various environments and locations, thereby maximizing its usability and practicality for a diverse range of travelers.

# Chapter 3 - Functional Decomposition

At the primary level of the SafeJourney Suitcase system shown in figure 3.1, data from various sensors is collected and fed into the central processing unit, referred to as the "SafeJourney Case." The outputs from this central unit are then transmitted to the "Mobile Application," providing users with real-time results and updates.

Diving deeper into the system architecture, the secondary level (figure 3.2) is composed of four distinct components, 3 of them integrated within the central unit (ESP 32). These 3 components comprise different types of sensors. The first set of sensors involves two ultrasonic sensors, leveraging ultrasound waves to measure distance. The output of these sensors, in the form of distance data, is routed to the ESP 32 for further processing. The second set includes three light-dependent sensors, responsible for reading light conditions and converting them into corresponding voltage data. These voltage outputs are subsequently channeled to the ESP 32 for subsequent analysis.

The third set is a load cell sensor that operates by measuring the force exerted on the luggage and converting it into analog weight data. To ensure precise measurements, the analog data is processed through an HX711 amplifier and analog-to-digital converter (ADC), yielding a digital weight output. This digital weight information is then relayed to the ESP 32 for integration into the overall data processing framework.

Finally, the system incorporates a Samsung Smart Tag, designed to track the location of the luggage. This comprehensive integration of various sensors within the ESP 32 and the Smart Tag enables the SafeJourney Suitcase system to provide users with a cohesive and robust tracking and monitoring mechanism, ensuring the safety and security of their belongings during travel.

We have joined forces for this visionary project, with a clear focus on the hardware and software components, respectively. This collaborative synergy ensures a comprehensive and specialized approach to the development of the SafeJourney Suitcase system. By harnessing our individual strengths and expertise, we are dedicated to delivering a fully integrated solution that sets new benchmarks in luggage security and tracking technology. Our collective commitment to excellence guarantees that every element of the system, from hardware engineering to software implementation, is meticulously tailored to provide optimal performance and an exceptional user experience.

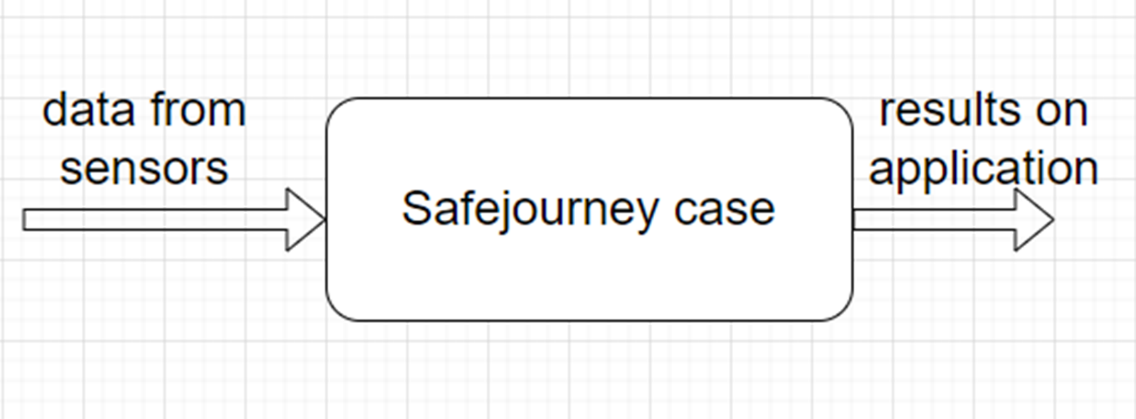


Figure 3.1: functional decomposition level 0

Table 3.1: Table showing inputs, outputs, and functionality of the SafeJourney case:

|  |  |
| --- | --- |
| **MODULE** | **SAFEJOURNEY CASE** |
| INPUTS | DATA FROM SENSORS |
| OUTPUTS | RESULTS ON APPLICATION |
| FUNCTIONALTY | Measure the weight of the luggage.  Check if the suitcase was opened.  Track the suitcase location. |

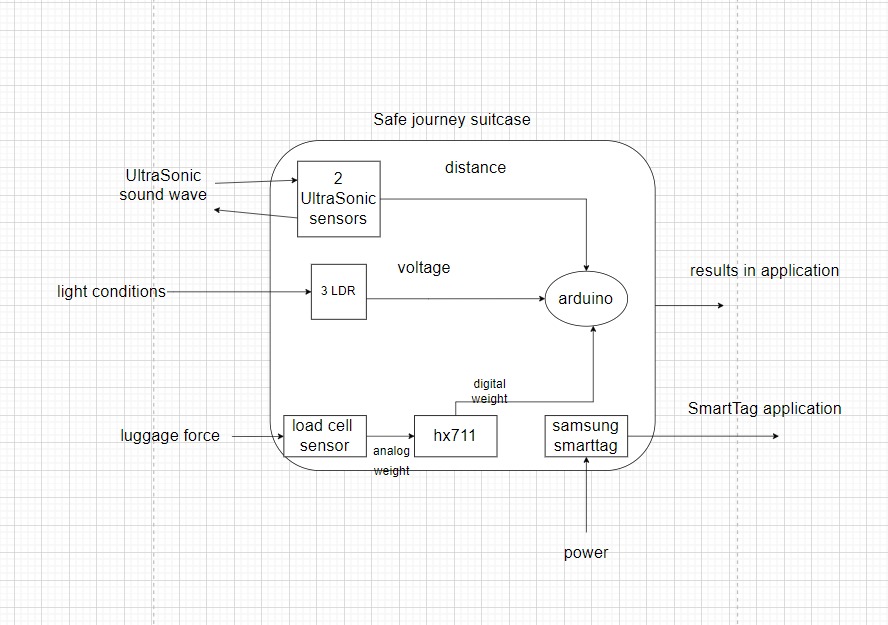


Figure 3.2: functional decomposition level 1

Table 3.2: Table showing inputs, outputs, and functionality of the ultrasonic sensor:

|  |  |
| --- | --- |
| **MODULE** | **ULTRASONIC SENSOR** |
| INPUTS | ULTRASOUND WAVES |
| OUTPUTS | DISTANCE |
| FUNCTIONALTY | Measure the distance between the ultrasonic and the inside of the suitcase |

Table 3.3: Table showing inputs, outputs, and functionality of the Light Dependent Resistor:

|  |  |
| --- | --- |
| **MODULE** | **LDR** |
| INPUTS | LIGHT CONDITIONS |
| OUTPUTS | VOLTAGE |
| FUNCTIONALTY | Measure the light intensity inside the suitcase |

Table 3.4: Table showing inputs, outputs, and functionality of the Load cell sensor:

|  |  |
| --- | --- |
| **MODULE** | **LOAD CELL SENSOR** |
| INPUTS | LUGGAGE FORCE |
| OUTPUTS | ANALOG WEIGHT |
| FUNCTIONALTY | Measure the weight of the luggage |

Table 3.5: Table showing inputs, outputs, and functionality of the HX711:

|  |  |
| --- | --- |
| **MODULE** | **HX711** |
| INPUTS | ANALOG WEIGHT |
| OUTPUTS | DIGITAL WEIGHT |
| FUNCTIONALTY | Amplify the signal of the load cell as well as converting the analog weight to digital weight |

Table 3.6: Table showing inputs, outputs, and functionality of the ESP32:

|  |  |
| --- | --- |
| **MODULE** | **ESP32** |
| INPUTS | DISTANCE  VOLTAGE  DIGITAL WEIGHT |
| OUTPUTS | RESULTS ON APPLICATION |
| FUNCTIONALTY | Ensures the real-time analysis of data and facilitates a secure and efficient connection with our dedicated mobile application. |

Table 3.7: Table showing inputs, outputs, and functionality of the Samsung smart tag:

|  |  |
| --- | --- |
| **MODULE** | **SAMSUNG SMART TAG** |
| INPUTS | POWER |
| OUTPUTS | LOCATION |
| FUNCTIONALTY | Tracking the location of the suitcase |

# Chapter 4-Project Management (Gantt chart, WBS, Cost Analysis)

The project spanned a duration of 5 months, during which we planned and executed their tasks, ensuring an equitable distribution of responsibilities. Leveraging the summer vacation period spanning June and July, we dedicated the efforts to procuring all the requisite sensors and essential materials crucial for the project's development.

As the project progressed into August, we diligently focused on the testing of the load cell sensor. Towards the middle of August, Jouna undertook comprehensive testing of the ultrasonic sensors. While Charbel was responsible in the testing of the light dependent resistors, marking a significant milestone in the project's development process.

The subsequent phase, encompassing the latter part of September, the entire month of October, and the initial week of November, was dedicated to the intricate task of seamlessly integrating all the individual components and sensors into a unified and cohesive system. Commencing this integration process in mid-October, we concurrently initiated the critical phase of software application development, underscoring our commitment to a comprehensive and fully functional end-to-end solution.

As the project neared its final stages, the entirety of November was allocated to testing, troubleshooting, and fine-tuning of the system's functionalities. Additionally, we dedicated concerted efforts to constructing the final project report and preparing the comprehensive presentation, ensuring a well-documented and articulate representation of our innovative solution. With a steadfast commitment to meeting the project deadline, we successfully concluded our collaborative efforts, presenting a comprehensive and fully functional SafeJourney Suitcase system on November 30.

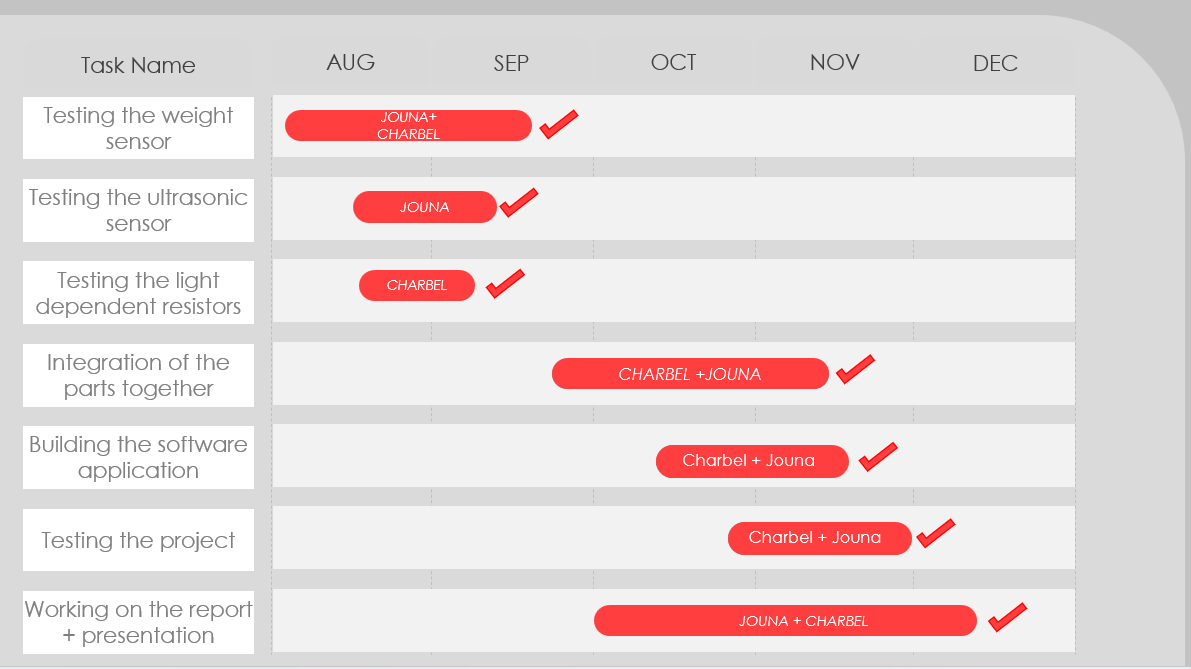


Figure 4.1: Gantt chart

As for the cost, it is detailed in the below table:

Table 4.1: Project Cost

|  |  |  |
| --- | --- | --- |
| **Components** | **Quantity** | **Price (in $)** |
| **Load Cell Sensor | 40kg** | 1 | 10 |
| **HX711 LOAD CELL AMPLIFIER** | 1 | 4 |
| **Ultrasonic HC-SR04** | 2 | 2.50 |
| **Light Dependent Resistor | LDR 6mm GL5506** | 3 | 0.25 |
| **ESP 32** | 1 | 10 |
| **Female to female wires** | 1 pack | 3 |
| **Female to male wires** | 1 pack | 3 |
| **male to male wires** | 1 pack | 3 |
| **Bread boards** | 2 | 4 |
| **Samsung smart tag** | 1 | 30 |
| **Suitcase** | 1 | 20 |
| **Resistor** | 1 | 0.25 |
| **Power bank** | 1 | 15 |
| **Total Cost** | | 112 |

The Work breakdown structure is shown in the table below:

Table 4.2: WBS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Activity | Description | Deliverables | Duration | People | Resources | Predecessors |
| 1 | **Purchasing components** | Get all the components needed for the project | Geeting the suitcase, the load cell, ultrasonic, LDR and smart tag | 2 weeks | Jouna and charbel | Katranji and  Bachir electronics |  |
| 2 | **Testing sensors** | Testing the sensors if they work correctly |  |  |  |  |  |
| 2.1 | Testing load cell sensor | Giving the load cell voltage and seeing the output | Placing a weight on the load cell and the output should be the right weight | 2 weeks | Jouna and charbel | Testing on Arduino IDE | 1 |
| 2.2 | Testing ultrasonic sensor | Giving the ultrasonic voltage and seeing the output if we place an object in front of it | The ultrasonic should give the right distance | 1.5 weeks | Jouna | Testing on Arduino IDE | 1 |
| 2.3 | Testing photoresistor | Giving the LDR voltage and seeing the output. | LDR should give 0 in the dark and a higher number whenever there is light | 1 week | Charbel | Testing on Arduino IDE | 1 |
| 2.4 | Testing smart tag | Seeing if the smart tag works | It should give the location correctly | 3 days | Charbel | Smart things app | 1 |
| 3 | **Hardware implementation** | Integrating parts together and building the prototype |  |  |  |  |  |
| 3.1 | Connecting the circuit | Connecting each sensor to the Esp 32 | ESP should give the right data on the Arduino IDE software | 3 weeks | Jouna and charbel | Testing on Arduino IDE | 2 |
| 3.2 | Fixing everything to the suitcase | Fixing the sensors on the suitcase | The sensors should be fixed on the suitcase and giving the right data | 2 weeks | Jouna and charbel |  | 3.1 |
| 4 | **The Arduino code** | Seeing the right data that the ESP 32 should give | ESP 32 is giving the right data and the Arduino IDE is displaying the data correctly | 1 week | Charbel | Testing on Arduino IDE | 3.1 |
| 5 | **Creating the application** | Creating a software application |  |  |  |  |  |
| 5.1 | Pubsbec file | Modifying the pubspec file | Defining dependencies and putting a background image | 2 days | Jouna | Flutter | 3 |
| 5.2 | Bluecall file | Creating a bluecall file | Managing Bluetooth on the application. | 1 week | Charbel | Flutter | 4 - 5.1 |
| 5.3 | Noti file | Creating a noti file | Managing the notifications on the application | 3 days | Jouna | Flutter | 4-5.1 |
| 5.4 | Main file | Creating the main file | the main interface for the application's home page, handling Bluetooth device scanning, notifications | 1 week | Charbel | Flutter | 5.1-5.2-5.3 |
| 5.5 | Android Manifest file | Modifying this file | Notifications and Bluetooth should work correctly now on the app | 2 days | Jouna | Flutter | 5.4 |
| 6 | **Presentation and report** | Working on a presentation and a detailed report | A very detailed explanation along with a very detailed report | 2 months | Jouna and charbel |  |  |

# Chapter 5- System Failure Modes and Effects Analysis (FMEA)

Risk assessment is a crucial process that allows for the identification, analysis, and mitigation of potential risks within a project. In the context of the current project, several critical failure possibilities have been identified, primarily concerning software bugs and hardware malfunctions. A bug in the software leading to the display of inaccurate weight poses a significant risk to the project's integrity and functionality. To counter this, stringent quality assurance procedures need to be in place, including thorough software testing and regular maintenance checks. Similarly, the risk of a software bug generating false notifications demands robust testing measures, such as boundary testing and error-checking protocols.

