

P.U.M.P.E.D: Precision Weighing, User-Friendly Heating, Mobile App Integration, Printed Labels, Expiration Tracking, and Data Management



Group 16 Authors:

Isaiah Robinson - *Computer Engineering*
Jason Devito - *Computer Engineering*
Jamieson Foley - *Electrical Engineering*
Zachary Celentano - *Electrical Engineering*

Reviewer Committee:

Dr. Suboh Suboh	ECE	Associate Lecturer
Dr. Wei Sun	ECE	Associate Professor
Dr. Hadi Kamali	ECE	Assistant Professor
Victoria Rodriguez	Sponsor/Engineering support	
Matthew Aberman	Sponsor/Engineering support	

Mentor:

Dr. Chung Yong Chan

Sponsors:

Victoria Rodriguez
Matthew Aberman

Table of Contents:

List of Tables:.....	6
List of Figures:.....	7
Chapter 1 - Executive Summary:.....	8
Chapter 2 - Project Description:.....	8
2.1: Project Background.....	8
2.2: Previous Work & Existing Products.....	8
2.3: Project Motivation.....	10
2.4: Goals & Objectives.....	11
Basic Goals:.....	11
Basic Objectives:.....	11
Stretch Goals:.....	12
Stretch Objectives:.....	12
2.5: Project Features.....	12
2.6: Required Engineering Specifications.....	13
2.7: Hardware Block Diagram.....	14
2.8: Software Flowchart.....	15
2.9: Prototype Illustration.....	16
2.10: House of Quality.....	17
Chapter 3 - Research:.....	18
3.1: Hardware Technologies.....	18
3.1.1: Temperature Measurement Methods.....	18
3.1.1.1: Contact Temperature Sensors.....	18
3.1.1.2: Non-Contact Temperature Sensors.....	19
3.1.1.3: Temperature Sensor Method Selection.....	20
3.1.2.4: Contact Temperature Sensors.....	20
3.1.1.5: DS18B20 (Waterproof Digital Temperature Sensor).....	20
3.1.1.6: PT100 (RTD - Resistance Temperature Detector).....	21
3.1.1.7: NTC Thermistor (Negative Temperature Coefficient Sensor - 10K Ω or 100K Ω).....	21
3.1.1.8: Temperature Detector Selection.....	22
3.1.2: Weight Measurement Methods.....	23
3.1.2.1: Load Cell (Strain Gauge-Based).....	23
3.1.2.2: Piezoelectric Force Sensors.....	23

3.1.2.3: Capacitive Force Sensors.....	24
3.1.2.4: Weight Measurement Method Selection.....	24
3.1.2.5: Load Cells.....	25
3.1.2.6: Phidgets CZL635 (5kg).....	25
3.1.2.7: SparkFun SEN-14729 (5kg).....	25
3.1.2.8: TAL220B (5kg).....	26
3.1.2.9: Load Cell Selection.....	26
3.1.3: Display Methods.....	27
3.1.3.1: LCD Display.....	27
3.1.3.2: LED Display.....	27
3.1.3.3: Display Method Selection.....	28
3.1.3.4: LCD Displays.....	28
3.1.3.5: 16x2 Character LCD (HD44780 - I2C or Parallel Interface).....	28
3.1.3.6: 128x64 Monochrome LCD (ST7920 - SPI/I2C/Parallel Interface).....	29
3.1.3.7: 2.8-inch TFT LCD (ILI9341 - SPI Interface, Full Color).....	29
3.1.3.8: LCD Display Selection.....	29
3.1.4: Label Printing Methods.....	30
3.1.4.1: Thermal Printing.....	30
3.1.4.2: Laser Printing.....	31
3.1.4.3: Inkjet Printing.....	31
3.1.4.4: Label Printer Selection.....	32
3.1.4.5: Thermal Printers.....	32
3.1.4.6: MUNBYN IMP001 58mm Thermal Printer (Serial/UART & USB).....	33
3.1.4.7: PRIMUZ Micro Thermal Printer.....	33
3.1.4.8: ASPIRON 58mm Embedded Thermal Printer Module (UART & TTL Interface).....	33
3.1.4.9: Thermal Printer Selection.....	34
3.1.5: Heating Methods.....	35
3.1.5.1: Induction Heating.....	35
3.1.5.2: Conduction Heating.....	35
3.1.5.3: Convection Heating.....	36
3.1.5.4: Heating Method Selection.....	36

3.1.5.5: Heating Elements.....	37
3.1.5.6: Electric Heating Pad - 14cm x 5cm by Adafruit.....	37
3.1.5.7: MECCANIXITY Silicone Heating Pad.....	38
3.1.5.8: MakerTechStore Cartridge Heater.....	38
3.1.5.9: Heating Element Selection.....	39
3.1.6: MCU.....	40
3.1.6.1: MCU vs FPGA.....	40
3.1.6.2: MCU Selection.....	40
3.1.6.3: ESP32.....	41
3.1.6.4: STM32.....	41
3.1.6.5: Nordic nRF52840.....	42
3.1.6.6: MCU Comparisons & Choice.....	43
3.1.7: Power Supply.....	43
3.1.7.1: Lithium Ion Battery Pack.....	43
3.1.7.2: Grid Power.....	44
3.1.7.3: Small Solar Panel with Battery Storage.....	45
3.1.7.4: Power Supply Selection.....	45
3.1.7.5: Grid Power.....	46
3.1.7.6: MEAN WELL LRS-100-24 (SMPS Adapter):.....	47
3.1.7.7: Alitove Power Supply Adapter(AC/DC Adapter):.....	47
3.1.7.8: Bel Signal Transformer with Rectifier Circuit:.....	47
3.1.7.9: Grid Power Selection.....	48
3.1.8: Communication Protocol.....	49
3.1.8.1: Serial Peripheral Interface (SPI).....	49
3.1.8.2: Inter-Integrated Circuit (I2C).....	49
3.1.8.3: Universal Asynchronous Receiver/Transmitter (UART).....	49
3.1.8.4: Communication Protocol Selection.....	50
3.2: Software Technologies.....	50
3.2.1: Mobile Application Languages.....	51
3.2.1.1: Swift.....	51
3.2.1.2: Flutter & Dart.....	52
3.2.1.3: React Native:.....	52
3.2.1.4: JavaScript (Backend Development).....	53
3.2.1.5: App Language Selection.....	54

3.2.2: Backend Frameworks.....	55
3.2.2.1: Node.js (JavaScript).....	56
3.2.2.2: Django (Python).....	57
3.2.2.3: Ruby on Rails.....	58
3.2.2.4: Backend Framework Comparisons.....	58
3.2.2.5: Backend Framework Selection.....	60
3.2.3: Database Options.....	60
3.2.3.1: PostgreSQL.....	60
3.2.3.2: MongoDB.....	61
3.2.3.3: MySQL.....	61
3.2.3.4: Database Selection.....	63
3.2.4: Cloud Storage.....	63
3.2.4.1: Benefits of Cloud Storage.....	64
3.2.4.2: Drawbacks of Cloud Storage.....	64
3.2.4.3: Amazon Web Services (AWS).....	64
3.2.4.5: Firebase.....	65
3.2.4.6: Google Cloud Platform (GCP).....	65
3.2.4.7: Evaluating Potential Cloud Services.....	66
3.2.5: Data Transfer Methods For ESP32.....	67
3.2.5.1: Bluetooth Classic.....	67
3.2.5.2: Bluetooth Low Energy (BLE).....	68
3.2.5.3: Barcodes.....	68
3.2.5.4: QR Codes.....	69
3.2.5.5: Data Transfer Evaluations & Comparisons.....	70
3.2.5.6: Data Transfer Conclusions.....	72
Chapter 4 - Standards & Design Constraints:.....	73
4.1: Standards & Regulations.....	73
4.2: Food & Drug Administration.....	74
4.2.1: Food Labeling Guide.....	74
4.2.2: Food Storage.....	74
4.3: Center for Disease Control & Prevention.....	74
4.3.1: Food Preparation.....	75
4.4: Communication Standards.....	75
4.5: User Constraints.....	76

4.5.1: Parent Constraints.....	76
4.5.2: Medical Faculty Constraints.....	77
4.6: Additional Constraints.....	78
4.6.1: Environmental Constraints.....	78
4.6.2: Power Constraints.....	79
5.1: ChatGPT in the Software Design Process.....	81
5.2: Cursor AI for Code Generation.....	81
5.3: Direct Comparison of ChatGPT and Cursor AI.....	82
5.4: ChatGPT in the Hardware Design Process.....	83
5.5: Gemini in Hardware Research and its Limitations.....	85
5.6: Testing ChatGPT & Result Analysis.....	86
Chapter 6 - Hardware Design:.....	86
6.1: Rectification for Transformer.....	86
6.2: 12V Regulator PSU.....	86
6.3: 5V Regulator PSU.....	87
6.4: 3.3V Regulator PSU.....	88
6.5: Load Cell Amplifier Design.....	89
6.6: MCU Design.....	91
Chapter 7 - Software Design:.....	92
7.1: Use Case Diagram.....	92
7.2: State (Activity) Diagram.....	93
7.3: Class Diagram.....	94
Chapter 8 - System Fabrication/Prototype Construction:.....	96
Chapter 9 - System Testing & Evaluation:.....	96
Chapter 10 - Administrative Content:.....	96
10.1: Budget Estimates.....	96
10.2: Project Milestones.....	97
Chapter 11 - Conclusion:.....	99
Appendix A - References:.....	99
Appendix B - Copyright Permission:.....	102
Appendix C - Data Sheet:.....	102
Appendix D - Software Code:.....	102
Appendix E - ChatGPT Prompts & Outcomes:.....	102

List of Tables:

Table 2.1 - Engineering Specifications.....	13
Table 2.2 - House of Quality.....	17
Table 3.1 - Temperature Detector Method Comparisons.....	20
Table 3.2 - Temperature Detector Comparisons.....	22
Table 3.3 - Weight Measurement Comparison.....	24
Table 3.4 - Load Cell Comparison.....	26
Table 3.5 - Display Method Comparison.....	28
Table 3.6 - LCD Display Comparison.....	29
Table 3.7 - Label Printer Comparisons.....	32
Table 3.8 - Thermal Printer Comparisons.....	34
Table 3.9 - Heating Method Comparisons.....	36
Table 3.10 - Heating Pad Comparisons.....	39
Table 3.11 - MCU Comparisons.....	43
Table 3.12 - Power Supply Comparisons.....	45
Table 3.13 - Grid Power Comparisons.....	48
Table 3.14 - Communication Protocol Comparisons.....	50
Table 3.15 - App Language Comparisons.....	54
Table 3.16 - Backend Framework Comparisons.....	59
Table 3.17 - Database Comparisons.....	62
Table 3.18 - Cloud Storage Comparisons.....	66
Table 3.19 - Data Transfer Comparisons.....	71
Table 6.1 - Hardware Components.....	91
Table 10.1 - Bill of Materials.....	96

List of Figures:

Figure 2.1 - Hardware Block Diagram.....	14
Figure 2.2 - Software Flowchart.....	15
Figure 2.3 - Prototype.....	16
Figure 6.1 - Rectifier Schematic.....	86
Figure 6.2 - 12V Regulator PSU Schematic.....	87
Figure 6.3 - 5V Regulator PSU Schematic.....	88
Figure 6.4 - 3.3V Regulator PSU Schematic.....	89
Figure 6.5 - Load Cell Amplifier Schematic.....	90

Figure 6.6 - MCU Schematic.....	91
Figure 7.1 - Software Use Case Diagram.....	93
Figure 7.2 - Software State Diagram.....	94
Figure 7.3 - Software Class Diagram.....	96

Chapter 1 - Executive Summary:

Chapter 2 - Project Description:

2.1: Project Background

Breastfeeding parents often face the challenge of measuring, labeling, heating, and storing breast milk effectively. Traditional methods require manually writing the weight, date, and time on a milk storage bag or container, which can be both time-consuming and prone to human error. In addition, accurate records are crucial not only for understanding how much milk is being stored and consumed but also for ensuring the milk's freshness and safety for the infant.

The purpose of this project is to design and implement a device and application that pair together to accurately measure, heat, catalogue, and provide feedback on the amount of breast milk stored, how long it has been stored for, and show possible allergens to be aware of. With this, parents and healthcare providers can easily manage the supply of breast milk available for infants without risk of human error, all while saving significant time that would have been used to accomplish the same task.

There is a growing need for systems that can store personal healthcare and dietary data. In the context of newborn dietary information, it is clear how this growing need translates; dietary data management for children is of the utmost importance for both the children themselves and also the parents, as being able to easily manage an infants amount of food available saves time and avoids issues that are otherwise unnecessary, like human error when recording pumping amounts and dates.

2.2: Previous Work & Existing Products

This project does not iterate directly from any previous work; the idea itself was formed by sponsors Victoria Rodriguez and Matthew Aberman in response to problems they encountered when managing the food supply with accurate labels for their children. The issue that they want resolved is that there were cumbersome amounts of expired milk that could no longer be used or even donated to hospitals, for the reason that they could not effectively track the data on what has been pumped.

There are few products that perform separate individual functions of the proposed project. Namely, there are many brands that make milk bags that have measurement markings externally to show the amount stored within the bag. The advantage these products have is their convenience. With labels and markings on the bag itself, it takes away the extra effort of needing to manually manage the milk that is pumped. Where our system differs is that it offers a mobile app that will provide reminders and updates on pertinent information, like approaching expiration of milk. Another product that can be used in this context is a label maker. A parent can pump their milk, record the amount, then put the information regarding the milk into a label maker. Our product would remove the intermediate step and directly record the data of the milk and print a label immediately. Furthermore, there are also milk bag organizers sold that allow milk bags to be separated and stored in freezers; these organizers exclusively help with storage, where ours would not necessarily aid in physical storage, but rather in reminding parents when their supply is low, expiring soon, or some combination of the two scenarios.

When using the breast milk that has been stored, many parents have to heat the milk up to a temperature suitable for the baby's consumption. What is common practice is to place cold or room temperature milk bags in hot water baths to heat the milk, which can take excessive amounts of time. Adding the capability to warm up the milk bags in our design while also monitoring the temperature of the bag could help even further by saving parents time needed to ensure their children's food is at the temperature needed. In our case, it would make sense to model our heater after technology used in mug warmers. Mug warmers are ideal to take inspiration from because they typically do not heat past a temperature of 200 °F, which is the temperature most polyethylene plastics begins to melt. In our design, using similar topology, we would monitor the temperature to ensure product safety, while keeping it hot enough that someone could place their milk on for a short duration and prepare the milk to consume in a reasonable time frame. The monitoring of the temperature would most

effectively be achieved through thermal coupling. Through this, we can send temperature data from the milk bag to our mobile app, where parents will be alerted of the temperature of their children's food.

Parents who use these products previously described must manually take inventory of what is pumped, the time it is pumped, and the amount they have stored, while also keeping track of potential expirations, ideal temperature, and allergens. All of these devices and aids share a common flaw; that being time consumption. The previously mentioned products require an extra step, and thus more time to manage and maintain a parent's supply of breast milk. While parents already have much to keep track of for their children, adding these additional few things can be cumbersome, which is what our device aims to minimize. While all of these systems accomplish their specifically designed task, there is no product that combines all of the strengths of the individual products to provide in-depth data monitoring and ease of use for parents who are not able to manage the many complexities that can arise when pumping breast milk. The system we aim to design would be available on a large commercial scale in a form factor that is cheap for parents, boast similar portability that of kitchen appliances, in both the device itself, as well as the application that allows consumers to track their data at any time.

2.3: Project Motivation

This project was motivated by:

1. **Sponsorship:** Having a sponsored project as college students is a substantial motivating factor because of our lack of funding, while also giving us valuable project experience for our field.
2. **Accuracy:** Providing consistent and precise measurement of breast milk weight.
3. **Traceability:** Using an integrated mobile app to track stored milk bags, expiration dates, and consumption over time.
4. **Peace of Mind:** Helping parents avoid confusion about which bag was pumped when and how long it remains viable, thus improving overall feeding management.
5. **Convenience:** Automating the labeling process to save time and reduce mistakes.

By combining a custom scale with printing functionality and automatically syncing data to a mobile application, this device addresses a real need for

modern, tech-savvy parents who want a seamless solution to manage their breast milk supply.

2.4: Goals & Objectives

Our project aims to design an affordable device that accurately measures the amount of milk in a breast milk storage bag. To ensure the user will not waste the milk they produce, we plan to include a label printer to create labels with the measured volume, current date, and time so that the user can keep track of their supply. Additionally, we will develop a mobile app that allows the user to interact with the data, which will be automatically stored in memory after the device prints a label. With milk storage being one of our main goals, we also want to include a milk warmer. The user will have the functionality to either heat or weigh their milk. Below are our general goals and specific objectives for achieving this plan.

Basic Goals:

- Build a product that allows parents to easily weigh, label, and store milk bags.
- Build an application that allows an accessible interface with the data of bagged milk.
- Build a system that ensures milk is stored and used at safe temperatures, helping prevent spoilage or degradation of nutrients.

Basic Objectives:

- Implement a load cell and ADC for accurate weight measurement. Convert the measured weight to fluid ounces for parents ease of use.
- Integrate a label printer to produce labels containing all relevant information, including fluid ounces, date, time, and temperature.
- Show real-time weight, date, and time on the LCD prior to label printing.
- Measures the temperature of bagged milk using thermal coupling.
- Implement a heating element to allow parents to more freely manage milk temperature.
- Implement a custom power supply to power the device and all peripherals.
- Any surface that makes contact with the milk bags will be able to be taken apart for sanitation purposes.

- Develop an iOS app to interface with a database of information regarding stored milk.
- Provide features for viewing, editing, and managing milk inventory.
- Send alerts for impending expiration dates and low general storage.

Stretch Goals:

- Include an option to track allergens in milk bags.
- Include a cloud storage system
- Track all bagged milk for up to a year.
- Suggest how much milk a Mother needs to pump.

Stretch Objectives:

- Build a custom thermal printer and its subsequent PCB to minimize form factor.
- Design the power supply to be able to work with batteries so the product can be portable.

2.5: Project Features

As stated previously, the overall functionality of the project is to provide a quick and easy alternative to manually writing down the current date, time, and weight of breast milk in a storage bag. To accomplish this task, and more, our project consists of five main features.

- **Scale:** The function of the scale is to measure the weight of the breast milk bag and convert that weight into weight. The device would then store that measured weight for use in all the other features.
- **LCD Screen:** The function of the LCD screen is to display the current date, time, and measured weight provided by the scale.
- **Button Functionality:** Buttons will serve a way for the user to choose which action they want the device to run.
- **Label Printer:** The function of the label printer is to, with the press of a button, print out a label with all the required values. This label could then be directly attached to the breast milk bag, allowing for easy access to all the information on each bag when stored.
- **Mobile App:** The function of the mobile app is to keep track of each breast milk bag in storage. It should be able to automatically load

information from a database. Additionally, the app should be able to track how much milk is left in each bag and send alerts about upcoming expiration dates. In the case of a trip or vacation, the app should be able to portion out the amount of breast milk required over a specified period of time.

- **Bluetooth Connectability:** The function of Bluetooth is to allow the device to directly connect to the users phone, allowing access to the mobile app
- **Temperature Sensor:** The function of the temperature sensor is to measure the temperature of the breast milk bag while it is on the scale.
- **Heating Element:** The function of the heating element is to provide a controlled warming feature for the breast milk bag. This allows users to gently heat milk to an optimal feeding temperature without needing a separate bottle warmer.
- **Custom Power Supply:** To power the heating feature, we will design a DC switching power supply, researching the appropriate topology to efficiently meet the heater's power requirements. The heating process will be closely monitored to prevent overheating and ensure uniform temperature distribution.

2.6: Required Engineering Specifications

Table 2.1 - Engineering Specifications

Parameter	Value	Description
Weight Accuracy	± 1 Grams	Accurate weight measurements.
Temperature Accuracy	± 1 °F	Accurate temperature measurements.
Label Size	2" x 1"	Large enough to display all data.
LCD Display Time	< 1 Second	Fast response time.
Power Consumption	< 80 Watts	Power all peripherals in the device.
Bluetooth Connectivity Response Time	< 5 Seconds	Efficient interface connectivity.
Local Data Storage Capacity	> 20 entries	Enough data storage capabilities to give parents enough information.

Heating Temperature	90-100 °F	Must heat milk to the proper temperature of consumption.
Cost Estimate	< \$800	Needs to be cost effective to manufacture.

