THE UNIVERSITY OF WESTERN AUSTRALIA

Smart IV Bag Monitoring and Alert System

Group 19

Adharsh Sundaram Soudakar (23796349)
Warren Wang (23680549)
Yuxiao Shi (23806063)
Zhiyang Cai (23488665)

Semester 2, 2023

CITS5506 - Internet of Things
The University of Western Australia

ABSTRACT

Considering burgeoning demands on healthcare infrastructure and the imperativeness of refined patient care, this manuscript delineates the conceptualization and application of an innovative Smart Saline/Intravenous (IV) Bag Monitoring and Alert System, aspiring to amplify patient care efficacy through the timely replacement of saline and IV bags. The architected system leverages economically viable, commercially available hardware constituents, inclusive of fluid level sensors. The sensors interface with an Arduino micro-controller, thereby facilitating the processing of data and engendering communication with an external server via the ThingSpeak platform. The server, assesses the status of the saline/IV bag and generates alerts when nearing depletion, thus ensuring the fluid remains at an optimal level. Healthcare practitioners are afforded the capability to monitor, transition operational modes, or confirm alerts through a dedicated mobile application, which establishes communication with the server backend through wireless fidelity (Wi-Fi). After successful implementation, the Smart Saline/IV Bag Monitoring and Alert System demonstrated proficient functionality across all delineated modes. Preliminary findings point towards the system's potential in augmenting patient care and mitigating nursing response durations; nonetheless, a comprehensive study is imperative to validate its overarching impact on hospital operations and patient outcomes holistically.

1. INTRODUCTION AND BACKGROUND

Since the inception of the COVID-19 pandemic in Australia in early 2020, sustained impacts have pervaded through emergency departments, admitted patient services, and elective surgery activities [1]. This backdrop has progressively exerted amplifying pressure on the Australian hospital system, particularly discernible over the decade leading up to the fiscal year 2021-22.

Documented data reveals that, over the decade culminating in 2021-22, the number of hospitalisations witnessed an upsurge from 9.4 million to 11.6 million. Concurrently, the rate of hospitalisation experienced a subtle ascent, escalating from 391 per 1,000 population to 405. Furthermore, the cumulative number of patient days expanded from 27.7 million to 31.8 million days [2]. This escalation in figures not only mirrors the conspicuous impact of the COVID-19 pandemic on the Australian health system but also augments the demand placed upon healthcare personnel and facilities.

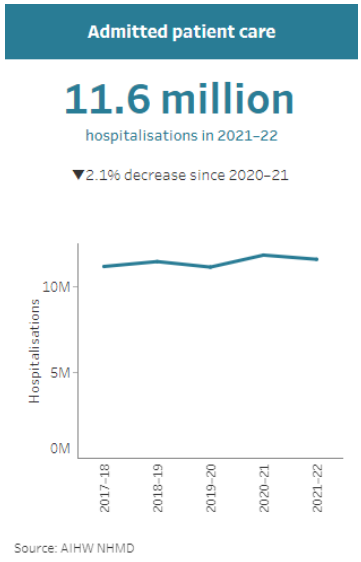

Fig1. Increase in patient care from 2017 to 2022.

With the augmentation in both hospitalisations and patient days, physicians, nurses, and ancillary healthcare workers have encountered a significant amplification in workload.

1.1. Primary Motivation for Monitoring a Smart IV bag

Amidst the global landscape, the COVID-19 pandemic has precipitated substantial pressures and formidable challenges, notably within the healthcare domain. A manifest deficiency of healthcare professionals, coupled with equipment shortages and a perpetually surging patient population, has emerged as a paramount concern during and subsequent to the pandemic era, thereby directly impinging upon the sustainability and operational efficiency of healthcare infrastructures. Consequently, the foundational motivation for the conception and development of the Smart IV Bag is underscored as a critical focal point within this context.

Firstly, a shortage of medical personnel: Amidst the pandemic's impetus, a significant majority of countries and regions worldwide have encountered a pronounced scarcity of healthcare workers. A substantial number of frontline medical staff have resigned or taken extended leaves, driven by the confluence of intense occupational stress, infection risks, and psychological burdens. This has invariably resulted in the remaining healthcare professionals bearing an augmented workload, particularly in performing certain routine and repetitive tasks, such as the regular inspection and replacement of IV bags.

Secondly, Elevating Medical Efficiency: Traditionally, the monitoring of IV bags has primarily relied upon periodic manual checks by medical staff. In situations where resources are stretched, this can divert healthcare professionals' attention, reallocating precious manpower from areas of critical need to tasks of comparatively lower priority. By deploying a Smart IV Bag monitoring and alert system, this process can be automated, ensuring timely and accurate IV replacements, while concurrently alleviating the workload of the medical staff.

Lastly, Enhancing the Quality and Safety of Patient Care: The Smart IV Bag has the capability to monitor intravenous fluid administration in real-time, transmitting data remotely and thereby permitting healthcare professionals to intervene instantaneously as needed, rather than awaiting the next scheduled check. This approach not only assures patient safety and comfort but also mitigates risks and potential complications that might arise from overlooked or delayed IV bag replacements.

In summary, the Smart IV Bag system epitomizes an innovative solution capable of mitigating some of the core challenges within healthcare systems through technological intervention. At this pivotal juncture, the question of how to enhance the efficiency, safety, and quality of healthcare services via technology, alleviate the burden on healthcare professionals, and optimize limited medical resources without compromising patient care quality becomes a paramount issue necessitating collective exploration and resolution among all pertinent stakeholders.

1.2. Secondary Motivations for Evaluating a reminding system of IV bag

1.2.1. Reliability

In traditional practices, both patients and healthcare providers were required to manually monitor the utilization of IV bags, a method that might be influenced by personnel constraints and divided attention, potentially compromising accuracy and timeliness. The introduction of the Smart IV Bag Reminder System ensures an automated and intelligent reminder and reporting mechanism, significantly enhancing its reliability. This system is capable of autonomously sending alerts when an IV bag needs

replacement or encounters issues, reducing the likelihood of oversights and delays, thereby elevating the precision and punctuality of patient care.

1.2.2. Real-time Processing

The Smart IV Bag Reminder System, through real-time monitoring of the IV bag's status, facilitates continual tracking of its usage and can issue timely alerts during critical moments, such as imminent depletion of medication or the occurrence of issues. This mechanism not only ensures that patients receive consistent and uninterrupted drug delivery but also promptly notifies healthcare providers to intervene in the event of potential problems or risks, thereby enhancing the safety and efficacy of the treatment.

1.2.3. Data Volume and Velocity:

Within the medical context, the transmission speed and volume of infusion data are pivotal. The Smart IV Bag System can collect, process, and transmit various data pertinent to IV drug delivery in real-time, such as the type of medication, dosage, delivery rate, and remaining volume. This information becomes crucial when swift decisions are needed, such as altering medications or adjusting infusion rates. Concurrently, as the system can handle and transmit substantial volumes of data, this accumulated information becomes invaluable when data analysis is required to optimize treatment plans or conduct medical research. Rapid and accurate data processing and transmission ensure that healthcare professionals obtain real-time, comprehensive information, thereby enhancing medical decision-making and improving the quality of care.

1.3. Project Objectives

This project endeavours to design and implement an Intelligent IV Bag Monitoring and Alert System, aimed at optimizing medical processes and enhancing the safety and efficiency of patient care. The system will employ highly sensitive sensing technology to monitor the status of IV bags in real time, ensuring the provision of accurate and timely data regarding fluid levels. Furthermore, through the integration of analytical and alert mechanisms, the system is engineered to intelligently interpret data and issue necessary alerts promptly, thereby facilitating immediate notification and requisite intervention guidelines for medical personnel. In essence, this project seeks to amplify the reliability and precision of IV therapy through technological means, alleviate the workload of healthcare staff, and assure patients of secure and continuous drug delivery.

1.4 Design Brief

There are two parts to the design of this system, the hardware and software part. The hardware part measures the weight of the IV/Saline bag in real time and transmits this data to a cloud server. The cloud server also acts as a dashboard. The Software part includes the mobile application that receives the weight data from the cloud server and based on the weight data makes decisions that alert the user and sends data back to hardware part, effectively controlling its functionality. The block diagram representing the above information with the components used to achieve it, is visualized through the figure below.