Furthermore, potential hardware malfunctions, such as the failure of weight and ultrasonic sensors, could compromise the project's reliability. To minimize these risks, routine calibration and maintenance of the sensors are imperative, coupled with the integration of redundant systems to ensure data accuracy even if one sensor fails. SmartTag malfunction and battery drainage issues in both the system and the SmartTag itself also pose considerable risks. Implementing regular checks along with effective power management strategies for the system can significantly reduce these risks. Moreover, the possibility of physical damage to the hardware during transit or operation especially to the base above the load cell sensor underscores the importance of robust protective measures. Implementing durable casing and shock-absorption techniques can safeguard the hardware from potential damage, ensuring the system's longevity and functionality. By proactively addressing these risk assessment factors and failures, the “SafeJourney Case” can enhance its reliability and sustainability, ensuring a seamless and effective travel experience.

Let’s summarize these risks in some bullet points and assess them in a table:

1. A bug in the software (display inaccurate weight).
2. A bug in the software that gives false notifications.
3. Weight sensor malfunction.
4. ultrasonic sensor malfunction.
5. SmartTag malfunction.
6. SmartTag run out of battery.
7. The system battery run out.
8. Hardware gets damaged while traveling.

Table 5.1: Table showing sensitivity, occurrence and detection ratings as well as the RPN for each risk above.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Sensitivity ratings | 8 | 6 | 8 | 8 | 10 | 8 | 8 | 10 |
| Occurrence ratings | 4 | 4 | 4 | 4 | 2 | 3 | 7 | 5 |
| Detection ratings | 8 | 10 | 6 | 6 | 5 | 8 | 8 | 6 |
| Risk Priority Number RPN ( S x O x D) | 256 | 240 | 192 | 192 | 100 | 192 | 448 | 300 |

# Chapter 6- Hardware Implementation

In the hardware implementation phase of our project, we have integrated various cutting-edge components to ensure comprehensive security and monitoring of the suitcase. The cornerstone of our system lies in the utilization of a load cell sensor combined with the amplifier HX711, which effectively measures the weight of the suitcase.

Moreover, our approach incorporates the deployment of two ultrasonic sensors, intelligently calibrated to detect any increase in distance, thereby instantly alerting users in the event of unauthorized access or tampering with the suitcase. Recognizing the evolving tactics of theft, especially with the rising instances of fabric suitcase vandalism, we have innovatively integrated light-dependent photoresistors across strategic points within the suitcase. This ingenious addition enables our system to detect any alteration in light conditions in case the suitcase was torn, promptly notifying users in cases where traditional security measures might fall short. The addition of the LDRs was necessary since in the case where the suitcase is damaged, the ultrasonic sensors will not detect the distance increasing.

To coordinate the seamless communication between these hardware components and our intuitive software interface, we have seamlessly integrated an ESP32 WROOM DevKit v1, serving as the central nervous system of our operation. This module ensures the real-time analysis of data and facilitates a secure and efficient connection with our dedicated mobile application.

Furthermore, boosting our security measures, we have employed a Samsung Smart Tag, a sophisticated tracking device that enhances the overall safety of the suitcase during travel. This multi-faceted implementation forms the foundation of our hardware infrastructure, promising a comprehensive and foolproof security system for modern-day travelers. In the subsequent sections, we will delve into the complex details of each component and their connections, clarifying their functionalities and seamless interconnectivity within our robust framework.

### 6.1 - Load cell sensor YZC-1B series | 40kg

An electro-mechanical sensor called a load cell is used to gauge weight or force. Its straightforward but efficient design depends on the widely acknowledged relationship between the flow of electricity, material deformation, and applied force. These are very adaptable gadgets that provide reliable and accurate performance in a variety of settings. It should come as no surprise that they are now necessary for a wide range of commercial and industrial operations, from weighing groceries at the register to automating the production of cars. Load cells are finding numerous new and fascinating uses as technology advances, all of which stand to gain from their use. Effective methods for measuring forces and weights are necessary for new developments in robotics, haptics, and medical prosthesis, to mention a few. [5]

This strain gauge-based sensor operates on the principle of the piezoelectric effect, where the deformation of the load cell due to applied force results in the generation of an electrical signal proportional to the applied load.

The YZC-1B series load cell, designed to accommodate a weight range of up to 40kg, comprises four distinct wires, each serving a crucial role in its functioning. The red wire, often denoting the excitation voltage, provides the necessary power supply to the load cell. The black wire, serving as the ground wire, enables the completion of the electrical circuit and ensures stable readings. The white wire, functioning as the signal positive wire, carries the electrical output signal from the load cell to the data acquisition system or instrumentation. Lastly, the green wire, serving as the signal negative wire, helps in completing the signal circuit, ensuring accurate transmission of the output signal. The collaborative functioning of these wires enables the load cell to deliver precise and reliable weight measurements, crucial for maintaining operational efficiency and quality control in diverse industrial environments.

The datasheet for the load cell is provided by this link https://www.katranji.com/tocimages/files/336273-574238.pdf



Figure 6.1.1: Load cell sensor | 40kg

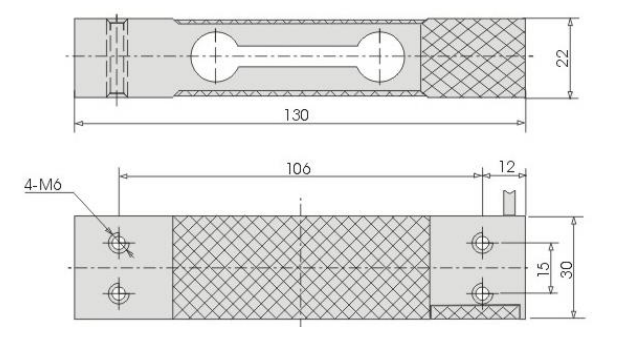


Figure 6.1.2: load cell installation size (mm)

### 6.2 - Load cell amplifier HX711

The necessity of the HX711 amplifier in conjunction with a load cell is rooted in the requirement for precise and accurate analog-to-digital conversion of the small electrical signals generated by the load cell. The HX711, a 24-bit analog-to-digital converter (ADC) specifically tailored for weigh scales and industrial control applications, acts as an intermediary between the load cell and the microcontroller, enhancing the precision and resolution of weight measurements. Operating on the principle of the Wheatstone bridge configuration, the HX711 utilizes a differential input voltage and a programmable gain amplifier to amplify and digitize the minute variations in the electrical output of the load cell.

The HX711 module comprises several essential pins, each playing a critical role in its operation. The E+ (Excitation Positive) and E- (Excitation Negative) pins facilitate the supply of the excitation voltage to the load cell, ensuring the consistent and stable provision of power. The A+ (Signal Positive) and A- (Signal Negative) pins receive the differential analog input signals from the load cell, which are then amplified and converted to a digital signal for further processing. The GND (Ground) pin serves as the reference point for the voltage signals, ensuring a common ground potential for stable signal processing. The DT (Data) and SCK (Serial Clock) pins establish the communication interface with the ESP, enabling the transmission of digital weight data. The VCC (Voltage Common Collector) pin provides the necessary supply voltage to power the HX711 module, ensuring seamless functionality and reliable data acquisition.

The comprehensive integration of these pins within the HX711 module enables the accurate conversion and transmission of weight measurements, crucial for a wide array of industrial and commercial applications, including precision instrumentation, robotics, and load cell-based weighing systems.

The datasheet for HX711 is provided by this link https://www.katranji.com/tocimages/files/349159-518277.pdf

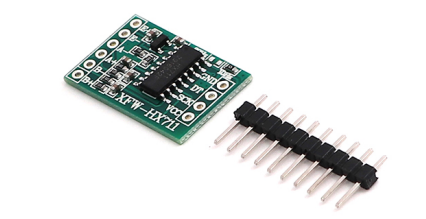


Figure 6.2.1: load cell amplifier HX711

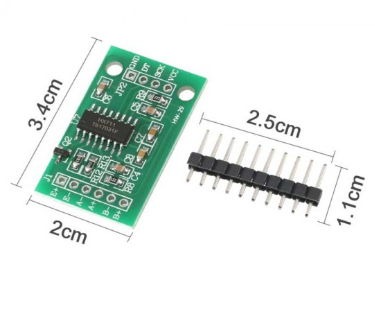


Figure 6.2.2: HX711installation size

### 6.3 - Ultrasonic sensor HC-SR04

High-pitched sound waves that are outside of the human hearing range are known as ultrasounds [6].

Sounds vibrating between 20 times per second (a deep rumbling noise) and 20,000 times per second (a high-pitched whistle) are audible to humans. However, humans cannot hear ultrasound because its frequency exceeds 20,000 Hz. Two ultrasonic transducers make up the HC-SR04 ultrasonic distance sensor. One serves as a transmitter, creating ultrasonic sound pulses at 40 KHz from the electrical signal. As a receiver, the other listens for the pulses that are transmitted. The width of the output pulse that the receiver generates in response to these pulses is proportional to the distance of the object in front of it [6].

The HC-SR04 ultrasonic sensor stands as a pivotal component in various technological applications, particularly in robotics, automation, and distance measurement systems.

This sensor employs ultrasonic waves to gauge the distance between the sensor and an object with exceptional accuracy. Operating on the principle of ultrasonic echo ranging, the HC-SR04 emits a high-frequency sound pulse through its transmitter, and upon encountering an obstacle, it measures the time it takes for the reflected pulse to return to the receiver. This time interval is then used to calculate the distance between the sensor and the object.

The HC-SR04 sensor comprises four essential pins, each serving a crucial function in its operation. The VCC (Voltage Common Collector) pin facilitates the supply of power to the sensor, ensuring its smooth and consistent functionality. The trig (Trigger) pin serves as the trigger input, enabling the initiation of the ultrasonic pulse transmission when a pulse signal is applied. The Echo pin, on the other hand, acts as the output signal, relaying the time taken for the ultrasonic pulse to return to the sensor after bouncing off an object. This time data is crucial for calculating the distance between the sensor and the object. The Gnd (Ground) pin serves as the reference point for the voltage signals, ensuring a common ground potential and stable signal processing.

The collaborative functioning of these pins within the HC-SR04 sensor facilitates precise distance measurements, making it an indispensable tool in a wide range of applications requiring reliable and non-contact distance sensing capabilities.

The distance between the HC-SR04 ultrasonic sensor and the object is calculated using the time-of-flight principle, considering the speed of sound in the medium. The speed of sound is usually considered constant under standard conditions. Given the time interval (T) it takes for the ultrasonic pulse to travel to the object and back, the distance (D) can be calculated using the formula:

*D* = (*T*×*V*)/2

Where:

* *D* represents the distance between the sensor and the object,
* *T* is the time interval it takes for the ultrasonic pulse to travel to the object and back,
* *V* is the speed of sound in the medium (typically in air at room temperature, approximately 343 meters per second).

The datasheet for the ultrasonic sensor can be found in this link <https://www.katranji.com/tocimages/files/357582-190289.pdf>



Figure 6.3.1: Ultrasonic HC-SR04 sensor

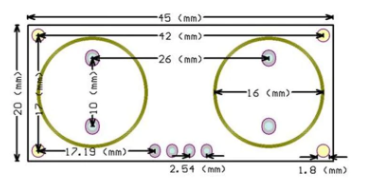


Figure 6.3.2: HC-SR04 installation size (mm)

### 6.4 - Light Dependent Photoresistors GL5506

The GL5506 Light Dependent Photoresistor, commonly known as a photoresistor or LDR (Light Dependent Resistor), is an essential component in various electronic circuits and light-sensing applications.

This semiconductor device operates on the principle of the internal photoelectric effect, where the intensity of incident light modulates the conductivity of the semiconductor material, leading to a change in its resistance. The GL5506 exhibits a high resistance in the absence of light and experiences a significant decrease in resistance when exposed to light, allowing it to serve as an effective light sensor [7]. This property makes it ideal for use in applications such as automatic lighting control systems, cameras, and light intensity measurement devices.

The GL5506 photoresistor typically features two pins. Together, these pins enable the photoresistor to provide a responsive and reliable output signal that accurately reflects the ambient light intensity and giving the corresponding voltage, facilitating seamless integration into a wide array of electronic systems and light-sensitive applications.

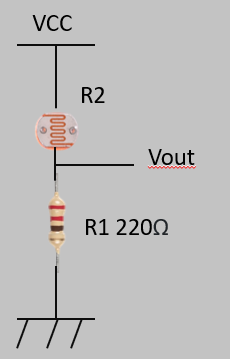


Figure 6.4.1: LDR voltage divider

By figure 10, The formula for calculating the voltage is:

The photoresistor’s datasheet can be found in this link https://www.handsontec.com/dataspecs/sensor/GL55-LDR.pdf

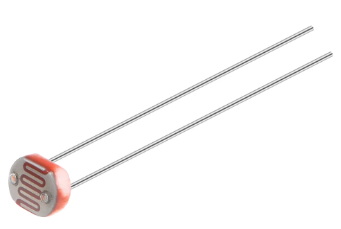


Figure 6.4.2: Light dependent resistor GL5506

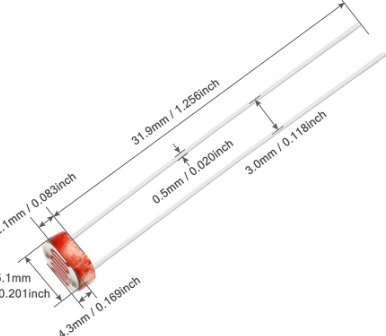


Figure 6.4.3: GL5506 installation size

### 6.5 - ESP 32 WROOM DEVKIT V1

The ESP32 Wroom DevKit v1, a powerful development board based on the ESP32 microcontroller, serves as a versatile and feature-rich platform for IoT (Internet of Things) applications, wireless communication, and embedded systems development [8]. Incorporating a dual-core Tensilica LX6 microcontroller, Wi-Fi and Bluetooth connectivity, and a rich set of peripheral interfaces, the ESP32 Wroom DevKit v1 facilitates seamless integration of IoT functionalities into a wide array of projects. Operating on low power consumption, it enables efficient and robust execution of various tasks, making it an ideal choice for battery-powered applications and projects demanding high computational capabilities.