2.7: Hardware Block Diagram

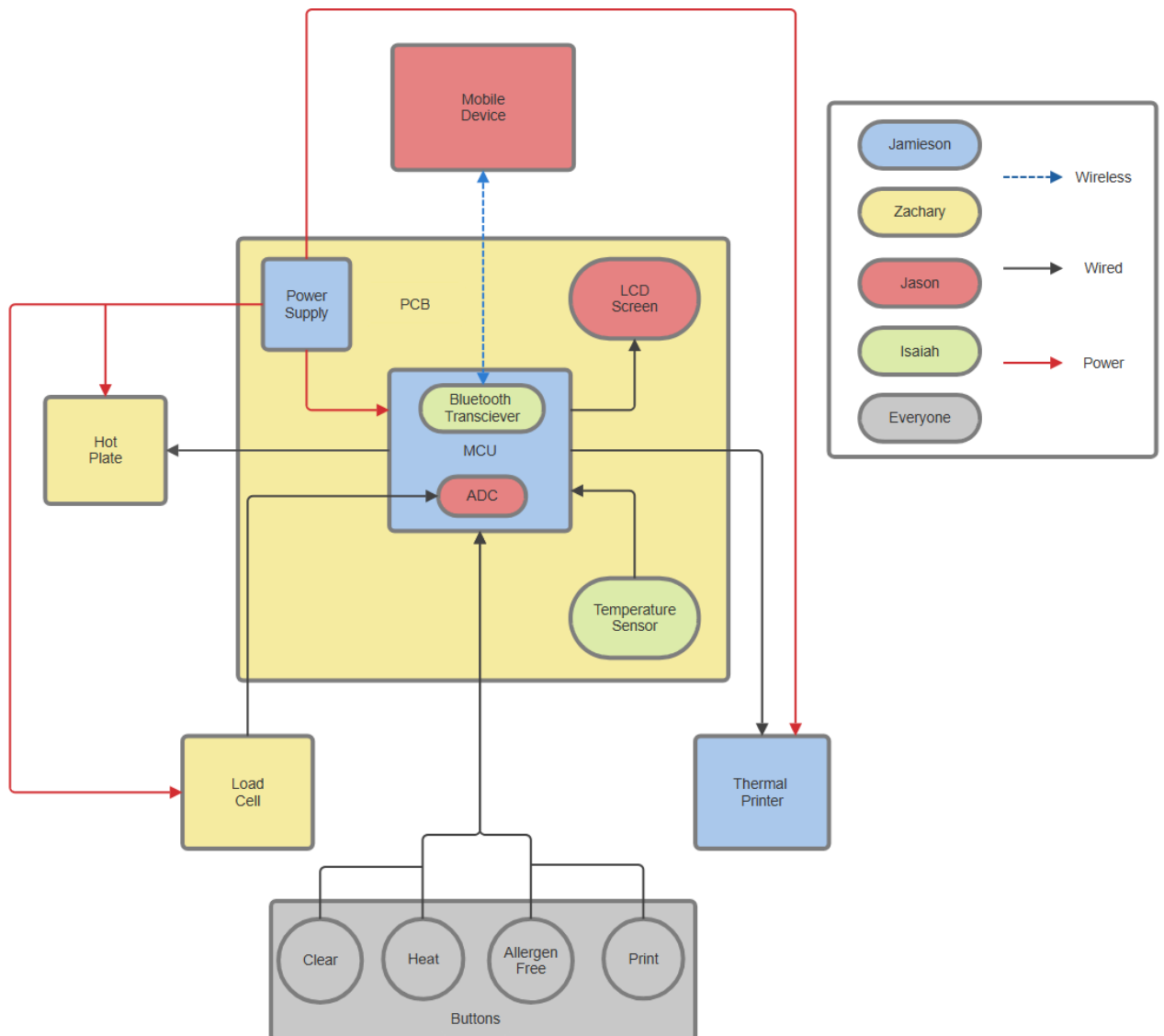


Figure 2.1 - Hardware Block Diagram

2.8: Software Flowchart

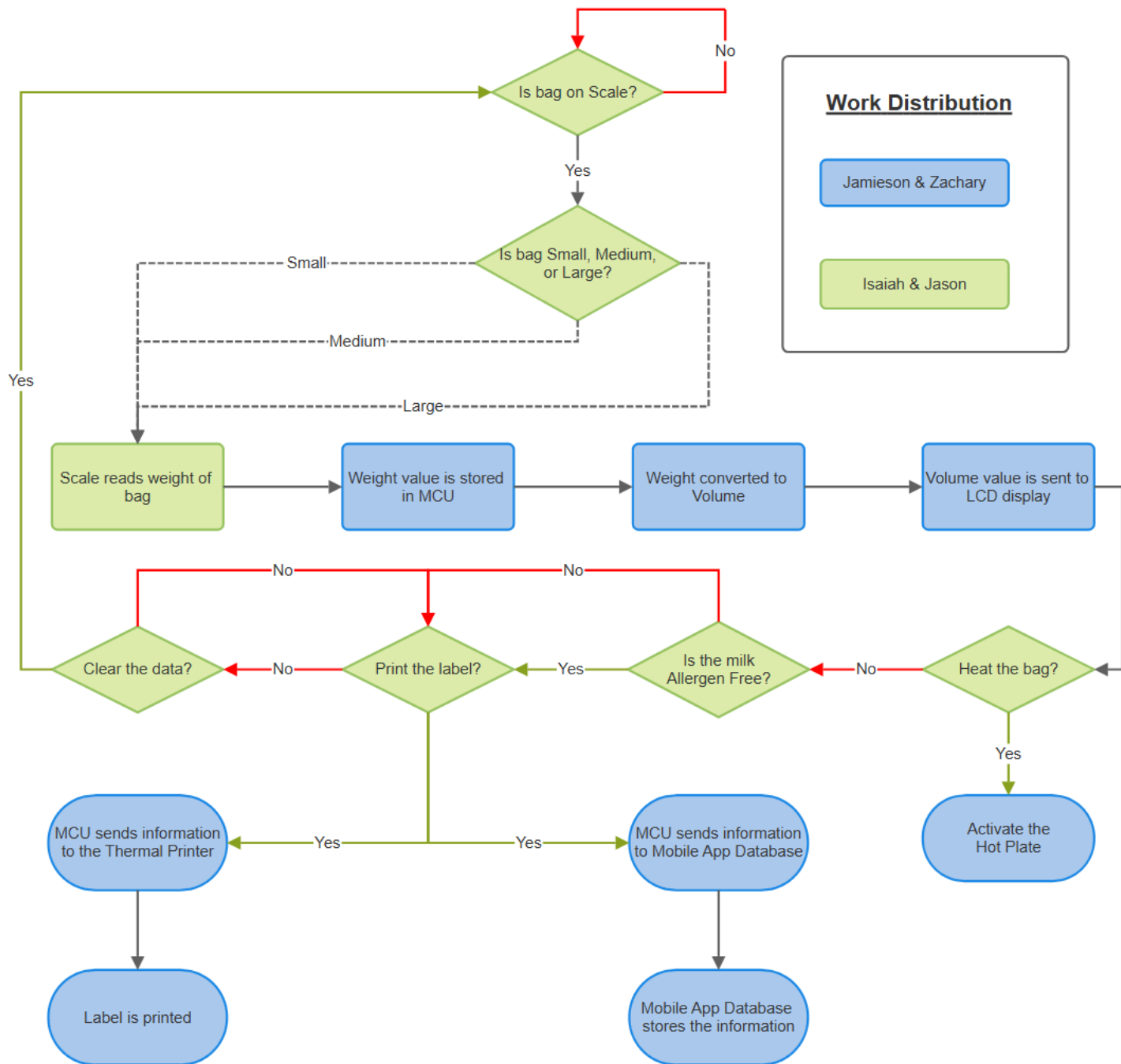


Figure 2.2 - Software Flowchart

2.9: Prototype Illustration

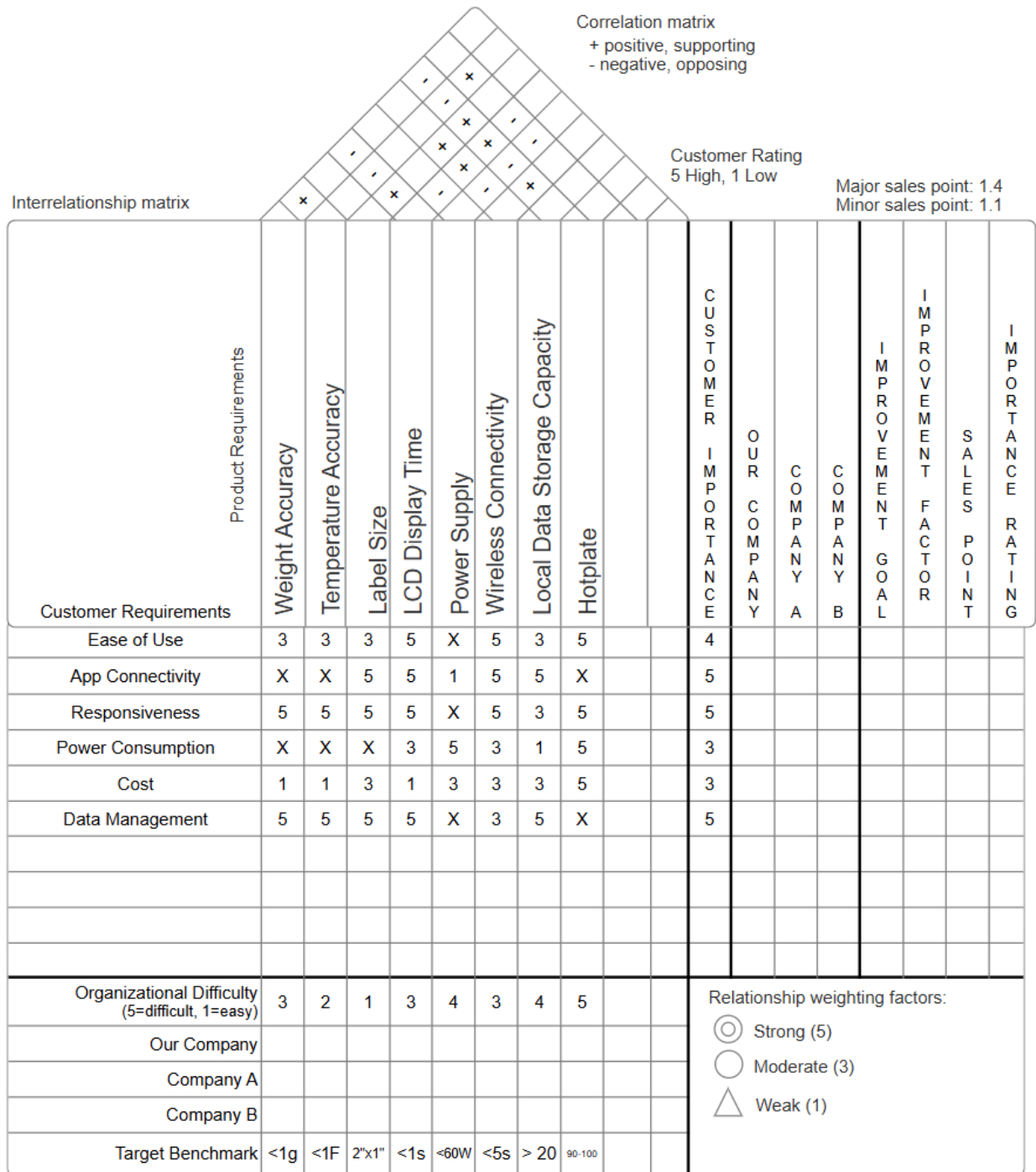


Figure 2.3 - Prototype

Diagram Caption: The illustration above shows a prototype of what we want the device to look like upon its completion. This image was created by AI, and given to us by our sponsors as what they would roughly like the product to look like once it's finished. The Milk bags will sit in a cylindrically encapsulated space, allowing for it not to fall. The bag will sit on top of the scale which will have functionality to both measure and heat its temperature using buttons. Additionally, there will be a button to print a label which will contain the bag's volume in fluid ounces, the current time, and date.

2.10: House of Quality

Table 2.2 - House of Quality



Chapter 3 - Research:

The purpose of this chapter is to investigate the technical aspects for this project's design and development. Each section contains analysis of components relevant to individual parts of our design as well as subsystems that will go into our design to measure, heat, and categorize milk. Contained in the subsections will be information regarding things such as physical architecture to power the system to the software that will enable the users to interface with the data the system intends to store. Sections will provide key technical knowledge to best guide the design of the system to meet all design requirements and constraints. Comparisons given in the chapter serve to provide the most optimal design choices that do not compromise functionality while remaining efficient and effective for all users.

3.1: Hardware Technologies

The first section of this chapter is dedicated to the research of design elements of the physical system. For this, we will be considering many factors for the physical design. Firstly, we will consider how the system will be powered, since that is what the system hinges upon. Afterwards, we will discuss components, such as what MCU will be used, how temperature will be measured, how the bags will be heated, how information will be displayed, among other design choices that must be investigated.

3.1.1: Temperature Measurement Methods

For this section of research for our design, we are considering the method for which we will implement measuring the temperature of the milk and heating elements for our device. This is essential for ensuring that the milk is safe for consumption, so choosing the ideal measurement device is of great importance. The following subsections compare different sensors that we compared to choose the ideal option in our implementation.

3.1.1.1: Contact Temperature Sensors

The first potential method for measuring the temperature of the milk is with a Contact Temperature Sensor. These sensors must physically touch the object to measure its temperature and from there that sensor's output is then converted into a temperature reading. When making contact they absorb heat from the object which causes a change in electrical resistance, voltage, or current. This change is often the heat transfer between the object and sensor reaching thermal equilibrium. Contact Temperature Sensors best measuring happens when contact is made between the sensor and the object. The advantages of using contact sensors is that they have high accuracy when properly calibrated, they're suitable for continuous monitoring, and they are the more cost effective option. The disadvantage of the contact sensors is that they require direct contact which could be invasive and also lead to physical degradation of the sensor.

3.1.1.2: Non-Contact Temperature Sensors

The next method that we considered for the temperature sensor is a Non-Contact Temperature Sensor. These sensors do not physically touch an object to measure its temperature rather they detect infrared radiation emitted by the object's surface. All objects above absolute zero (-273.15°C) emit infrared radiation, so the milk would be able to have its temperature taken by a Non-Contact Temperature Sensor as long as its temperature remains above absolute zero. After the sensor captures this radiation it converts it into an electrical signal which is then sent to a microprocessor or analog circuit which then calculates the temperature. This temperature calculation is often based on the intensity of the infrared radiation. Non-Contact Temperature Sensors are ideal for measuring moving objects or hard to reach areas due to the lack of contact being present. They also have a fast response time with almost instant readings. With the ability to not have to make contact to measure temperature they are also great at measuring very high temperatures. Some disadvantages of these types of sensors is that they can be affected by the ability of a surface to emit infrared radiation, they are also less accurate for reflective surfaces like metals, and are more expensive than contact sensors.

3.1.1.3: Temperature Sensor Method Selection

Table 3.1 - Temperature Detector Method Comparisons

Method	Cost	Implementation	Accuracy
Contact Temperature Sensor	Low (between \$ and \$\$)	Easy to Moderate	High (Loses accuracy over time due to physical exposure)
Non-Contact Temperature Sensor	Moderate (between \$\$ and \$\$\$)	Moderate to Hard	High

The final choice for our method of Temperature Sensing comes down to accuracy and cost. Contact Temperature Sensors provide excellent accuracy overtime, but due to its physical exposure it will degrade. That is a problem we will have to overcome by choosing the right sensor material and to apply a protective coating or encapsulation. With the Contact Temperature Sensor being at a low cost and easy implementation, the physical exposure to the sensor is the only major disadvantage of using this sensor.

3.1.2.4: Contact Temperature Sensors

Now that we have chosen our method for which we will measure the temperature of the milk, we must consider what contact sensor we will implement into our PCB design. There are many options for Contact Temperature Sensors, so for this section we will discuss several that could be used.

3.1.1.5: DS18B20 (Waterproof Digital Temperature Sensor)

The DS18B20 is a common choice for water heating applications due to its availability in a waterproof stainless steel probe variant. It operates within -55°C to 125°C and provides $\pm 0.5^\circ\text{C}$ accuracy, making it practical for monitoring and regulating water temperature.

The 1-Wire digital communication allows multiple sensors to be connected to a single microcontroller pin. Additionally, it supports parasitic power mode, which means it can draw power from the data line, reducing wiring complexity. This sensor is commonly used in home water heaters, brewing systems, and aquarium temperature regulation.

For the simple water heating aspect of our design, the waterproof DS18B20 is the best choice due to its ease of use and digital output.

3.1.1.6: PT100 (RTD - Resistance Temperature Detector)

The PT100 is a highly accurate Resistance Temperature Detector (RTD) that is widely used in industrial applications. It can measure temperatures from -200°C to 850°C with an accuracy of $\pm 0.1^{\circ}\text{C}$ to $\pm 0.5^{\circ}\text{C}$, depending on the quality of the sensor.

Unlike thermistors, RTDs have a nearly linear resistance-to-temperature relationship, making them very precise. However, PT100 sensors require a signal conditioning circuit, such as a Wheatstone bridge and an amplifier, to interface with a microcontroller. They are ideal for precise temperature control in water heating systems, industrial boilers, and laboratory setups.

For a high-precision industrial heating device, the PT100 RTD is the best option, although it requires additional signal conditioning.

3.1.1.7: NTC Thermistor (Negative Temperature Coefficient Sensor - $10\text{K}\Omega$ or $100\text{K}\Omega$)

An NTC thermistor is a simple and cost-effective sensor that changes its resistance inversely with temperature. A commonly used variant is the $10\text{K}\Omega$ or $100\text{K}\Omega$ NTC thermistor, which can provide a temperature range of -40°C to 150°C with moderate accuracy.

Thermistors are highly sensitive but non-linear, meaning you need to use a lookup table or Steinhart-Hart equation for accurate readings. They are often used in thermostats, electric kettles, and water heaters for basic temperature regulation.

For a cheaper solution, an NTC thermistor is sufficient but requires calibration and compensation for non-linearity.

3.1.1.8: Temperature Detector Selection

Table 3.2 - Temperature Detector Comparisons

Temperature Detector	Cost	Integration	Maintenance
DS18B20 (Waterproof Digital Temperature Sensor):	Moderate \$2-\$10	Easy to integrate. Requires a single-wire digital interface and minimal external components	Durable and waterproof (if sealed correctly) minimal calibration needed
PT100(RTD-Resistance Temperature Detector)	High \$10-\$100+ Varies significantly with accuracy and material	Moderate to Complex. Requires an excitation current, specialized amplifier and signal processing	Moderate to High as it is more fragile and needs careful wiring. Needs to be calibrated overtime
NTC Thermistor (Negative Temperature Coefficient Sensor - 10K Ω or 100K Ω)	Low \$0.50-\$5	Easy to Moderate. Requires voltage divider, but is more sensitive to noise	Moderate as it is prone to aging and drift. This also needs to be recalibrated over time

The final temperature detector selection was based on cost efficiency, simple integration, and low maintenance. With the three options displayed above, We decided to choose the DS18B20(Waterproof Digital Temperature Sensor). Although it is moderately priced, the extra cost is made up for because of its low maintenance and easy integration. The PT100(RTD-Resistance Temperature Detector) is the exact opposite, with its high cost, moderate to complex integration, and its fragile durability. The PT100 was put last on the list of options as its high cost and lack of durability is not ideal for our device. The close runner up to the DS18B20 was the NTC thermistor. With its low cost and easy integration it seemed like the ideal option, but the problem arises due to the

sensor being prone to aging. The sensor aging would not be ideal for temperature readings that are needed for safe consumption and storing. So the overall choice is the DS18B20 because it meets a moderate to low cost, integrates easily, and has the lowest maintenance due to it being durable and waterproof.

3.1.2: Weight Measurement Methods

A vital piece of information in our design is the weight of the milk that is being measured. For this system, we want to be able to give parents the most accurate measurements for the amount of food they have, as this will aid in keeping track of when their child will need more milk based on how much they consume, while also allowing our database to give alerts to when their supply is running low. This section will discuss several methods to measure the weight of the milk for our system, as well as implementation.

3.1.2.1: Load Cell (Strain Gauge-Based)

The first potential method of measuring the weight of the milk in our device is through a Load Cell. Essentially, this works by measuring the force caused by the deformation of the element. This deformation generates a voltage difference, which is proportional to the amount of force on the element. What makes this method for measurement ideal is that it has high precision in its measurements, with a large range for weights that it can measure accurately. When it comes to implementation, there are a few drawbacks that must be considered. Most notably, the signal output of a Load Cell is typically small, in the mV range. This means that in our PCB design, we will need to include an amplifier. Furthermore, Load Cells are susceptible to error due to temperature variation; this is because resistance changes with temperature, which could potentially be problematic in our design as we are going to be involving heating in our design. In addition to this, Load Cells being sensitive measuring tools means that their mounting is important to their measurement accuracy, which could mean that assembly of the final design is more troublesome.

3.1.2.2: Piezoelectric Force Sensors

The next method that we considered for the weight measurement aspect of our system is Piezoelectric Force Sensors. Piezoelectric Force Sensors utilize the Piezoelectric Effect, which allows materials to generate a voltage in response to some applied force. The key difference between the Piezoelectric Force Sensor and the Load Cell is that the load cell measures force in terms of difference in resistance caused by an applied force, whereas the Piezoelectric Force Sensor generates a charge when a force is applied, and needs a charge amplifier to actually utilize the measurement, as the charge generated is incredibly small. Piezoelectric Force Sensors are best utilized where the force applied is temporary, which for our design is not ideal, as we will be dealing with objects that will not be changing force applied during measurement. An even deeper disadvantage in our case is the fact that the charge generated by the sensor depletes over time, making them even less suited for our use. Aside from these disadvantages, the Piezoelectric Force Sensor is also typically very expensive compared to other measurement options.