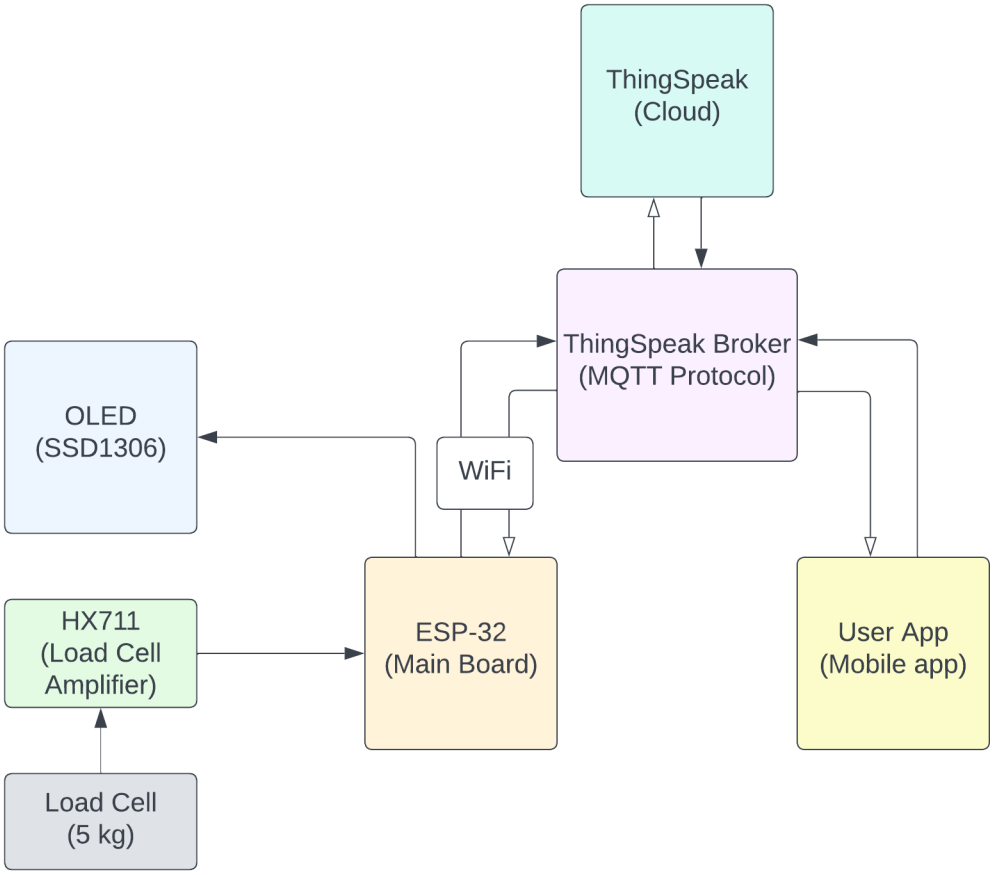

Fig2. Block diagram.

2. SOLUTION AND IMPLEMENTATION

2.1. Hardware Solution

Key criteria in hardware selection for this project were:

- *Cost*

The prototype cost about $36.84, which does not include the ESP-32-E, and one OLED display which is provided by the Engineering department of UWA.

Since sensors and other components make up a large amount of the final production cost, designing a prototype for the IV bag may initially appear to be more expensive. But it's crucial to take a wider view into account. The commercialization phase might result in significant cost savings once the prototype has been successfully created and improved. The economy of scale is largely used to accomplish this. Transport expenses per unit drop dramatically when IV bags are produced on a bigger scale for commercial use since bulk shipments are less expensive than individual parts. In addition, suppliers frequently offer price breaks on materials when orders are submitted in big volumes, lowering the cost of production per unit. Because of this, even though the initial cost of prototyping may seem high, the long-term advantages of commercialization can result in significant cost reductions in the creation and distribution of IV bags.

The marketability of crucial components significantly influences selecting the most suitable sensor and amplifier for a given project. To facilitate a seamless and efficient development process, the preference is often for components that can be readily procured from local or reputable regional distributors. Nevertheless, there are instances where, despite exhaustive searches, the required components remain elusive. In such circumstances, online procurement becomes the ultimate recourse, offering access to a broader spectrum of manufacturers and suppliers, albeit potentially leading to project timeline extensions. This approach underscores the importance of adaptability and persistence in the face of component scarcity, ultimately enabling this project to keep on track.

Table1. Cost of each component and hardware used.

	Parts Name	**Price**
Sensor	5KG Load Cell Weight Sensor Module for Electronic Scale Arduino Project	9.95
Amplifier	Load Cell Amplifier Module HX711 for Arduino Projects	3.95
Other external cost	Shipment + Tax	13.65
	CUP HUOK EUERHANG18MM 25PK 72324	4.89
	LEG FURNITURE PINE ADOORED47MM BALL 40179	4.4
Total		**36.84**

- *Availability*

Initially, the main goal was to buy all necessary components locally, motivated by convenience concerns and a desire to support small businesses. This method, however, ran into significant restrictions because there weren't many possibilities accessible on the local hardware market. To broaden the range of options and gain access to a wider variety of possibilities, it was decided to purchase the weight sensor and amplifier through Internet channels. The need to find specialized components that were difficult to find from local sources forced this strategy change. Although there was still a preference for supporting small firms, the need for diversification and the acquisition of specialized components prompted a switch to online procurement, which finally allowed the project's unusual criteria to be met.

- *Size*

Due to the ESP-32-E inclusion and the use of a common weight sensor and an amplifier part, the prototype's dimensions are a little bit larger. It is important to remember that these dimensions can be reduced through modification. It is possible to lower the total size of the prototype by customizing the components to the particular needs of the project. This modification improves the prototype's compactness while also optimizing its functioning and bringing it closer to the required requirements and design goals.

- *Power requirement*

Energy efficiency is meticulously integrated into the design of the power supply for the IV bag monitoring system. The chosen approach leverages the utilization of a USB port on a computer for power supply, which not only proves to be a cost-effective and pragmatic solution but also underscores a commitment to sustainability. This strategic choice not only aligns with the overarching project objective of low-energy

maintenance but also significantly diminishes the system's environmental footprint and minimizes ongoing operational costs. By circumventing the necessity for dedicated power sources or batteries, this approach reflects a conscientious consideration of both ecological and economic factors, making it a judicious choice in the context of the IV bag monitoring system's design.

The ESP-32-E, serving as the primary component of the system, operates on a voltage of 3.6 volts. The pivotal rationale behind the selection of this microcontroller module is its intrinsic energy-saving capacity, achieved through the integration of Bluetooth and Wi-Fi networking functionalities. The ESP-32-E emerges as an exceptional choice for this specific application, primarily due to its adept consolidation of multiple wireless communication technologies within a singular device. This strategic amalgamation significantly mitigates power consumption in contrast to the deployment of distinct modules for each wireless function. This discerning choice of the ESP-32-E underscores a commitment to optimizing energy efficiency, a fundamental consideration in the design of the system.

Moreover, the holistic design of the entire system is underpinned by a steadfast commitment to minimal power consumption. To uphold this pivotal principle, careful consideration is given to the selection of the monitoring system's sensors and components, with a distinct focus on their energy-efficient attributes. Additionally, the system's operational profile exhibits a notably restrained current flow, well within the parameters accommodated by the computer's USB-powered energy supply.

- *Measurement range and accuracy*

The sensor that is utilized in the system has a sturdy design made of an aluminium alloy. It has strain gauges that are pre-adhered to strain-relieved wires, increasing the durability and dependability of the sensor. This sensor is very good in accuracy; it can measure forces between 0 and 5 kg with ease. Its remarkable accuracy threshold enables the detection of minuscule force differentials as small as +/-100 grams (0.02%F.S), provided it is calibrated based on industry calibration methods. The sensor's high degree of accuracy makes it ideal for uses where precise force measurement is crucial.

- *Ease of integration*

Each element that has been incorporated into this project is in perfect alignment with the hardware platform, guaranteeing a high level of compatibility. This smooth integration makes assembly easier and improves system performance overall since every part works well together inside the assigned hardware structure. The elements' natural compatibility highlights the careful planning and design considerations made to guarantee the project's efficient and successful functioning.

Table2. Functionality of each component and their availability in the University

S.No.	Component	Cost	Avail at UWA	Operating Current/Voltage	Function
1	ESP-32-E	13.99	TRUE	240 mA / 3.3 V (Max)	Microcontroller with Wi-Fi and Bluetooth component
2	5KG Load Cell Weight Sensor Module for Electronic Scale Arduino Project	9.95	FALSE	1.5 mA / 5.25 V (Max)	Weight sensor

3	Load Cell Amplifier Module HX711 for Arduino Projects	3.95	FALSE		Amplifier for increasing the signal and 24-bit ADC
4	OLED Display SSD1306	13.55	TRUE	20.7 mA / 5 V (Max)	Display for message

2.1.1. Hardware Design

The ESP-32, which has inbuilt Wi-Fi functionality and sufficient number of GPIO pins, was chosen as the main board. Apart from built-in communication functionalities, there are several power modes which enables us to effectively save power.

We used C to program the ESP32-E with the help of Arduino IDE. The entire code is divided into segments and attached to the appendix (**Appendix-A**). Laptop connected through USB-C to the ESP-32-E was used to power it. The circuit design is shown is shown below.