The ESP32 Wroom DevKit v1 encompasses a multitude of pins, each dedicated to specific functionalities crucial for its operation and versatility. These pins include digital input/output pins (GPIO), analog-to-digital converter (ADC) pins for analog sensor interfacing, pulse-width modulation (PWM) pins for controlling analog circuits, and serial communication pins such as UART, SPI, and I2C for seamless interaction with external devices.

Additionally, the board features dedicated power supply pins for stable voltage regulation and ground pins for reference potential, ensuring reliable and consistent performance. Moreover, the integrated Wi-Fi and Bluetooth modules enable seamless wireless communication, facilitating data exchange and remote control functionalities. The comprehensive pin layout and rich set of features make the ESP32 Wroom DevKit v1 an indispensable tool for IoT development, home automation, smart appliances, and various other projects demanding robust connectivity and processing capabilities.

The datasheet regarding the ESP 32 is provided by this link https://www.mouser.com/datasheet/2/891/esp-wroom-32\_datasheet\_en-1223836.pdf



Figure 6.5.1: Esp 32 Wroom devkit v1

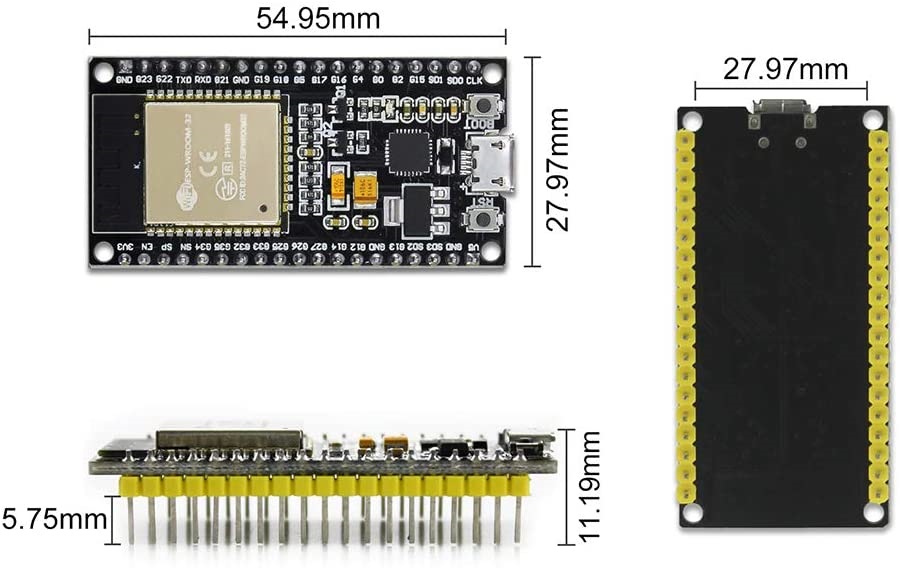


Figure 6.5.2: esp32 installation size

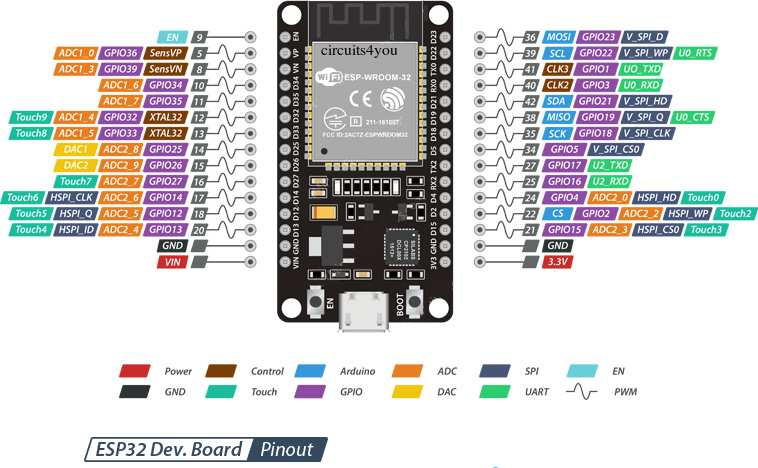


Figure 6.5.3: ESP32 Pinout

### 6.6 - Samsung smart tag

Samsung SmartTag is a cutting-edge tracking device that allows users to locate and keep track of their valuable belongings through a connected mobile application. Built with the latest Bluetooth technology, the SmartTag functions as a compact and versatile beacon, enabling users to attach it to commonly misplaced items such as keys, wallets, or bags. The device operates by establishing a wireless connection with the user's smartphone via the SmartThings Find app, facilitating real-time tracking and proximity alerts. The SmartTag connect to the Samsung account, and can never be connected to another Samsung account, however, the user can share it with other accounts enabling them to use its functionality. The device can then be located with SmartThings mobile app, using Bluetooth Low Energy.

When the gadget is within the 120-meter Bluetooth range, it can use its built-in piezoelectric speaker to play a ringtone at a volume of 85 to 96 dB (unobstructed) to acoustically notify the user of its precise location. Samsung's SmartThings Find Network can be used to locate an item even if it is not within Bluetooth LE range. This network uses the GPS location of nearby Samsung Galaxy phones and internet connections to anonymously locate and identify the location of the SmartTag for its owner. Additionally, the gadget contains a programmable button that can be used to operate smart home appliances that are compatible with SmartThings [9].

The Samsung SmartTag updates its location using not only Samsung phones but also other Android devices that have the SmartThings app installed and are part of the SmartThings Find network. This network is a crowd-sourced system that uses the power of nearby devices to help locate your SmartTag.

Here's how it works:

SmartThings Find Network: When a SmartTag is out of Bluetooth range of its owner's phone, it sends out a signal that can be detected by other devices in the SmartThings Find network.

Samsung and Other Android Devices: While Samsung phones are a significant part of this network, other Android devices with the SmartThings app installed and enabled also contribute to the network. These devices can anonymously and securely pick up

the signal from the lost SmartTag.

Privacy and Security: The process is encrypted and anonymous, ensuring the privacy and security of all users within the network.

Location Update: If a SmartTag is detected by someone else's device within this network, its location is updated and the owner is notified of its new location through the SmartThings app.

It's important to note that while other Android devices can participate in the SmartThings Find network, the overall effectiveness and range of features may be more robust within the Samsung ecosystem, due to better integration and optimization with Samsung devices and services.



Figure 6.6.1: Samsung SmartTag



Figure 6.6.2: Samsung smart tag installation size

39.1 mm

### 6.7 - Connecting everything to ESP32 and building the prototype

The process of integrating various components into the ESP32 microcontroller begins with the connection of the load cell to the HX711 module, where the load cell's red wire, representing the excitation voltage, connects to the E+ (Excitation Positive) pin, and the black wire, functioning as the ground wire, links to the E- (Excitation Negative) pin. Furthermore, the white wire, serving as the signal positive wire, is connected to the A+ (Signal Positive) pin, and the green signal negative wire connects to the A- (Signal Negative) pin on the HX711 module.

While we were building our prototype, we wanted to place the load cell with a strong base, so it doesn’t break with the weight pressure on it. So, we decided to put an Aluminum base that can fit inside the suitcase on the bottom without taking much space of the suitcase. We strongly bolted this base to the case then we firmly fixed the load cell onto this base. Next, we placed a wooden plate on top of it with 2 hex nuts leaving space between the plate on the load cell, so they don’t touch preventing us from getting accurate measures.

Although, when we were trying to calibrate the load cell, the plate was bending and touching the load cell and consequently we weren’t getting an accurate number when the weight was above 6kg. That’s why we decided to add 2 metal bars above the metal hex nuts, so the plate won’t bend and touch the load cell.

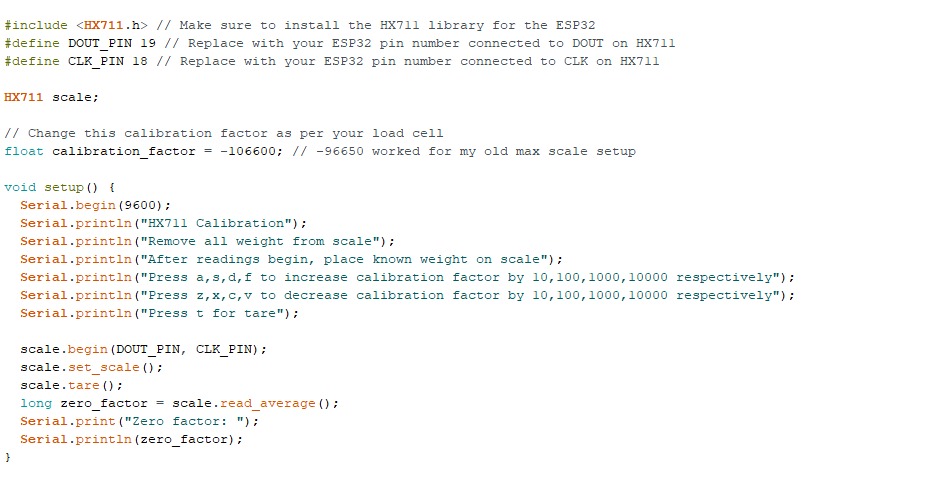


Figure 6.7.1: calibrating HX711 pt1

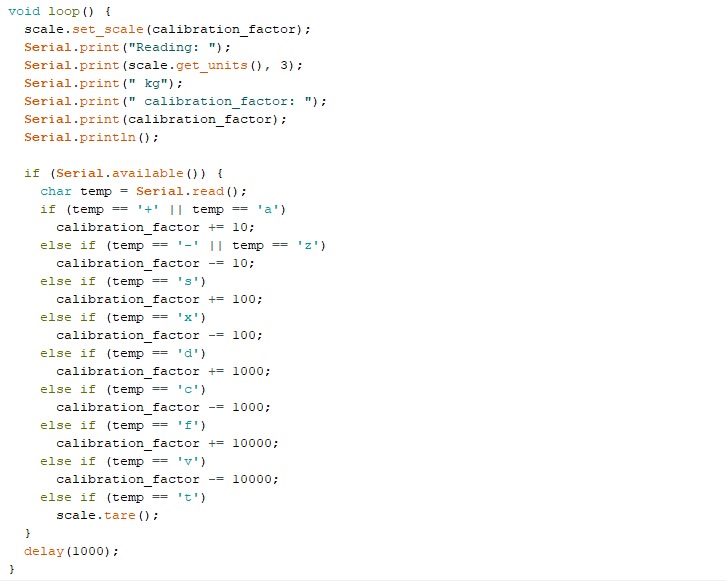


Figure 6.7.2: calibrating HX711 pt2

The figures above (figure 6.7.1-6.7.2) are the calibration of the HX711. We start by removing all weights from the load cell and then we start putting known weights on it. We adjust the calibration factor, by entering a character as a serial input (the character are found in the setup(),and they change the factor as printed in the message), so it give us the most accurate weight.

For the connection between the HX711 and the ESP32, the GND pin of the HX711 is connected to the GND pin of the ESP32, ensuring a common ground reference. The VCC pin of the HX711 is linked to the Vin pin of the ESP32 to provide a stable power supply. The DT (Data) pin of the HX711 is connected to D19 (GPIO 19) of the ESP32, facilitating data communication between the two components, while the SCK (Serial Clock) pin of the HX711 connects to D18 of the ESP32, ensuring synchronized data transmission.

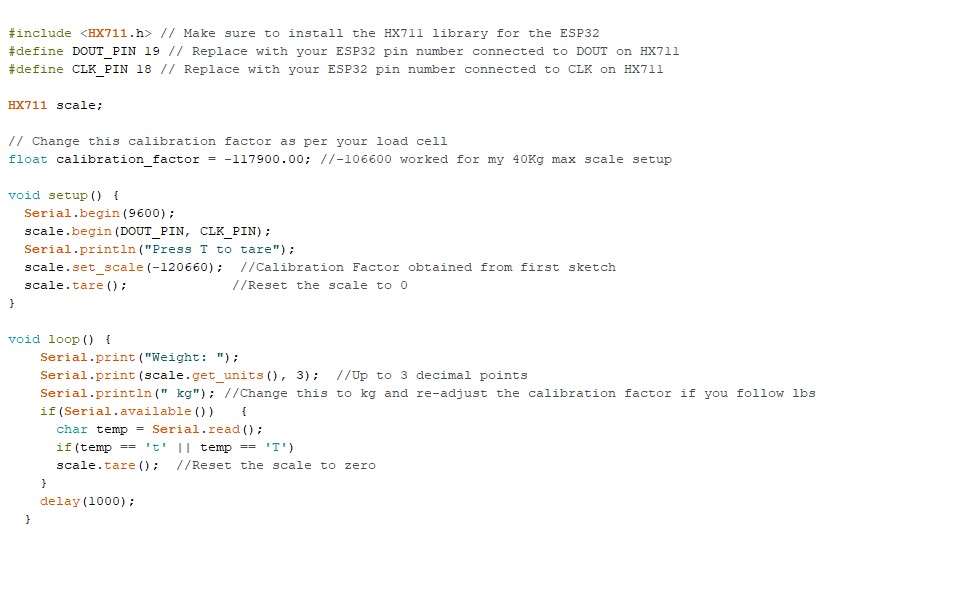


Figure 6.7.3: The load cell Arduino code

In figure 6.7.3, is the code to test the load cell.

In the case of the two ultrasonic sensors, both sensors have their GND pins connected to the GND pin of the ESP32, while their VCC pins are connected to the Vin pin. The trigger and echo pins of the first ultrasonic sensor are linked to D26 and D27, respectively, while the corresponding pins of the second ultrasonic sensor connect to D22 and D23, enabling distance measurement and obstacle detection. They are on both sides of the bottom of the lid compartment of the suitcase. We stitched them to the fabric of the suitcase and fixed them firmly with electrical tape.

A computer code with text

Description automatically generated with medium confidence

Figure 6.7.4: Ultrasonic sensor Arduino code

In figure 6.7.4, is a simple code for testing the ultrasonic sensor.

Regarding the photoresistors, their LDR pins are connected to the Vin pin of the ESP32, with the other pin connected to a 220-ohm resistor. This pin (where LDR and resistor are connected) is linked to the D2 pin of the ESP32, and its remaining pin is connected to GND. All photoresistors utilized in the project follow the same configuration, connecting to the same pins of the ESP32, facilitating light intensity measurements and ambient light sensing.

For the LDRs, we distributed them in the case as follows, one on the left side of the suitcase, one on the top and one on the right slightly above the wooden plate. We fixed these LDRs using strong electrical tape.



Figure 6.7.5: Photoresistor Arduino code

In figure 6.7.5, is a simple code to test the photoresistors.

All the other components, like the breadboard, the ESP32, The HX711 are all placed at the bottom of the suitcase next to the load cell.

All these codes were integrated into one which is explained in chapter 7 “Software Application”.