3.1.2.3: Capacitive Force Sensors

The last method for which we considered to measure the weight of milk is through the use of Capacitive Force Sensors. Similarly to Load Cells, Capacitive Force Sensors work by measuring variation in impedance, instead now using variation in capacitive changes instead of resistive changes with the Load Cell. Capacitive Force Sensors are of particular interest because they also provide incredibly precise measurements. However, the precise measurements come with the drawback that they are best used in compact PCB design where the expected force measured is expected to be small. Since Capacitive Force Sensors are very precise, they are also going to be subject to more noise in their measurements. If we were to utilize Capacitive Force Sensors, we would also need to implement a separate IC for capacitive sensing, adding to the cost.

3.1.2.4: Weight Measurement Method Selection

Table 3.3 - Weight Measurement Comparison

Method	Cost	Implementation	Accuracy
Load Cell	Moderate (Anywhere between \$ & \$\$\$)	Moderate (Requires an amplifier IC)	High (Does not lose accuracy over time)

Piezoelectric Force Sensors	High (Between \$\$ & \$\$\$)	Difficult (Requires Charge amplifier)	Moderate (Loses accuracy with time, best for alternating forces)
Capacitive Force Sensors	Low (Between \$ & \$\$)	Moderate (Requires Capacitive Sensing IC)	Moderate (Sensitive detection, best for low force detection)

The final choice for our method weight measurement comes down primarily to accuracy and cost. Load Cells provide excellent accuracy whilst not completely sacrificing in terms of implementation difficulty. Furthermore, while Load Cells can potentially be costly, on average, they are cheaper than Piezoelectric Force Sensors and equal in most cases to Capacitive Force Sensors. A problem that we will have to face in our design with respect to Load Cells is their sensitivity to temperature variations; to combat this, we will likely have to consider insulation for the load cell, depending on how it is impacted by our heating element.

3.1.2.5: Load Cells

As we have chosen the method for which we will measure the weight of the milk in our system, we now must consider the actual load cell which we will implement into our PCB design. There are many options for Load Cells, so for this section we will discuss several that could potentially be used.

3.1.2.6: Phidgets CZL635 (5kg)

The Phidgets CZL635 (5kg) is a load cell that has a maximum capacity of 5 kg. What is beneficial about this is that while all load cells will require amplification and an ADC for accuracy, this manufacturer has an IC that can be purchased to decrease the overall design time that can be directly connected to the Load Cell.

3.1.2.7: SparkFun SEN-14729 (5kg)

Similarly, the SparkFun SEN-14729 (5kg) is a load cell with a maximum capacity of 5 kg. What makes this option potentially more viable is the competitive cost. As we would be using only the load cell and no proprietary IC for amplification.

3.1.2.8: TAL220B (5kg)

The final Load Cell option is the TAL220B (5kg). Again, the maximum capacity is 5kg, which is ideal for our design. This option benefits similarly to the previous SparkFun SEN-14729 (5kg), in that we would be designing the amplifier on our own, thus saving money in the manufacturing process.

3.1.2.9: Load Cell Selection

Table 3.4 - Load Cell Comparison

Product	Cost	Implementation
Phidgets CZL635 (5kg)	\$7 per unit + \$30 - \$95 for proprietary IC	Easy (Can implement amplifier IC into PCB design)
SparkFun SEN-14729 (5kg)	\$13.12 per unit	Moderate (Requires amplifier design and implementation)
TAL220B (5kg)	\$11.95 per unit	Moderate (Requires amplifier design and implementation, which is provided by the seller)

The load cells all function identically and all require an amplifier IC. What this means is that the main selling point for which one we choose is the cost. While the Phidgets CZL635 is the lowest cost per unit, it also necessitates the purchase of a proprietary IC that could potentially cost hundreds of dollars if one malfunctions. To that end, the TAL220B is the cheapest per unit comparatively to the other options. Additionally, the manufacturer offers the schematic design of the amplifier IC necessary for the load cell to operate. For these reasons, the TAL220B is the best option for our load cell selection.

3.1.3: Display Methods

The next consideration for our device is the means to which information about measured material and the device in general will be displayed. We will be comparing LCD and LED displays because they are the most commonly used display technologies. Below is information on both of them including their working principles, advantages, and disadvantages.

3.1.3.1: LCD Display

LCDs (Liquid Crystal Displays) use liquid crystals that are between two layers of glass or plastic. These crystals do not emit light directly but rather manipulate it from a backlight source to create an image. The structure of the display is in a grid of pixels which then consist of subpixels (Red, Green, and Blue) controlled by an electrical current to allow different amounts of light to pass through. The advantages of the LCD displays are their low cost, less power consumption, better in bright environments, do not suffer from burn in problems, and they can last a long time with minimal degradation in display quality. The disadvantages of the LCD display is their slower response time compared to the LED display

3.1.3.2: LED Display

LED (Light Emitting Diodes) displays use light emitting diodes to produce an image. Each pixel is an LED display which consists of three LEDs (red, green, and blue) that light up at different intensities to form the colors needed to display an image. LED displays do not require a separate backlight because they produce their own light. The advantages of the LED is their superior picture quality (due to providing better contrasts), energy efficiency, faster response times, slim design, and longer lifespan compared to LCD displays. The disadvantages of LED displays are their higher costs and risk of burn-ins due to use.

3.1.3.3: Display Method Selection

Table 3.5 - Display Method Comparison

Display Method	Cost	Implementation	Maintenance
LCD Display	Lower Cost	Easy	Moderate (may require backlight replacement or repairs overtime)
LED Display	Higher Cost	Moderate	Lower (LEDs have longer lifespans and require minimal maintenance)

3.1.3.4: LCD Displays

As we have chosen the method for displaying the data, we now have to consider what LCD display we want to implement into our design. For the following subsections, we will compare various LCD display options and determine which is most well suited for our implementation.

3.1.3.5: 16x2 Character LCD (HD44780 - I2C or Parallel Interface)

The 16x2 LCD is a widely used display that can show two lines of 16 characters each. It uses the HD44780 driver, which allows easy interfacing with microcontrollers like MSP430, Arduino, and ESP32. You can connect it using a parallel 4-bit/8-bit interface or an I2C module for simplified wiring.

The refresh rate is very fast (milliseconds range), ensuring an update time well below 1 second. This display is great for showing temperature readings, system status, and basic control options.

The HD44780 is a simple LCD screen that is easy to read and implement. This choice, although not the best looking, is practical and can work with our design to display the needed information.

3.1.3.6: 128x64 Monochrome LCD (ST7920 - SPI/I2C/Parallel Interface)

The 128x64 graphical LCD provides more flexibility than a character LCD, allowing you to display numbers, text, and simple graphics such as temperature graphs or heating status indicators. The ST7920 driver supports SPI, I2C, and parallel communication, making it easy to interface with microcontrollers.

The update speed is typically under 100ms per full screen refresh, ensuring real-time data display. This type of screen is practical if you need more detailed information or a better user experience compared to a simple character display.

3.1.3.7: 2.8-inch TFT LCD (ILI9341 - SPI Interface, Full Color)

The 2.8-inch TFT LCD with the ILI9341 driver is a great choice. It offers a 320x240 pixel resolution with full-color support, allowing you to create a graphical user interface (GUI) with buttons, charts, and real-time temperature readings.

This display communicates via SPI, ensuring fast updates typically 50-200ms per refresh. This is all dependent on the complexity of the graphics that are implemented. The ILI9341 is ideal for a touchscreen-based control panel. This would allow for a more modern feel to our device.

3.1.3.8: LCD Display Selection

Table 3.6 - LCD Display Comparison

LCD Display	Cost	Integration	Maintenance
16x2 Character LCD (HD44780 - I2C or Parallel Interface)	Low \$3-\$10	Easy integration with simple character display and uses I2C	Low as it is durable with no complex graphics. Has minimal issues
128x64 Monochrome LCD (ST7920 -	Moderate \$6-\$15	Moderate integration and requires SPI/I2C/Parallel interface. Also needs a graphics library to	Low as it is reliable, but backlight or contrast may degrade overtime

SPI/I2C/Parallel Interface)		support simple graphics and texts	
2.8 inch TFT LCD (ILI9341 - SPI Interface, Full Color)	High \$10-\$30	Complex integration, requiring graphics library, more memory and processing power needed	Moderate as it is more fragile. Color display may degrade and the touchscreen may wear out

The final LCD Screen selection was based on cost efficiency, simple integration, and low maintenance. While the 2.8 inch TFT LCD (ILI9341 - SPI Interface, Full Color) is a more modern approach to an LCD screen its higher cost, lower durability, and complex integration make it a less optimal choice when looking for an LCD screen that can achieve the three specifications set above. The 128x64 Monochrome LCD (ST7920 - SPI/I2C/Parallel Interface) is a good middle ground when it comes to cost and integration. Its maintenance is also low which makes it a suitable fit for our device. The only problem is the backlight or the contrast could degrade overtime due to use. With the three specifications we set, the 16x2 Character LCD (HD44780 - I2C or Parallel Interface) is the most optimal LCD for our device. With its low cost, simple integration for character displays using I2C, and its high durability make it optimal for our device.

3.1.4: Label Printing Methods

In this section, we will be considering the type of printer that we will use for printing the labels for the milk bags. When considering the printer, we must consider how it will be integrated in our final PCB design, cost comparatively to other options, and power consumption.

3.1.4.1: Thermal Printing

Thermal printing uses heat to create images or text on special thermal paper. It is widely used in point of sale systems such as grocery receipts, barcode printing, and shipping labels. Direct Thermal Printing uses the special thermal paper to heat specific parts of the paper through a heated printhead to create an image or text. This process requires no ink, ribbon, or toner to produce text or images. Thermal printers have fast printing speeds, high durability, low

maintenance due to fewer moving parts, and are ideal for barcodes, receipts, and industrial labels. The downside to thermal printers is that the labels created by them fade over time especially when exposed to heat and excess light. They are also limited to monochrome printing and special thermal paper is required for this type of printer.

3.1.4.2: Laser Printing

Laser printers use a laser beam and toner to create high quality prints at an efficient pace. They are commonly used in offices for documents, shipping labels and marketing materials. Laser Printers work by charging a rolling drum with static electricity, which is then shot at by a laser beam in specific places to remove the static charge. After some of the static has been removed in specific areas the roller is then covered in toner (a fine powder). The drum then rolls the toner onto the paper, which is then put into a fuser unit that heats and presses the toner onto the paper making the print permanent. The advantages of the laser printer is the high speed printing which is helpful for bulkier jobs, sharper professional quality text, no smudging due to fusing the toner to the paper, and toner lasts the longest on paper. While those sound ideal for a printer the advantages are quickly outweighed by the disadvantages. Those being that a laser printer is the most costly printer, it is also limited to standard paper, and laser printers are bulkier and heavier than other printers.

3.1.4.3: Inkjet Printing

Inkjet Printers spray tiny ink droplets onto paper to create text or images. They are widely used for home and office printing, as well as for photo and label printing. The printer is pretty basic when it comes to the process of printing. Inkjet Printers have a nozzle that releases ink droplets in a controlled pattern. The ink then quickly dries forming an image or text. The advantages of the Inkjet Printer is that it is affordable and widely available, good for high quality color printing and photos, can print on a variety of surfaces, and the printer is compact which saves space. The disadvantages of an Inkjet Printer is the cost of ink cartridges, its slow printing speed compared to a laser printer, the ink can smudge if it is not dried properly, and the printheads can clog if not regularly maintained.

3.1.4.4: Label Printer Selection

Table 3.7 - Label Printer Comparisons

Label Printer	Cost	Implementation	Maintenance
Thermal printer	Low \$50-\$100	Easy	Low (fewest moving parts, no toner or ink)
Laser Printer	High \$100-\$500	Moderate	Moderate (toner needs to be replaced often. Regular cleaning needs to take place)
Inkjet Printer	Moderate \$50-\$300	Easy	Moderate (regular cleaning of the printhead to avoid clogging)

After examining our options, we chose to go with the Thermal Printer. The laser printer and the inkjet printer are great choices, but with their costs going into the hundreds we want a cheaper alternative. Not only is the cost a deterrent, but the maintenance is something we do not want to pass on to the customer as one more thing to do. With the Thermal Printer there is low maintenance at an affordable cost. Not only that, but with that it has the easy implementation out of the three printers. The only obstacle we will have to overcome with the Thermal Printer is finding one that can fit in our device's size dimensions.

3.1.4.5: Thermal Printers

As we have chosen to use thermal printers for our project, the following subsections will focus on comparing various thermal printers and choosing the one that can best be implemented into our project.

3.1.4.6: MUNBYN IMP001 58mm Thermal Printer (Serial/UART & USB)

The MUNBYN IMP001 is a compact 58mm (2.28 inches) thermal printer, making it well-suited for printing small labels or receipts. It supports serial (UART) and USB communication, allowing easy interfacing with microcontrollers like MSP430, Arduino, and Raspberry Pi.

The printer offers fast printing speeds (90mm/s) and high resolution for clear text and simple graphics. It is ideal for printing temperature logs, timestamps, and heater status updates. Additionally, it operates on DC 5V-9V, making it adaptable to embedded systems.

For direct microcontroller integration (TTL Serial/UART), the Adafruit Mini Thermal Printer is the best option.

3.1.4.7: PRIMUZ Micro Thermal Printer

The PRIMUZ Micro Thermal Printer is a small and efficient thermal printer designed for embedded applications. It communicates using a serial port and usb port, making it easy to integrate with a microcontroller. This printer supports thermal label paper sizes of 58 millimeters.

It prints efficiently and supports basic text and graphics. Since it operates at 12V-24V, it is well-suited for low-power applications like a water heater system where periodic data printing is required.

For high-speed printing and reliable data logs, the PRIMUZ Micro Thermal Printer is a solid choice.

3.1.4.8: ASPIRON 58mm Embedded Thermal Printer Module (UART & TTL Interface)

The ASPIRON 58mm Embedded Thermal Printer Module is designed for direct PCB integration. It supports TTL UART and RS232 serial communication, making it easy to connect with microcontrollers like MSP430, Arduino, or ESP32. This printer is commonly used in embedded systems, ATMs, kiosks, and industrial devices where real-time data printing is needed.

It supports 2-inch (58mm) thermal paper and can print at a speed of 50-80mm/s, ensuring quick updates on temperature logs, timestamps, and system alerts. The module operates on DC 5V-9V, making it compatible with standard power supplies in PCB-based designs.

The Aspiron also has mounting holes for secure attachment to a PCB or enclosure.

3.1.4.9: Thermal Printer Selection

Table 3.8 - Thermal Printer Comparisons

Thermal Printer	Cost	Integration	Maintenance
MUNBYN IMP001 58mm Thermal Printer (Serial/UART & USB)	Moderate \$40-\$80	Moderate as it uses UART or USB connection	Moderate maintenance as needed depending on use of paper, also dependent on jams.
PRIMUZ Micro Thermal Printer	Moderate \$45-\$65	Moderate, can be connected through USB or through serial port	Ventilation may be a concern, as overheating could possibly occur.
ASPIRON 58mm Embedded Thermal Printer Module (UART & TTL Interface)	Moderate \$45-\$70	Moderate; connected through USB or through serial port	Similar ventilation problems are possible, also may need additional protection from dust buildup.

Ultimately, what is the primary concern for our printer is the cost. All of the printers perform nearly the same, with similar communication methods. What differentiates them all is the ease of integration with respect to their cost. The best choice for our integration is going to be the Aspiron Embedded Thermal Printer, as it allows for serial communication and USB connection if desired. At its cost, it is extremely competitive with our other options, and is ideal for our design.

3.1.5: Heating Methods

This section will dive into the aspect of heating the milk. The safest and recommended way of heating breast milk is heating water to desired temperature and placing the milk in the water until heated to the desired temperature. With this consideration in mind, we must find a method to heat the water that is safest for the user and most effective. The following sections are the research into methods of heating water and their key differences.

3.1.5.1: Induction Heating

Induction Heating is the method of heating utilizing electromagnetic fields to heat elements. Using induction, an alternating current is passed through some coils of wires, which generates a magnetic field around the coil of wires. The magnetic field generated by the bundle of wires generates a current in another conductive material. This current experiences a resistance in the material, which generates heat. This method for heating is incredibly efficient, as it is not heating the heating element itself, only the conductive material, and environment immediately surrounding the conductive material. Since induction is an efficient method of heating, consequently it is typically energy efficient, as less energy is wasted in the process of heating the conductive material. However, a major drawback of using induction is the physical space required in order to accommodate the induction coils. Furthermore, powering such a circuit would be complicated, as we need to have an alternating high frequency current to sufficiently generate enough current in the conductive material.

3.1.5.2: Conduction Heating

The next method viable in our project that we can implement for heating is conduction heating. Conduction heating uses direct contact to transfer heat from an element to a subject. Conduction heating is effective where localized heating is best, and solid materials are the subject of heating. In our case, conduction heating is an incredibly simple and effective potential solution to heating the milk. Since we will likely utilize a cup of sorts with water inside, having it in contact with a conductive heating element would be an effective means for heating the water. While Conduction is viable, it is not necessarily without flaws. Conduction as a heating method is inefficient, as it loses heat to its environment, which in our use case, is a significant drawback where we are trying to save energy and maintain the other components in our PCB.

3.1.5.3: Convection Heating

Convection heating is the final method to discuss, which uses the motion of fluids to heat a specific subject. This motion of fluids is caused by hotter elements rising in an area and cooler elements sinking in an area, which ideally distributes the heat evenly through a subject as the air moves constantly. By forcing further movement of the fluids through a fan or some other means, the heating efficiency can be increased without needing significantly more energy. The drawbacks of convection come with the nature of our project. Convection requires significant space in order to sufficiently circulate the fluid that is being heated and moved. Additionally, convection takes more time to heat the subject because it depends on the motion of the fluid. Since our project does utilize a fluid medium in order to heat the milk, this method will inherently be used. While it will be used by virtue of the fact that we will be using a medium to heat the milk, it is important to consider how other methods will interact with this method, while also potentially acknowledging ways to improve the efficiency of our system.

3.1.5.4: Heating Method Selection

Table 3.9 - Heating Method Comparisons

Method	Efficiency	Implementation
Conduction	High (Element makes direct contact with subject, and direct heat transfer is better for small area heating)	Easy as the heating element is directly in contact with the water which would transfer the heat efficiently
Convection	Moderate (heats large spaces well, requires consistent fluid motion)	Moderate (implementation of forced convection would require moving parts to move fluid)
Induction	High (requires significant space for coils generating magnetic	Difficult (requires complex power management due to AC

	current, energy efficient)	requirement for operation)
--	----------------------------	----------------------------

We chose conduction for our heating method due to its efficiency in heating. If our conduction is efficient enough then convection will also be present as we are heating a stainless steel cup filled with water. Although the operational cost of using conduction over the other two heating methods is higher, its efficiency and ability to heat an element directly is the primary reason for it being chosen.

3.1.5.5: Heating Elements

The following subsections will compare the various options for heating elements that we want to incorporate into our design. We also want to ensure that these heating elements are able to heat the milk in our system with efficiency and reliability.

3.1.5.6: Electric Heating Pad - 14cm x 5cm by Adafruit

The Electric Heating pad is a small rectangular resistor that heats up when a voltage is applied to it. This heating pad is flexible, thin, and is able to be mounted to a surface using an adhesive. The use of this pad is mainly for low-power heating applications which is ideal for our device. Control over this device can be done through a MOSFET or relay, which is connected to a microcontroller.

At only 14 cm x 5cm it is a relatively small heating pad which could be mounted on the stainless steel cup and used for conduction heating. The power required to heat up this heating pad is 5W. Which is a lower wattage for a heating application. The heating pad itself is also going to have to be paired with a temperature sensor to regulate how much voltage is being applied to it.

This Heating Pad is mainly used for maintaining mild warmth over a surface or heating small amounts of water in insulated conditions.

3.1.5.7: MECCANIXITY Silicone Heating Pad

The Silicone Heating Pad is a flexible pad that is used to distribute heat evenly over its surface. For its application it is mainly attached directly to a metal or plastic container using an adhesive backing. Much like the previous Electric Heating Pad it requires a MOSFET or a relay to control power input.

This device will also be used for conduction heating. At 10 watts of power, which is moderate out of the three heating applications, the Silicon Heating Pad is a break through the hard and weak points of traditional heaters. Due to its flexibility it is a more convenient way of getting a heating application closer to the heated object.

The Silicone Heating Pad is best used for small scale applications such as, for lab equipment or preheating surfaces.

3.1.5.8: MakerTechStore Cartridge Heater

Cartridge Heaters are metal cylindrical heaters that are designed to be inserted into a metal block or a reservoir of water. In our design the cartridge heater would be inserted into the reservoir of water placed into the stainless steel cup. Their application is based on high temperature heating applications due to their ability to heat up rapidly. Due to their ability to heat up quickly, a heat sink or proper mounting would be necessary to prevent overheating. Much like the other two applications mentioned above, a MOSFET or relay is required to handle the higher current draw.