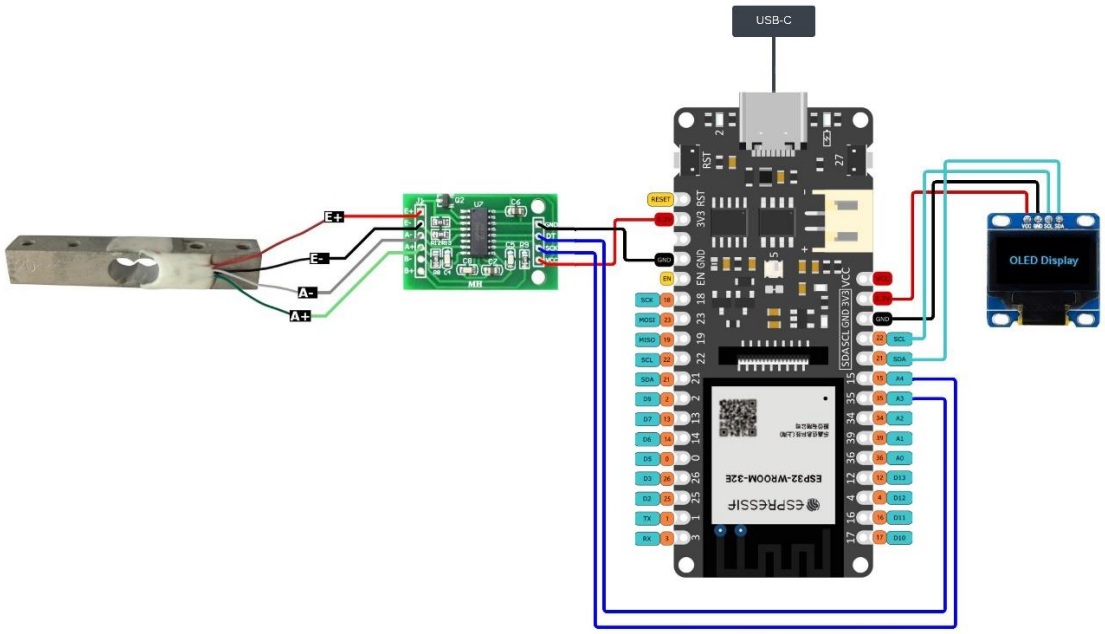

Fig3. Circuit Diagram

The 5kg load cell was chosen considering the magnitude of the weight being measured and budget. Hospitals, typically, do not cross the 1000mL mark when it comes to saline/IV bags which weighs around 1kg. The load cell has an accuracy of +/-0.02 on full scale, this means that we could get weight measurements ranging +/- 100 grams from the actual value. To take this into account, the load cell requires industry level calibration as the load cell's accuracy is subject to change with temperature and mounting position. There is one problem with using this load cell, which is that it is not built for measuring weight by hanging objects but on applications that use a flat platform as the scale fixed to weighing end of the load cell. Again, this makes the measurements fall off from the original values.

Considering the above limitations, we proceeded to calibrate the load cell using the guide available from sparkfun [3]. (Calibration explained in detail in Section 3.1).

HX711 was a definite choice when it came to ADC, which converts the analog input from the load cell to digital output, which can then be used by the ESP-32, performing different operations based on the values received. We could have used HX710 instead of the HX711 but when it comes to compatibility, which matters when trying to integrate it with other components, there are readily available modules that make it more compatible and easier to use. The modules used here were from bodge [4]. The circuit connection between load cell and the HX711 was adapted from Indrek Luuk [5].

Weighing the IV/Saline bag

To weigh the IV/Saline bag, a scale cannot be used. From a practical point of view, the bag loses its weight over time and this depletion occurs with the help of gravity. To make this work, the only solution is to hang the bag through a hook from the load cell and fixing the other end of the load cell to something rigid.

Hence, we decided to screw a hook to the weight sensing end of the load cell and fix the supporting end to a sofa leg that not only acts a sturdy platform but also provides elevation. To fix the load cell to the sofa leg, super glue was used. Screwing the load cell to the sofa leg was not necessary as the common saline bags that are administered for patients do not cross 1000mL which weighs around 1kg. We found that to support such a small amount of weight, gluing the load cell to the sofa leg was sufficient.

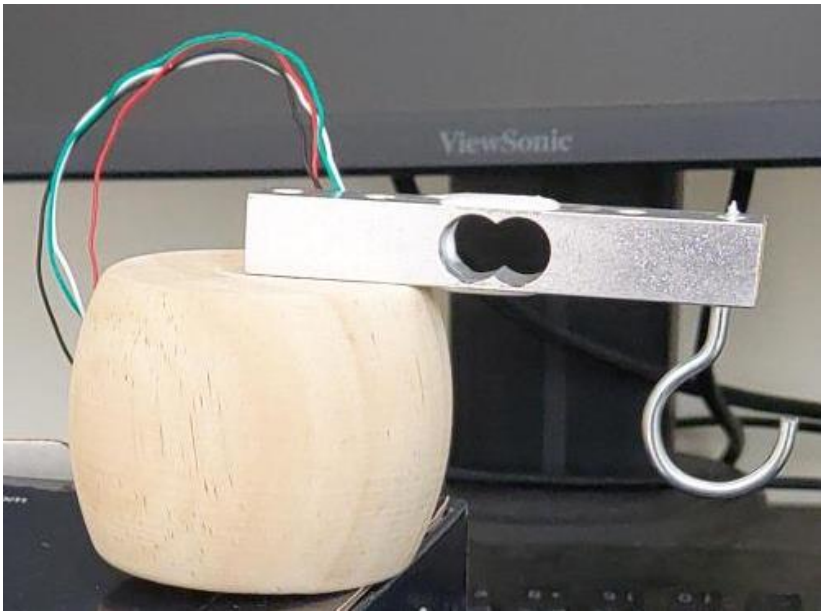

Fig4. Load cell Mounting

Displaying actions

To display appropriate messages, give the user a handsfree experience and facilitate easy understanding of the ESP-32-E's actions, we decided to use a small 128x64 OLED display which serves as an on-console display. The size of the display is quite small when compared to typical displays used in other hospital applications. But our design does not require a big display as the information displayed on the screen is very minimal. It only displays the state of the ESP-32-E, whether its sleeping or not, the Wi-Fi connection status and the weight of the saline/IV bag based on which appropriate alerts are

displayed. We programmed the display by referring to the work by Sara Santos from Random Nerd Tutorials [6].

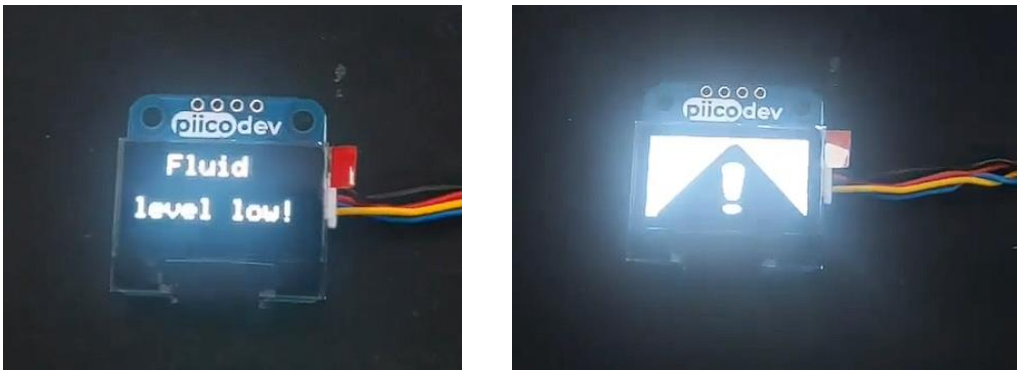

Fig5. Use case of the OLED 128x64 display

With the calibration done, we had to come up with a logic that will help us weigh the IV/Saline bag and display appropriate messages that make sense. This was done by building flow charts and implementing that flow into code using Arduino IDE. After several revisions and changes, we finally decided on a flow chart that works as intended.