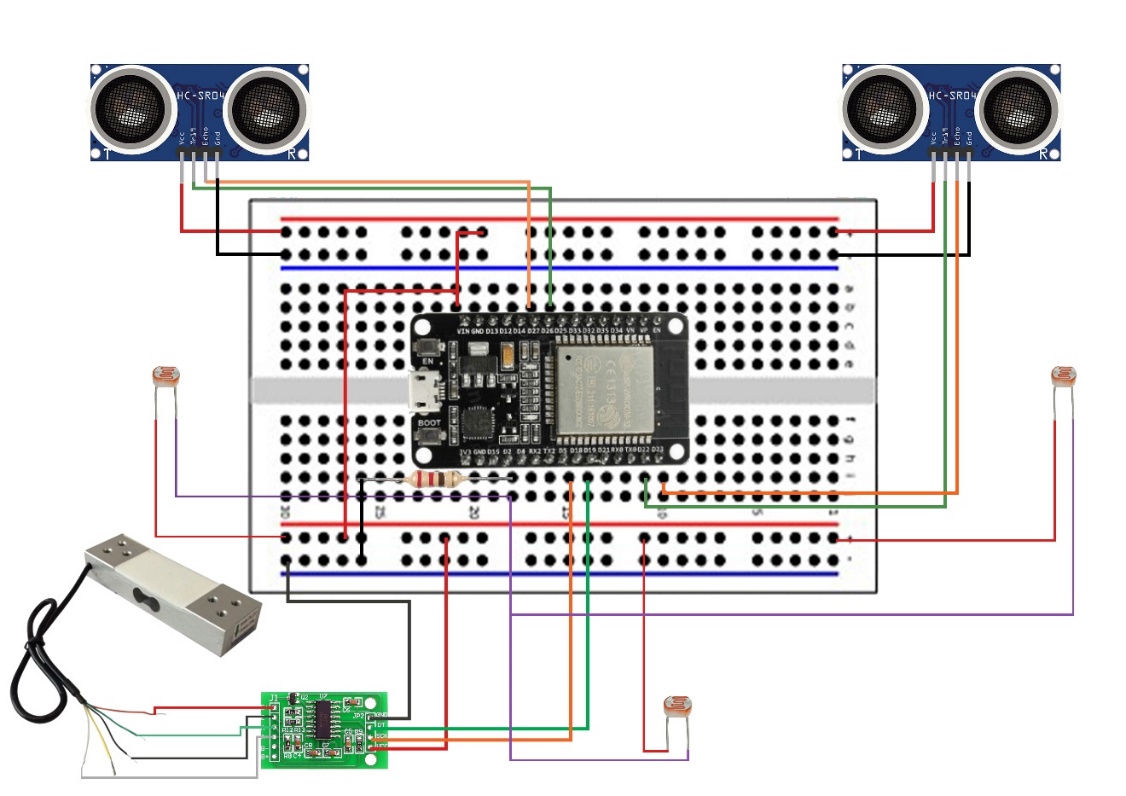


Figure 6.7.6.: the full circuit

Figure 6.7.6 illustrates the full circuit showing all the connections talked about before in this chapter.

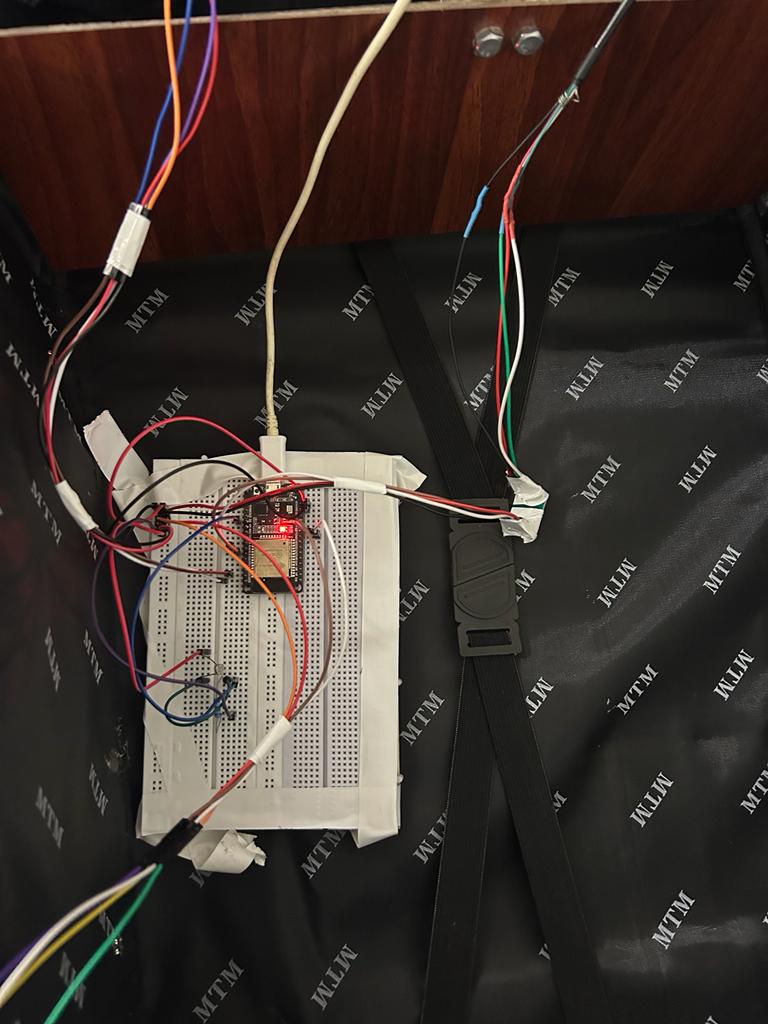


Figure 6.7.7: testing out all the connections together pt1

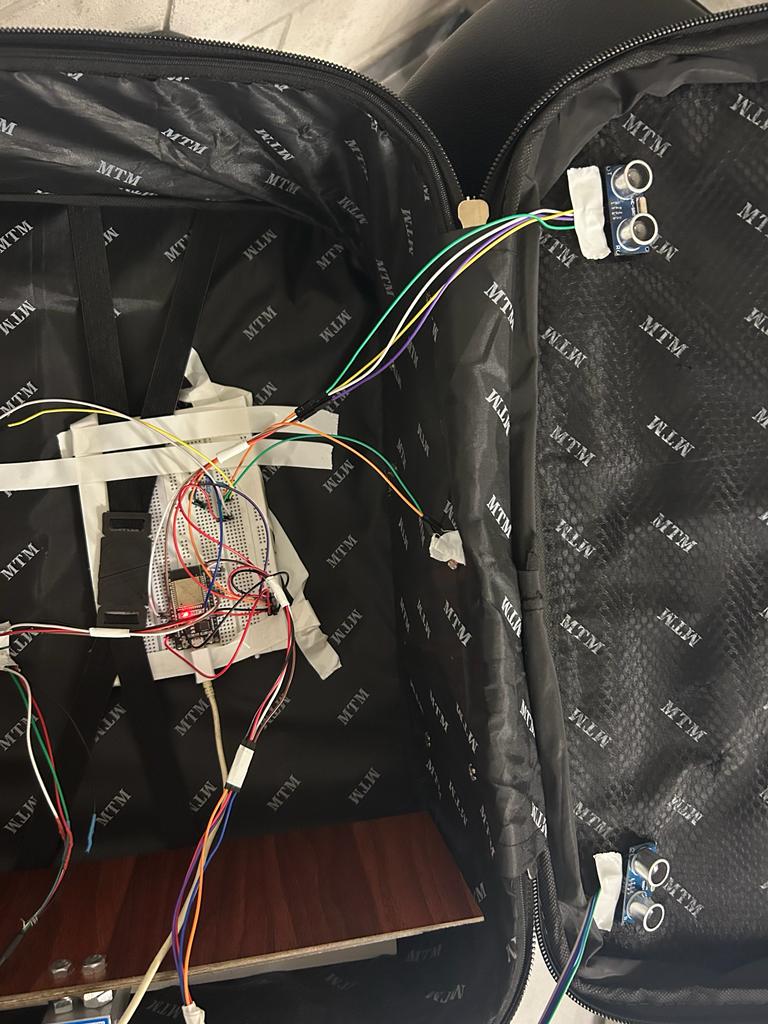


Figure 6.7.8: testing out all the connections pt2



Figure 6.7.9: case closed

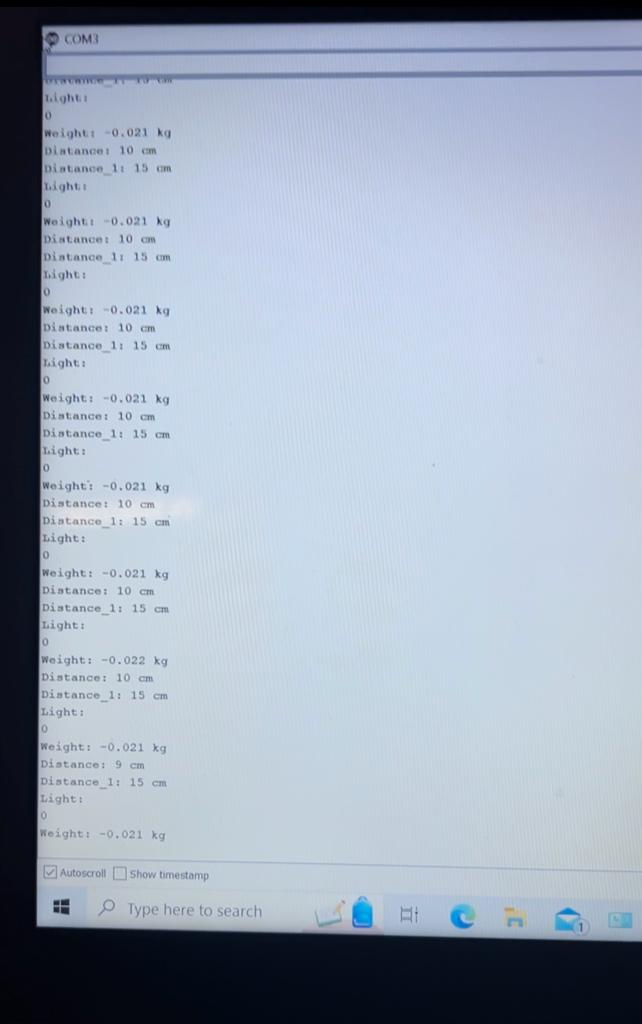


Figure 6.7.10: results from case closed

As seen in the results from figure 6.7.7. through 6.7.10, the light sensors are giving the value 0, which means no light in case the suitcase is closed. In addition, we can notice as well, the ultrasonic sensors giving a small distance between 9 cm to 15 cm. note that in this testing, no weight was put on the load cell and the suitcase was empty.



Figure 6.7.11: case opened

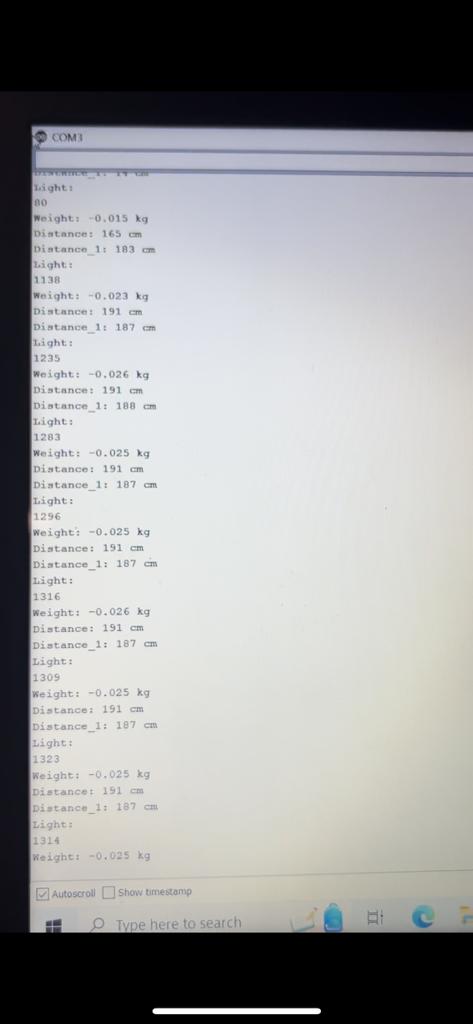


Figure 6.7.12: results of case opened

As seen in these results (figure 6.7.11-6.7.12), when the case was opened the light sensors are giving a huge number (1296 – 1323) indicating that there is light now and that the case is opened. And we can also notice how the ultrasonics are also giving a big number compared when the case was closed (187-191 cm) indicating that the case was opened.

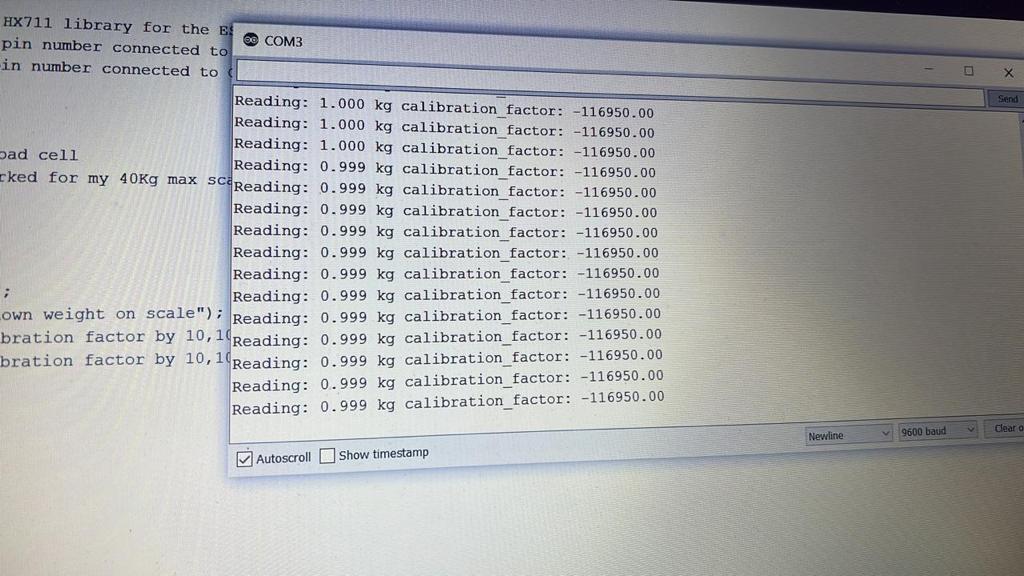


Figure 6.7.13: testing out a 1kg weight on the load cell

In figure 6.7.13, is what the load cell is giving values when 1kg weight was put on it.

# Chapter 7- Software application

In the software application section, our initial strategy involved utilizing the Arduino IDE software and establishing a connection between our system and the Flutter application through the serial communication method. However, we encountered challenges with this approach, which will be elaborated upon in subsequent discussions. To address these issues, we transitioned to an alternate methodology by employing the Arduino IDE and establishing connectivity with the Flutter application using the Bluetooth Low Energy (BLE) protocol. This shift allowed us to overcome the encountered obstacles and provided a more robust and effective means of interaction between our hardware system and the Flutter application, ensuring smoother data transmission and communication.

### 7.1 - Using serial connection

The following code is the Arduino IDE software code when we used the serial connection method.

A screenshot of a computer code

Description automatically generated

Figure 7.1.1: serial IDE code pt1

The first part of the code shown in figure 7.1.1 starts by including the libraries needed to work with the ultrasonic sensor (Bonezegei\_HCSR04.h), the weight sensor along with the HX711 (HX711.h) and finally a library needed to establish a Bluetooth communication (BluetoothSerial.h).

Next, we define the pins connected to the esp32; pins 18-19 for the HX711, pins 26-27 for the first ultrasonic, pins 22-23 for the second ultrasonic and pin 2 for the photoresistors.

The “HX711” object is initialized, configuring it to work with specific data and clock pins. “ultrasonic and ultrasonic\_1” are also initialized to work with the ultrasonic sensors.