The Cartridge heater would display conduction like we intend, but would also cause convection in the water. This is ideal as it would create an equally heated vat of water that is warm enough to heat up the milk. Its temperature control is best handled with a thermocouple or a high temperature thermistor. These would allow for the Cartridge heater to avoid overheating and inefficiency.

Cartridge Heaters are best used for direct water heating in a metal enclosure or heating metal surfaces that then transfer the heat to water.

3.1.5.9: Heating Element Selection

Table 3.10 - Heating Pad Comparisons

Heating Pad	Cost	Integration	Maintenance
Electric Heating Pad 14cm x 5cm by Adafruit	Low \$6	Easy to integrate as it is thin and flexible and can be attached to a surface by an adhesive. Requires MOSFET or Relay for control	Low maintenance as there are no moving parts. The adhesive will weaken over time
Silicone Heating Pad 15cm x 5cm by Meccanixity	Medium \$10	Easy to integrate as it is thin and flexible and can be attached to a surface through an adhesive. Requires MOSFET or Relay for control	Low maintenance as there are no moving parts. Very durable choice. Adhesive may degrade overtime
Cartridge Heater by MakerTechStore	Low \$4	Moderate to integrate as it requires a hole or slot to be inserted into a heating block or reservoir of water. Requires MOSFET or Relay for control	Medium maintenance as it can accumulate limescale. Requires regular cleaning.

The Heating Element that we want to implement into our design is the Electric Heating Pad by Adafruit. Although it does not have a silicon protective layer, we are choosing to look past that for a cheaper option that can do without the protective layering. The thickness of the silicon layer is also another constraint we would have to work around when it comes to optimizing our device. We also chose the Electric Heating Pad because there is no maintenance when it comes to cleaning the heating application. With the Cartridge Heater the user will have to clean the surface for maintenance. While it is the best application to heat a vat or water we do not want our heating application to be a burden on the user. Electric Heating Pad is sufficient in what we want to incorporate into our design due to its low cost, easy integration, and low maintenance. We will have to look

into how to prevent the degradation of the adhesive over time due to heat and use.

3.1.6: MCU

This section goes into considerations and requirements necessary for Microcontroller Unit (MCU) selection. The MCU is the central processing unit for embedded systems, which will be responsible for processing data and interfacing with sensors, communicating with outside systems, as well as any other potential processing needs. There are plentiful options to choose from in terms of MCUs, so comparison of options will give a better idea of which MCUs strengths best suit the needs of our design.

3.1.6.1: MCU vs FPGA

Something to be considered when designing our system is choosing whether to utilize an MCU for our design or an FPGA (Field-Programmable Gate Array). FPGAs offer significantly greater processing power and the potential for parallel processing. However, using an FPGA comes with certain disadvantages for designing our system. FPGAs are highly complex and often difficult to program. Furthermore, FPGAs require a large amount of power to function.

With respect to our design, an MCU is likely to be the best option. MCUs are much cheaper than FPGAs, as they are power efficient and cost effective as they have fixed architecture. MCUs are also much easier to program, as you can use a high level language such as C. MCUs are ideal in situations where there is simple sensor interfacing, which is the case of our design. As the primary purpose of the MCU in the case of this design is to interface with the peripherals and not high-speed processing, MCU will be our choice.

3.1.6.2: MCU Selection

With the previous information, there are several factors to consider when choosing an MCU. The project entails several design specifications; firstly that the device itself can interface with a mobile app and database with ease.

Additionally, the device must also be able to interface with our sensors that we will be integrating.

There are several microcontrollers that fit the basic requirements for our design, including bluetooth capability and the ability to interface with multiple sensors. Given the wide variety of choices, this allows us to choose the MCU that will best fit our needs and allow us flexibility in the future as the project evolves and develops.

3.1.6.3: ESP32

The ESP32 is an option for our project that stands out among other options for its cost effectiveness and capabilities. In terms of features, the ESP32 is incredibly feature dense, sporting Wi-Fi and Bluetooth capability, allowing for a wide range of wireless applications. Additionally, it houses a 2-core processor that can run at up to 240 MHz, potentially allowing for demanding applications where real-time data management and multitasking are of importance. The ESP32 can also have SRAM up to 520 KB of SRAM with support for I2C and SPI protocols.

Most important of the features is pins allowing for ADC and PWM. Seeing as we are looking for an MCU that will allow for multiple sensors to be integrated, the ESP32 has great potential. However, there are several drawbacks to the ESP32. Where the ESP32 lacks comparatively is in terms of power consumption; the ESP32 consumes more power comparatively due to the WiFi and Bluetooth capabilities. Furthermore, the accuracy of the ADC in the ESP32 is comparably worse to the other MCUs. Despite these drawbacks, the ESP32 is still potentially an excellent choice in circumstances where wireless communication and real-time processing are primary considerations.

3.1.6.4: STM32

The STM32 is a family of microcontrollers from STMicroelectronics. They offer high performance MCUs with low power consumption, while also supporting a wide variety of peripheral sensors. There are various models of the STM32 that support specific integrated circuit requirements, such as low-power consumption, or chipsets with expanded memory. The STM32 family

outperforms other MCUS in real-time processing, accurate ADC results, and power consumption.

One major drawback of the STM32 family is that they lack Wifi/Bluetooth capability in the chip. However, it outperforms most MCUs in this context when it comes to accuracy. Furthermore, it can support significant peripherals, such as ADC channels. The chip is most competitive in scenarios in which precision in measurements is of great importance, such as ours.

Another drawback with the STM32 comes with the difficulty of programming. Compared to something like the ESP32, the STM32 requires specific software tools, namely STM32CubeIDE. Programming with the STM32 is much more industrial focused, and thus has less community and availability in terms of educational information to access. While the barrier to entry is higher due to difficulty to program, the device's performance largely makes up for this drawback. Ultimately, the STM32 family of microcontrollers is an excellent choice for a microcontroller in use cases where peripheral GPIO support and accuracy are high importance design factors, but lacks a key feature in WiFi and Bluetooth support without external devices.

3.1.6.5: Nordic nRF52840

The Nordic nRF52840 is another microcontroller option that we considered. It is designed with Bluetooth Low Energy (BLE) as a main priority, making it an ideal choice in networking projects with low power consumption. While it does offer BLE support, it does not support bluetooth classic.

While the MCU does not possess WiFi capability without external devices, it does utilise BLE 5.3 which is a longer range Bluetooth that is also a low-power variation of Bluetooth. Further in the vein of power consumption, the microcontroller has the ability to run on ultra-low power, and is a superior choice compared with other options for applications in which the device is going to be powered with a battery.

While the Nordic nRF52840 is a great choice for applications that may be constantly powered, requiring low power utilization, or for devices that are going to be portable, there are several downsides. As previously stated, it does not natively possess WiFi capabilities, only BLE. Furthermore, it has lower processing power than the previously mentioned microcontrollers. Comparatively, it is more expensive than the ESP32, but less expensive than

certain models of the STM32. All considered, the Nordic nRF52840 is a viable option for applications in which the device is going to be consistently powered, and where long range local connectivity is important.

3.1.6.6: MCU Comparisons & Choice

Table 3.11 - MCU Comparisons

Name	WiFi	Bluetooth	RAM	Clock Speed	Cost
ESP32	Yes	Yes	520 KB	240 MHz	Low (\$ per unit)
STM32	No	No	512 KB - 1MB	84-216 MHz	High (\$\$ per unit)
nRF52840	No	Yes	1 MB flash, 256 KB RAM	64 MHz	Medium (\$-\$\$ per unit)

Based on the table above, we will choose the ESP32. For our project, the ESP32 series of microcontrollers is most feature complete; it enables both bluetooth support as well as WiFi support, which is a main concern for our MCU of choice. While the primary concerns of ADC reliability are valid for our project, sacrificing Bluetooth and WiFi capability is not ideal, as adding separate modules for those could be potentially costly in the final design.

3.1.7: Power Supply

For this section, we will be delving into the details of powering our system. Our system will feature a variety of peripheral devices and measurement tools that we will need to power. The following section will discuss the methods and specific technological methods we considered for the design of our system.

3.1.7.1: Lithium Ion Battery Pack

The Lithium Ion Battery Pack is one of the most commonly used power sources for small electronic devices. They have various forms, such as prismatic,

cylindrical, and pouch cells and typically a single lithium ion cell operates around 3.7 Volts. This voltage can be changed through series or parallel configuration to reduce or increase voltage for the demand needed. Their high energy density allows them to store a significant amount of power relative to their size and weight, making them perfect for a portable electronic device.

However, Lithium Ion Batteries require careful management to ensure safe and efficient operation. A Battery Management System (BMS) would need to be implemented to prevent overcharging, deep discharge, and overheating. The BMS would prevent damage done to the lithium ion battery and prevent safety risks like thermal runaway. When using the Lithium Battery pack the charging circuits must also match the specific requirements of the battery pack to prevent damaging the battery cells. Even with these complications, Lithium Ion Batteries are favored for their rechargeability, long life cycle, and their relatively low self discharge rate.

The application for the Lithium Ion Battery Packs mainly include handheld devices and remote sensors.

3.1.7.2: Grid Power

Using Grid Power is a simple and effective way to power an electrical device as long as there is a power outlet available. An adapter must be implemented so that a standard input AC voltage of 110 or 220 can be converted into a lower DC voltage that is suitable for the device, such as five, nine, twelve, or twenty-four volts. The range of power can vary from a few watts to our intended 60 watt maximum.

The main advantage of using Grid Power is its reliability and convenience. Unlike the previously mentioned Lithium Ion Batteries, there is no need for recharging or replacement. When it comes to using Grid Power there is no uninterrupted power supply as long as the device is plugged into a power outlet. The plugs that are in the wall have adapters that come with a built-in voltage regulator to ensure a consistent output voltage, which would prevent fluctuations that could damage the electronics in the device. While Grid Power is reliable and convenient, the dependency on a wall outlet makes our device unsuitable for being a portable device.

Additionally, when using Grid Power we need to select an adapter that matches the voltage and current requirements of the device to prevent damage. Some

high-quality adapters include protections against overvoltage, overcurrent, and short circuits, further ensuring the safety of connected electronics.

3.1.7.3: Small Solar Panel with Battery Storage

For a greener and offgrid option, a small solar panel system with a battery storage could be implemented. The solar panel converts sunlight into electrical energy, which then could directly power the device or store in the battery for later use. The power output of the solar panel depends on its size and efficiency, but most of the small panels have a power output ranging from 10 watts to 60 watts, which is ideal for our device.

The biggest constraint when using solar panels is the dependency on sunlight to power the device, as well as a battery storage system. To ensure overheating, overcharging, and deep discharge do not occur a charge controller is necessary to regulate these functions. The stored energy from the battery will allow a constant and reliable power source that can be accessed during cloudy days or nighttime. While it is a greener option from the two previously mentioned options, the efficiency of the machine depends on the solar panel orientation, weather conditions, and the battery capacity.

Solar-powered systems are commonly used for off grid and remote applications, such as outdoor sensors and renewable energy projects. Compared to the other two previously mentioned options, the solar panel will be more costly to implement, but they offer a long-term benefit through lower operational costs and sustainability. With advancements in solar panel technology and battery storage continuing to improve in efficiency, this could make solar energy increasingly viable for powering our device.

3.1.7.4: Power Supply Selection

Table 3.12 - Power Supply Comparisons

Power Supply	Cost	Integration	Maintenance
Grid Power	\$5-\$50, depending on voltage and current requirements	Requires a power jack and voltage regulation circuit	Low, as it relies on existing power infrastructure

Battery Storage	\$20-200, depending on capacity, chemistry, lifespan and energy needs	Requires charging circuit, battery management system and protective enclosures	Medium, because of charging cycles and the battery degrading over time
Solar with Battery Storage	\$50-\$500, depending on solar panels, charge controllers, a battery storage system, and power output	Requires solar panels, charge controllers, and power management circuitry	High, because of environmental exposure the solar panels and batteries degrade overtime

The final power supply selection was based on cost efficiency, simple integration, and low maintenance. While the battery pack is a phenomenal idea we wanted to incorporate into our design initially, we chose not to due to the included battery management system and possible degradation of our devices battery over time. The solar panel with battery storage was also a great idea and was the greener option when it came to all three of our choices, but due its dependency on light and the problems with battery degradation we chose to go an alternative route. We wanted something that could sufficiently power our device with efficiency, be reliable to constant use, and have low maintenance. We opted to go with Grid Power because it hit all three of our categories listed above. The Grid Power, although it makes the device not portable, is the most reliable and low maintenance option while also being the most efficient. In the long run it is also the lowest cost to implement being that it relies on an existing power infrastructure and all it requires is a power jack and voltage regulation circuit.

3.1.7.5: Grid Power

For our power supply we have chosen the method of Grid Power. We must now consider how we want to connect our device to the grid. There are many options, so for this section we will discuss several options that could potentially be used.

3.1.7.6: MEAN WELL LRS-100-24 (SMPS Adapter):

This open frame power supply provides 24V at 4.5 amps (108 Watts). This is ideal for a medium power application such as our device. With a built-in Universal AC input meaning it can accept 110V (60 Hz) or 220V (50 Hz) making it suitable for grid power world wide. The LRS-100-24 also has a high efficiency at 88% which is great for reduced heat generation. Not only does it have reduced heat generation, but being that it is in a ventilated metal case it can allow for passive cooling without a fan. The dimensions of the LRS-100-24 are also a great feature as it is compact which is easy for installation purposes. It also comes with some protections such as short circuit, overload, and overvoltage to ensure the components in our device do not get harmed.

3.1.7.7: Alitove Power Supply Adapter(AC/DC Adapter):

The Alitove AC/DC Power Supply provides a stable 24V DC output at 10A (240 Watts). Also with a built-in Universal AC input it too can accept 110V (60 Hz) or 220V (50 Hz). This adapter can power the necessary requirements for our project being that the power output is 240 Watts. With its compact design, it is easy to integrate into the design process as there will not be any bulky wires to interfere with space. This AC/DC Adapter comes with some built in safety protections, including overvoltage, overcurrent, and short circuit protection which is a preventative measure to ensure no damage to our devices circuit. Another great feature about this adapter is there is no internal fan making it noise free. Now the heating aspect is taken care of with a passive cooling feature and an efficiency of 85% which will also reduce heat generation.

3.1.7.8: Bel Signal Transformer with Rectifier Circuit:

The Bell Signal Transformer is a step down transformer designed to convert 120V AC input into a lower AC Voltage. This lower AC voltage is then put through a Rectifier circuit. This Rectifier Circuit will take the input AC voltage and then only allow the positive half of the AC sinusoidal wave to pass through. Once that is completed, capacitors will be put into place to flatten out the sine

wave to create a constant DC voltage that we can use in the circuit. If we go with this power supply we will have to build and connect everything by hand, but that allows us to create our power supply to the specific requirements needed by our PCB. The downside of doing this power supply is that this will be the most tedious as we will have to build and connect everything by hand. Another problem we will have to face if we try this power supply is the cost of all of the individual

3.1.7.9: Grid Power Selection

Table 3.13 - Grid Power Comparisons

Product	Cost	Implementation	Maintenance
Bel Signal Transformer with Rectifier Circuit	Moderate (\$30-\$50)	Hard (We will have to self build everything to our design specification)	High (with a lot of individual parts, there is a possibility of having to fix individual parts if problems arise)
LRS-100-24 (SMPS Adapter)	Low (\$17)	Easy to Moderate (Comes pre-built)	Moderate (cleaning may need to occur to prevent dust build up)
Alitove Power Supply(AC/DC Adapter)	Low (\$23)	Easy to Moderate (Comes pre-built)	Moderate (cleaning may need to occur to prevent dust build up)

We chose to go with the Bel Signal Transformer with Rectifier Circuit because it will allow us to create a power source that can fulfill our device's power consumption. Although implementation will be hard, as well as maintenance and a higher cost, we want to go this route so that we have a more hands on approach with our power supply creation.

3.1.8: Communication Protocol

For the final section of our hardware research, we will be going over the various communication protocols and determining which one will best be implemented for our project. Communication protocols are a set of rules that allow for multiple devices in a system to communicate with each other and enable data to be transferred efficiently.

3.1.8.1: Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a synchronous serial communication protocol that can be used for high-speed data transfer between a microcontroller (Master) and peripherals (Slaves), such as sensors, displays, and memory chips. SPI follows a Full-Duplex configuration, meaning that data can be simultaneously transmitted in both directions between the master and slaves using MOSI/MISO communication. This allows for the devices to function continuously without interruption.

3.1.8.2: Inter-Integrated Circuit (I2C)

The Inter-Integrated Circuit (I2C) is a synchronous serial communication protocol that can be used for short-distance data transfer between multiple masters and multiple slaves using an I2C bus structure. Such a structure is especially important when there are multiple microcontrollers in a device system. I2C follows a Half-Duplex configuration, meaning data can be transmitted in both directions, but not at the same time.

3.1.8.3: Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) is an asynchronous serial communication protocol that transmits data bit by bit, without using a shared clock signal to synchronise data between the transmitter and receiver. What makes UART different from other communication protocols, is that it transmits data using two wires connected between the transmitter and receiver. UART has multiple distinct ways of configuration, such as Simplex, Half-Duplex, and Full-Duplex.

3.1.8.4: Communication Protocol Selection

Table 3.14 - Communication Protocol Comparisons

Protocol	Communication	Configuration	Data Transfer Rate	Amount of Devices	Complexity
SPI	Synchronous	Full-Duplex	>10 Mbps	Multiple Slaves	Moderate
I2C	Synchronous	Half-Duplex	100 Kbps	Multiple Masters & Slaves	High
UART	Asynchronous	Simplex, Half-Duplex, Full-Duplex	1 Mbps	Single Master to Slave	Low

For our project, we decided that the Serial Peripheral Interface (SPI) communication protocol was the best fit for our system. Compared to the other protocols, it has a significantly higher data transfer rate, with a moderate amount of complexity depending on the scale of the system. The SPI protocol is also best for our system as we plan on having only a single microcontroller (Master) control multiple other devices (Slaves).

3.2: Software Technologies

A crucial factor in our design is ensuring accurate and timely data flow from the milk-weighing device to the caregivers who rely on this information. Beyond simply measuring the milk's weight, the system must seamlessly record, store, and process these readings so parents can know precisely when they need to replenish their supply. To achieve this, our software must not only retrieve data from the hardware but also maintain a robust and user-friendly interface for real-time updates and historical tracking. In this section, we explore the various software technologies—ranging from front-end development tools to backend frameworks and database options—that collectively form the backbone of our solution. By carefully selecting, comparing, and implementing these technologies, we ensure our system is responsive, accurate, and capable of delivering notifications whenever inventory falls below user-defined thresholds.

3.2.1: Mobile Application Languages

In order for our milk-weighing system to provide real-time updates and easily accessible information, a reliable mobile application is essential. This app must present users with up-to-date measurements, track historical usage, and seamlessly interface with our backend for database alerts on milk supply and expiration. To accomplish these goals, we need to select a development language capable of delivering responsive performance, rich user interfaces, and simple integration with the hardware's data output. In the following section, we will examine various mobile application languages—each with its own strengths, ecosystems, and potential trade-offs—as they relate to building a robust, user-friendly platform for parents who rely on continuous insight into their child's feeding schedule. By carefully evaluating these options, we can ensure the final app design is well-suited for both day-to-day convenience and long-term system maintainability.

3.2.1.1: Swift

Swift is Apple's modern programming language tailored for iOS development. It compiles to native code, providing high performance and efficient memory management. Consequently, Swift-based applications are generally fast and reliable, achieving robust performance without sacrificing quality.

A key advantage of native development with Swift is the direct access to iOS SDK features and hardware. This capability can lead to optimal performance for tasks requiring intensive processing. Swift also benefits from Apple's well-supported toolchain, primarily Xcode, which streamlines development workflows. However, this approach is inherently platform-specific, meaning Swift apps are limited to iOS devices unless additional frameworks or platforms are leveraged.

Swift's community continues to expand at a notable pace (around 21% annual growth), positioning it as the predominant language for iOS development over its predecessor, Objective-C. This broad base of experienced iOS developers can ease the process of sourcing talent, although cross-platform flexibility may be limited compared to other frameworks.

3.2.1.2: Flutter & Dart

Flutter, developed by Google, is a cross-platform UI toolkit that utilizes the Dart language to build both iOS and Android applications from a single codebase. It achieves near-native performance through ahead-of-time (AOT) compilation and by rendering its interface directly via the Skia graphics library. In practical tests, Flutter's runtime performance on iOS approaches that of Swift, showing only minor variances in CPU, memory, and GPU usage. While heavy, graphics-intensive animations may perform slightly better under purely native solutions like Swift, most use cases will experience comparable responsiveness and frame rates.

From a development standpoint, Flutter streamlines the process with features such as hot reload and a comprehensive set of pre-built UI widgets. This often results in reduced code volume and faster iteration cycles, enabling quicker time-to-market. The single codebase also supports deployment to multiple platforms, lowering maintenance overhead for projects targeting both iOS and Android.

Dart, the language powering Flutter, has seen swift growth—approximately 33% in 2023—leading to around three million users. While this increase is driven largely by Flutter's popularity, the pool of experienced Dart developers still lags behind more established languages like Swift. Another consideration is app size: Flutter's engine includes all the necessary components for cross-platform operation, resulting in binaries that are generally heavier than those produced through purely native development. Nevertheless, with Google's backing, a rapidly expanding community, and an extensive repository of open-source packages, Flutter remains a strong contender for cross-platform app development.