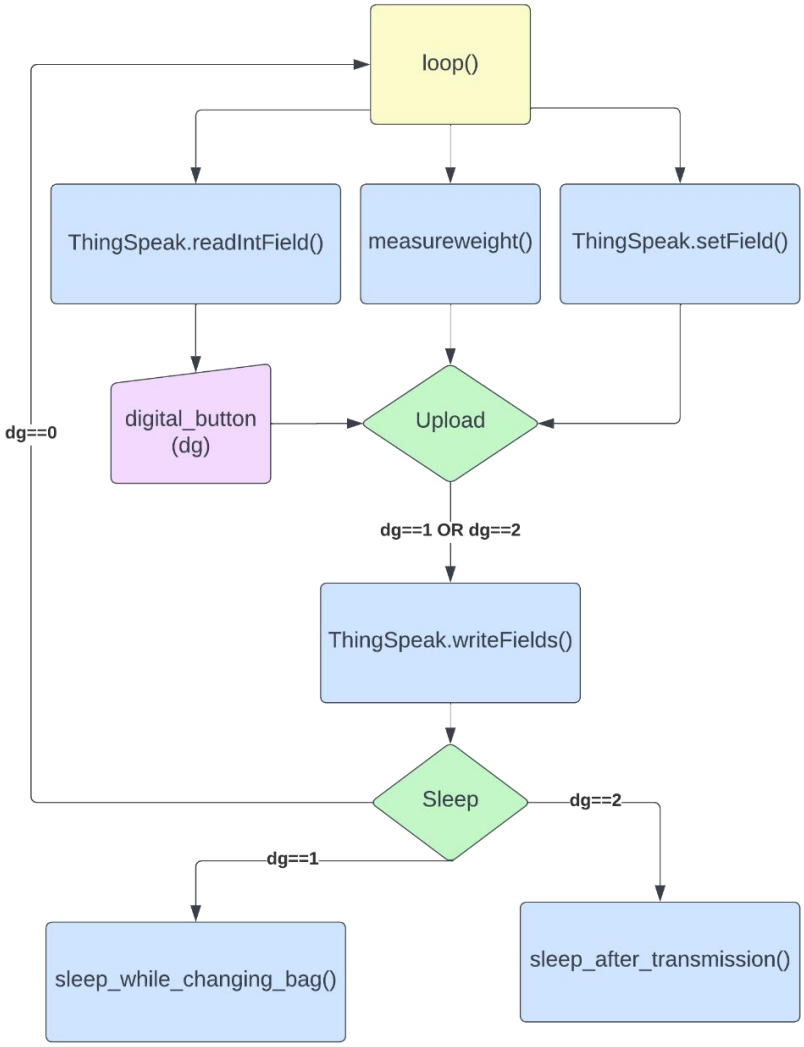

Fig6. Main flow chart

The digital_button variable is read by the ESP-32-E from the server, this data is sent to the server by the user. The user must manually press a button for the value of the digital_button variable to be changed. (This is explained in detail in section 2.2.2.). This variable is important as it controls the flow. We introduced sleep function to save power as when the ESP-32-E is in deep sleep it consumes 0.15mA max. Considering this fact, we managed to find to situations where the ESP-32 could be put to sleep, after there is a successful transmission of sensed data and when the bag is being changed. The inbuilt sleep function uses the RTC (Real Time Clock) that helps with setting the duration of sleep. Programming the ESP-32-E to connect to Wi-Fi and sleep was learnt and adapted from the YouTube channel Simply Explained [7].

After waking up from deep sleep, the ESP-32-E goes through the whole process of connecting to Wi-Fi and all the initialisations. This means that for the prototype to work, there must be continuous and reliable Wi-Fi network.

The measureweight() function was written to control the ESP-32-E, to decide what it has to do with the sensed data. The figure below visualises the process.

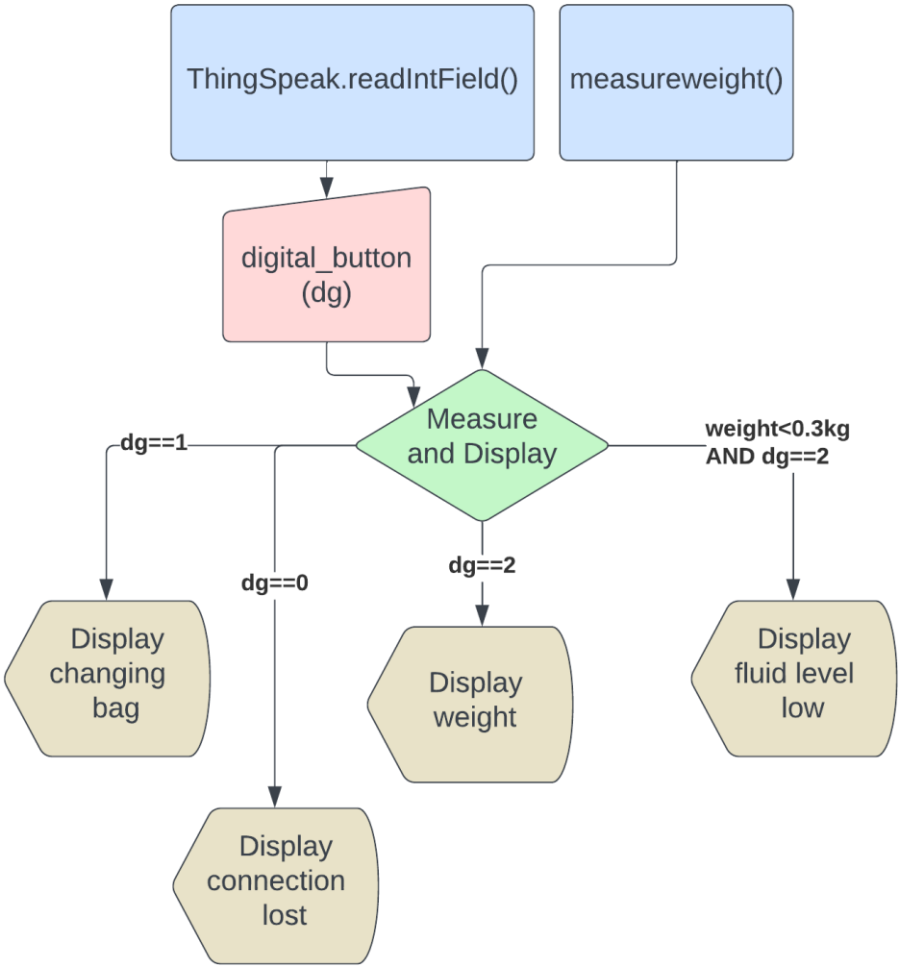

Fig7. measureweight() function logic

As explained earlier, the digital_button variable plays a crucial role in controlling the whole of hardware. Here, we have set a threshold value for the weight data to be 300 grams, below which an alert is displayed on the OLED display.

Each component of the prototype had to be fixed to something rigid and should not move from its place for the sensing part to work properly. So, we decided to come up with a case that not only fits the specifications but also the budget. Ideally, a hanging load cell application would have all its components fixed far away from the load cell. This helps with preventing any physical interruptions when it comes to sensing the weight of the hanging object. But, constrained by time and resources, mainly time, we decided to use a coffee glass package box as a quick and easy DIY solution. This fitted the size specifications as all the components had room to be fitted.

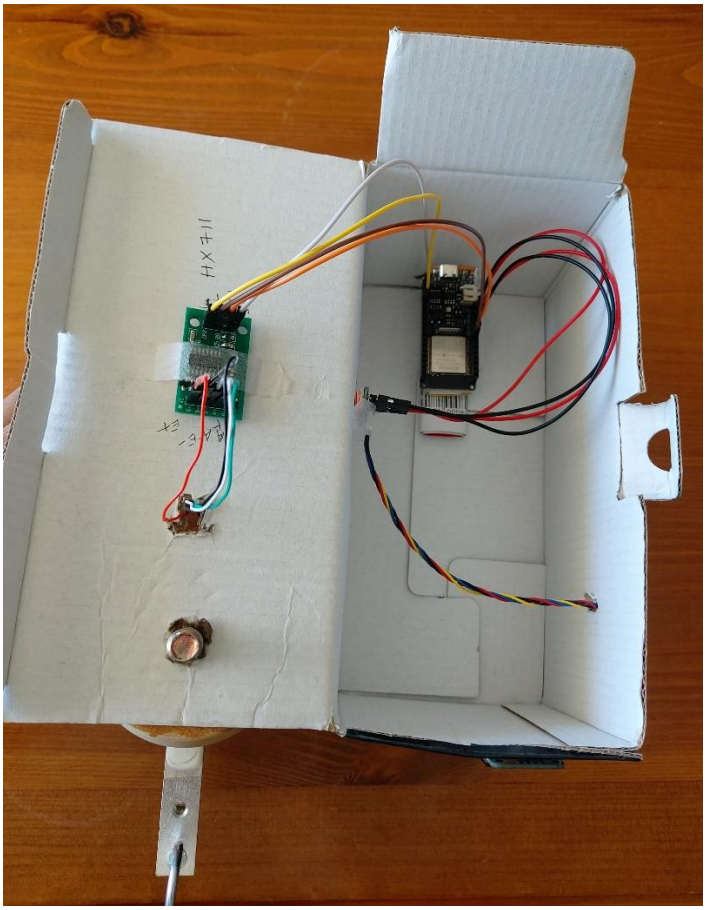

Fig8. Inside of the coffee glass package box.

Fitting the display to the front of the box seemed to help with its visibility. One of the situations where this placement proved helpful was when the user must come to the front of the prototype to change the bag during which the display would be in their field of vision. The dimension of the prototype is approximately 11.5cm in height, taking into account the sofa leg and the load cell mounted on top of it, 6cm in width and 13cm in length. The figures below show the final design in different perspectives.

Fig9. Side view of the prototype

Fig10. Front view of the prototype.

2.2. Software Implementation

2.2.1. User interfaces and use logic.

The app's user interfaces are composited with three screens as shown below.