In the “setup()” function, the serial communication is started at a baud rate of 9600, allowing communication between the ESP32 and a computer via USB. The Bluetooth serial communication is also initiated with a device name of "ESP32\_BT". The HX711 sensor is initialized and calibrated for weight measurements. This code prepares the microcontroller and sensors for data collection and communication with other devices using Bluetooth.



Figure 7.1.2: serial IDE code pt2

The second part of the Arduino IDE code shown in figure 7.1.2 is a loop that constantly gathers data from the different sensors used. It retrieves the distance from the ultrasonic sensors and stores them in variables d and d1.

Additionally, it reads the light intensity from the photoresistors and stores it in a variable called light. Furthermore, it collects weight data from the HX711 weight sensor and stores it in the weight variable.

The function then prints these values to the serial monitor, displaying the distance readings from both ultrasonic sensors in centimeters, the light intensity value, and the weight in kilograms with three decimal places.

Also, it checks if there's a Bluetooth connection established using the SerialBT.connected() function. If a Bluetooth connection exists, the function sends this sensor data in a specific format via Bluetooth. Each sensor reading is labeled (D1, D2, L, W) and followed by the respective sensor value.

These values are sent sequentially with periods as separators, and the data ends with a period to mark the completion of a data entry. If there's no Bluetooth connection, it prints a message indicating the absence of a Bluetooth connection.

Finally, the function includes a delay of 1000 milliseconds (1 second) using delay(1000) to regulate the rate of data transmission and processing.

Regarding the flutter code, we searched and tried many Bluetooth serial/classic libraries that we can find on pub.dev where we can find all the libraries for flutter. We chose a library called flutter\_bluetooth\_serial, which is the best rated and most used library for serial Bluetooth. In it we go its GitHub repository where we can find examples to test it and then integrate it into our codes. This is the link for the repository <https://github.com/edufolly/flutter_bluetooth_serial> and in this link, there is an example from which we got the flutter code for the serial Bluetooth connection.

However, this code was running with no errors on flutter but whenever we open the application it crashes. In Github, there is a Issues tab, in it a discussion box is open where people talk about the same issue that they are facing with the code. And from these discussions, it was found that this code crashes on Android 12, but works fine with android 11 and older versions, and till now, no solution was found, so we decided to use BLE.

### 7.2 - Using the BLE method

After the issue we faced, we decided to use BLE which is Bluetooth Low Energy. The following code is the Arduino IDE code to send the data through BLE

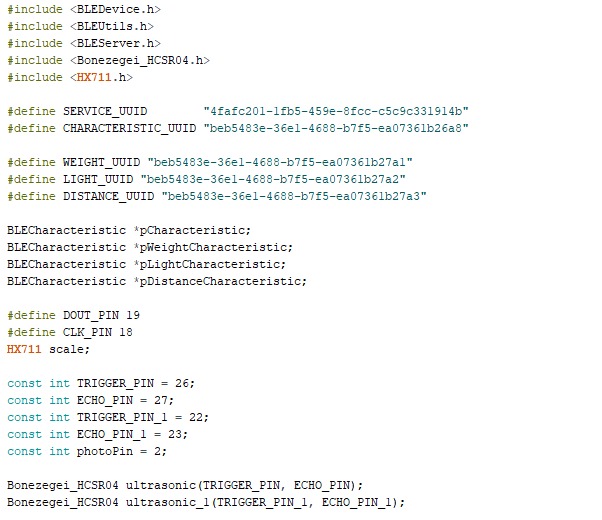


Figure 7.2.1: BLE IDE code pt1

The first part of the code (figure 7.2.1) configures an ESP32 microcontroller to utilize many sensors and Bluetooth Low Energy (BLE) capabilities. It contains the libraries required for BLE communication, as well as the weight sensor (HX711), ultrasonic sensors, and the definition of particular UUIDs (Universally Unique Identifiers) for BLE features and services.

The UUIDs defined represent the service and characteristics that the ESP32 will offer when connected via BLE. SERVICE\_UUID is a unique identifier for the main service, and CHARACTERISTIC\_UUID is a unique identifier for the main characteristic used to exchange data. To further characterize particular data kinds that will be broadcast over BLE, three distinct characteristic UUIDs (WEIGHT\_UUID, LIGHT\_UUID, and DISTANCE\_UUID) are defined: weight, light intensity, and distance measurements.

The code then initializes BLE characteristics (‘BLECharacteristic’), weight characteristic (‘pWeightCharacteristic’), light characteristic (‘pLightCharacteristic’) and finally a distance characteristic (‘pDistanceCharacteristic’).

Then different pins were defined (TRIGGER\_PIN, ECHO\_PIN, TRIGGER\_PIN\_1, ECHO\_PIN\_1, photoPin). The HX711 object (‘scale’) is initialized, and objects for two ultrasonic sensors (‘ultrasonic’ and ‘ultrasonic\_1’) are created using the ‘Bonezegei\_HCSR04’ library, setting up triggers and echo pins for each sensor.

A screen shot of a computer code

Description automatically generated

Figure 7.2.2: BLE IDE code pt2

In the part of the code above (figure 7.2.2), we created a class ‘MyServerCallBacks” that is responsible for restarting the esp so we can reconnect to it without the need to turn the esp off and on again, since if we disconnect from the esp32, we were unable to reconnect to it without restarting it.

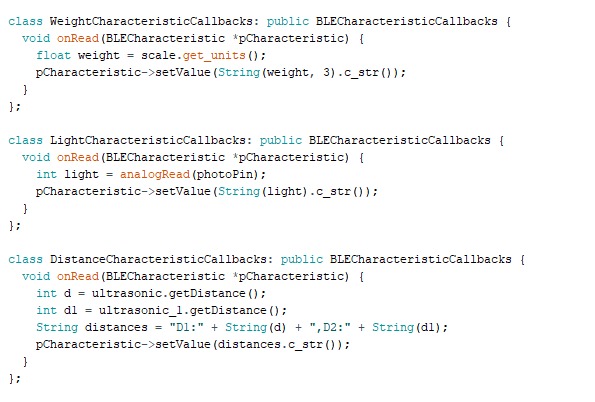


Figure 7.2.3: BLE IDE code pt3

Next, 3 classes were created (figure 7.2.3), one for each type of sensor (‘WeightCharacteristicCallbacks’,‘LightCharacteristicCallbacks’, ‘DistanceCharacteristicCallbacks’). In these classes, ‘onRead’ function is triggered when either the weight, the light and the distance characteristic is read. It then retrieves the specified data needed from the sensors and send it by a characteristic when the flutter app is called.



Figure 7.2.4: BLE IDE code pt4

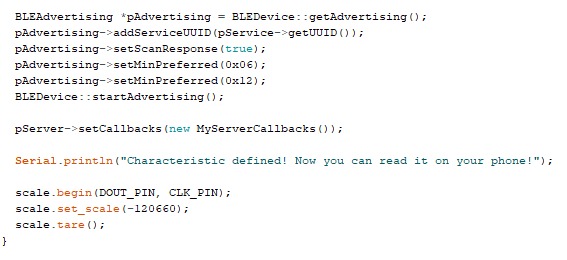


Figure 7.2.5: BLE IDE code part5

The above code (figure 7.2.4-7.2.5) starts by initializing serial communication at a baud rate of 115200, allowing interaction with the serial monitor for debugging purposes.

The code then initializes BLE functionality by initializing a BLE device with the name "Safe Journey SuitCase" using BLEDevice::init(). It creates a BLE server and a service with a defined UUID (SERVICE\_UUID).

Callbacks for each characteristic are set using the respective callback classes (WeightCharacteristicCallbacks,LightCharacteristicCallbacks,DistanceCharacteristicCallbacks). These callbacks define the behavior when the characteristics are read.

After configuring the characteristics, the service is started, and advertising settings are configured to make the ESP32 discoverable over BLE. The ESP32 then begins advertising itself as a BLE device.

The setup completes by initializing the scale (HX711) and setting it up to measure weight using particular data and clock pins. This prepares BLE capabilities and sensor readings for BLE-based data transmission and communication.

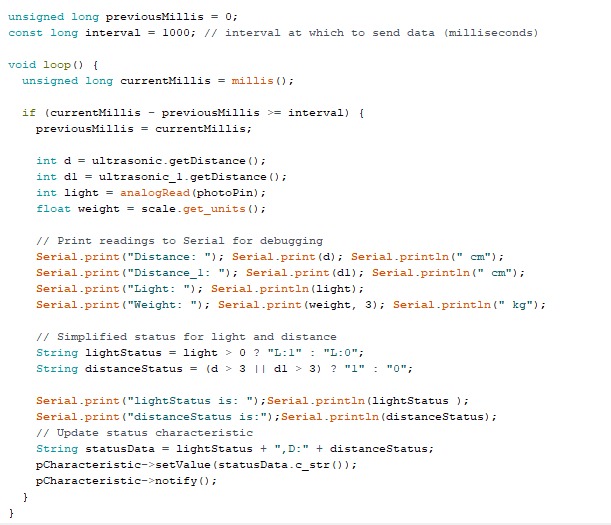


Figure 7.2.6: BLE IDE code pt6

At the core of this segment in figure 7.2.6 is a timed routine, governed by the interval variable, which determines the frequency of sending data over BLE. The millis() function is employed to keep track of time intervals.

Within this routine, sensor readings are collected: ultrasonic distance measurements (d and d1), a photoresistor light intensity reading (light), and weight from the HX711 sensor (weight). These readings are printed to the Serial monitor.

Light and distance status were introduced, and they play the role of flags. After determining these statuses, the code updates a BLE characteristic (‘pCharacteristic’’) with a combined status string (‘statusData’) containing both the light and distance statuses. This characteristic is then notified (‘pCharacteristic->notify()’) to inform connected devices of the updated status. The ‘notify’ function sends the data all the time without flutter calling them.

Now for the flutter code, when we first create a new project in flutter, many files including the 3 files called “pubspec.yaml” and “AndroidManifest.xml” and “main.dart” are automatically created. We added 2 files: “Bluecall.dart” and “noti.dart”. These files will be discussed below.

## 7.3 - Pubspec.yaml

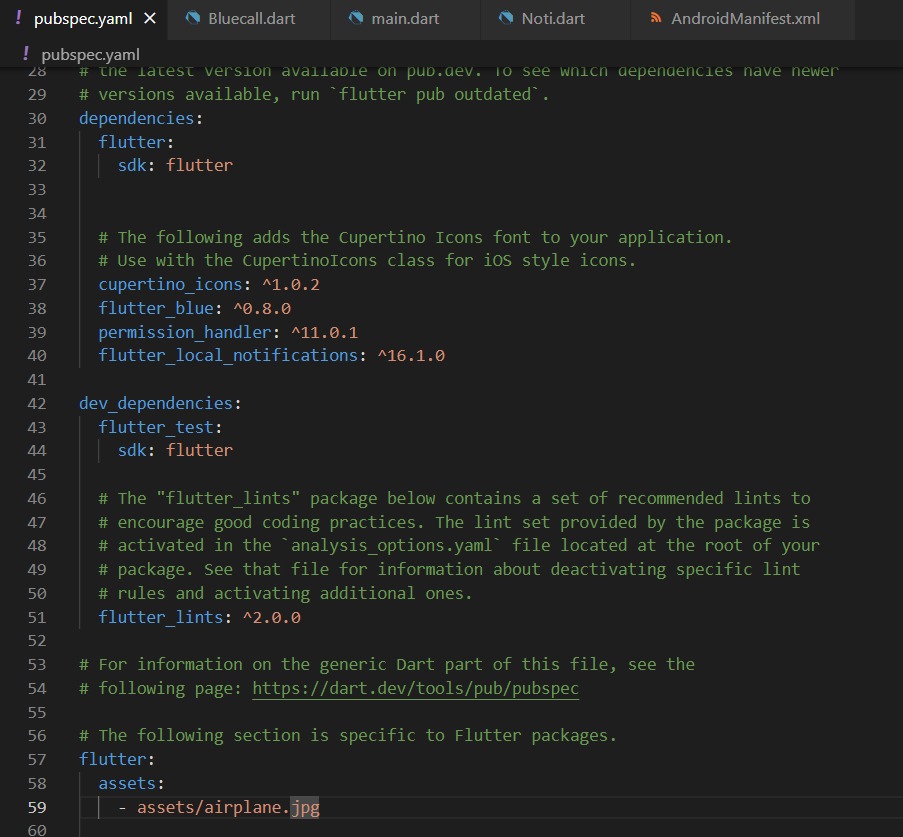


Figure 7.3.1: pubspec code

In this part of the code above (figure 7.3.1), we define our dependencies: flutter\_blue, permission\_handler, and flutter\_local\_notifications. We also add assets for the background image used in the app. We the use flutter pub get command to install all the libraries needed.

## 7.4 - Bluecall.dart

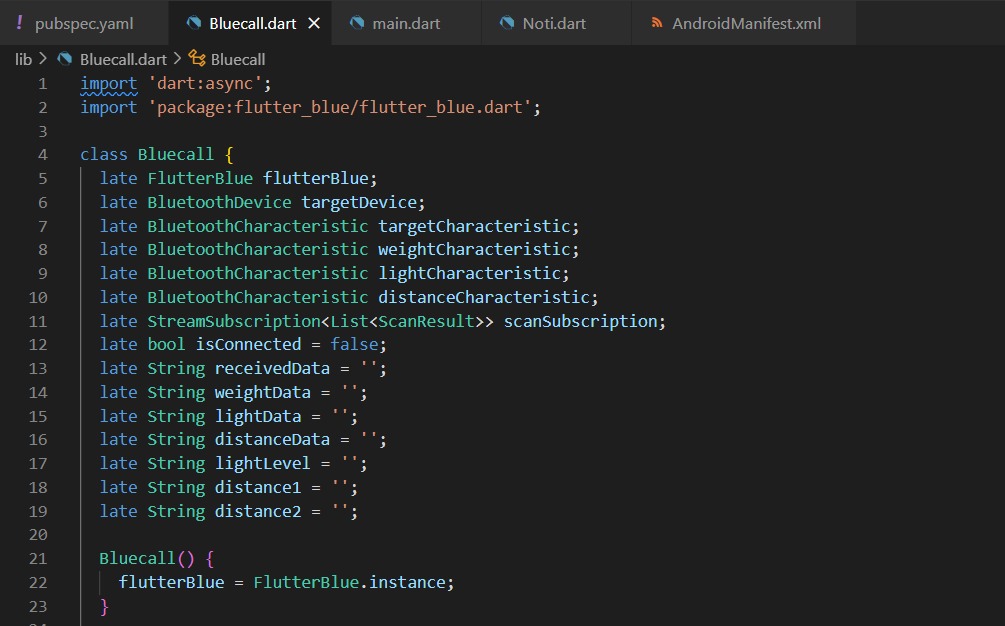


Figure 7.4.1: Bluecall code pt1

The part of the code in the figure 7.4.1 sets up the necessary variables and instances required for managing Bluetooth-related operations, such as device discovery, connection, and communication with specific characteristics of a Bluetooth device using the FlutterBlue package in a Flutter application.

A computer screen with text

Description automatically generated

Figure 7.4.2: Bluecall code pt2

These methods in figure 7.4.2 provide functionality for initiating and stopping Bluetooth device scanning within the Flutter application. The startScan() method triggers the scan for nearby devices, while stopScan() halts the ongoing scanning process when necessary.