3.2.1.3: React Native:

React Native is a popular open-source framework developed by Meta that allows developers to build mobile applications using JavaScript and React. It enables one codebase to run on both iOS and Android devices, which can significantly reduce development time and maintenance overhead. For our project, React Native was considered as a possible approach for the mobile companion app that connects to the breast milk weighing device. Its component-based structure and live reloading features make it ideal for rapidly building and testing user interfaces, especially for data-driven apps like ours that

require regular updates and real-time display of weight, date, and inventory alerts.

React Native also has a strong ecosystem of libraries, including tools for handling Bluetooth and QR code scanning, which could align well with the core functions of our system. However, the major limitation we encountered was related to Swift-specific development. Since our final mobile app is focused exclusively on iOS, using Swift directly in Xcode provided more native functionality and tighter integration with Apple's APIs. This is especially important for device pairing, background tasks, and accessing low-level Bluetooth features required for a smooth and reliable user experience. While React Native offers many advantages in cross-platform development, we ultimately chose to prioritize a native Swift-based solution due to better long-term compatibility with the hardware and the operating system.

Another challenge with React Native is performance optimization. While it performs well for general UI and typical mobile tasks, it may fall short when dealing with high-frequency real-time data, hardware communication, or advanced animations. This is because React Native uses a JavaScript bridge to communicate with native components, which can introduce delays or performance bottlenecks, especially in tasks requiring rapid state updates—something that could impact the responsiveness of our app when receiving new weight or temperature readings from the device.

3.2.1.4: JavaScript (Backend Development)

While JavaScript isn't typically utilized for native iOS interfaces, it remains a central technology in backend development through platforms like Node.js. Teams with a background in front-end web development may find leveraging JavaScript on the server particularly appealing due to the prospect of code reuse and maintaining a single, unified language across both the frontend and backend.

Node.js, which runs on Google's V8 engine, is known for its high throughput and non-blocking I/O model. This event-driven, asynchronous architecture allows efficient handling of numerous concurrent connections, making it ideal for real-time applications, streaming data, or IoT scenarios. However, Node.js performs less optimally with CPU-intensive tasks, as these can monopolize the single-threaded event loop. One common approach is to offload such operations to worker threads or external microservices.

JavaScript’s flexibility and extensive library ecosystem—primarily through the npm registry—further streamline development. With roughly 22+ million JS developers worldwide, there is robust community support and ample resources. Notably, TypeScript, which adds static type-checking to JavaScript, is also compatible with Node.js, offering an extra layer of reliability for complex projects. Given its performance profile and vast community, JavaScript (via Node.js) is well-suited for server-side components in this project, though it remains separate from the native iOS interface development.

3.2.1.5: App Language Selection

Table 3.15 - App Language Comparisons

App Languages	Mobile Applications	Developer	Strengths	Weakness
Swift	iOS	Apple	High performance & memory management	Limited to iOS devices
Flutter & Dart	iOS & Android	Google	Reduced code volume & faster iteration cycles	Binaries generally heavier leading to larger app sizes
JavaScript	iOS & Android	Netscape	Allows for unified language across frontend & backend	Not generally used for iOS interfaces
React Native	IOS & Andriod	Meta(FaceBook)	Access to a big library eco system	Code syntax more complex, performance optimization

Based on our project requirements—particularly the need for native performance, seamless integration with iOS hardware, and a future-proof technology stack—our team has chosen Swift as the primary language for iOS application development. Swift’s compilation to native code provides the high efficiency and memory management necessary for a responsive and stable user experience, which is critical for applications that demand real-time functionalities. Additionally, Apple’s robust tooling around Swift, particularly Xcode, streamlines the development process and simplifies the integration of iOS frameworks, sensors, and system-level features.

While alternative frameworks and cross-platform solutions exist, Swift’s direct access to Apple SDKs offers a clear performance advantage and minimal overhead—an important consideration for battery usage, user interface responsiveness, and secure data handling. Though adopting Swift naturally limits our codebase to the iOS ecosystem, this trade-off aligns with our goal of delivering an optimized, platform-specific experience that leverages the full capabilities of iOS devices. As the project evolves, we will continue to evaluate ongoing language advancements and ecosystem developments, but Swift presently remains the most suitable choice for an efficient, high-quality iOS application.

3.2.2: Backend Frameworks

In order to store, process, and securely manage the data coming from our breast milk weighing and labeling device, a reliable backend system is essential. This backend must be able to handle frequent updates—such as new weight readings—while also providing robust data storage, user authentication, and security measures. It serves as the central hub that bridges the hardware device and the mobile app, ensuring that measurements are consistently logged, easily retrievable, and fully protected. In the following subsections, we examine several popular backend technologies—Node.js, Django, and Ruby on Rails—to determine which framework might best meet our needs in terms of performance, scalability, developer productivity, and real-time capabilities.

3.2.2.1: Node.js (JavaScript)

Node.js is a runtime environment for server-side JavaScript that utilizes an event-driven, non-blocking I/O model. This architectural choice makes Node especially adept at handling numerous simultaneous connections without monopolizing system resources, which proves advantageous in scenarios requiring real-time data transfer or frequent updates to multiple devices. For instance, if our breast milk measurement device streams temperature data or sends continuous status notifications to the mobile app, Node's efficient concurrency management would facilitate real-time event handling.

Under the hood, Node runs on the Chrome V8 engine and uses a single-threaded event loop to coordinate incoming requests. Rather than spinning up a new thread for each connection, Node places requests into a queue and processes them asynchronously. This approach yields high throughput for I/O-bound workloads such as database queries or sending notifications. Well-known organizations have adopted Node for its lightweight design and ability to handle numerous open connections concurrently.

A key benefit of Node.js is the unified language stack. Development teams can code both client-side interactions and backend logic using JavaScript, simplifying collaboration and reducing the need to switch between languages. Node itself provides only the core runtime; security features such as request validation, authentication, or rate limiting require external modules like Express.js and specialized middleware. Incorporating these measures is crucial if the application handles sensitive information—particularly relevant in health-related contexts.

Regarding computationally intensive tasks, Node's single-threaded model can introduce bottlenecks. Demanding operations, such as complex image analysis or resource-heavy data processing, may block the event loop and delay other tasks. However, in the context of measuring breast milk volume or performing routine database operations, this limitation is unlikely to pose a major issue. If the project eventually expands to require heavier processing, resource-intensive jobs can be delegated to external services. Overall, Node.js offers strong

performance, supports real-time interactions, and aligns well with a modern backend approach for connected devices like our milk-weighing system.

3.2.2.2: Django (Python)

Django is a high-level Python framework distinguished by its “batteries-included” philosophy, offering an extensive feature set from the outset. This includes an object-relational mapper (ORM) for seamless database interactions, a configurable admin panel for efficient record management, and built-in safeguards against common exploits like SQL injection, cross-site scripting, and cross-site request forgery. These default features allow projects to begin rapidly without relying heavily on third-party integrations.

For applications with significant data storage needs—such as detailed logs of temperature readings, timestamps, or user records—Django’s ORM can substantially simplify database queries and handling. Highly trafficked platforms such as Instagram and Pinterest have successfully scaled on Django, demonstrating its ability to accommodate large volumes of data and user requests. The framework’s architecture also supports horizontal scaling by distributing load across multiple web servers, with caching and database layers absorbing heavier workloads.

Security is a cornerstone of Django, making it particularly appealing if your application requires user authentication or the storage of sensitive health information. Though Python may not match compiled languages in raw speed or Node.js in asynchronous capabilities, its performance is typically sufficient for standard web service scenarios. Meanwhile, features like the auto-generated admin panel significantly streamline development, enabling the team to focus on core functionality rather than configuring administrative interfaces. If the team values a structured environment with built-in security measures and straightforward user management—and is comfortable with Python—Django offers a powerful and efficient choice for backend implementation.

3.2.2.3: Ruby on Rails

Ruby on Rails, is a web framework built on the Ruby programming language that emphasizes rapid development and streamlined workflows. Adhering to the Model-View-Controller (MVC) pattern and the principle of “convention over configuration,” Rails allows teams to launch web applications with minimal manual setup. The scaffolding tools can swiftly generate base code for models, views, and controllers, enabling a quick start and letting developers focus on fine-tuning specialized features—such as a web dashboard to accompany a breast milk monitoring device.

Rails is supported by a vast collection of Gems, or pre-built libraries, which can handle tasks ranging from file uploads to background job processing. This ecosystem accelerates feature implementation without the overhead of writing everything from scratch. Well-known services like Shopify and GitHub operate on Rails, demonstrating its capacity for handling substantial workloads when optimized effectively. However, Rails generally consumes more server resources at scale compared to Node.js, so additional server capacity may be necessary to maintain performance under heavy traffic.

On the security front, Rails defaults to safe practices, automatically escaping user input to guard against vulnerabilities like SQL injection and cross-site scripting. It can also be configured to meet stricter standards where sensitive data—such as personal health information—demands encryption and rigorous authentication procedures. While Rails does not specialize in high-frequency real-time updates, it remains well-suited for projects requiring rapid prototyping or moderate concurrency. For teams aiming to build and deploy a robust backend with minimal setup time, Rails offers a polished, developer-friendly experience that can swiftly bring a product to market.

3.2.2.4: Backend Framework Comparisons

Choosing the right backend technology ultimately depends on our team’s expertise, project timeline, and specific requirements for speed and security. Node.js, with its event-driven architecture, excels at handling multiple concurrent connections, making it especially suitable if the breast milk device

needs real-time updates or push notifications to a mobile app. On the other hand, Django’s battery-included philosophy and robust default security measures allow for rapid development of data-centric applications, which can be a huge advantage if the system frequently stores and processes a large volume of records. Meanwhile, Ruby on Rails stands out for its streamlined setup and convention-over-configuration approach, reducing the time it takes to implement and maintain new features—ideal for projects with tight deadlines or smaller teams. Each framework has proven capable of scaling to accommodate growing user bases, but the path to scalability can differ—Node.js emphasizes non-blocking I/O, Django leverages its ORM and caching layers, and Rails relies on multiple servers and optimized database usage. By considering both the nature of our device’s data flow and the level of real-time interaction required, we can identify which backend technology best aligns with our development goals.

Table 3.16 - Backend Framework Comparisons

Framework	Language	Best Fit	Development Speed	Performance	Security	Scalability
Node.js	JavaScript	Real-time apps & concurrency	Medium	Excellent for I/O-bound task	Requires additional libs (e.g., Express + middleware)	Horizontal scaling; easy concurrency mgmt
Django	Python	Data-heavy or content-driven projects	High	Good for web applications	Built-in protections; strong defaults	Horizontal scaling via more servers
Rails	Ruby	Fast prototyping & startup-friendly	High	Slower than Node.js, can handle large traffic with optimizations	Built-in secure defaults	Scales via multiple instances & caching

3.2.2.5: Backend Framework Selection

We ultimately chose Node.js because it delivers the best balance of speed, scalability, and compatibility with real-time data streams coming from our milk-weighing device. Among the frameworks we compared, Node's event-driven architecture is especially well-suited for handling multiple concurrent connections without the overhead of threading, meaning it can gracefully manage frequent updates from the hardware device (such as weight or temperature readings) and swiftly push notifications to the mobile app. The ability to write both frontend(if needed) and backend components in JavaScript also streamlines development, allowing the team to reuse skills and libraries across the entire stack rather than switching between languages. Although Node requires extra diligence in configuring security through frameworks like Express.js and associated middleware, its lightweight I/O model and active community more than compensate by offering an adaptable ecosystem for building and scaling backend services. This combination of real-time capabilities, language consistency, and community support makes Node.js the most suitable choice for our project's backend.

3.2.3: Database Options

The PUMPED project includes a hardware device that weighs breastmilk and prints labels with information about volume, date, and time. A companion mobile app then tracks storage details and sends notifications when supplies are low or about to expire. To support these features, a reliable database must be chosen to store user data, breastmilk bag information, and any additional properties that might evolve in the future. The choice between different database solutions depends on data structure requirements, potential scalability, and ease of integration with both the backend and the iOS application.

3.2.3.1: PostgreSQL

PostgreSQL is an open-source relational database that organizes data in structured tables, ensuring consistent handling of transactions. This can be especially beneficial for storing precise records, such as the total number of milk bags or any updates regarding the relocation of stored items. PostgreSQL also supports JSON fields, which allows for storing more flexible data if new

attributes need to be added later, without sacrificing the core reliability of traditional relational schemas.

Vertical scaling is straightforward, and replication can be set up to manage a growing number of read requests. Advanced features like complex queries, indexing, and multi-table joins mean that PostgreSQL remains strong if the application logic requires intricate data relationships and transactional integrity. Integration is widely supported across major backend frameworks, and the iOS app would simply communicate through an API layer. This makes PostgreSQL a dependable solution when data integrity and structured queries are vital for the PUMPED system.

3.2.3.2: MongoDB

MongoDB is a document-based database that stores data in JSON-like documents rather than structured tables. This setup provides a high degree of flexibility, allowing the application to add or modify fields with minimal effort. A breastmilk inventory app might benefit from this approach if it intends to include varying types of information, such as nutritional content or additional metadata about each milk bag, because each record can adapt as needs change.

MongoDB was also designed to scale horizontally, so distributing data across multiple servers is more straightforward if the application must eventually handle a high volume of writes or very large datasets. The system favors quick writes and simple document retrieval, which can be advantageous in scenarios where performance is crucial and data relationships are relatively simple. However, for certain cases involving multi-document updates, MongoDB transactions have to be used carefully to maintain consistent data. If a Node.js backend is chosen, MongoDB integrates seamlessly, making it a reasonable option for projects that require flexibility and potential horizontal growth.

3.2.3.3: MySQL

MySQL is a long-established SQL database known for stability and speed, particularly under read-heavy workloads. It uses a structured table format, where data is organized in rows and columns, ensuring that transactions involving breastmilk inventories and user information remain consistent. This is especially useful if the PUMPED system relies on straightforward queries for tasks such as retrieving available milk volume or checking expiration dates.

Scaling MySQL typically involves vertical enhancements or adding replication for read-heavy scenarios. Although sharding MySQL is not as direct as with some other databases, it has been done successfully in large-scale deployments, indicating that with sufficient engineering effort it can accommodate substantial growth. MySQL also supports JSON fields in newer versions, which can be helpful if semi-structured data needs to be stored. It integrates well with most development frameworks and can be set up on managed cloud services, making it an accessible and reliable choice for teams that prefer a well-known relational approach.

Table 3.17 - Database Comparisons

Database Option	PostgreSQL	MySQL	MongoDB
Data Model	Relational (SQL)	Relational (SQL)	Document (NoSQL)
Structure	Strict schema; can store semi-structured data via JSON fields	Strict schema, well-suited for basic tables/joins	Flexible schema (JSON-like documents)
Performance	Good for complex queries and large datasets	Typically fast reads, widely used	Excellent for high write loads and large-scale, unstructured data
Scalability	Vertical scaling common; horizontal replication and sharding supported	Vertical scaling + read replicas; sharding possible with extra tooling	Horizontal scaling and sharding built-in
Transaction	ACID-compliant; advanced SQL features like window functions	ACID (with InnoDB engine); robust for standard operations	Multi-document transactions introduced more recently, optional use
Use Cases	Ideal for structured data, complex queries, strict integrity	Perfect for typical web applications or smaller, simpler database needs	Great when flexibility is paramount or data models evolve frequently
Security	Offers robust authentication/roles; encryption at rest supported	Various security layers (depending on version and engine)	Document-level security in newer versions, requires careful config

3.2.3.4: Database Selection

We decided on PostgreSQL not only for its robust ACID compliance, advanced querying capabilities, and integration with modern frameworks, but also because the group has prior experience with the SQL-based workflow, and finds it more straightforward. Even though MongoDB was effective for certain projects in the past, our team ultimately prefers PostgreSQL's relational structure and powerful SQL features for handling the structured data of our bagged milk inventory. This familiarity with SQL makes schema design, query writing, and general maintenance more intuitive, ensuring that future development and scaling can proceed smoothly. Its ACID compliance ensures reliable handling of critical updates—like inventory counts or expiration dates—while JSON field support allows for semi-structured data if we need more flexibility in the future. Although other databases like MySQL or MongoDB perform well for many applications, PostgreSQL's combination of relational consistency, querying capabilities, and compatibility with modern frameworks made it the ideal choice to pair with our Swift-based mobile app. This approach allows us to maintain strict control over data accuracy and security while also offering room to expand the schema if our project scales to include more detailed analytics or user-driven features later on.

3.2.4: Cloud Storage

Storing breastmilk inventory data in the cloud can bring valuable benefits to an app like PUMPED, but it also raises some important trade-offs. Our team is considering implementing cloud storage as part of the system's long term functionality. However, for the time being, we will focus on building a reliable, local connection between the hardware and the mobile app. Cloud-based solutions let users synchronize data across multiple devices, prevent data loss through remote backups, and enable features like real-time alerts or shared access among parents and caregivers. However, these advantages must be balanced against potential drawbacks such as relying on constant internet connectivity, navigating privacy regulations, and incurring costs for hosting and maintenance. Ultimately, the decision to adopt a cloud solution depends on whether the project needs multi-user access, real-time syncing, and robust backup capabilities, or if it could suffice with a simpler local-only approach.

3.2.4.1: Benefits of Cloud Storage

Cloud storage makes data available across any device with an internet connection, allowing users to check and update the milk inventory in real time, even if they switch phones or if multiple people need to coordinate the same supply. Having the data off-device also means it is protected against phone loss or damage, as cloud providers typically replicate data across multiple servers for reliability. Sharing data with other family members or healthcare professionals becomes straightforward, because everyone can access the central cloud store rather than passing files around. If the system grows to include more complex information—like nutritional stats or photos—cloud services can usually be scaled up with a few settings, sparing the user’s device from storing large amounts of data. Another advantage is that once the data is in the cloud, it can be integrated with backend processes such as automated expiration reminders or analytical tools, enhancing the overall user experience.

3.2.4.2: Drawbacks of Cloud Storage

A primary concern with cloud storage is the need for an internet connection at all times. Users who are offline cannot update or retrieve the latest information unless the app includes an offline caching strategy. Security and privacy are also vital considerations, especially if the system handles data that may be viewed as sensitive or subject to health regulations. Storing data on third-party servers introduces the possibility of unauthorized access if the system is not configured with proper encryption and authentication. In addition, there can be hidden costs once the free-tier limits are exceeded or if the project scales beyond the initial prototype. Maintaining a cloud-based backend also requires more setup and ongoing management than a purely local solution, though modern services have simplified much of this process.

3.2.4.3: Amazon Web Services (AWS)

Amazon Web Services provides an extensive collection of cloud services that can fit a range of application needs, making it a compelling choice for a system like PUMPED. Developers can deploy a relational database through Amazon RDS and use Amazon S3 for file storage, which allows for separating structured data (such as milk bag details) from unstructured files (such as labeled images or logs from the hardware device). AWS also includes a variety of tools for

handling more specialized requirements, such as real-time data processing or IoT integration, which can be particularly useful if the hardware device transmits data to the cloud in real time. While this breadth of features is a major advantage, AWS can have a steeper learning curve because it often involves configuring multiple services that work together. In a production context, AWS can scale efficiently to accommodate significant growth in data or user activity. However, for a smaller-scale or student project, the setup complexity and potential cost for additional services might be more than some teams prefer, unless they make use of frameworks like AWS Amplify, which simplify the process of linking front-end applications to AWS back-end components.

3.2.4.5: Firebase

Firebase is a Google service specifically focused on mobile and web development, offering streamlined, serverless tools that reduce much of the usual overhead associated with building and maintaining a back-end. It centers around NoSQL databases like Firestore or Realtime Database, which sync data in near-real-time and can keep multiple devices up to date, even if they briefly go offline. This functionality suits the PUMPED project if the team wants each user's app to receive instantaneous updates on milk quantities or expiration alerts. Firebase also provides integrated user authentication, file storage, and serverless functions, all accessible through straightforward SDKs. This simplicity means developers can concentrate on building user-facing features rather than juggling different back-end infrastructures. Although Firebase is flexible enough for many use cases, it may feel limiting if the app demands complex relational queries or requires a wider range of services. It also involves reliance on Google's ecosystem, which can bring pricing concerns when usage exceeds the free tier, but for many smaller projects, Firebase's convenience and scalability make it an appealing choice.

3.2.4.6: Google Cloud Platform (GCP)

Google Cloud Platform is a broader suite of services beyond Firebase, giving developers the ability to deploy virtual machines, run managed relational databases through Cloud SQL, and leverage serverless platforms such as Cloud Functions or Cloud Run. These resources can be assembled to create custom architectures that handle a variety of tasks, from detailed data analytics on milk usage trends to automated notifications for low stock or expired inventory.

GCP's approach can feel more cohesive than AWS for some teams, but it still presents the typical considerations of setting up and managing several services. Whether the PUMPED system requires traditional relational data handling or specialized machine learning APIs, GCP can accommodate these demands while retaining the option to integrate with Firebase for real-time synchronization. As with AWS, using multiple GCP services can result in increased complexity and costs, so teams should weigh whether the advantages of Google Cloud's services warrant the additional configuration steps.