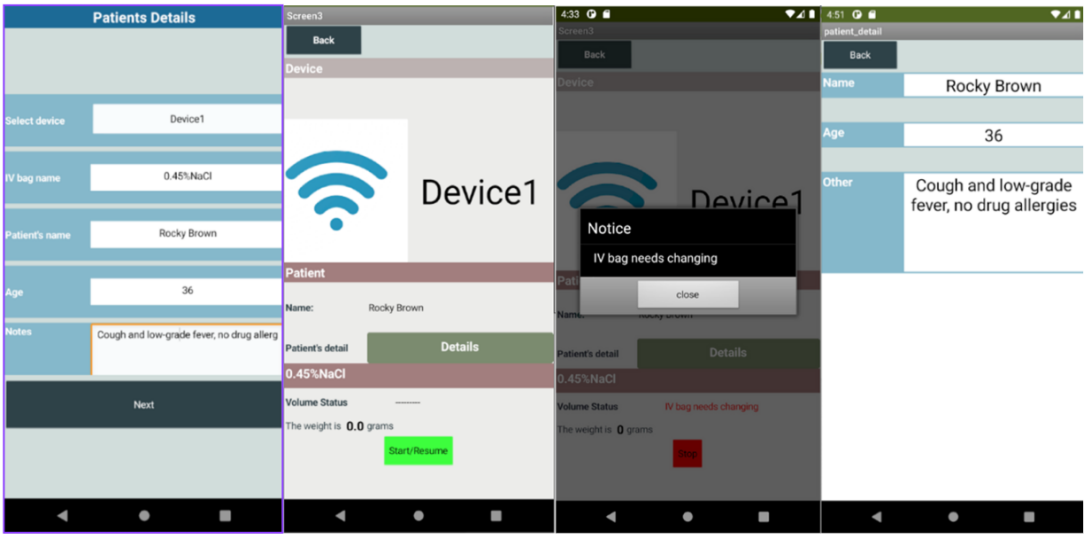

Fig11. User interfaces

In the first screen, users choose the device for connection, enter the patient's information and IV bag's name. Click on "Next" and jump to the second screen which contains basic information got from the first screen. The weight will update in real time once users press the "start" button. If the weight data read is lower than 200 grams, a notification will appear to let nurses know IV bags need to be changed. Click the "Details" button, more patient's information will be shown in the third screen.

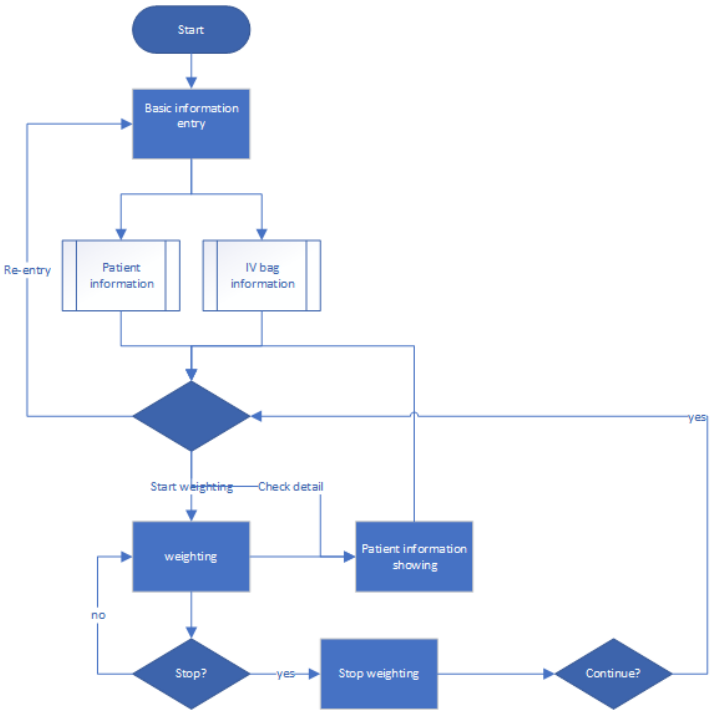

Fig12. Use logic

2.2.2. Cloud service - ThingSpeak

ThingSpeak [8] is an IoT platform that offers services such as data collection and analysis. It allows user to upload data using MQTT protocols. ThingSpeak provides convenient APIs for data reading and writing. Consequently, we decided to use ThingSpeak as the server to facilitate interaction between the ESP-32-E and the app. Specifically, two interactions will happen in the project.

For the first interaction between hardware and software, the ESP-32-E collects the sensors data and writes it to a ThingSpeak channel. Screenshots below will show the dashboard of channels of Thinkspeak.

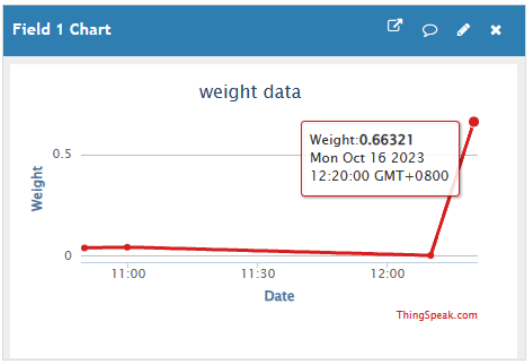

Fig13. Thinkspeak channel weight data

After that, the app read the JSON form data through the channel. The logic of data reading is that the app will always read the latest entry of the data.

```
    "created_at": "2023-10-16T04:18:07Z",
    "entry_id": 884,
    "field1": "0.64708"
},
{
    "created_at": "2023-10-16T04:18:25Z",
    "entry_id": 885,
    "field1": "0.64210"
},
{
    "created_at": "2023-10-16T04:18:45Z",
    "entry_id": 886,
    "field1": "0.00000"
},
{
    "created_at": "2023-10-16T04:19:04Z",
    "entry_id": 887,
    "field1": "0.00000"
},
{
    "created_at": "2023-10-16T04:19:43Z",
    "entry_id": 888,
    "field1": "0.66132"
},
{
```


Fig14. Weight data in JSON form

For the second interaction, the app sends signals that hint some operation to other channel, which the ESP-32-E reads and responds accordingly. For example, app will send signal of 1 which represents the weighting process should stop, while signal of 2 represents it should start. ESP-32-E will reads the signal through the other channel's field.

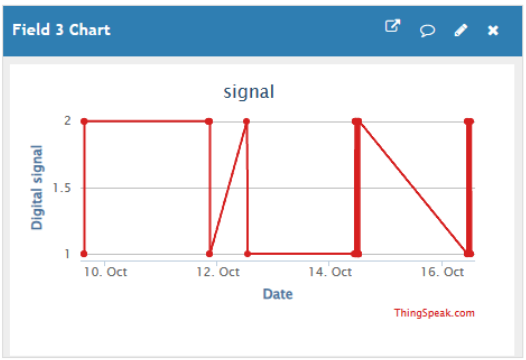

Fig15. signal data

2.2.3. App Develoment tool - MIT APP Inventor

The selected tool for app development by the project team is the MIT App Inventor [9]. The utilization of this tool facilitates the expeditious development of Android applications, particularly for novice programmers. Its user interfaces for developers are shown below.

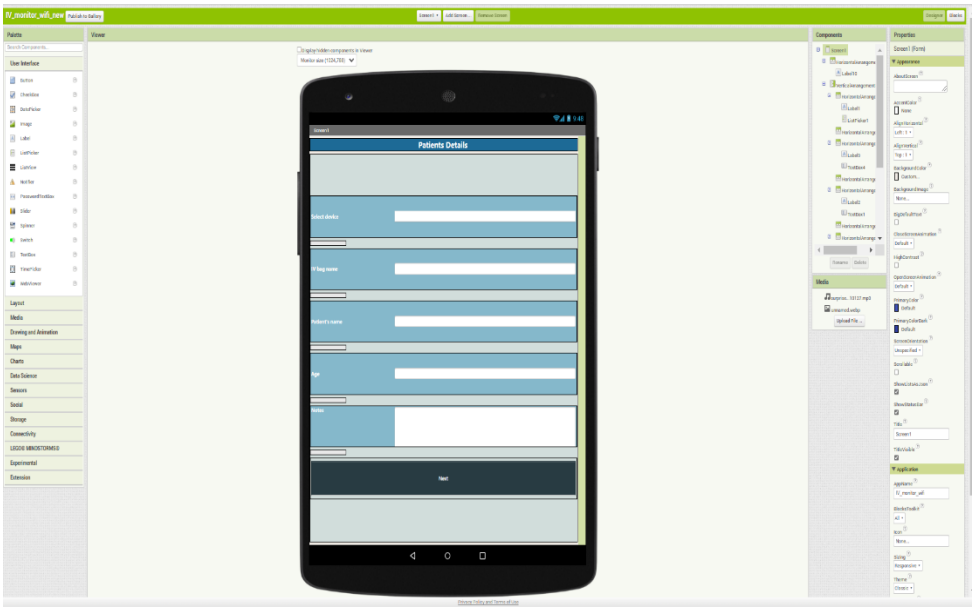

Fig16. User interfaces for developers

Fig17. User interfaces for developers

The project aims to streamline the procedures involved in UI interface design, component logic design, and network interaction setup. Additionally, this aids the project team in expeditiously generating the application prototype and carrying out functional testing and validation. Nevertheless, the platform's

development interfaces, which rely on dragging and dropping, pose certain limits, particularly in terms of controlling the app's development.