The method includes an await keyword, indicating that it's an asynchronous function. It awaits the completion of the scan, which is set to last for a duration of 4 seconds (timeout: Duration(seconds: 4)), without blocking the main thread of the application (UI remain responsive).

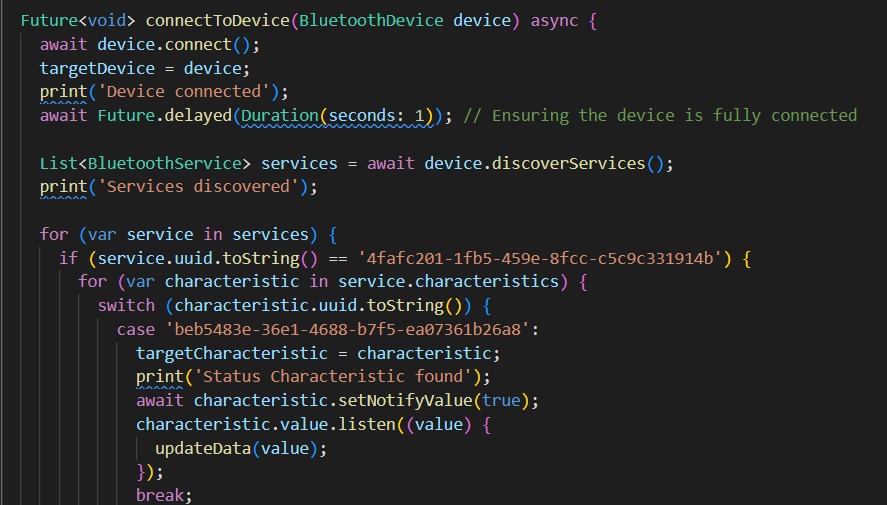


Figure 7.4.3: Bluecall code pt3

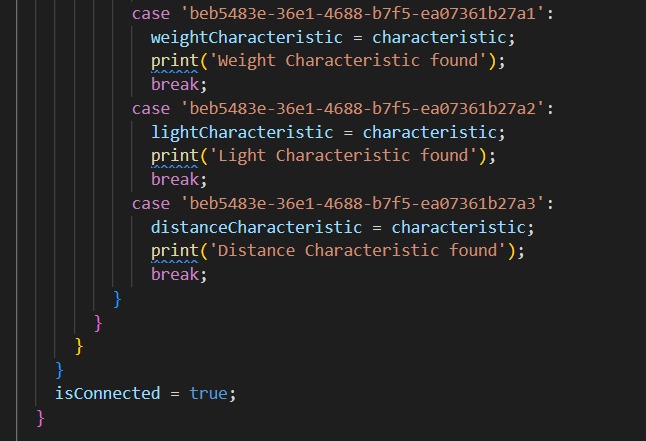


Figure 7.4.4: Bluecall code pt4

In figure 7.4.3 and 7.4.4, ConnectToDevice() is a Dart method that connects a Flutter application to a designated Bluetooth device. First it establishes a connection with the selected device and prints 'Device connected' to indicate that it has done so (the prints are for debugging purposes, it does not appear to the user of the app, and only appear in the terminal). It waits a second to make sure there's a steady connection before continuing. Next, it finds out what services the connected device offers. It searches these services for particular traits connected to UUIDs, or unique identifiers. It recognizes and sets the target attributes of the Flutter app for various reasons after locating matching UUIDs.

it identifies 'Weight', 'Light', and 'Distance' characteristics, assigning them to their respective variables for future interactions. Once these processes are complete, it sets a flag indicating the successful connection to the device, marking isConnected as true.

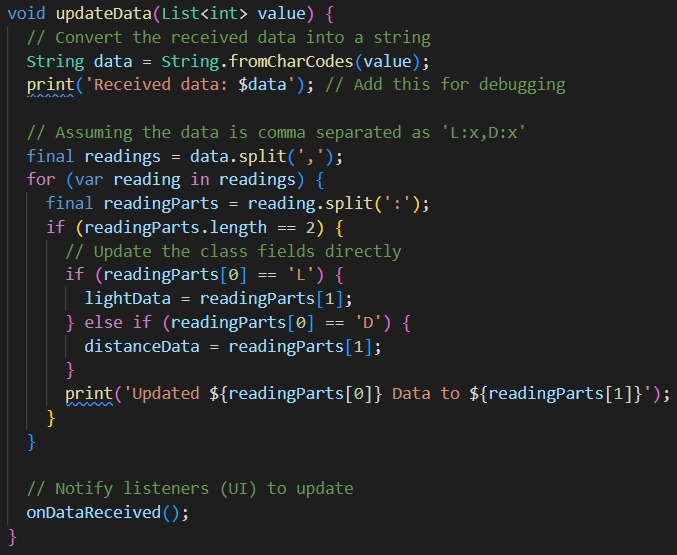


Figure 7.4.5: Bluecall code pt5

This procedure in figure 7.4.5 basically parses the received data string from the notification (which are the light and distance flags that are continuously being sent from the esp32), divides it into discrete readings, and modifies two variables (lightData and distanceData) according to the received light and distance values. Additionally, it calls the onDataReceived() method, which updates the presented information by informing the user interface of the new data.



Figure 7.4.6: Bluecall code pt6

These methods shown in figure 7.4.6 are in charge of starting read operations from particular Bluetooth properties and updating weightData, lightLevel, distance1, and distance2 variables with the information they get. Additionally, they alert any listeners to update the user interface or take actions in response to the most recent data.

A screen shot of a computer code

Description automatically generated

Figure 7.4.7: Bluecall code pt7

onDataReceived (figure 7.4.7) is a function that serves as a placeholder and is later assigned to another function elsewhere in the code to trigger actions upon data reception.

The disconnectFromDevice() method is responsible for disconnecting the app from the currently connected Bluetooth device (targetDevice) and then sets the isConnected flag to false, indicating that the app is no longer connected to any device.

## 7.5 - Noti.dart

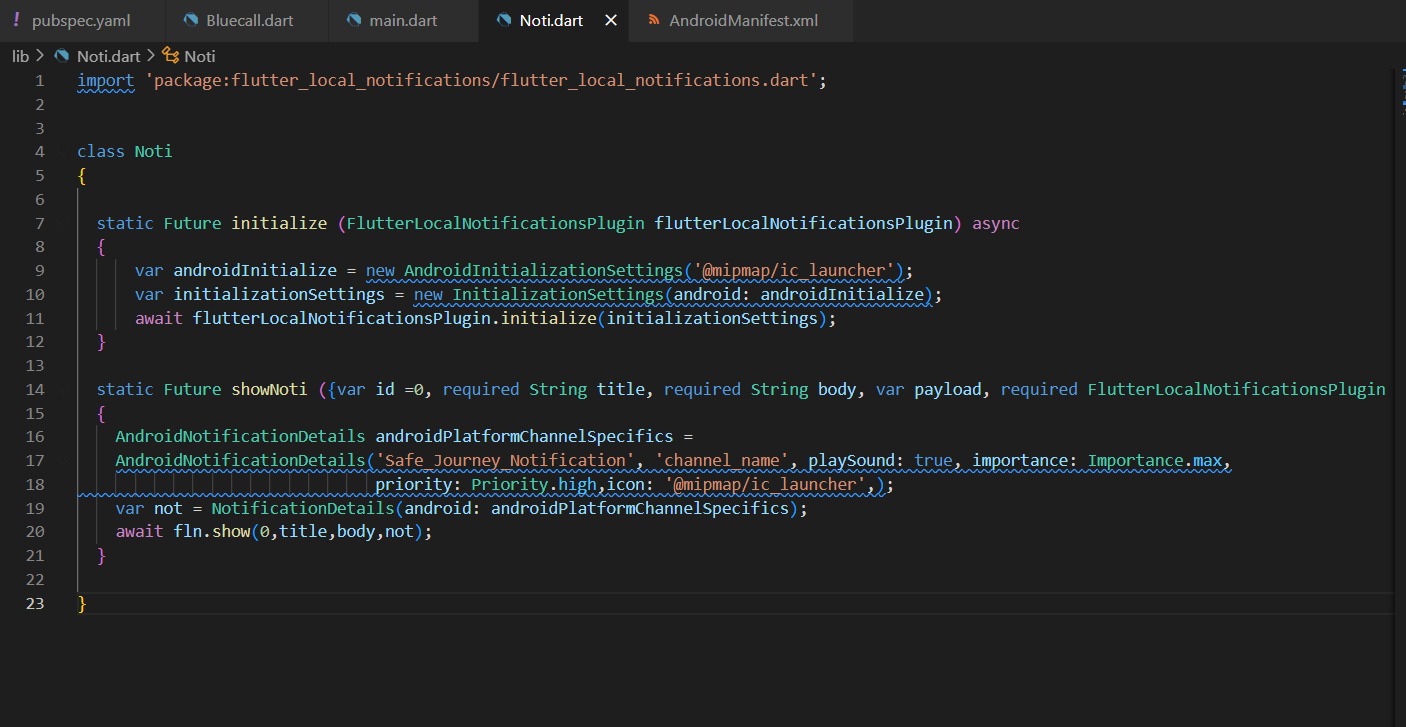


Figure 7.5.1: Noti code

This part of the code in figure 7.5.1 defines a class called Noti, which encapsulates methods to manage notifications using the flutter\_local\_notifications package in a Flutter application.

It uses two techniques:

initialize(): Prepares the system for notifications. It uses settings unique to Android to configure the look and feel of notifications, including the app icon and tone.

showNoti(): Displays a message on the phone's display. The notice's title and content are handled by this function, which also uses the initialize() settings to show the message. It functions similarly to a tool that converts your message into a notification that appears on your phone.

## 7.6 - AndroidManifest.xml

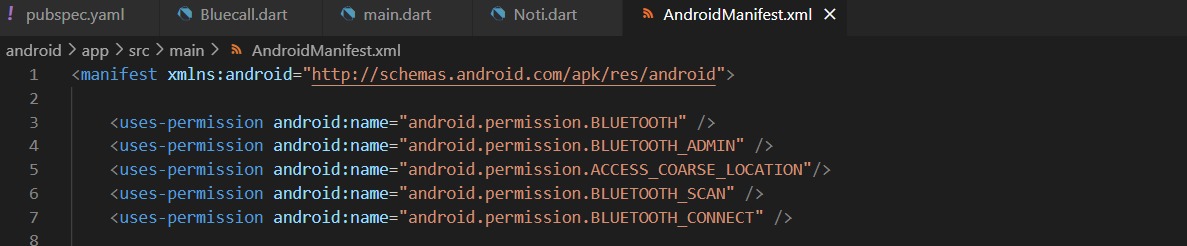


Figure 7.6.1: AndroidManifest code pt1

In this code (figure 7.6.1), we ask for permissions that are needed for Bluetooth to work.

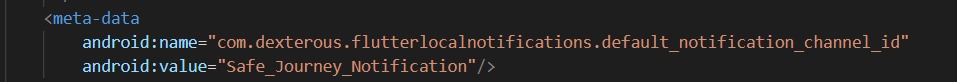


Figure 7.6.2: AndroidManifest code pt2

And then we add this meta-data illustrated in figure 7.6.2, that are needed for the notification to work. We should make sure here that android:value is the same as the channel\_id which is defined in the showNoti function in the Noti.dart file.

## 7.7 - main.dart

This is the main code of flutter. When we run “flutter create project\_name”, the main.dart is automatically created, and filled with starter code or boilerplate code. We modify on it by adding the packages and changing the page state.

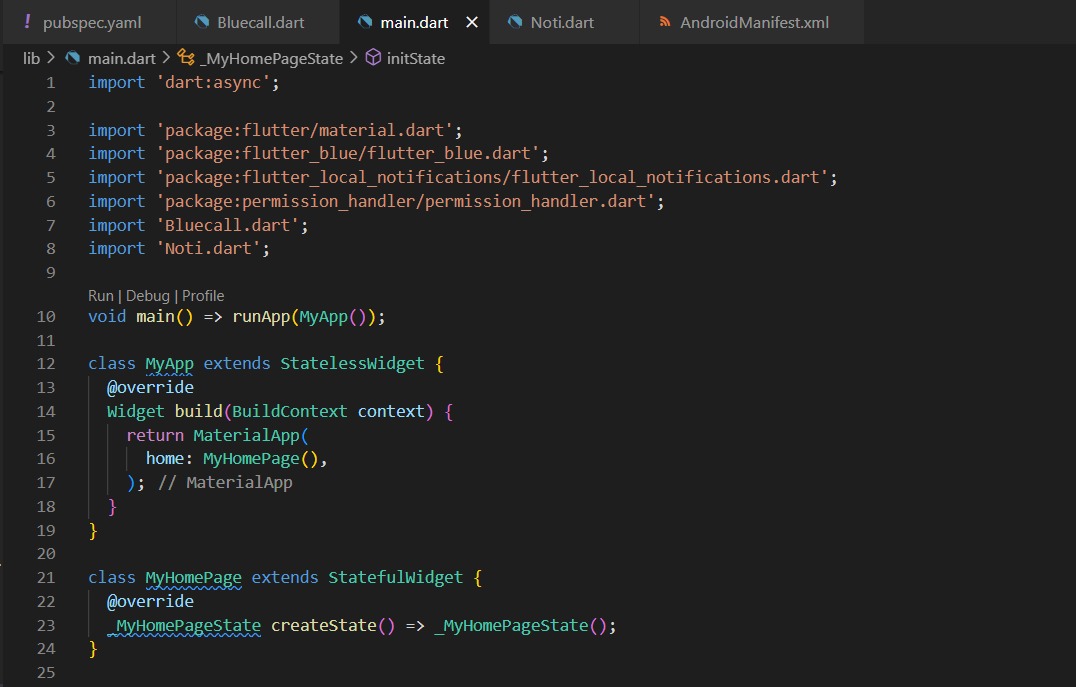


Figure 7.7.1: main code pt1

This code below (figure 7.7.1) serves as the main interface for the application's home page, handling Bluetooth device scanning, notifications, and managing the state related to these functionalities. It interacts with the Bluecall class for Bluetooth-related tasks and utilizes the flutterLocalNotificationsPlugin for local notification management within the app.

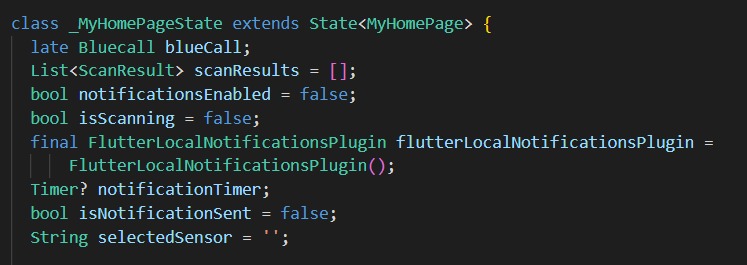


Figure 7.7.2: main code pt2



Figure 7.7.3: main code pt3

Now, in figure 7.7.1-7.7.3, “initstate” initializes the class “bluecall” whenever the app is launched.then we enter the function onDataRecieved, which update the state (rebuild the UI without resetting the values), and call the methode to handle the flags received from bluetooth. This onDataRecieved is also called whenever we receive data from Bluetooth, which will help update the status data as well as the sensor data when they are called. Next, we call “initializeNotification” for the notifications and “requestPermissions” which is responsible for asking the user for the required permissions.