3.2.4.7: Evaluating Potential Cloud Services

Table 3.18 - Cloud Storage Comparisons

Cloud Platform	Developer	Database	Strengths	Weaknesses	Cost
Amazon Web Services (AWS)	Amazon	Amazon RDS	Variety of tools to fit a range of application needs	Requires configuring multiple services together	High (\$ per service)
Firebase	Google	NoSQL	Focuses on mobile apps, syncing data & quick development	Limited depending on apps complexity	Low (free tier)
Google Cloud Platform (GCP)	Google	CloudSQL	Provides more services beyond Firebase	Requires setting up & managing several services	Medium (\$ per service)

For a project like PUMPED, choosing the right cloud platform can hinge on the desired mix of simplicity, flexibility, and cost considerations. A general-purpose ecosystem such as Amazon Web Services (AWS) is favored for its extensive range of services, including options for running a PostgreSQL or MySQL database, storing images, and integrating IoT devices. AWS is extremely

powerful and can accommodate significant growth but has a steeper learning curve.

Firebase, offered by Google, is popular for mobile apps that need real-time syncing and minimal back-end coding. Its straightforward SDKs, built-in authentication, and NoSQL database features enable quick development, making it suitable for prototypes or student projects. If the team wants more control and broader infrastructure options, Google Cloud Platform (GCP) provides additional services such as managed relational databases, serverless computers, and machine learning tools, all under Google's ecosystem.

3.2.5: Data Transfer Methods For ESP32

In a home environment, an ESP32-based device that weighs bagged breast milk must send basic data—weight, date, and time—to a companion mobile app. Several options exist for transmitting small data packets over short distances or through a quick scan. These include Bluetooth Classic, Bluetooth Low Energy (BLE), and methods involving either traditional barcodes or QR codes. Each approach differs in terms of hardware cost, ease of integration, efficiency, security, and user experience.

3.2.5.1: Bluetooth Classic

Classic Bluetooth was originally designed to replace wired connections like serial ports, and is still widely used for applications such as audio streaming. The ESP32 chip supports dual-mode Bluetooth, meaning it can handle both Classic and BLE. This implies minimal additional hardware costs for Bluetooth-based data transfer. In many use cases, Classic Bluetooth offers faster data throughput than BLE, which is valuable for streaming large files or audio.

However, the relatively high energy consumption and limited iOS integration create potential complications for a milk-weighing device intended for home use. Although Android permits easy data exchange via the Classic Bluetooth Serial Port Profile, iOS enforces stricter controls. Integrating an ESP32 with iPhones over Classic Bluetooth typically requires Apple's special licensing program, increasing complexity. Classic Bluetooth also has a generally more

elaborate pairing process than BLE, though it remains straightforward in simple single-device setups. Because the volume and date data is small, the higher bandwidth of Classic Bluetooth offers little advantage over BLE. In a scenario emphasizing ease of use across all smartphones, Classic Bluetooth can become a limiting factor on iOS.

3.2.5.2: Bluetooth Low Energy (BLE)

BLE is a more recent standard optimized for low-power, intermittent data transfer—ideal for battery-powered or smaller devices. Despite a lower overall bandwidth, BLE’s speed is sufficient for sending short messages, such as the time and weight of a milk bag. Its low energy usage is a distinct advantage if the device runs on battery. BLE also connects quickly, enabling a phone to discover and read data from a device within seconds. The ESP32’s built-in dual-mode radio means no additional hardware is needed.

From an integration viewpoint, both Android and iOS come with robust BLE APIs, and many development frameworks provide examples for discovering, pairing, and reading data from a BLE peripheral. Once the device and phone are paired, the data exchange happens securely through encryption. This enhances privacy if the measured data is considered sensitive. Another benefit lies in user convenience: the ESP32 can automatically deliver or “notify” the phone of new weight data in near real time, removing the need for manual scanning or multiple steps. While BLE still requires initial pairing, it is generally considered easier to integrate across different operating systems than Classic Bluetooth when building custom mobile applications. The seamless nature of BLE makes it highly suitable for a consumer-friendly experience, where the phone automatically collects data whenever the user is in range.

3.2.5.3: Barcodes

A barcode-based solution avoids wireless connectivity altogether by printing an encoded label. Since the device already prints information about milk volume or timestamps, embedding that information in a linear (1D) barcode is straightforward. The user would scan it with their phone’s camera. This technique is especially cost-effective and maintenance-free from a hardware

standpoint: the printer exists regardless, and modern smartphone cameras, with the right app code, can decode barcodes reliably.

Implementation on the device side is not complex. Many microcontroller-friendly label printers can produce barcodes. Alternatively, a small library can convert a string of data (for instance, “4oz/03-17-2025 10:15 AM”) to the corresponding series of bars to print. On the mobile side, developers integrate a barcode scanning feature that leverages the phone’s camera. Once scanned, the data is extracted and recorded. Barcodes can be read quickly, but each data transfer requires a manual process: the user lines up the camera with the label and waits for a scan confirmation. Doing so for multiple bags can become time-consuming. Barcodes have no intrinsic encryption, so if privacy is a concern, the label’s visible data might be accessible to anyone who sees or scans it. Given that the data (milk amounts, times) is relatively simple, this level of risk may be acceptable in a home context. Overall, barcodes are an easy, hardware-free approach but lack real-time data flow or automation.

3.2.5.4: QR Codes

A QR code builds on the same scanning principle as a barcode but allows more data storage and includes error correction. The device side can generate a two-dimensional code containing weight, date, time, or even structured text. Most smartphones can recognize a QR code at various angles, making scanning faster and more forgiving than linear barcodes. Error correction capabilities also help if a label is partially smudged or folded, allowing successful decoding even when part of the code is damaged.

Generating a QR code is slightly more involved than generating a barcode, but open-source libraries can handle the conversion from text to a grid pattern. The user experience around scanning can be smoother than with barcodes, as modern phone cameras often natively detect QR codes. Many users now associate QR codes with quick interactions, such as menus or authentication links. Still, the device data only appears in the mobile app upon completion of a manual scan. This manual step must be repeated every time a new label is printed. As a result, while QR codes enhance scan reliability compared to barcodes, they do not overcome the core limitation of needing to physically scan each new label. For low-frequency tasks, this is tolerable. If multiple measurements or high-frequency logging is anticipated, the scanning routine quickly becomes a hassle. As with barcodes, QR code data is generally stored

in a visible, unencrypted form. For a controlled home environment, this may be acceptable.

3.2.5.5: Data Transfer Evaluations & Comparisons

An important consideration for a milk-weighing device is whether the data transfer should be instantaneous and continuous, or whether it is acceptable for users to scan each label at their leisure. Bluetooth, especially in its BLE form, enables a near-real-time, hands-off approach. The phone can automatically save each measurement whenever a bag is weighed, eliminating repeated manual actions and potential scanning errors. This is especially attractive in a setting where parents may be busy or where convenience is paramount.

By contrast, barcodes and QR codes excel in simplicity and reliability while introducing a manual scanning step. For single measurements done infrequently, scanning is harmless and avoids the complexities of wireless pairing. The trade-off is that scanning becomes more tedious if a user needs to transfer multiple measurements a day, and it does not provide immediate notifications or background syncing.

Security matters to varying degrees depending on how private the milk data is. BLE can use built-in encryption to protect the data in transit. Neither barcodes or QR codes inherently protects data: once printed, the code can be scanned by anyone who has physical access to the label. In a home environment, however, the user may accept that risk, treating the label itself as a non-sensitive item. Additional obfuscation or encryption within a QR code is possible but introduces more development overhead.

Hardware costs for all these methods are minimal, given the ESP32 includes dual-mode Bluetooth by default and the printer already produces labels. The primary investment lies in integration effort. BLE demands the most coding to establish pairing, define characteristics, and handle data flow on the phone. Barcodes and QR codes require the phone app to incorporate camera scanning, which also involves code but is arguably simpler or at least more familiar to many developers. Maintenance over time might favor BLE if frequent data updates are desired, as scanning libraries or usage patterns can become cumbersome in high-volume scenarios.

In terms of user experience, BLE stands out if the goal is to “remove friction” from the data logging process. After an initial setup, measurements can be auto-recorded without user interaction. Barcodes and QR codes, while approachable, force an extra task—scanning—for each data transfer event. This is not necessarily inconvenient if usage is sparse, but it does not match the ease of background synchronization. On the other hand, scanning approaches are platform-agnostic and do not depend on firmware versions or potential pairing issues. Any caregiver’s phone can quickly read the code, whereas BLE might require pairing each phone that needs direct data access.

Ultimately, the final decision hinges on the value placed on immediate, hands-free data transfer versus the acceptability of manual scans. Many users in a home environment appreciate a device that seamlessly logs measurements without scanning, which tips the scales in favor of BLE. Others might prefer a scan-based method if the device is used infrequently, and they want to avoid any wireless complexities. Both barcodes and QR codes remain viable, low-cost methods, but do not fully match Bluetooth’s automation or near-instant data push.

Table 3.19 - Data Transfer Comparisons

Criteria	Bluetooth Classic	Bluetooth Low Energy	Barcode(1D)	QR Code(2D)
Technology Type	Wireless (2.4 GHz), dual-mode Bluetooth on ESP32 (Classic)	Wireless (2.4 GHz), dual-mode Bluetooth on ESP32 (Low Energy)	Optical (printed labels + phone camera)	Optical (printed labels + phone camera, 2D code)
Hardware Requirements	ESP32’s built-in Bluetooth (may require MFi licensing for iOS Classic support), no extra module needed but possibly more power usage	ESP32’s built-in Bluetooth LE radio, no extra module needed, extremely low power draw	Printer to generate 1D barcodes, phone camera to scan	Printer to generate QR codes, phone camera to scan (or device display if going paperless)
Data Speed	Moderate–High throughput (suitable for streaming or file transfer),	Moderate throughput (optimized for	Instant data retrieval upon scanning, but only	Quick scanning even at an angle, supports more data than 1D barcodes (due to

	but more power consumption than BLE	smaller, frequent data packets), connects/disconnects quickly	minimal data can be encoded; user must do each scan	2D format), but still manually scanned
Security & Encryption	Encrypted link is possible, but pairing can be more complex; iOS typically requires special MFi program for full Classic usage	LE Secure Connections offer strong encryption and authentication, typically simpler pairing experience on both iOS and Android	Printed data is visible; no built-in encryption (plain text in the barcode). Physical possession needed to scan	Data is plainly visible unless encrypted or obfuscated. Built-in error correction, but still no inherent encryption for sensitive info
User Interaction	Typically requires manual pairing in OS settings, better on Android than iOS if no MFi license	Usually handled in-app; minimal user intervention once paired (automated data push possible)	User must open scanning app and align the barcode each time; manual scanning of each label	User must open scanning app; faster and more forgiving to scan than 1D barcodes, but still a manual step for every data transfer
Advantages	Higher bandwidth than BLE, widely supported	Ultra-low power consumption, straightforward iOS/Android support, great for small, frequent updates	Extremely low implementation cost, no radio configuration needed, easy to integrate basic scanner libraries	Faster scanning with built-in error correction, can store more info than 1D barcodes, user familiarity with QR codes is high
Disadvantages	More power-hungry than BLE; iOS support is limited without MFi licensing, pairing can be cumbersome	Lower peak data rate than Classic, still requires initial pairing steps, must maintain BLE code logic in the app	Manual scanning becomes tedious if frequent updates are needed, limited data capacity, no real-time push	Manual scanning still required, data is only transferred at the moment of scan, no real-time updates, label printing or display is mandatory

3.2.5.6: Data Transfer Conclusions

For an ESP32-based milk-weighing device that prints labels, Bluetooth Low Energy offers the most modern, user-friendly, and energy-efficient solution to transfer weight and time data into a companion mobile application. The user

gains a near-real-time, hands-off experience, while developers benefit from a robust ecosystem of cross-platform BLE tools. In contrast, barcodes and QR codes function well for smaller-scale or infrequent tasks where manual scanning remains acceptable. These scanning methods minimize wireless constraints but require additional user input and do not update automatically. In a domestic setting where the device is likely to be used repeatedly, BLE's seamlessness may justify the minor initial overhead of pairing and characteristic definition. Barcodes and QR codes can serve as simpler or backup options that rely solely on the camera and printed labels, but they cannot surpass the convenience of automated, background data transfer.

Chapter 4 - Standards & Design Constraints:

In this chapter, we will explore standards and design constraints that will guide and assist in the development of the project. The standards that will be discussed are key in ensuring that our system is in compliance with all regulations, including those that govern safety, as well adhering to industry guidelines. Such standards that will guide our design include, but are not limited to, the Food and Drug Administration (FDA), Centers for Disease Control and Prevention (CDC), and the National Fire Protection Association (NFPA). In addition, constraints regarding physical size, energy efficiency, and environmental factors will help in meeting user expectations to be utilized in household and other settings.

4.1: Standards & Regulations

Firstly, we will discuss the standards and regulations that must be evaluated when designing our products. Since our product is going to be in contact with food for infants, and moreover, deal with the preparation of the food as well as its catalogue and storage, there will be several regulatory standards to consider in order to ensure product safety and reliability. Additionally, our product will be designed to utilize the mains electricity, which requires regulatory evaluation.

4.2: Food & Drug Administration

The Food and Drug Administration (FDA) is a governing body within the United States that is responsible for the protection of public health by regulating and supervising the safety of food, drugs, cosmetics, medical devices, and other products. [13]

The purpose of the FDA is to set guidelines for the handling, consumption, and distribution of food and other materials for the safety of the general public. The articles and regulations within the FDA are concerned with the material in which food is stored in, how food packaging is labeled, and the condition in which food is stored.

4.2.1: Food Labeling Guide

The Food Labeling Guide as published by the FDA is a guide to how food packaging should be labeled, whilst ensuring that the food within is properly categorized. Although our product will not be dealing with mass distribution of food, using this article as a guideline will protect users from unsafe food. [14]

4.2.2: Food Storage

In the article Once Baby Arrives (Food Safety for Moms-to-Be) as published by the FDA, there are guidelines included with regards to how breastmilk is to be stored to ensure the safety of the food and of the infant. This guideline will be adhered to to ensure that monitoring of food is within acceptable ranges for consumption. [15]

4.3: Center for Disease Control & Prevention

The Center for Disease Control & Prevention (CDC) is an agency within the United States under the Department of Health and Human Services that is in place to protect the public from potential hazards regarding diseases that are currently threatening individuals or could potentially be a threat.

The primary goal of the CDC is to control and prevent disease within the US and across the world. The CDC alerts the public of potential hazards with regards to food borne illnesses, and is crucial in ensuring the safety of food products.

4.3.1: Food Preparation

The CDC has published guidelines that pertain to the preparation of food for infants, specifically the storage, heating, and thawing of breast milk. Our device will have the capability to prepare breastmilk for consumption; ensuring our product is within the guidelines of preparation is essential, as ensuring the safety of the food for an infant is of the highest priority. Furthermore, keeping in mind the ideal consumption temperatures of the breast milk will aid when designing the heating elements within our system, ensuring that it is as efficient as possible. [16]

4.4: Communication Standards

Communication standards play a crucial role in designing how our system connects to mobile devices and remote servers (if needed). It is necessary for there to be effective communication between the microcontroller, sensors, internal devices, and external devices (mobile app). To accomplish this, our project is going to use protocols such as Serial Peripheral Interface (SPI). SPI is a communication protocol that is used to facilitate reliable and efficient short-distance data transfer between microcontrollers and other components. By following these communication standards, our system will have appropriate control, minimize latency, and provide compatibility with multiple peripherals.

Beyond SPI, other widely recognized standards such as I2C (Inter-Integrated Circuit) can also come into play, especially if we decide to integrate additional sensors that benefit from multi-master or multi-slave setups. Because both SPI and I2C are standard protocols, numerous off-the-shelf components like temperature sensors, EEPROMs, and display modules can communicate with our microcontroller almost out of the box. Moreover, by selecting open, well-documented interfaces instead of proprietary ones, we can more easily troubleshoot connection errors, achieve consistent data rates, and upgrade hardware in the future. Ensuring each hardware link, whether it is an internal bus

like SPI or a short-range wireless channel like Bluetooth conforms to known best practices will ultimately enhance reliability, minimize downtime, and help us maintain an efficient data pipeline from the sensor level up to the user's mobile application.

4.5: User Constraints

When analyzing how our system will impact the users quality of life, there are several factors that must be considered in the design process. The groups that will use the system will be parents and potentially hospital faculty. For parents, the primary concerns that we wanted to address are affordability, safety, and ease of use. As for healthcare professionals, the concerns overlap in many regards, but also are concerned with device efficacy and regulatory compliance.

4.5.1: Parent Constraints

As parents are the primary audience for the system, the constraints and requirements for them are of great importance in the overall design. First and foremost, parents who would have a need for this system want it to save them money where possible. As discussed previously, there are separate solutions that solve individual components of the problem this device aims to solve; that being to make the processes involved with managing the food supply of an infant. While these separate solutions exist, they are individual solutions that only temporarily solve a problem that requires a more practical and complete solution. That being said, we want the device to supplement these other options, and to warrant their replacement, it needs to be cost effective. On the surface level, the system will be able to save parents money if we design it using the most cost effective parts while not compromising the functionality. However, our design must also account for power consumption of the device. In our design, we want to minimize potential costs by designing in a fashion that consumes only power required, and adding functionality that can reduce power consumption, such as having a 'sleep mode' that turns all non essential peripherals off in cases where the device is not actively used.

In addition to the device needing to be affordable, there also needs to be considerations in place to ensure safety for parents, and safety for children. As mentioned previously, when it comes to preparing breast milk for infants, the

major points of contention are when it comes to safely preparing the breast milk and storing the food. When preparing the breast milk for an infant, there are guidelines to ensure the food is at a proper temperature. When designing our system, the temperature measurement needs to be accurate to ensure the safety of consumption, reaching a balance between hot enough as to not potentially breed harmful illness, and not too hot as to harm an infant. This balance will be achieved through ensuring accurate temperature measurements, and through the design of the heating element as to not allow temperatures above a threshold that is acceptable for consumption, while maintaining the ability to heat the breast milk.

Lastly, for parents, our thought process is to ensure that the system is as easy to use for parents without sacrificing performance, features, or safety. While designing our system, our goal is to alleviate any complications that can arise with overly complex solutions. While categorizing and ensuring the safety of breast milk for infants is a nuanced issue, we aim to make our system complex in its execution of our solution whilst giving an uncomplicated user experience. To accomplish this, our app will provide necessary information as it becomes pertinent, such as expiration dates of food that has been stored in long term storage. Additionally, the physical form factor of our device will display and request only required information. This potentially may include the date and time, or intended use, like if the milk is going to be frozen, or immediately fed to the infant.

4.5.2: Medical Faculty Constraints

As for medical faculty constraints, we wanted to investigate how medical faculty would interact with our device, or specific concerns they have with its functionality. After doing research to determine what constitutes usability for medical equipment, as well as consulting healthcare professionals related to group members, there were several factors that needed consideration. A major constraint that medical faculty have with equipment is efficacy. The device must be accurate in its measurements, especially with regards to temperature measurement. Any temperature deviations in the milk can be potentially dangerous for an infant to consume, either in terms of burning and infants mouth, or if improperly heated or stored potentially causing foodborne illness. With this in mind, the device design should incorporate alternative methods for measurement as a means for checking measurements as a safety procedure.

Furthermore, medical workers must use equipment that adheres to hospital standards and guidelines. Since hospital guidelines differ on a case by case basis in terms of acceptable medical devices, depending on the state, or even country, our design must adhere to the nationally accepted protocols and standards for medical devices to have the system in place for hospitals and other healthcare facilities to use.

4.6: Additional Constraints

While the constraints of users of our system are most important in our design effort and decisions, there are further constraints that must be considered when designing our system. Such constraints of note will be mentioned in this section with specific details on how certain design elements may be limited by constraints, while discussing how we will best tackle acknowledging constraints that are going to impact how we design our project.

4.6.1: Environmental Constraints

Since our project directly involves the manufacturing of PCBs and their assembly, it is key to understand how it will affect the environment. Ensuring that our project adheres to constraints associated with the environment, such as reducing energy wasted, using safe materials during assembly, or designing our project in a way that its lifespan will be extended, will be helpful in the design process. In addition, we must acknowledge how the environment will interact with our design, and how we will account for factors to ensure our product's utility in different conditions.

Our system will be utilizing standard wall adapters for power, which means it needs to be considered how much energy our system will be consuming. It is desirable for our system to not consume excessive power, as this will save parents from excessive power bills if they have great use for our system. Furthermore, by designing our system to use only necessary power, this will reduce energy waste, which is heavily intertwined with our design, since we will be using conduction heating, which naturally wastes energy in the form of excess heat to the environment.