3. RESULTS AND DISCUSSION

3.1. Calibration and Initial Testing

The main goals of initial testing and calibration in this particular context are to optimize the precision of the sensor, guaranteeing precise weight readings, and to verify the smooth operation of the data processing process. These essential phases play a pivotal role in ensuring the seamless advancement of the complete system.

The process of calibrating the weight sensor has resulted in the determination of a permissible margin of error, which is equivalent to 150 grams. The error range is carefully considered in relation to the overall weight measurement range of the sensor, which reaches a maximum of 5 kilos. The degree of inaccuracy observed can be considered reasonable and pragmatic, taking into account the specific scope and precision needs of the system.

Calibrating the load cell accurately requires industry level environmental control and equipment as the load cell's behaviour is subject to vary based on the temperature. To perform any calibration of a device that takes a measurement, we need to know the true measurement value of the object that is being measured. To accomplish this, we used a kitchen scale that shows almost precise weight of the object in grams. The graph below shows our deviation from the ground truth. (This test data and graphs can be found in **Appendix-B**).

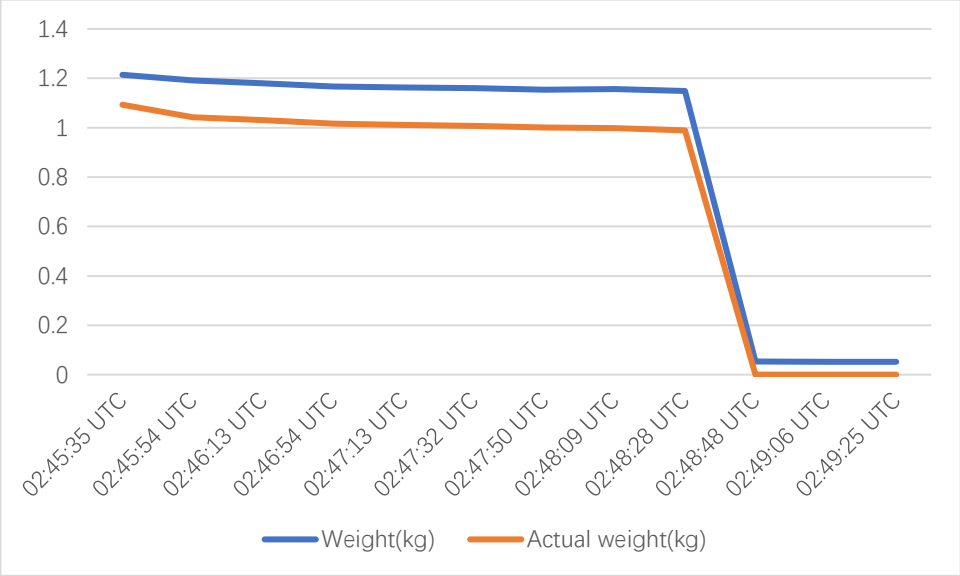

Fig18. Actual vs Measured weight.

From the graph, we can infer that there's a clear difference between the actual weight and the measured weight. This was a collective error, the manufacturing defects, using a load cell not suited for measurement of weight by hanging, the hook used (weighed around 5g), the mounting and the deformation of the cardboard package box, all added up to approximately 150 grams more than the actual weight. Although the calibration factor and the zero factor or the offset value of the load cell (which takes into the account the weight of the hook and the mounting position error) could be found,

the deformation of the structure was random each time a weight was hanged. This clearly shows the poor choice and construction of prototype structure. Repeating the test gave the same results.

Regarding the interface between the ESP-32-E and the server, preliminary testing has indicated a time gap of 15 seconds between successive data transactions. This empirical observation enhances comprehension of the system's operating tempo and offers significant insights into its capabilities for real-time data transfer.

Finally, a problem arose in the interface between the hardware components and the software during the process of replacing the IV bag. The notification component of the system remained active until the bag replacement operation was completed. The issue was overcome by subsequent changes and refinements to the architecture of the system, resulting in the smooth and efficient working of the interface between the hardware and software components. The aforementioned adjustments highlight the iterative aspect of system development, in which recognized issues are resolved in order to improve the system's operation and better the user experience.

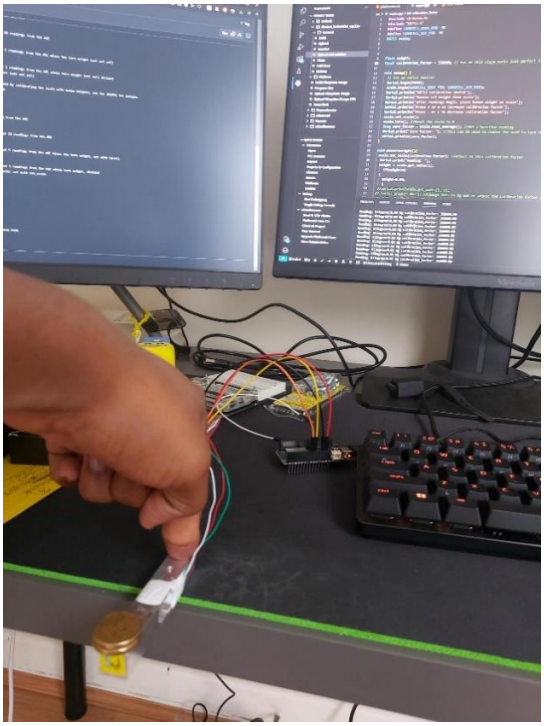

Fig19.Testing load cell functionality.

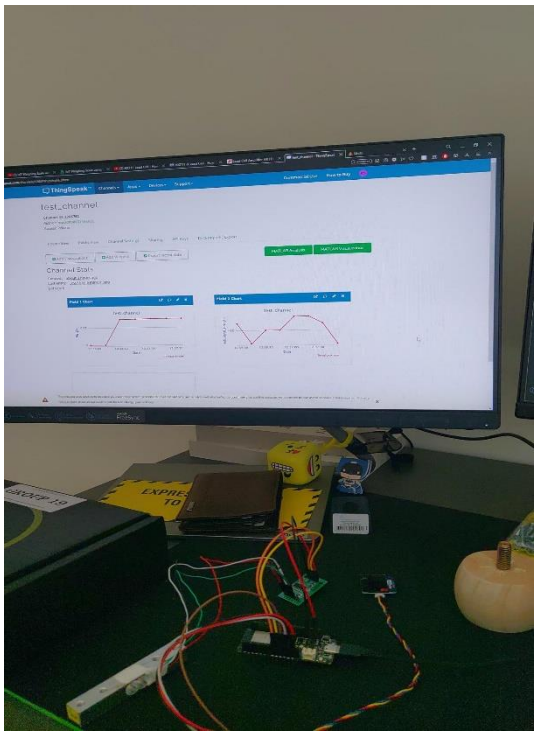

Fig20. Sending data to the server for the first time.

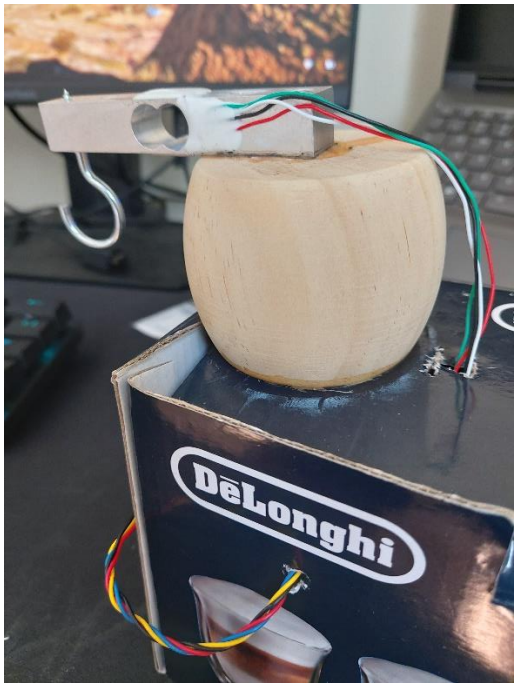

Fig21. Structure’s deformation

3.2. Outcomes

If the recorded weight is found to be lower than the pre-established threshold of 300 grams, the application is designed to carry out a certain set of instructions. The major purpose of the system, in the given context, is to expeditiously transmit a notice to swiftly inform the nurse on duty. This warning acts as an important reminder for the nurse to perform the duty of replacing the intravenous (IV) bag, therefore assuring the continuous administration of necessary fluids or medicine to the patient.

The program, as part of its notification system, leverages numerous user interface components to deliver the message efficiently. Significantly, the device integrates an OLED (Organic Light-Emitting Diode) display as a visual medium for presenting the procedure of IV bag changing. The inclusion of this visual representation is of utmost importance as it serves to uphold transparency and offer a dynamic portrayal of the bag-changing process, hence augmenting the lucidity and comprehensibility of the method.

Moreover, after the successful completion of the change of the intravenous (IV) bag, the application guarantees the continuity of the signal. The continuous signal functions as a form of validation and reinforces the importance of successfully completing the job, affirming that the IV system is again functioning efficiently once more. The application's functionality, which is determined by the threshold weight, serves the dual purpose of facilitating patient care and enhancing the efficiency of nurses' tasks by providing timely notifications and clear visual feedback throughout the process of changing IV bags.

3.3. Recommendations and Future Work

One potential route for future enhancement entails decreasing the transaction delay between data transmissions. By reducing the interval between sensor readings and data transmission, the system may offer increased frequency and fast dissemination of information. This feature has significant value in dynamic contexts characterized by the necessity for timely monitoring and response to rapid changes.

There are further measures that may be used to improve the precision of weight measurements. This may involve optimizing the structure of the prototype, sensor calibration procedure, enhancing the accuracy of data processing algorithms, or investigating cutting-edge sensor technology. The endeavour to achieve enhanced precision in measurement guarantees that the system is capable of satisfying rigorous criteria in crucial domains, such as medical or industrial environments.

An Analysis of Potential Commercialization Opportunities: In order to enhance the reach and influence of this system, it is imperative to take into account the potential for its commercialization. Assessing market demand, doing feasibility studies, and investigating prospective collaborations with manufacturers are crucial measures for expanding the reach of this technology. The process of commercialization has the potential to enhance the accessibility of this beneficial monitoring system, allowing it to be utilized by a wider array of users. Consequently, this expansion in accessibility can provide benefits for diverse businesses.

Future research should prioritize the enhancement of the connectivity between sensors and their corresponding applications. This entails the optimization of data transmission methods, the reduction of delay, and the establishment of a smooth and dependable interchange of information. The system's ability to monitor in real-time relies heavily on a strong and prompt connection.

The logical organization and presentation of facts are of utmost importance. In subsequent rounds, it is important to prioritize the enhancement of data display by ensuring its intuitiveness and ease of interpretation. An interface that is designed to be easily navigable and intuitive, along with effective visualizations and data representation, promotes the usability of the system and guarantees that users can understand and effectively utilize the data provided.

In summary, the recommendations and prospective avenues emphasize the dedication to furthering the system's functionalities, augmenting its applicability, and broadening its scope. By focusing on these specific areas, the system has the potential to act as a beneficial tool across several industries and make a meaningful contribution to the larger field of monitoring and data management technology.

4. CONCLUSION

The preceding discourse elucidates a methodical and proficient methodology employed in the conception and implementation of the monitoring system for intravenous (IV) bags. The project effectively demonstrates the successful incorporation of both hardware and software elements, perfectly connecting them with the main goal of assuring precise and timely monitoring of essential medical indicators. The system has a significant focus on energy efficiency, realistic power supply options, and precision in measurements.

Moreover, the project's methodical approach to calibration and preliminary testing aims to authenticate its functioning and dependability, with a particular emphasis on error margins and intervals for data transmission. The meticulous examination of these factors demonstrates a dedication to providing a solution that is both grounded in scientific principles and feasible in practical terms.

The proposals and future work delineate a coherent trajectory for continuous enhancement and enlargement of the system. The proposed measures involve the reduction of transaction intervals, improvement of measurement accuracy, investigation of commercialization opportunities, enhancement of sensor-to-app communication, and optimization of data display.

In summary, this project offers a noteworthy illustration of a successful philosophy that forms the foundation for the creation of a monitoring system. The system effectively combines theoretical rigor

with practical functionality, placing equal emphasis on its current capabilities and its potential for expansion and broader use across diverse sectors. The project's notable characteristics lie in its methodical approach, accuracy, and attention to energy saving. These qualities render it a significant addition to the realm of monitoring and data management technology.

5. REFERENCES

[1] Australian Institute of Health and Welfare, “Impact of COVID–19 on hospital care”, 03-Oct-2023. [Online]. Available. **https://www.aihw.gov.au/reports/hospitals/australias-hospitals-at-a-glance/contents/impact-of-covid-19-on-hospital-care** [Accessed: 31-Sept-2023].

[2 Australian Institute of Health and Welfare, “Admitted patients”, n.d. [Online]. Available. **https://www.aihw.gov.au/reports-data/myhospitals/sectors/admitted-patients#:~:text=from%202012%E2%80%9313%20to%202016,and%201.8%25%20in%20private%20hospitals.** [Accessed: 31-Sept-2023].

[3] Sarah Al-Mutlaq, Alex the Giant, sparkfun, “Load Cell Amplifier HX711 Breakout Hookup Guide”. [Online]. Available. https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all. [Accessed: 31-Sept-2023].

[4] bodge, “HX711 modules”, github.com, 24-Nov-2021. [Online]. Available. https://github.com/bogde/HX711. [Accessed: 30-Sept-2023].

[5] Indrek Luuk, “4-Wire Load Cell (1/5/10/20/200kg) with HX711 and Arduino”. [Online]. Available. https://circuitjournal.com/four-wire-load-cell-with-HX711. [Accessed: 30-Sept-2023].

[6] Sara Santos, Random Nerd Tutorials, “ESP 32 OLED Display with Arduino IDE”. [Online]. Available. https://randomnerdtutorials.com/esp32-ssd1306-oled-display-arduino-ide/. [Accessed: 30-Sept-2023].

[7] Simply Explained, “ESP32 + Arduino”. [Online]. Available. https://www.youtube.com/playlist?list=PLzvRQMJ9HDiQ3OIuBWCEW6yE0S0LUWhGU. [Accessed: 31-Sept-2023].

[8] K. Anipindi, "An Introduction to ThingSpeak," CodeProject, Nov. 23, 2014. [Online]. Available: https://www.codeproject.com/Articles/845538/An-Introduction-to-ThingSpeak. [Accessed: 30-Sept-2023].

[9] Simple Tutorial, “Beginner tutorials”. [Online]. Available. Our Tutorials! (mit.edu) [Accessed: 31-Sept-2023].

6. APPENDICES

Appendix A: ESP-32 Code sheet.

<u>Please read comments to understand the usage of variables and functions.</u>

Final.ino

Modules, variable definitions, and class objects:

```
//All modules required to run the program
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ThingSpeak.h>
#include "HX711.h"
//-------------------------------------------------------------------------------------------------------------------
#define LOADCELL_DOUT_PIN  35 //HX711 DT PIN
#define LOADCELL_SCK_PIN  15 // HX711 SCK Pin
#define CHANNEL_ID1 2286761 //Write Channel ID
#define CHANNEL_ID2 2287107 //Read Channel ID
#define CHANNEL_WRITE_API_KEY "EPBNJOD1B8ZXNTRM" //API key for write channel
#define CHANNEL_READ_API_KEY "Q682151FXCS2RFOY" //API key for read channel
#define calibration_factor 211000 // Calibration factor for the selected load cell
#define zero_factor 193373 //Offset value for the selected load cell
#define WIFI_NETWORK "Adharsh" //Mobile phone hotspot name(SSID)
#define WIFI_PASSWORD "oiky2507" //Mobile phone hotspot password
#define WIFI_TIMEOUT_MS 10000 //U dont want ESP32 to keep trying to connect infinitely in case the name and password dont match, or if the network is not available.
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define micS_to_S_Factor 1000000
#define time_to_sleep 10 //10 seconds sleep (After every data transmission) or (when the bag is being changed)
//-------------------------------------------------------------------------------------------------------------------
WiFiClient client; //WiFiClient object
HX711 scale; //HX711 object
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1); //SSD1306 object
//-------------------------------------------------------------------------------------------------------------------
```

Global variables:

```
float weight; //Global weight variable that holds the weight data
int digital_button = 0; //Global variable that holds the data from ThingSpeak generated by the user through the Application
static const uint8_t image_data_Image[1024] = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x03, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x01, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x7f, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x0f, 0xff, 0xf8, 0x00, 0x0f, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x0f, 0xff, 0xf8, 0x00, 0x07, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x0f, 0xff, 0xf8, 0x00, 0x03, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x0f, 0xff, 0xf8, 0x00, 0x01, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x0f, 0xff, 0xf8, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x1f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00, 0x07, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x03, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x03, 0xff, 0xff,
```

```
    0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x01, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x01, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff,
    0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x01, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff,
    0xff, 0xff, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x01, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xff,
    0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xff,
    0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff,
    0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff,
    0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xff,
    0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff,
    0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f,
    0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f,
    0xff, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x1f,
    0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f,
    0xff, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07,
    0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03,
    0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
    0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
}; //128x64 monochrome image (caution symbol) information stored in an array (Block-size:8bit) (Byte-order:Little-eldian)
//---------------------------------------------------------------------------------------------------------------------
```

Connecting to network using Wi-Fi:
Function connectToWiFi():

```
// Function that connects the ESP-32E to Wi-Fi
void connectToWiFi(){
  Serial.print("Connecting to Wifi");
  WiFi.mode(WIFI_STA); //Station mode, to connect to available networks
  WiFi.begin(WIFI_NETWORK,WIFI_PASSWORD); //try to connect to the specified wifi SSID

  unsigned long startAttemptTime = millis(); //millis() function returns the current uptime of the ESP32

  while(WiFi.status() != WL_CONNECTED && millis() - startAttemptTime < WIFI_TIMEOUT_MS){ //try for 10 seconds
    Serial.print(".");
    delay(100);
  }
  if(WiFi.status() != WL_CONNECTED){ // The case if failed to connect, try again.
                                    // Displaying appropriate messages in the 128x64 OLED display
  Serial.println("Failed to connect to the network!");
  Serial.println("Trying again");
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Not connected");
  display.println();
  display.println("to Wi-Fi!");
  display.display();
  delay(1000);
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Trying");
  display.println();
  display.println("again!");
  display.display();
  connectToWiFi(); //calling the function again to try again
  }
  else{ // The case if successfully connected.
        // Displaying appropriate messages in the 128x64 OLED display
  Serial.println("Connected to the network!");
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("   Wi-Fi");
  display.println();
  display.println("connected!");
  display.display();
  Serial.println(WiFi.localIP()); //displays the IP of the connected network
  }
}
//---------------------------------------------------------------------------------------------------------------------
```

Measuring weight and displaying appropriate messages:

Function measureweight():

```
  else if(weight<0.3 and digital_button == 2){ //This case is when the saline/IV bag goes to low level and a notification is to be sent to the user
                                               //Notice that the digital_button value is 2, which means the user wants the weight to be measured.
                                               //We have set the threshold weight to be 300 grams, anything less and the user is notified.
                                               //This case only displays an alert in the 128x64 OLED display and not in the app, the notification/alert that the user gets is through the app's logic.
                                               //On the 128x64 OLED display,a CAUTION SYMBOL followed by "Fluid level low!" message are displayed.
    display.clearDisplay();
    display.drawBitmap(0, 0, image_data_Image, 128, 64, 1);
    display.display();
    delay(1500);
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0,0);
    display.println("  Fluid  ");
    display.println();
    display.println("level low!");
    display.display();
  }
  else{                        //This case is when the weight is above the threshold and the user just wants to measure the weight.
                               //It just displays the weight data in the 128x64 OLED display.
                               //Interestingly, this is the case that gets executed under normal use.
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.print("Weight: ");
  display.setTextSize(3);
  display.setCursor(0, 20);
  display.print(weight);
  display.print(" kg");
  display.display();
  }
  delay(500);    //A small delay is required for the load cell and HX711 to function properly.
                 //This acts as a delay between consecutive measurements.
}
//-------------------------------------------------------------------------------------------------------------------
```

```
// Function that measures weight and controls the display under different scenarios
void measureweight(){
  weight = scale.get_units(); //Function that returns analog data converted to human understandable form.
    if(weight<0)              //If the wheatstone bridge thats supposed to experience compression undergoes expansion and vice versa,
                              //we get a negative value, to avoid this we implement this condition
  {
    weight=0.00;
    }
    //Displaying weight data in the serial monitor
  Serial.print("Kilogram:");
  Serial.print( weight);
  Serial.print(" Kg");
  Serial.println();

  if(digital_button == 0){  //If we get no input data from ThingSpeak, it means that connection is load between the server and the APP or the server and the ESP-32E,
                            // this case displays that information. On the 128x64 OLED display, "Connection lost!" message is displayed
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0,0);
    display.println("Connection");
    display.println();
    display.println("  lost! ");
    display.display();
  }
  else if(digital_button == 1) // We get a value of one from the server if the User wants to stop measuring the weight and change the saline/IV bag,
                               // this case corresponds to this scenario. On the 128x64 OLED display, "Changing bag!" message is displayed
  {
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0,0);
    display.println("Changing");
    display.println();
    display.println("  bag! ");
    display.display();
    delay(1000);
  }
```

Sleep function after the data has been transmitted:

Function sleep_after_transmission():

```
//Function that puts the ESP-32E to deep sleep after transmitting data successfully.
//All the time when the ESP-32E is in deep sleep, "Sleeping!" is displayed on the 128x64 OLED.
//This helps with knowing the state of the microcontroller.
void sleep_after_transmission(){
  Serial.println("Sleeping after sending data!");
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Sleeping!");
  display.display();
  delay(1500);
  esp_sleep_enable_timer_wakeup(time_to_sleep*micS_to_S_Factor); //In-built function that calculates the amount of time the ESP-32E needs to sleep.
  esp_deep_sleep_start(); //In-built function that puts the ESP-32E to sleep for the calculated amount of time.
}
//-------------------------------------------------------------------------------------------------------------------
```

Sleep function while the bag is being changed:

Function sleep_while_changing_bag():

```
//Function that puts the ESP-32E to deep sleep while the bag is being changed.
//All the time when the ESP-32E is in deep sleep, "Sleeping!" is displayed on the 128x64 OLED.
//This helps with knowing the state of the microcontroller.
void sleep_while_changing_bag(){
  Serial.println("Sleeping until bag is changed!");
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Sleeping!");
  display.display();
  delay(1500);
  esp_sleep_enable_timer_wakeup(time_to_sleep*micS_to_S_Factor); //In-built function that calculates the amount of time the ESP-32E needs to sleep.
  esp_deep_sleep_start(); //In-built function that puts the ESP-32E to sleep for the calculated amount of time.
}
//-------------------------------------------------------------------------------------------------------------------
```

Setup function:

Function setup():

```
// Function that is used to setup all the components and begin their functionality if required.
void setup() {
  //Specifying I2C address for the 128x64 OLED display.
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  delay(2000); // Delay for the display to configure.

  Serial.begin(9600); //Start serial monitor at baud rate = 9600

  connectToWiFi(); //Function call that helps with getting connected to the Wi-Fi.
  ThingSpeak.begin(client); //Function call that helps with connecting to the ThingSpeak server with the help of the connected Wi-Fi network.

  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN); //Setting up the load cell and the HX711 at pins 35(DT) and 15(SCK)
  scale.set_scale(calibration_factor); //Setting the calibration factor, found by calibirating the load cell and HX711.
  scale.set_offset(zero_factor); //Setting the zero factor or the scale offset value, this is also found during calibration. This helps with applications that need weight to be monitored continuously.
}
//-------------------------------------------------------------------------------------------------------------------
```

Loop Function:

Function loop():

```
//Function that runs repeatedly.
void loop() {
  digital_button = ThingSpeak.readIntField(CHANNEL_ID2,3,CHANNEL_READ_API_KEY); //Reading value from ThingSpeak server, generated by the user. (the function returns 0 if failed to retrieve data)

  measureweight(); //Function call to measureweight();

  ThingSpeak.setField(1,weight); //Setting field1 value as weight data in ThingSpeak
  ThingSpeak.setField(2,WiFi.RSSI()); //Setting field2 value as Wi-Fi signal strength data in ThingSpeak

  Serial.println(digital_button); //Printing the digital_button value to serial monitor, useful to understand the workflow/debugging.

  if(digital_button == 2 || digital_button == 1){ //Only if digital_button values is either 1 or 2, we write the data to ThingSpeak server.
  ThingSpeak.writeFields(CHANNEL_ID1, CHANNEL_WRITE_API_KEY); //Function that writes the set field values to the server given the channel name and write API key.
  }

  if(digital_button != 0){ //ESP-32E goes to sleep only if it successfully transmits the weight data.
                           // If not connected to the server, it waits until the connection is re-established.
    if(digital_button == 1){
      delay(1000);
      sleep_while_changing_bag(); //Sleep during the change of bag.
    }
    else{
      delay(1000);
    sleep_after_transmission(); //Sleep after successfully transmitting data.
    }
  }
}
```

Appendix B: One of the test data.

	Time	Weight(kg)	Actual weight(kg)	8/10/2023	Time	wifi_signal
2	02:45:35 UTC	1.21421	1.093		02:45:35 UTC	-47
3	02:45:54 UTC	1.19137	1.042		02:45:54 UTC	-54
4	02:46:13 UTC	1.18016	1.031		02:46:13 UTC	-42
5	02:46:54 UTC	1.16635	1.017		02:46:54 UTC	-45
6	02:47:13 UTC	1.16248	1.011		02:47:13 UTC	-32
7	02:47:32 UTC	1.15974	1.008		02:47:32 UTC	-34
8	02:47:50 UTC	1.1539	1.001		02:47:50 UTC	-35
9	02:48:09 UTC	1.15671	0.998		02:48:09 UTC	-40
10	02:48:28 UTC	1.14914	0.989		02:48:28 UTC	-48
11	02:48:48 UTC	0.05263	0		02:48:48 UTC	-32
12	02:49:06 UTC	0.05191	0		02:49:06 UTC	-24
13	02:49:25 UTC	0.05225	0		02:49:25 UTC	-28