“setSelectedSensor” is the method called when the user chooses a sensor from the drawer (the code of the drawer is shown later on in the report), it then update the UI so that the \_buildSensorDataDisplay method (explained later), return the needed widget.

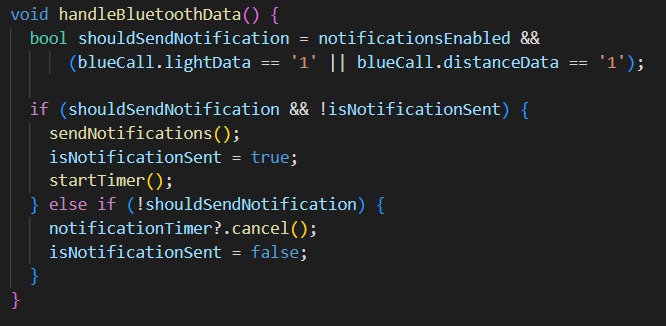


Figure 7.7.4: main code pt4

The function “handleBluetoothData” (figure 7.7.4) checks if the notifications switch is enabled (notificationsEnabled) and if certain conditions are met: either the light or distance data from the Bluetooth device indicates a trigger ('1'). If notifications are enabled and either the light or distance data indicates a trigger, and if a notification hasn't been sent yet (isNotificationSent is false), it initiates sending notifications by calling sendNotifications(), sets isNotificationSent to true, and starts a timer to manage notification intervals. If the conditions for sending notifications aren't met or are no longer valid (for instance, if the light and distance data both indicate '0'), it cancels any existing notification timers (notificationTimer) and resets isNotificationSent to false to ensure proper notification handling based on the changing Bluetooth data.

A screen shot of a computer code

Description automatically generated

Figure 7.7.5: main code pt5

The logic for showing notifications depending on the Bluetooth data received from sensors is handled by this Dart method, sendNotifications() (figure 7.7.5). It shows a combined alarm notice titled "Combined Alert," showing that both light and distance are above their respective threshold, if both the light and distance data indicate a trigger ('1'). It displays distinct alerts if just one sensor data point indicates a trigger ('1'). For example, when light ('1') is detected by the light sensor, a notification labeled "Light Alert" is displayed to notify the presence of light. In a similar vein, should the distance sensor register a trigger ('1'), the device will display a warning labeled "Distance Alert," signifying that the distance has exceeded the threshold.

A computer screen with text on it

Description automatically generated

Figure 7.7.6: main code pt6

This code above (figure 7.7.6), called startTimer(), creates a timer that checks every 10 seconds if there's a need to send notifications based on Bluetooth data. It first stops any ongoing timer to prevent conflicts, then sets up a new timer to repeat every 10 seconds. Within this timer, it looks at whether notifications are turned on and if either the light or distance data from Bluetooth shows something important (like a '1' indicating an alert). If there's something crucial, it sends the corresponding notification, but if the data doesn't indicate an issue, it stops the timer and makes sure that no notification has been sent, keeping everything in sync with the changing Bluetooth information.



Figure 7.7.7: main code pt7

These are “initializeNotifications” and “requestPermissions” that we mentioned and explained before in the “initstate” function (figure 7.7.7).

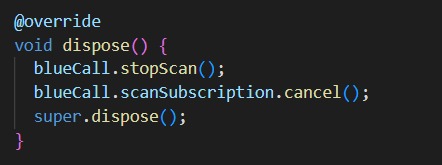


Figure 7.7.8: main code pt8

Regarding figure 7.7.8, Dispose() stops any ongoing Bluetooth scans initiated by blueCall using blueCall.stopScan() to release resources and prevent unnecessary scanning.



Figure 7.7.9: main code pt9

The toggleScan(), shown in figure 7.7.9, function regulates how Bluetooth scanning occurs. In the event that scanning is in progress, it updates the app's state to indicate the completion of scanning, ends the current scan, disconnects from any connected devices, and clears any saved Bluetooth data. It determines whether Bluetooth functionality is enabled and available when scanning is not in use. If so, it begins searching for Bluetooth devices in the vicinity, listens for scan results, and updates the user interface of the app with the items found. If there is no Bluetooth device nearby or Bluetooth is not on, we print in the terminal that Bluetooth is not available. This was only added for debugging purpose and the users will not be able to see it.

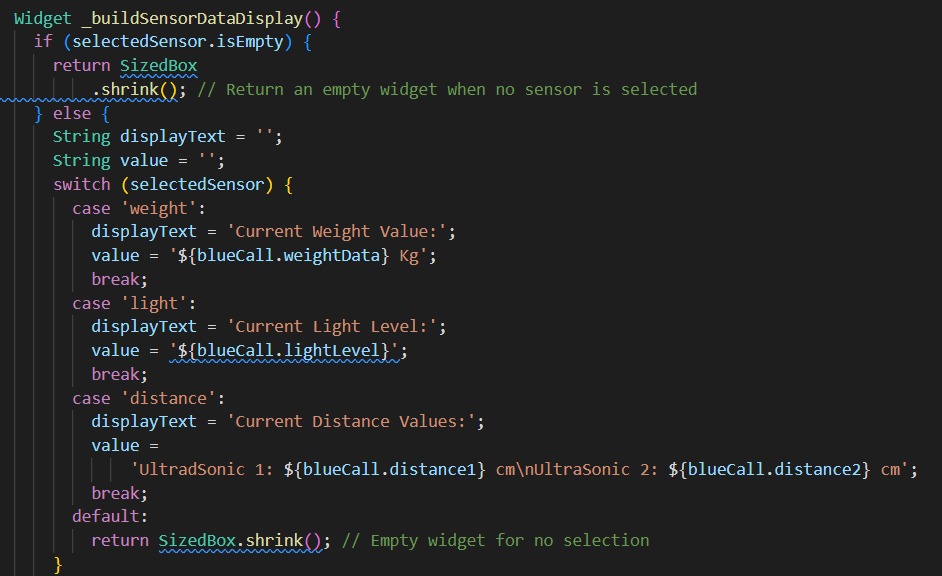


Figure 7.7.10: main code pt10

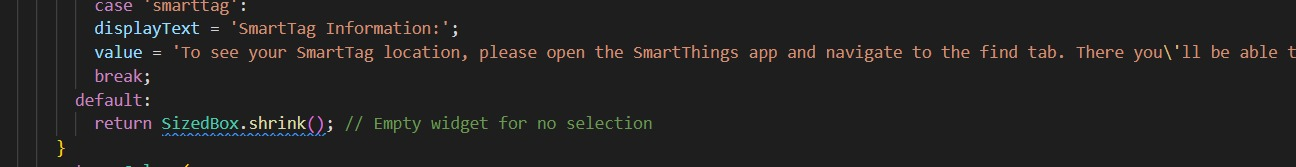


Figure 7.7.11: main code pt11

A computer screen shot of a program code

Description automatically generated

Figure 7.7.12: main code pt12

These 3 figures in page 78 (figure 7.7.10, 7.7.11, 7.7.12), are just for building the structure and fixing the appearance of the application depending on the specified sensor. If there is no sensor, it calls the default which is an empty widget. If a sensor is selected, depending on the sensor it returns the text specified.

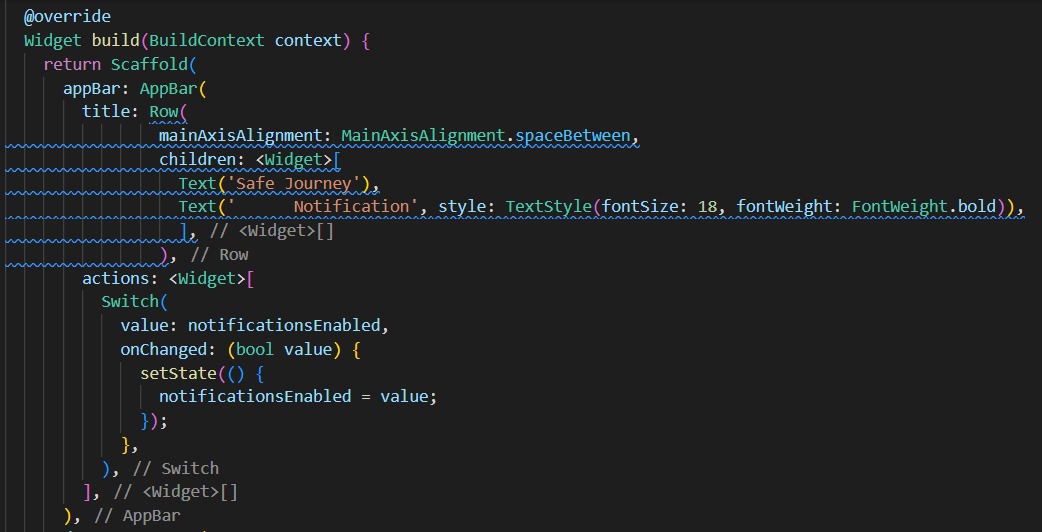


Figure 7.7.13: main code pt13

The figure 7.7.13 above is also building the body’s appearance (putting the title on top, adding a notification switch…).

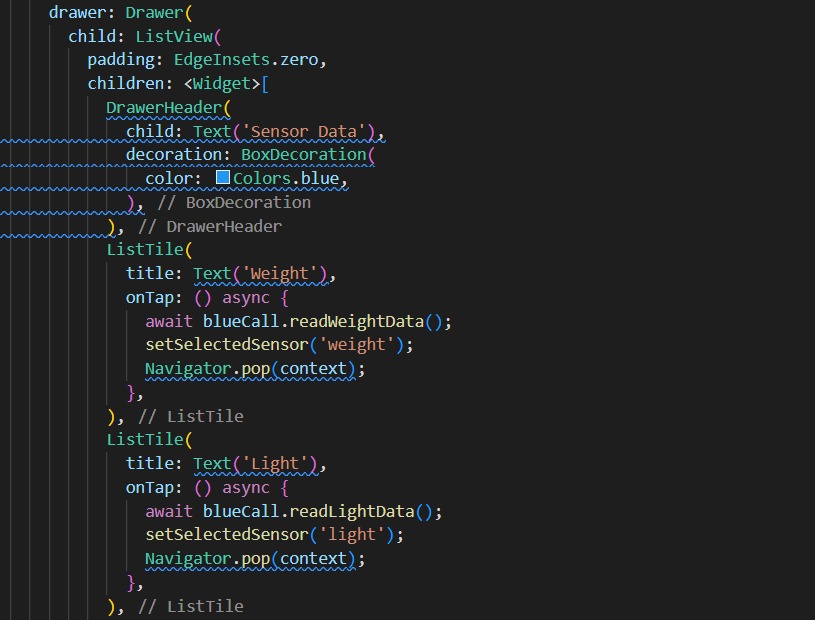


Figure 7.7.14: main code pt14

A screen shot of a computer code

Description automatically generated

Figure 7.7.15: main code pt15

A computer code on a black background

Description automatically generated

Figure 7.7.16: main code pt16

These 3 figures 7.7.14, 7.7.15 and 7.7.16, are for creating a drawer with ListTitle that has many options for the user to check: weight, light, distance and smarttag, and when a sensor is chosen, it calls the setSelectedSensor method.

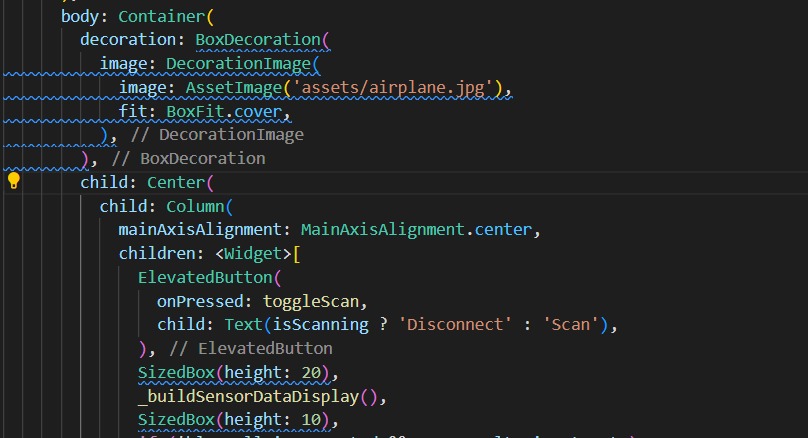


Figure 7.7.17: main code pt17

The figure 7.7.17 is the code for the decorationof the app, and initializing a scan/disconnect button depending on the isScanning flag. The \_buildSensorDataDisplay returns the specified widget depending on the sensor as explained before.

A screen shot of a computer screen

Description automatically generated

Figure 7.7.18: main code pt18

This part of the code in figure 7.7.18 is for displaying nearby devices when we scan, the results are put in a list, and the onTap is for when we tap a Bluetooth device, the application call the connectToDevice method in the Bluecall class. It then updates the isScanning flag and rebuild the UI.

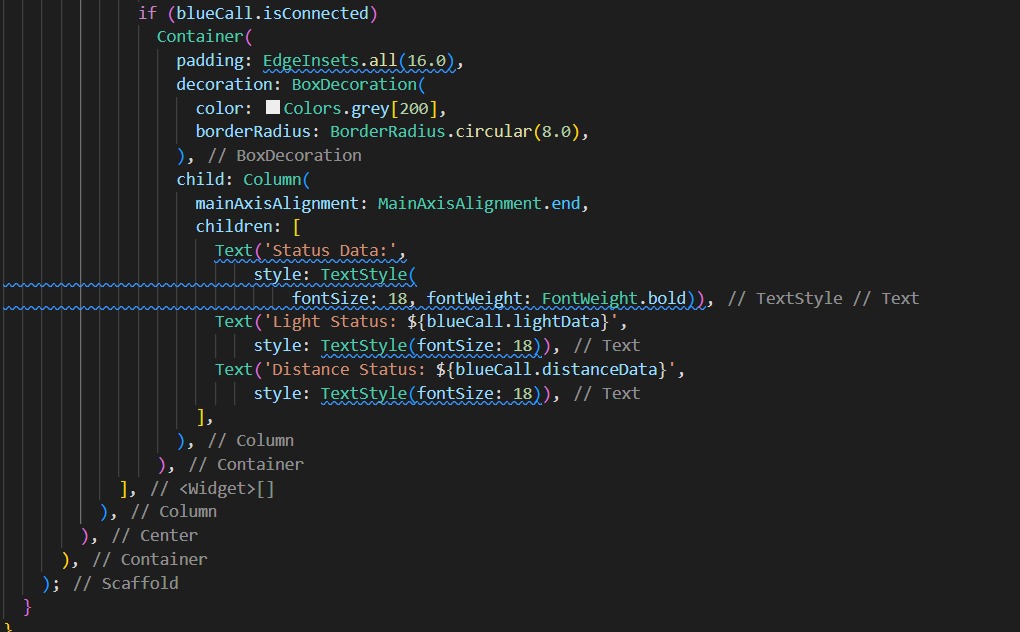


Figure 7.7.19: main code pt19

And finally, the figure 7.7.19 shows the code that behaves in a way, when we are connected to a device (which can only be our esp32 since we specified the UUID in the Bluecall class), the box goes down to the middle and the flags appear on the screen.

Before finishing up the code part, we were trying to make the application more efficient to use by integrating the tracking of the suitcase in our application. Unfortunately, this was not feasible.