As technology has evolved and PCB manufacturing with it becomes more necessary, a concern with our project is raised in the form of electronic waste, or

e-waste. E-waste typically results from electronic components being disposed of when they reach the end of their lifespan, or when a consumer loses use for them. The problem with this is obvious, as we want our product to last as long as it is needed. In our design, we must design it in a fashion that maximizes its lifespan; we can accomplish this by making as many individual parts of the system replaceable, such as individual components. When choosing components for manufacture, we must also be mindful of hazardous materials, like lead or mercury. Since we are working with a device that potentially can come in contact with food, we will be utilizing lead-free solder, which functions identically to lead-based solder.

Lastly, our system must be able to operate in a wide range of conditions. The heating element in our device will cause the temperature at which the device operates to rise, which must be considered in our design, since several components are sensitive to temperature changes. Bearing this in mind, our design must incorporate some insulative materials to ensure that there are no areas that experience rapid changes in temperature, potentially resulting in our device being compromised.

4.6.2: Power Constraints

Power constraints are critical to consider when designing an electronic device, as they impact efficiency, performance, and safety of a circuit. Power constraints are the limitations imposed by the available power supply, energy consumption, and thermal dissipation of components used. Managing power constraints is vital to a circuit so that it does not exceed voltage or current limits set to prevent damaging internal components.

One of the primary factors in power constraint is the power supply limitations. Every electronic system has a specific operating voltage and current rating and if they are exceeded then component failure is likely as well as erratic behavior by the device. If multiple heating elements are connected, then the total power must be carefully calculated to ensure that the supply can handle the load. Voltage regulators and power management integrated circuits are also commonly used to efficiently distribute power within the system while preventing overload.

Thermal Management is also another critical aspect when it comes to power constraints. Excess power consumption generates heat which must be properly dissipated in order to prevent overheating. Heating elements like resistors

convert electrical energy into heat and if not properly managed can cause the performance to degrade or cause permanent damage to the device. Heat sinks, cooling fans, and proper ventilation are great ways to manage heat dissipation in power-constrained systems. Due to how small we intend to make our device, we will also have to look into other passive ways to cool our device down, such as through heat spreading materials and optimized airflow.

Power efficiency plays a role in determining how effectively power is being used within a system. Inefficient components or poor design can cause significant energy loss which is often in the form of heat dissipation. Switching regulators will be implemented to convert power as they are more efficient with less energy loss than linear regulators. Components like MOSFETs, relays, and resistors will also be chosen to minimize unnecessary power dissipation and create a more energy efficient circuit.

Finally system-level power constraints must be considered when integrating multiple components that draw power simultaneously. If multiple sensors, heaters, and other electrical components are drawing from the same source, then the total power must be calculated to ensure enough power is being supplied to the components. Power sequencing and load balancing must be implemented to prevent sudden power spikes that could destabilize the system. Proper power constraint management ensures reliability, safety, and longevity of electronic systems.

Chapter 5 - Comparison of ChatGPT with Similar Platforms:

In the development of our senior design project—a device to measure and track bagged breast milk—AI tools played a significant role in both the research and coding phases. We specifically utilized two platforms: ChatGPT, a conversational AI for research and planning, and Cursor AI, an AI-powered coding assistant integrated with Visual Studio Code. While both tools were extremely helpful in different contexts, they also presented unique limitations that shaped how and when we used them. This chapter explores specific examples of how each platform contributed to our project and the areas where they fell short.

5.1: ChatGPT in the Software Design Process

ChatGPT was used heavily in the early phases of our project for both technical research and architectural planning. One of the first useful examples came when we were unsure what kind of microcontroller could handle both temperature sensing and Bluetooth communication. ChatGPT helped us narrow our choices to a few realistic candidates, including the ESP32, explaining the pros and cons of each and why the ESP32's dual-core processor and built-in BLE support made it ideal for our use case. This saved us hours of manual searching.

Another concrete example occurred when designing our backend. We asked ChatGPT what types of databases would work best for a small, structured data set involving timestamps and fluid ounce values. ChatGPT compared PostgreSQL, MySQL, and MongoDB, and provided a side-by-side analysis that helped us select PostgreSQL due to its ACID compliance and our group's familiarity with SQL. The ability to get real-time answers to layered questions—like “What database works best with Node.js on iOS?”—was significantly faster than searching and piecing together answers manually from forums or documentation.

However, ChatGPT also showed limitations. For instance, when we asked it how to implement BLE communication on iOS, it suggested outdated or oversimplified Swift syntax that didn't align with modern Xcode requirements. This caused some confusion and led us to double-check everything it provided against Apple's documentation. Another example of its limitation was in suggesting generic data flow diagrams that didn't quite apply to how our physical device and mobile app would communicate asynchronously. These inaccuracies didn't derail our progress, but they highlighted the importance of treating ChatGPT as a starting point rather than a final authority.

5.2: Cursor AI for Code Generation

Cursor AI was primarily used to support the development of our backend and database systems, and it proved to be a valuable tool during this phase. Since our backend was written using Node.js with Express, and our database was built on PostgreSQL, Cursor's real-time code suggestions and integration with Visual Studio Code allowed us to develop functionality quickly and efficiently. For example, when implementing our user registration and login endpoints,

Cursor helped scaffold Express routes, add middleware, and suggest how to use packages for password hashing and input validation. It even flagged missing try-catch blocks in our asynchronous functions, helping us make our code more stable and secure.

Cursor also enhanced our PostgreSQL integration. As we wrote SQL queries using the pg library in Node.js, Cursor provided real-time assistance with parameterized query structure. It helped catch mistakes such as forgetting to return a response after a successful database insertion or incorrectly formatting SQL strings. These small suggestions saved us significant time during development and prevented many common bugs related to database interaction.

However, when we attempted to use Cursor to assist with Swift development for our iOS app, its usefulness dropped significantly. Since Swift is not natively supported in Visual Studio Code, Cursor struggled to offer accurate or up-to-date SwiftUI code. When we asked it to generate a basic login screen using SwiftUI, the code it provided often used outdated syntax or missed key elements like state binding and navigation handling. This made the process more frustrating than helpful. Additionally, because our app had to be built and tested in Xcode, we were forced to copy and paste code from VS Code to Xcode, and manually resolve any syntax errors or missing functionality. This back-and-forth workflow was inefficient and undermined the benefits Cursor provided during backend development.

In summary, while Cursor was a powerful tool for writing and refining backend and database logic, it proved unreliable for mobile development in Swift. Its lack of deep integration with Xcode and limited understanding of Swift-specific patterns made it more of a hindrance than a help on the iOS side. Nonetheless, its contributions to our backend efficiency were significant and helped us stay on track during the implementation phase.

5.3: Direct Comparison of ChatGPT and Cursor AI

ChatGPT and Cursor AI offered value in different phases of the project. ChatGPT was most helpful during the initial planning and research, while Cursor excelled once we started writing and testing code. For instance, when trying to understand what kind of backend architecture would suit our project best,

ChatGPT explained REST APIs, recommended using Express with PostgreSQL, and helped map out the database schema. Once we made those decisions and opened our code editor, Cursor became the go-to tool for actually implementing those ideas.

When it came to documentation and writing explanations for reports, ChatGPT provided polished, professional language that helped us meet our deliverables more quickly. Cursor, on the other hand, helped us write Express middleware, fix syntax issues, and even auto-complete complex SQL queries, which would've taken longer to write and debug manually.

The key difference is that ChatGPT worked best as a thinking partner for brainstorming and research, while Cursor acted as a coding assistant embedded within our development environment. Both tools had clear limitations—ChatGPT's code suggestions often lacked practical context or accuracy, and Cursor struggled with platform-specific syntax, especially outside of the JavaScript ecosystem—but used together, they formed a powerful support system that accelerated our development process.

5.4: ChatGPT in the Hardware Design Process

ChatGPT played a pivotal role in guiding our hardware design process by rapidly providing detailed comparisons and specifications for critical components. For instance, when evaluating heating methods for our device, we consulted ChatGPT to understand the relative benefits of different heating options. It highlighted several commercially available products such as the 14cm x 5cm Electric Heating Pad by Adafruit, the 15cm x 5cm Silicone Heating Pad by Meccanixity, and the Cartridge Heater offered by MakerTechStore. ChatGPT explained that electric heating pads generally operate via conduction, while other methods like convection and induction—although more common in industrial settings—may introduce unnecessary complexity for a home-use device. With this context, our team was able to weigh factors such as energy efficiency, heat distribution, and ease of integration, leading us to lean towards a

PTC-based electric heating pad that would provide rapid and controlled temperature increases.

In addition to heating components, ChatGPT assisted us in exploring power supply options for our system. It compared various choices including the Bel Signal Transformer with a Rectifier Circuit, the LRS-100-24 SMPS Adapter, and the Alitove AC/DC Power Supply. ChatGPT provided insights into the voltage and current requirements of these options, emphasizing that a stable power source is critical for both the heating element and the microcontroller. This discussion helped our group understand the trade-offs between using a robust, regulated supply like the LRS-100-24 versus a more budget-friendly option like the Bel Signal Transformer, allowing us to determine the best fit for maintaining consistent performance under continuous operation.

Moreover, ChatGPT proved valuable when we evaluated different microcontrollers for our device. By comparing the ESP32, STM32, and Nordic's nRF52840, it offered a clear overview of the features, such as integrated Wi-Fi/BLE support, processing power, and peripheral interfacing capabilities. This comparative analysis was particularly useful because it allowed us to focus on the ESP32, which provided the necessary balance of cost-effectiveness, low-power operation, and robust connectivity options essential for a compact, home-based device.

While ChatGPT offered detailed and specific information that greatly expedited our research, there were instances where its recommendations required further verification. For example, its initial suggestions on integrating a rectifier circuit with a Bel Signal Transformer lacked the granular component specifications we ultimately needed, prompting us to consult manufacturer datasheets and industry forums. Nonetheless, these shortcomings were minor compared to the overall benefit: ChatGPT enabled our team to quickly compile and compare a wide range of hardware options, ensuring that our design decisions were both well-informed and aligned with the project's performance and safety requirements.

5.5: Gemini in Hardware Research and its Limitations

Gemini AI was evaluated as an additional resource during our hardware research, particularly when exploring printing options for our milk-weighing device. Our project required reliable label printing to display critical data such as weight, date, and time. We looked into thermal printing solutions—specifically, the MUNBYN IMP001 58mm Thermal Printer (Serial/UART & USB), the PRIMUZ Micro Thermal Printer, and the ASPIRON 58mm Embedded Thermal Printer Module (UART & TTL Interface). Gemini AI provided initial insights into these options, but many of its recommendations turned out to be inaccurate or misleading. For instance, when queried about the MUNBYN IMP001, Gemini AI reported incorrect interface requirements and power specifications that conflicted with the manufacturer’s datasheet, leading us to question its reliability in this domain.

Moreover, Gemini AI offered a comparative overview of printing methods, including thermal printing, laser printing, and inkjet printing, but its discussion was marred by several inaccuracies. While it correctly noted that thermal printers are ideal for producing fast, smudge-resistant labels suitable for our home environment, it confused the benefits and drawbacks of laser and inkjet printers by overemphasizing their use in high-volume industrial applications. This resulted in recommendations that did not align with our specific needs for a compact, energy-efficient device. Additionally, when discussing the PRIMUZ Micro Thermal Printer and the ASPIRON module, Gemini AI sometimes interchanged interface standards—suggesting, for example, that a UART interface was comparable to a TTL interface without acknowledging the subtle differences in voltage levels and signaling.

These inaccuracies forced our team to cross-reference every piece of hardware information with official datasheets and manufacturer resources, which in turn increased our workload and slowed down the design process. Ultimately, while Gemini AI showed potential as a quick reference tool, its frequent errors in detailed hardware specifications underscored the necessity for rigorous human verification—especially in a project where component compatibility and accurate power requirements are critical to performance and safety.

5.6: Testing ChatGPT & Result Analysis

Chapter 6 - Hardware Design:

Chapter 6 will focus on the design elements necessary in the schematics for our eventual PCB's. This chapter is key for the design as it will determine which parts we will choose when assembling the final design, and will allow us to make modifications in the future as needed.

6.1: Rectification for Transformer

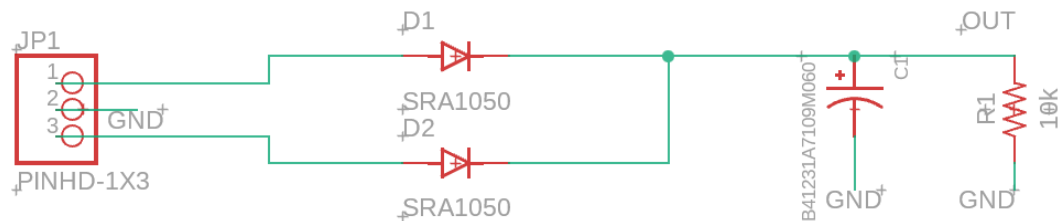


Figure 6.1 - Rectifier Schematic

The first schematic shown in Figure 6.1 is the rectification schematic. Since our design relies on grid power, we had to choose how we would go about transforming the power into usable DC power for our design. We chose to implement a signal transformer. Since we use a signal transformer, we need to rectify this output from the transformer into a more stable, consistent signal for our regulators which will be shown later. The two diodes will rectify the signal into a bumpy signal rather than a sinusoid. Afterwards, a capacitor of sufficient size will flatten the rectified waveform to an approximate DC value of 17 volts.

6.2: 12V Regulator PSU

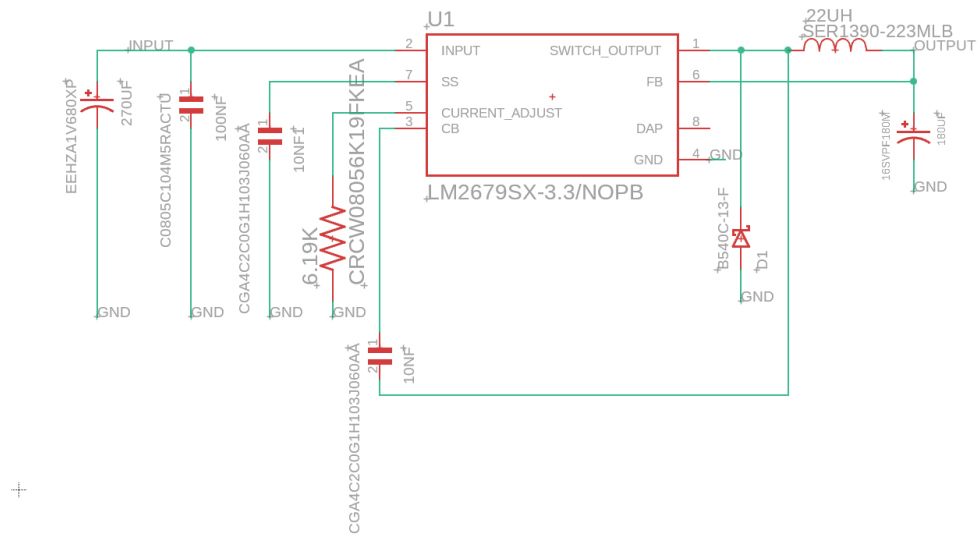


Figure 6.2 - 12V Regulator PSU Schematic

The 12 Volt PSU schematic in figure _ is solely responsible for powering peripherals that require 12 volts in our design. This design was acquired through the Texas Instrument Power Designer software. By recommendation of peers, the LM2679 IC is an excellent option for regulators, and will be used for all regulators in the future of this design. By putting this IC into the power designer with the parameters of 12 volts output at 4 amps, we attain the design above. [schematic citation]

6.3: 5V Regulator PSU



Similarly to the 12 volt regulator, this will be used to power all components of the design that require 5 volts instead by dropping the input voltage from the rectified input to the usable 5 volts. This design was also obtained from the WEBench power designer software, with the parameters changed to 5 volt output at 5 amps. The reason for the high current is because this regulator will power our heating elements, which draw considerable power in the design. This regulator design could potentially change in the future as we test the efficacy of the heating elements. Furthermore, this regulator is needed to power the LCD, which contributes to the power consumption further. [schematic citation]

6.4: 3.3V Regulator PSU

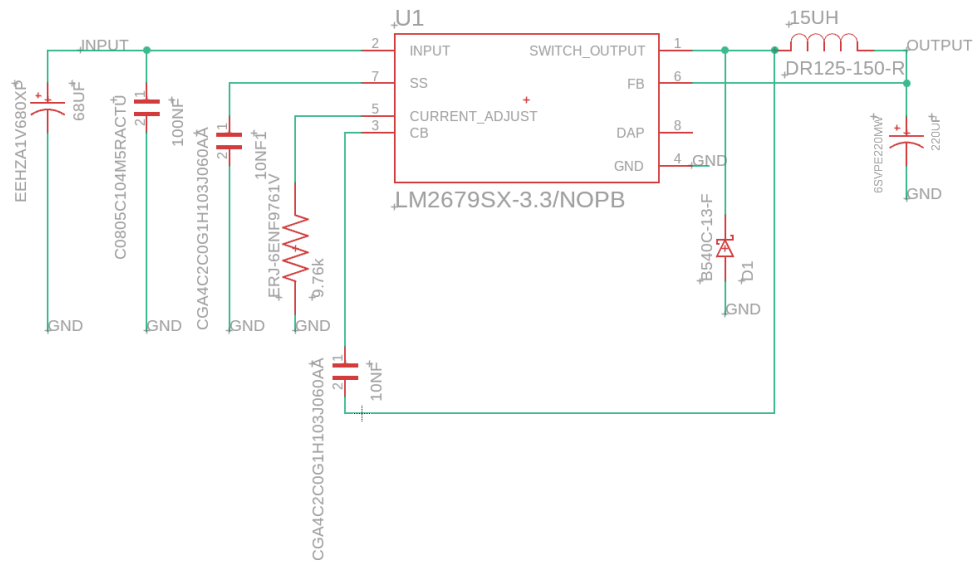


Figure 6.4 - 3.3V Regulator PSU Schematic

Our final regulator is most important of all, as it will power the MCU, thermal sensor, and load cell. This design was similarly acquired through the WEBench power designer software, which changed the components to reach the desired output of 3.3 volts at 2.5 amps, which should be sufficient power for all components that require 3.3 volts. [schematic citation]

6.5: Load Cell Amplifier Design

6.6: MCU Design

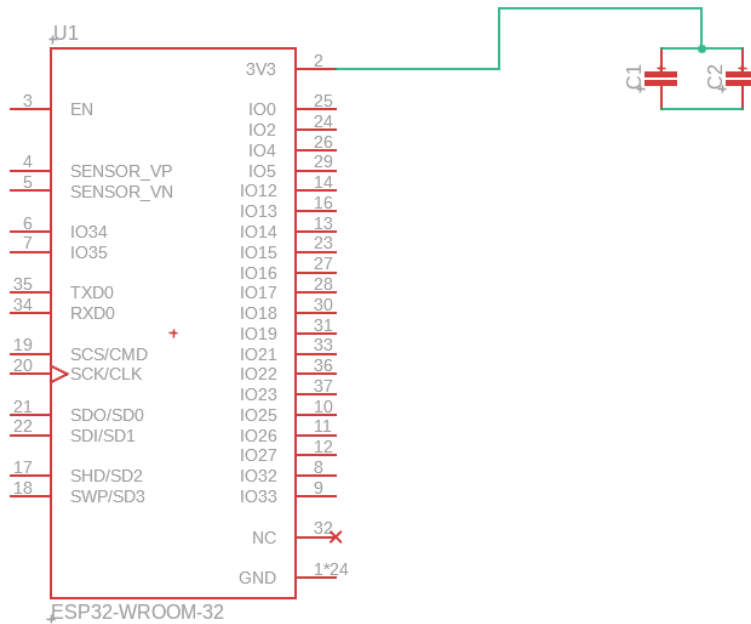


Figure 6.6 - MCU Schematic

This schematic is for the design of the MCU to show how the MCU will communicate with the other components. [In progress]

Table 6.1 - Hardware Components

Component	Part Number	Price	Voltage
Power Supply	AC-DC Wall adapter	\$5 - \$50	
Waterproof Digital Temperature Sensor	DS18B20	\$2 - \$10	3.3V - 5.0V
Load Cell	TAL220B	\$11.95 per unit	3V - 10V

LCD	HD44780	\$3 - \$10	3.3V - 5V
Thermal Printer	PRIMUZ Micro Thermal Printer	\$45 - \$65	12V - 24V
Heater	Electric Heating Pad 14cm x 5cm by Adafruit	\$6	
MCU	ESP32	Low \$\$\$	3.3V

Chapter 7 - Software Design:

The focus of this chapter is on providing an overview of the elements that make up the software development process for our project. To accomplish this, each section incorporates a diagram that represents the software's design and coding architectures that, when put together, form our software's system.

7.1: Use Case Diagram

The use case diagram is a UML (Unified Modeling Language) diagram that provides a visual representation of the interactions between users (actors) and a system. This diagram provides an easy to understand overview of the various functions of a system that the user can interact with, as well as how these functions can interact with each other. Our use case diagram focuses on the different functions of our device that a user directly interacts with, such as weighing the milk bag, heating the bag, and printing the label with the appropriate information.