To integrate it we had 2 options. The first one is if the smartthings app supports URL schemas, we can use the URL launcher package on flutter. We could get this URL from either play store (when opened on google we can see it as a parameter), or we can download an application called "package name viewer 2" that can retrieve it. However, this did not work since smartthings app does not support URL schemas and while searching we found out that it was requested by many users, but no response was ever replied to them.

The other option is to register as a Samsung developer, and after registering with our Samsung account, we can request for the API information for our specific samrttag to test. If we want to use this Api in our application, we need to request it stating that we are going to use it in an application and provide which information are we going to use and why we are going to use them.

Additionally, we should also provide the code of our application for Samsung to check and make sure it is safe. Once we get the Api, we can use it in our code to get the location coordinates of our smart tag using https requests and open them using a package that allow us to open google map inside the app.

This process, however, will take a lot of time and was not feasible in our time frame.

Now, we will provide some screenshots from the application.

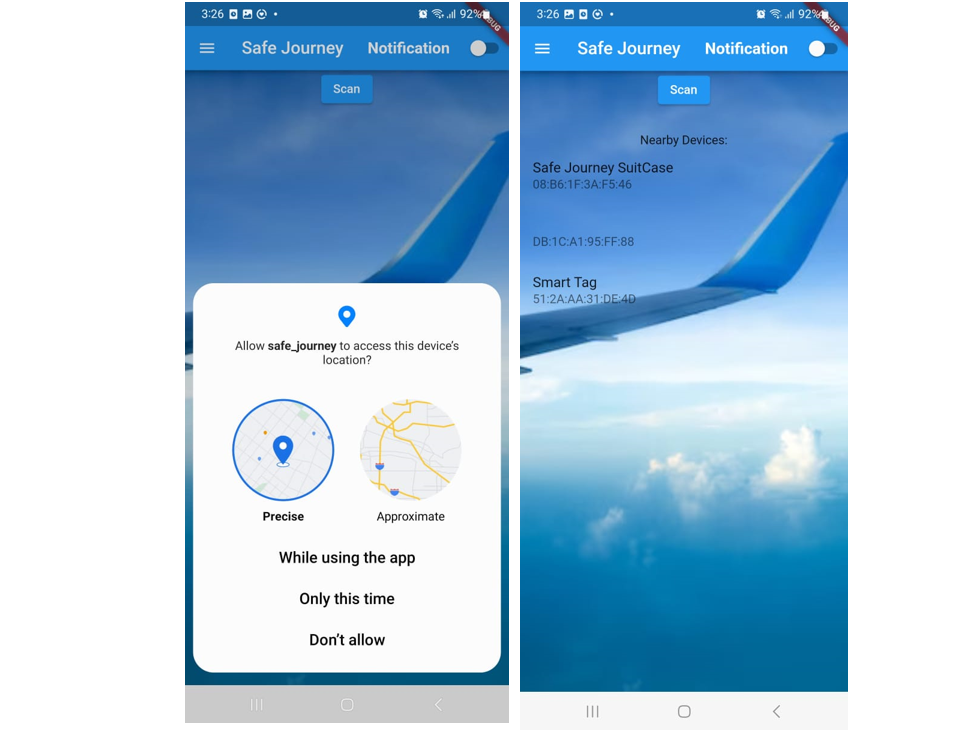


Figure 7.7.20: upon opening the application.

As shown in figure 7.7.20, upon opening the application, we will get a request if we want to allow permission to access the device’s location. Now when we press “scan”, we get all the nearby devices.

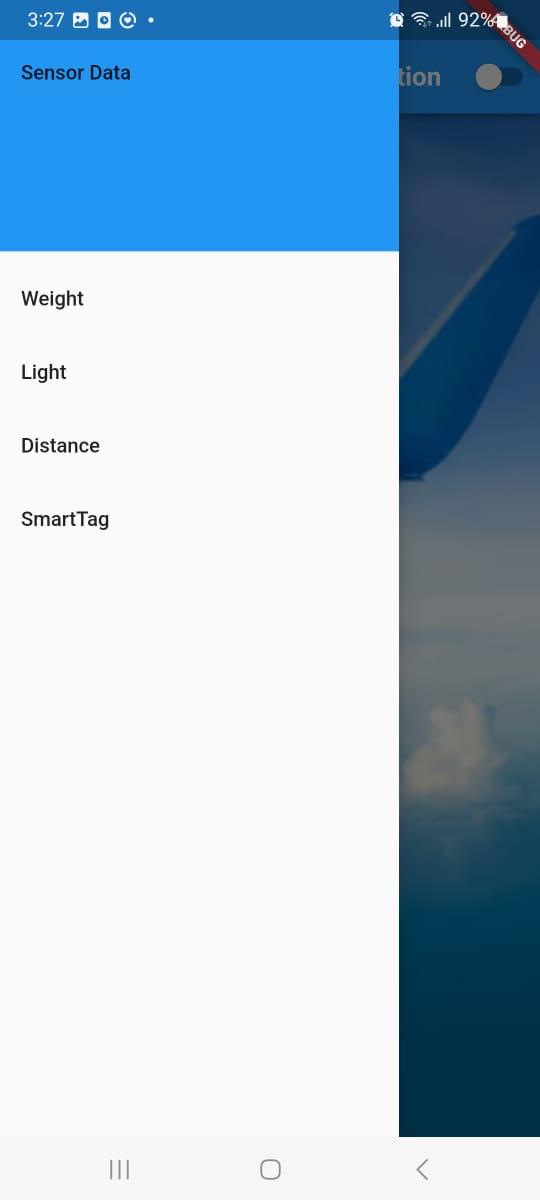


Figure 7.7.21: the drawer

Figure 7.7.21 is a screenshot of the drawer, where the user chooses what sensor he wants to check or if we wants to check.

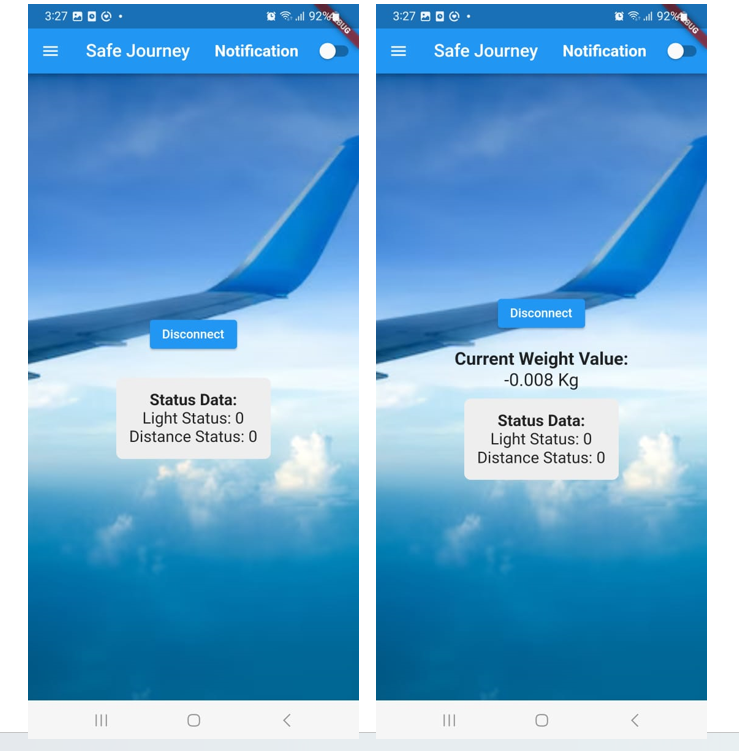


Figure 7.7.22: initial state of the closed empty suitcase

As for figure 7.7.22, it is showing that when the case is closed, the distance status and the light status are both 0. Additionally, the weight is approximately 0 when no weight is put on the load cell.



Figure 7.7.23: seeing the initial state of the ultrasonic sensor

Figure 7.7.23, is when choosing the ultrasonic sensor from the drawer, and it is giving 2 cm on both ultrasonics meaning that the suitcase is closed.

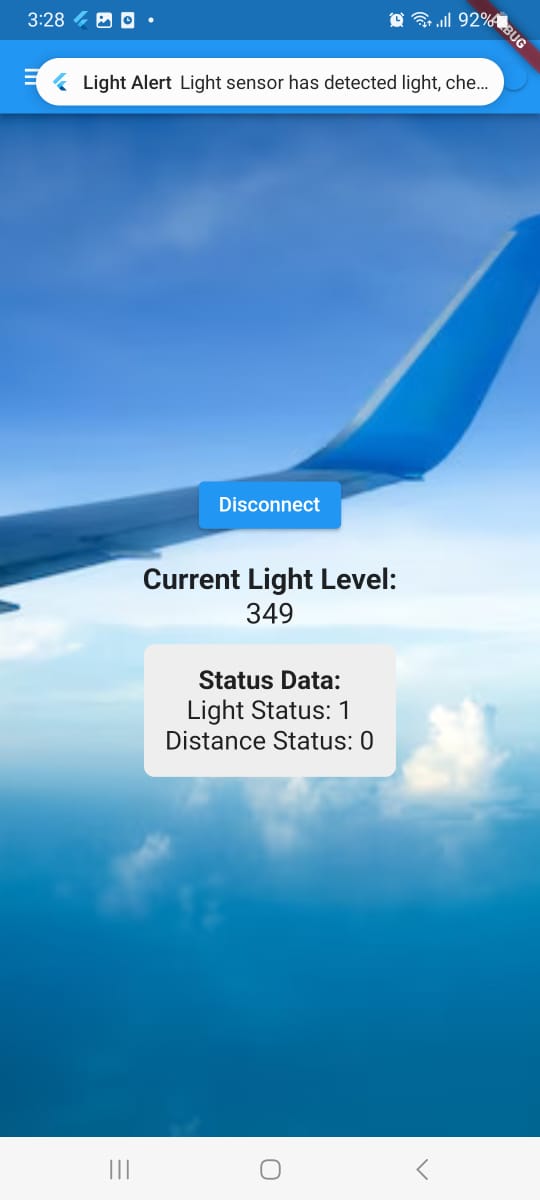


Figure 7.7.24: LDR when case is opened

In figure 7.7.24, the case is opened giving a light level 349 and updating the light status to 1. As we can also see in this figure, a notification was sent to the user, that light was detected.

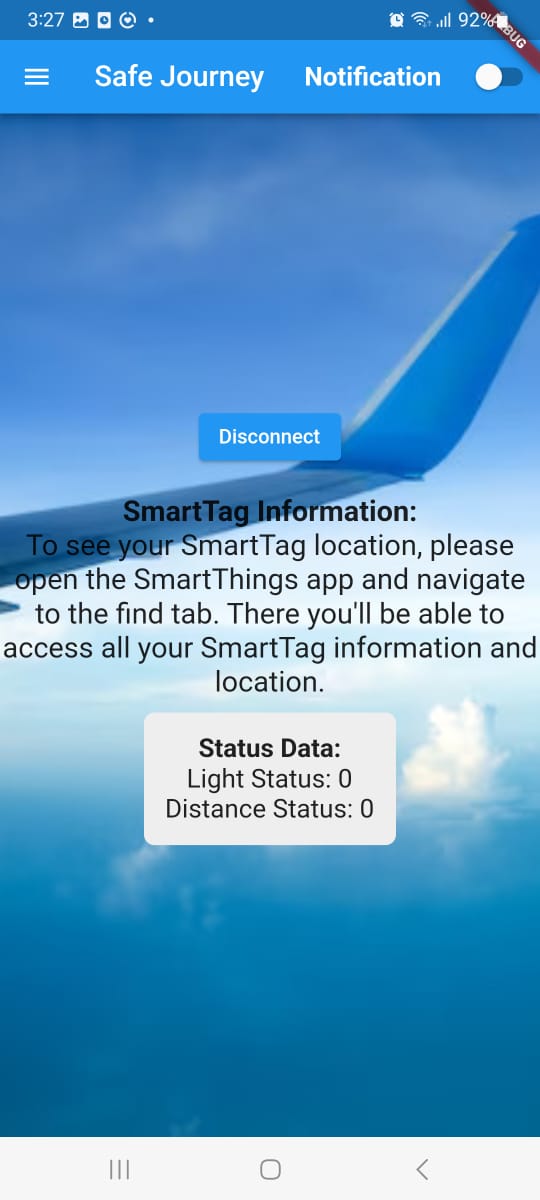


Figure 7.7.25: the smart tag page in the app

And lastly, figure 7.7.25 shows that if we press the option smarttag in the drawer, we will get the smart tag information.

**Chapter 9- Conclusion**

In conclusion, the SafeJourney Suitcase project represents a culmination of planning, dedicated teamwork, and innovative problem-solving. Through a strategic one-year project, we undertook the challenge of revolutionizing luggage security and tracking. From the initial phase of a full research from January 2023 to the sensor procurement during the summer months to the rigorous testing and integration of various components in September, October and November, each stage was met with precision and dedication.

Together, our knowledge of both software and hardware development ensured a well-rounded strategy. The methodical allocation of responsibilities and the coordinated efforts resulted in the successful development of a comprehensive and user-friendly system that promises to redefine luggage security and enhance the travel experience. Despite facing challenges inherent in innovative projects, the team persevered and delivered a functional and reliable solution by the November deadline.

Looking ahead, the SafeJourney Suitcase stands as a testament to innovation in the realm of travel technology. Its potential to increase luggage security, track belongings, and provide travelers with peace of mind speaks volumes about its relevance in modern travel.

**Chapter 10- References**

* + 1. SITA. (2023, May). Baggage Insight Report. Retrieved from <https://www.sita.aero/resources/surveys-reports/baggage-it-insights-2023/>
    2. Bureau of Transportation Statistics. (2021). FAQ: Is Lost Luggage Always Found? Retrieved from <https://lovethemaldives.com/faq/is-lost-luggage-always-found>
    3. Forte, J. (2021, June 9). Airport security now warning baggage thefts as travel numbers pick up. Retrieved from <https://www.newsnationnow.com/us-news/airport-security-now-warning-baggage-thefts-as-travel-numbers-pick-up/>
    4. Zamost, S., Griffin, D., & Devine, C. (2015, September 15). Hidden cameras reveal airport workers stealing from luggage. CNN. Retrieved from <https://edition.cnn.com/2015/04/13/us/airport-luggage-theft/index.html>
    5. FLINTEC. (n.d.). Weight Sensors - Load Cells: What is a load cell and how does it work? Retrieved from <https://www.flintec.com/weight-sensors/load-cells/what-is-a-load-cell>
    6. Last Minute Engineers. (n.d.). How HC-SR04 Ultrasonic Sensor Works & Interface It With Arduino. Retrieved from <https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/>
    7. EE Power. (n.d.). Chapter 3 - Resistor Types: Photoresistor. Retrieved from [https://eepower.com/resistor-guide/resistor-types/photo-resistor/#](https://eepower.com/resistor-guide/resistor-types/photo-resistor/)
    8. Espressif Systems. (n.d.). ESP32-WROOM-32 (ESP-WROOM-32) Datasheet (Version 2.4). Retrieved from <https://www.mouser.com/datasheet/2/891/esp-wroom-32_datasheet_en-1223836.pdf>
    9. Samsung US. (n.d.). Samsung Galaxy SmartTag, 1-Pack, Black. Retrieved January 16, 2021, from https://en.wikipedia.org/wiki/Samsung\_Galaxy\_SmartTag.