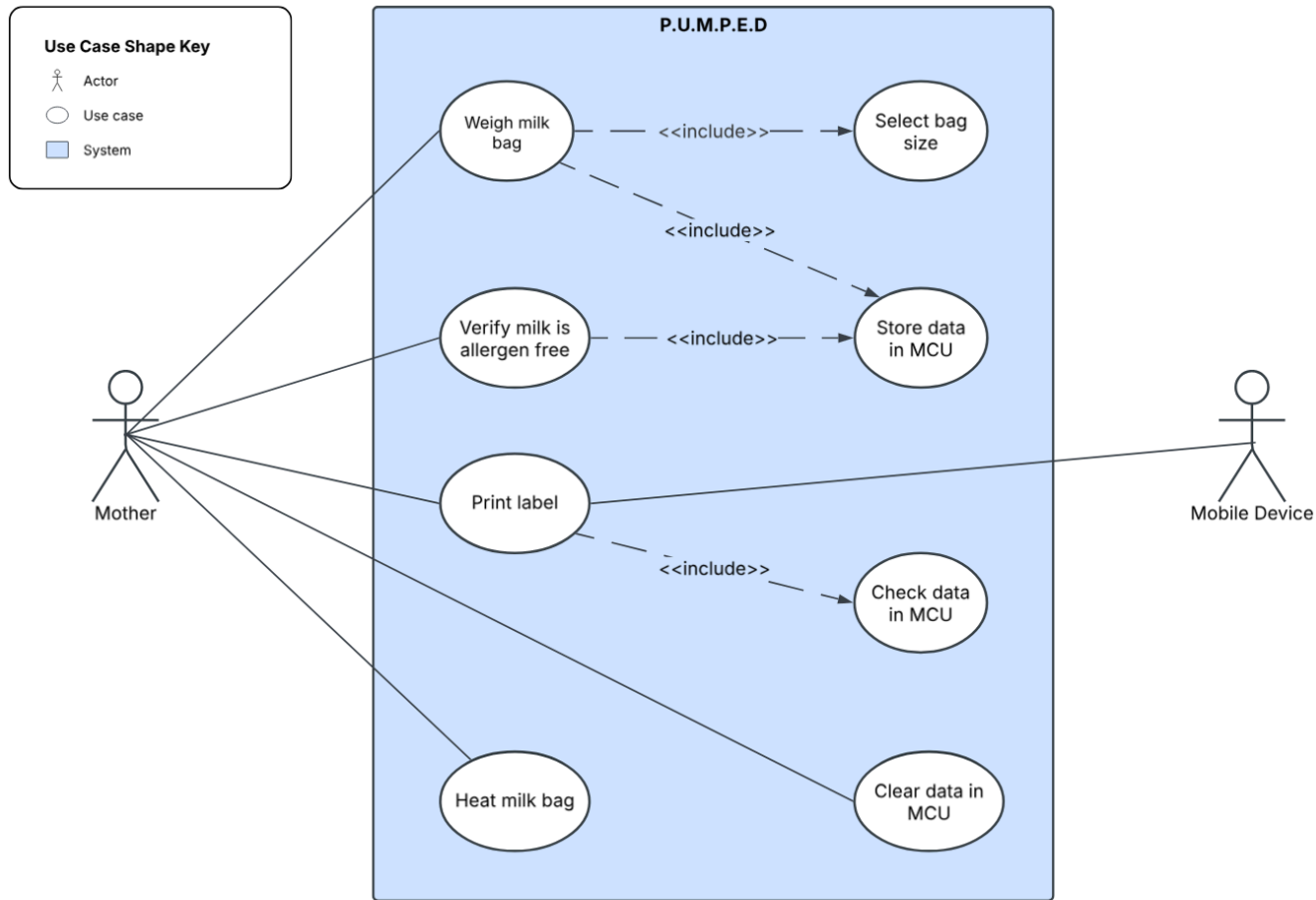


Figure 7.1 - Software Use Case Diagram

7.2: State (Activity) Diagram

The state (activity) diagram is also a UML (Unified Modeling Language) diagram that provides a representation on the condition of the entire system, or parts of the system, at certain instances of time or changes caused by external stimuli (user interactions). This diagram shows the transitions of states that occur within the system that are triggered by certain conditions or interactions. Our state diagram is mainly focused around the idle state, where the device will spend most of its time, until the user interacts directly with the device (clicking the appropriate button). Although there are various composite states, which represent states with internal activities, it's important to note that our state

diagram shows each of them transitioning back to the idle state. This is important because our device is meant to be continuously used and store/rewrite additional data to be printed out as a label. Alternatively, the device can also be used to heat up a milk bag completely independent of the other functions, but should still be able to freely transition between all functions of the device.

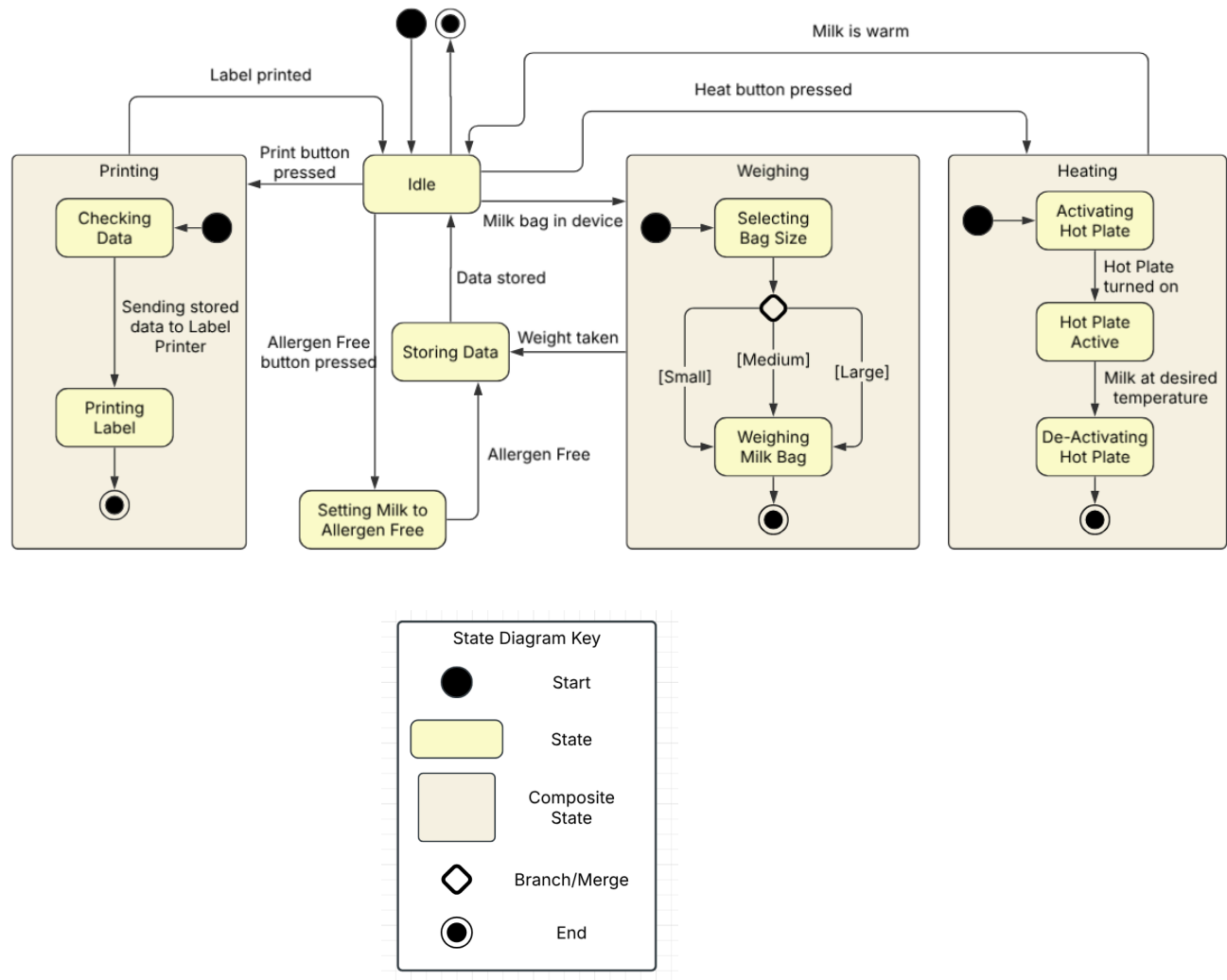
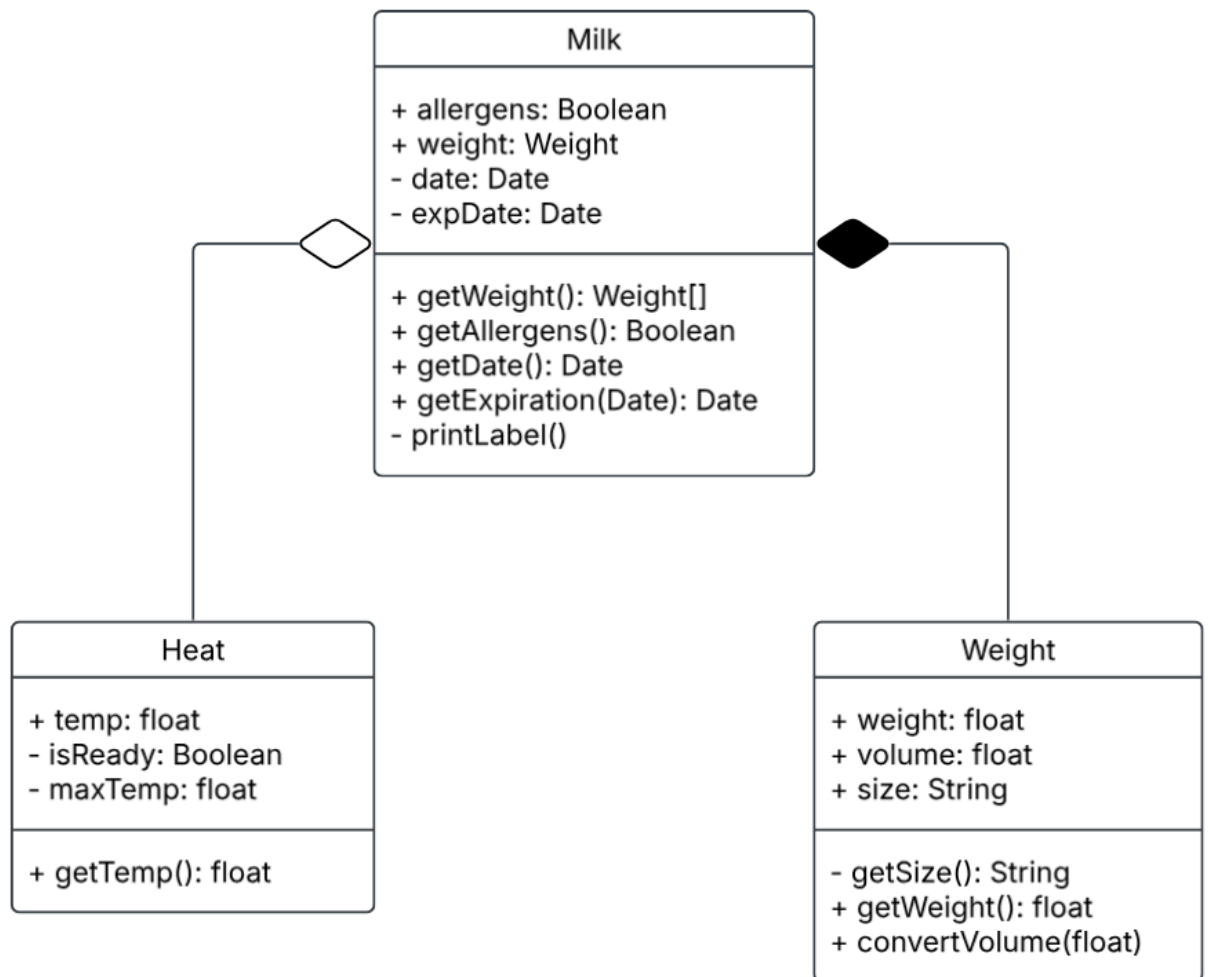


Figure 7.2 - Software State Diagram

7.3: Class Diagram

The class diagram is a UML (Unified Modeling Language) diagram that provides a visual representation of the structure of a system and the relationships between its classes, attributes and methods. Unlike the previous UML diagrams, the class diagram provides an overview of the object-oriented system design and can be used as a representation of the software development for our system. Our class diagram incorporates three classes that make up our system, with the Milk class being the whole (container) class composed of the other two part (contained) classes. There is a difference between the two part classes, with the Weight class representing a composition relationship with the Milk class and being unable to exist independently of its whole class. However, the Heat class represents an aggregation relationship with the Milk class and is able to exist independently of its whole class.



Chapter 8 - System Fabrication/Prototype Construction:

Chapter 9 - System Testing & Evaluation:

Chapter 10 - Administrative Content:

10.1: Budget Estimates

Table 10.1 - Bill of Materials

<u>Component</u>	<u>Estimated Cost (USD)</u>
Microcontroller (esp 32, MSP430, RaspberryPi, Arduino)	\$10 - \$30
ADC Module(If not embedded into MCU)	\$10 – \$15
Scale Platform / Mechanical Parts	\$10 - \$20
LCD Display	\$10 – \$20
Thermal Label Printer	\$20 - \$50
Bluetooth Transceiver (If not included in MCU)	\$10 - \$20
Power Supply	\$15 - \$40
PCB & Prototype Materials	\$80 - \$200
Misc. (cables, connectors, adhesives, buttons)	\$10 - \$20
Temperature Sensor	\$4 - \$30
Heater	\$20 - \$50

Building parts(MechE)	\$40 - \$100
Total (Approx.)	\$244 - \$595

10.2: Project Milestones

Weeks 1–2 (Jan 6 – Jan 19)

- Requirements Gathering & Research
 - Finalize project scope with sponsor/mentor.
 - Conduct feasibility analysis for hardware (scale, printer, LCD, etc.) and mobile app integration.
 - Develop initial problem statements and identify key user needs.

Weeks 3–5 (Jan 20 – Feb 9)

- System-Level Design
 - Create and review the hardware block diagram and software flowchart.
 - Draft House of Quality to correlate customer requirements with engineering specs.
 - Establish measurable specs (weight accuracy, label printing speed, etc.).
 - Research Power Supply.

Weeks 6–7 (Feb 10 – Feb 23)

- Component Selection & Ordering
 - Research and select load cell, ADC module, microcontroller, label printer, heater, and LCD screen.
 - Confirm and design power requirements .
 - Order all long-lead-time components to ensure timely delivery.

Weeks 8–10 (Feb 24 – Mar 16)

- Prototype Assembly (Hardware)
 - Assemble and mount the load cell onto a stable platform.
 - Mount heater to the load cell and insulate accordingly.
 - Begin integrating microcontroller, printer, and LCD.
 - Perform initial tests for power distribution and device safety.

Weeks 11–13 (Mar 17 – Apr 6)

- Firmware & Software Development (Phase I)
 - Write code for reading the load cell (via ADC), converting weight to weight, and displaying values on the LCD.
 - Integrate Real-Time Clock (RTC) or system time to record date/time.
 - Begin label printing code and basic user interface logic.

Weeks 14–16 (Apr 7 – Apr 27)

- Firmware & Software Development (Phase II)
 - Refine user interaction (button press to print, error handling, calibration routines).
 - Optimize label formatting and printing speed.
 - Conduct intermediate testing (accuracy, response time, label clarity).

Weeks 17–19 (Apr 28 – May 18)

- Mobile App Integration (Phase I)
 - Set up basic iOS app structure (user accounts, data model).
 - Implement Bluetooth or Wi-Fi communication to sync weight/weight data.
 - Test data transfer for speed and reliability.

Weeks 20–22 (May 19 – Jun 8)

- Mobile App Integration (Phase II)
 - Implement advanced app features (inventory tracking, expiration date alerts, portioning).
 - Conduct usability testing with mock data.
 - Ensure synchronization logic handles offline mode (caching measurements on device).

Weeks 23–25 (Jun 9 – Jun 29)

- System-Level Testing & Validation
 - Perform end-to-end tests (scale -> label -> app).
 - Perform additional tests on the temperature sensor and heater.
 - Validate accuracy, label content, and data consistency.
 - Gather feedback from peers or target users; iterate as needed.

Weeks 26–27 (Jun 30 – Jul 13)

- Final Adjustments & Optimization
 - Refine mechanical design (enclosure, cable management, ease of cleaning).
 - Optimize firmware/app for performance.
 - Prepare documentation for final demonstration (user manual, technical specs).

Weeks 28–29 (Jul 14 – Jul 31)

- Final Presentation & Demonstration
 - Complete final project report, design website, and any required submission materials.
 - Conduct a polished demonstration of the working device (hardware + mobile app).
 - Receive and integrate final feedback from the sponsor and review committee.

Chapter 11 - Conclusion:

Appendix A - References:

[1] ExpertAppDevs. “Swift vs Flutter – Which Is Better for Your Next iOS App?” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[2] Flatirons. “Flutter vs. Swift: Comparison in 2025.” Flatirons, 2025, <https://flatirons.com>. Accessed 26 Feb. 2025.

[3] Developer Nation. “Developer Nation Community.” DeveloperNation.net, 2025, <https://developernation.net>. Accessed 26 Feb. 2025.

[4] ExpertAppDevs. “Swift vs Flutter – Which Is Better for Your Next iOS App? (Additional Comparisons).” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[5] Flatirons. “Flutter vs. Swift: Comparison in 2025 (Performance Metrics).” Flatirons, 2025, <https://flatirons.com>. Accessed 26 Feb. 2025.

[6] Flatirons. “Flutter vs. Swift: Comparison in 2025 (Animation Tests).” Flatirons, 2025, <https://flatirons.com>. Accessed 26 Feb. 2025.

[7] ExpertAppDevs. “Swift vs Flutter – Build Time, UI/UX Differences.” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[8] ExpertAppDevs. “Swift vs Flutter – Widget System vs. Native UI Components.” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[9] ExpertAppDevs. “Swift vs Flutter – Time-to-Market and Performance Notes.” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[10] ExpertAppDevs. “Swift vs Flutter – Single Platform vs. Cross-Platform Trade-offs.” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[11] Developer Nation. “Dart as the Fastest-Growing Language Community in 2023.” DeveloperNation.net, 2025, <https://developernation.net>. Accessed 26 Feb. 2025.

[12] ExpertAppDevs. “Swift vs Flutter – App Size Considerations.” ExpertAppDevs, 2025, <https://www.expertappdevs.com>. Accessed 26 Feb. 2025.

[13] U.S. Food and Drug Administration. “FDA Mission.” U.S. Food and Drug Administration, 8 Mar. 2024, www.fda.gov/about-fda/what-we-do#:~:text=Information%20for%20Consumers-,FDA%20Mission,and%20products%20that%20emit%20radiation. Accessed 15 Mar. 2025.

[14] U.S. Food and Drug Administration. Guidance for Industry: Food Labeling Guide. U.S. Food and Drug Administration, 14 Jan. 2020, www.fda.gov/regulatory-information/search-fda-guidance-documents/guidance-industry-food-labeling-guide. Accessed 15 Mar. 2025.

- [15] U.S. Food and Drug Administration. Once Baby Arrives: Food Safety for Moms-to-Be. U.S. Food and Drug Administration, www.fda.gov/food/people-risk-foodborne-illness/once-baby-arrives-food-safety-moms-be. Accessed 15 Mar. 2025.
- [16] Centers for Disease Control and Prevention. Proper Storage and Preparation of Breast Milk. Centers for Disease Control and Prevention, 25 Apr. 2023, www.cdc.gov/breastfeeding/breast-milk-preparation-and-storage/handling-breastmilk.html. Accessed 15 Mar. 2025.
- [17] EspressifSystems. “ESP32 Series Datasheet.” Espressif.com, 2023, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Accessed 17 Mar. 2025.
- [18] Espressif Systems. “ESP-IDF Programming Guide: Bluetooth Classic and BLE Examples.” Github.com, 2023, github.com/espressif/esp-idf/tree/master/examples/bluetooth. Accessed 17 Mar. 2025.
- [19] Nordic Semiconductor. “Bluetooth Low Energy Fundamentals and Security Overview.” Nordicsemi.com, 2023, www.nordicsemi.com/Products/Low-power-short-range-wireless/Bluetooth-Low-Energy. Accessed 17 Mar. 2025.
- [20] ZXing Contributors. “ZXing (Zebra Crossing) Barcode/QR Scanning Library.” Github.com, 2023, github.com/zxing/zxing. Accessed 17 Mar. 2025.
- [21] Google. “CameraX and ML Kit: Building a Barcode and QR Code Scanner.” Developers.android.com, 2023, developers.android.com/codelabs/camerax-mlkit-barcode-scanning. Accessed 17 Mar. 2025.
- [22] Nielsen Norman Group. “The Impact of Scanning-Based User Interfaces on User Experience.” Nngroup.com, 2022, www.nngroup.com/articles/scannable-interfaces. Accessed 17 Mar. 2025.
- [23] University College London. “Security Considerations for IoT and Home Devices.” Ucl.ac.uk, 2021, discovery.ucl.ac.uk/id/eprint/10117141. Accessed 17 Mar. 2025.

Appendix B - Copyright Permission:

Appendix C - Data Sheet:

Appendix D - Software Code:

Appendix E - ChatGPT Prompts & Outcomes: