ECE 445
SENIOR DESIGN LABORATORY
DESIGN DOCUMENT

Secure Food Delivery Dropbox

Team No. 64
Rohan Samudrala
(rohans11@illinois.edu)
Dhruva Dammanna
(dhruvad2@illinois.edu)
Taniah Napier
(tnapier2@illinois.edu)

TA: Chi Zhang
Professor: Michael Oelze
May 7, 2025

Abstract

This document focuses on designing a Secure Food Delivery Dropbox that provides a safe and reliable holding area for food delivery. By integrating a robust authentication and heating subsystem, the device is easy for drivers to access while providing constant information to the user about their order status from a secure location.
The following document provides and explains the logic, details, and diagrams involved in designing, implementing, and verifying the Secure Food Delivery Dropbox. In addition, this document contains information about the parts used, a cost analysis, a weekly schedule of completion, and an examination of the safety & ethics for our project.

Table of Contents

1. Introduction

1.1 Purpose

About 70% of college students order food from third-party delivery platforms such as Uber Eats and Doordash[1]. The problem with using these delivery services is that upon arrival, the food order is left outside and is susceptible to theft, getting cold, or getting damaged, but the students might be able to retrieve their food immediately due to being busy with classes or working or might have possibly ordered ahead to have food ready when they get home. In 2024, there were approximately 311.1 million users of the online food and grocery delivery market in the United States alone, and this number is projected to continuously rise.[2]

To address this issue, we designed a secure food delivery dropbox that will keep the food safe, and if necessary, warm. The box will remain locked until the driver opens the box by using a one time use keycode which is provided by the user. The user has access to an app that will show whether the food is inside the box. The user can also turn on heating inside the box to keep their food warm and monitor the temperature inside the box through the app. Once the user is ready to retrieve their food, they must enter a master keycode, that they can set via the website, and scan a registered RFID tag to open the box.

1.2 Functionality

For our design we had 3 high level requirements that our product should meet in order to be considered a complete success.

1. Authentication: Authentication should initiate unlocking the box with correct inputs and keep the box locked with incorrect inputs. Specifically, authentication using the temporary code on the keypad should unlock the lock for 30 seconds 100% of the time, and authentication using the RFID sensor should require the master keycode to be entered within 30 seconds and unlock the lock 100% of the time when both are correct.
2. Box Mechanism: The load cell in the box should detect the presence of an object of at least 40 grams and should indicate on the website that there is either no food or there is food in the box within 20 seconds of the object being placed or removed 100% of the time.
3. Code generation: A master keypad code should only be generated when the user decides to change the code on the website, and only the latest master code generated should work. The master code should work within 30 seconds of making a new master code. Temporary keypad codes should be generated when the previous temporary code was used, and the old temporary code should not work. The temporary code should be sent to the user within 30 seconds of generation.

The first requirement is important to the overall project because it highlights the process of opening the box and making sure food can be placed inside and retrieved safely and securely. The second requirement ensures that the box relays the presents of food inside the box to the user, further establishing the security aspect of the box. The third requirement is important to ensure that the driver or other people can not reuse a one time use keycode to open the box and take the food inside the box unauthorized.

1.3 Block Diagram

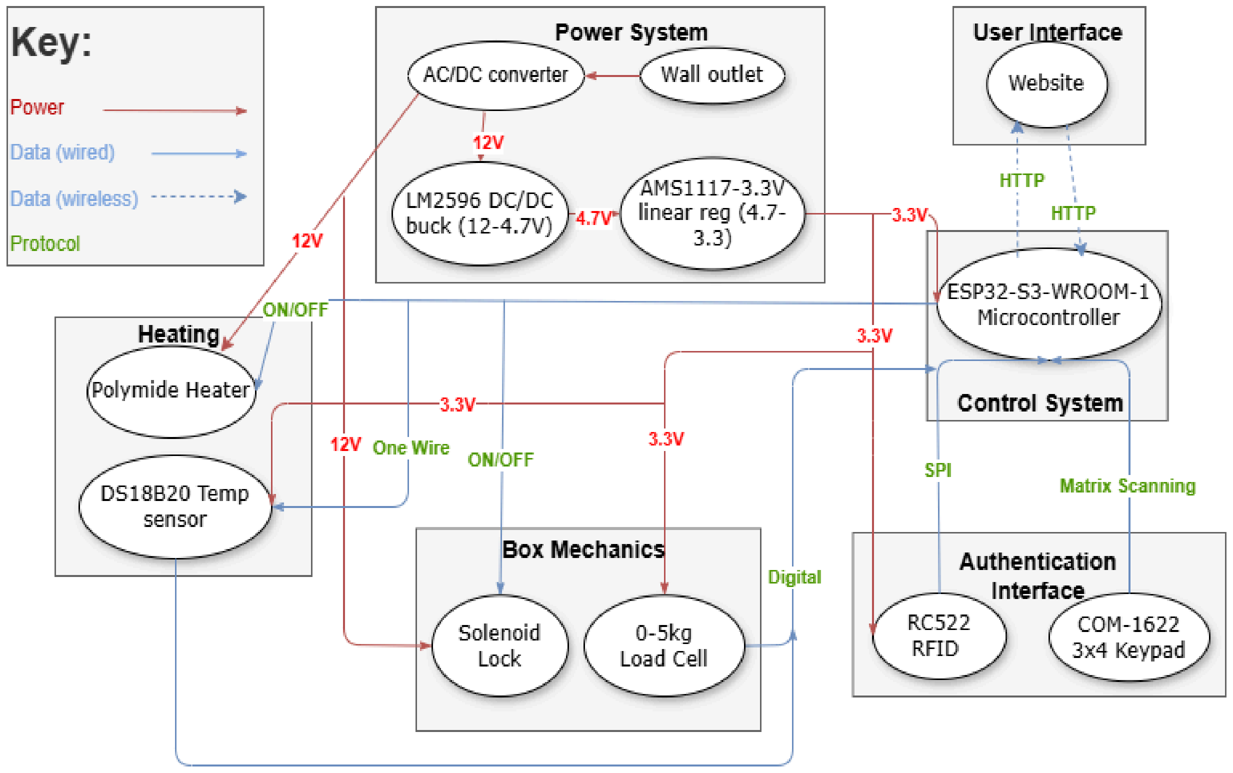


Figure 1: Block Diagram for Secure Delivery Dropbox

To describe the Block Diagram shown in Figure 1, the power subsystem will provide 12V to the heater and the solenoid lock and then 3.3V to everything else. The control subsystem will communicate with the user interface subsystem via HTTP and perform all of the logic of the project. The heating subsystem will read the temperature and toggle on and off the heater. The box mechanics subsystem will lock/unlock the box as well as read if there is food on the scale. The authentication subsystem has a one time use temporary code for the delivery driver to use as well as a 2 factor authentication system with an RFID tag and a master keycode for the user to use. More details about the subsystems and how they work will be provided in the design section.

2. Design

2.1 Design Details

Our design process involved splitting the box into multiple subsystems as described in the block diagram. By splitting up the PCB into simpler sections, it was much easier to digest each subsystem and then put it all together at the end. We will now go into more detail on each subsystem as well as the design of the box.

2.1.1 Power subsystem

The power subsystem provides the needed power to every component in the PCB. The power subsystem has 3 parts to it: an AC to 12V DC wall converter, an LM2596-4.7V[3] DC to DC Buck Converter, and 2 AMS1117-3.3V[4] Linear Regulators, one for the ESP32 and one for everything else using 3.3V. The solenoid lock and heater require 12V, and everything else on the PCB requires 3.3V. The AC to 12V DC wall converter provides power to the solenoid lock and heater, and the AMS1117-3.3V Linear Regulator provides power to everything else. The reason why the LM2596-4.7V DC to DC Buck Converter is used is the step down from 12V to 4.7V for the AMS1117-3.3V. A linear regulator burns off excess voltage as heat, so going from 12V down to 3.3V will create much more heat than going from 4.7V down to 3.3V. The reason why 4.7V was chosen is that the AMS1117-3.3V has a dropout voltage of 1.1V, meaning that to get a voltage of 3.3V consistently, a voltage of 3.3V + 1.1V = 4.4V or greater must be input. 4.7V fits this need and allows the AMS1117-3.3V to deliver 3.3V consistently. The max current drawn from everything using 3.3V is about **~337mA**, and for everything it is about **~1.83A**. The current draw and voltage for each component are as follows:

ESP-32-S3-WROOM-1 N4R2: 300mA, 3.3V
DSB1280: 3mA, 3.3V
RC522 RFID Reader: 30mA, 3.3V
3X4 Keypad: 1mA, 3.3V
HX711: 3mA, 3.3V
Solenoid Lock: 500mA, 12V
Heater: 1A, 12V

The equation

$$P_{heat} = (V_{in} - V_{out}) * I_{load} \qquad (1)$$

can be used to calculate how much power is being converted to heat by the AMS1117-3.3V Linear Regulator when 12V is inputted versus when 4.7V is inputted. Below are equations comparing the heat generated for each scenario

$$P_{heat} = (12V - 3.3V) * 337mA = \sim\mathbf{2.93\ W}$$

$$P_{heat} = (4.7V - 3.3V) * 337mA = \sim\mathbf{0.47\ W}$$

Dropping from 12V down to 3.3V generates **~2.36W** of extra power being converted to heat when compared to dropping from 4.7V down to 3.3V.

Another option would have been to drop the voltage from 12V down to 3.3V directly using the DC to DC Buck Converter. DC to DC Buck converters do not generate nearly as much heat as a linear regulator. The only problem with this is that DC to DC Buck Converters create a lot of switching noise. In our case, the LM2596 has a switching noise of **~150 kHz**. The switching noise would interfere with the ESP32's WiFi communication. Using a linear regulator will filter out the noise from the DC to DC Buck Converter, allowing the WiFi to work on the ESP32.

Another design choice made here was to have one AMS1117-3.3V Linear Regulator for the ESP32 and one for everything using 3.3V. The AMS1117-3.3V is capable of drawing at most 800mA, but it is recommended to have a separate regulator for the ESP32 because sharing it among other components using 3.3V can cause the noise from those to interfere with the WiFi of the ESP32.

The AMS1117-3.3V and LM2596 were implemented on the PCB using breakout boards, this is because we did not know if we could get the circuits that we developed to work as intended. With this design choice, we were able to switch between breakout boards we bought and tested that worked and breakout boards that we designed ourselves. The breakout boards that we developed were not outputting the correct voltages; the AMS1117-3.3V was outputting 2.7V consistently, and the LM2596 was outputting 5.7V consistently. Because of this, we decided to use breakout boards that we bought, as they worked as intended. Voltage is never 100% stable, so to account for this, we added a tolerance analysis for the 12V of ± 0.3V and for the 3.3V of ± 0.1V. Using the multimeter, we were able to verify that our power subsystem satisfied these constraints.

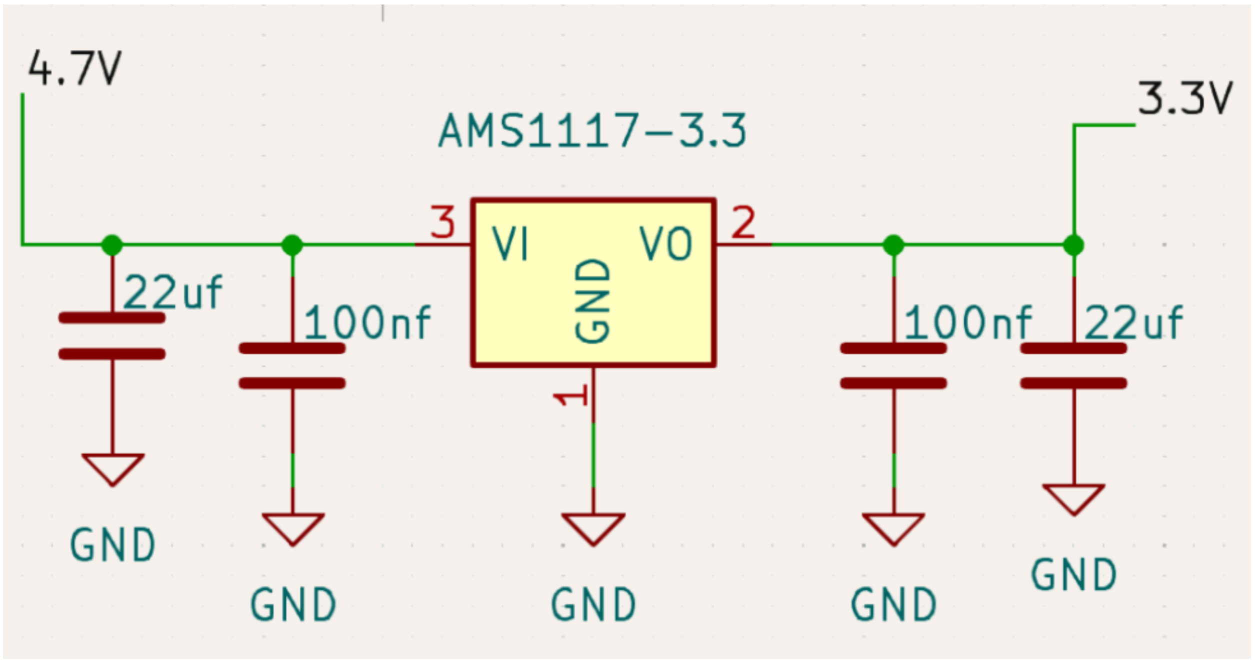


Figure 2: AMS1117-3.3V Breakout Board Schematic

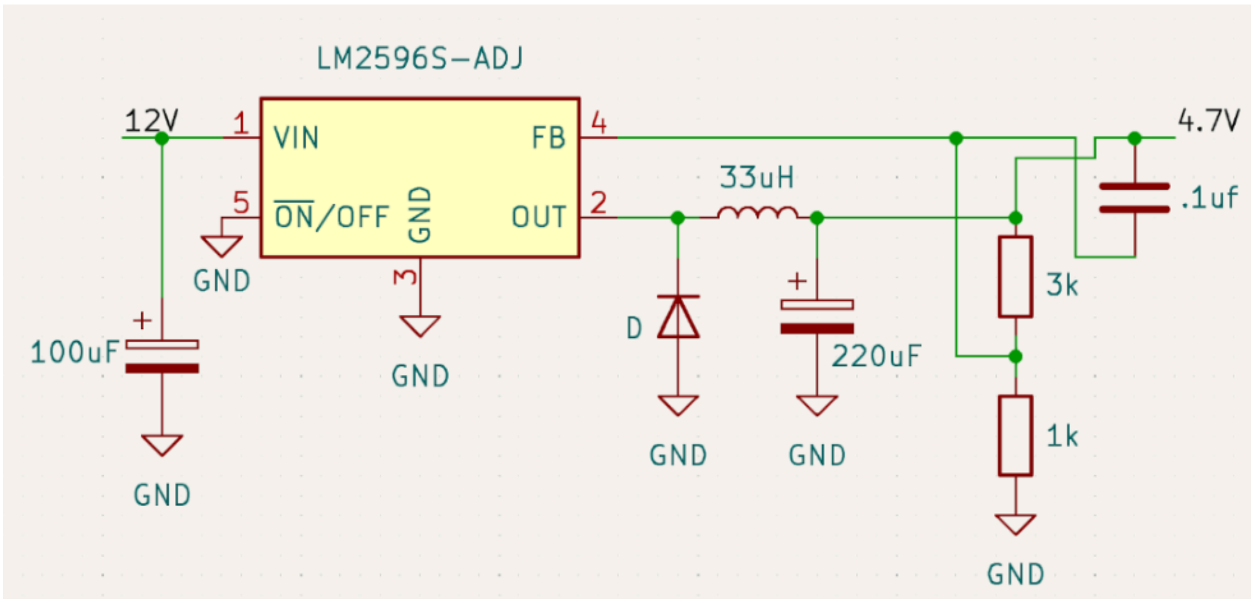


Figure 3: LM2596 Breakout Board Schematic

2.1.2 Control subsystem

The control subsystem contains the ESP-32-S3-WROOM-1-N4R2[5] and the UART circuit to program the ESP-32-S3-WROOM-1-N4R2. It takes in 3.3V and has its own linear voltage regulator. The control subsystem is responsible for all program logic, which includes sensor data processing, keypad and RFID input handling, hosting the web server, toggling the heater, locking/unlocking the solenoid lock, and keycode generation. We chose the ESP32 because of its built-in WiFi, which enabled remote access through our website, as well as the large number of GPIO pins to support all of our components. The ESP-32-S3-WROOM-1-N4R2 also has a 240 MHz dual-core processor and 4 MB of flash memory, providing our project with the processing power it needs.

Our original design was to program the ESP32 by routing a micro-USB's D+ and D− traces directly to GPIO 19 and 20, bypassing the need for a USB-UART chip. The issue with this was that the length of D+ and D- traces needed to be within 1 mm of each other to work properly, which was very difficult to achieve in our PCB with all the other traces we had. Also, another problem with using a micro-USB's D+ and D− traces was that they needed to have extra ESD protection; without it, static discharge from human touch could potentially fry the ESP32.

After realizing that using a micro-USB to program the ESP32 was not ideal, we decided to use an external USB-UART bridge. The USB-UART bridge allowed us to easily program the ESP32 by just exposing the TX, RX, 3V3, GND, IO0, and EN pins and plugging in our USB-UART bridge. The USB-UART bridge that we used was the ESP PROG[6]. It is a USB-UART bridge that is built for the ESP 32, so all we had to do was just plug in the pins of the ESP PROG to the corresponding ESP 32 pins. The ESP PROG uses the FTDI UART communication protocol, so we had to install the drivers on our computer to be able to program the ESP32 using the ESP PROG.

Once we set the way to program the ESP32 we then set up the coding environment in Arduino IDE. We used the Arduino programming language to program the ESP32 as well as the Serial Monitor in Arduino IDE to debug our code.

The ESP32 is capable of using different communication protocols, which we used to communicate with our components. Below are the communication protocols used for everything connected to a GPIO pin of the ESP32:

On/Off: Heater, Solenoid Lock
SPI: RC522 RFID Reader
Matrix Scanning: 3X4 Keypad
Digital Input: HX711
One Wire: DSB1280

With everything connected and the ESP 32 being able to be programmed, we then wrote the logic of the program, which can be seen in Appendix D, Figure 11 as a flow chart. To get data from the sensors, we used different polling rates that we found would best fit the project: the load cell was polled every 5 seconds, the temperature sensor every 15 seconds, the RFID reader every 100 ms, and the keypad every 10 ms. This approach allowed the box to function as intended without overloading the ESP32 with excessive polling. Our control subsystem was able to run our entire program and was able to be programmed, showing that it worked as intended.

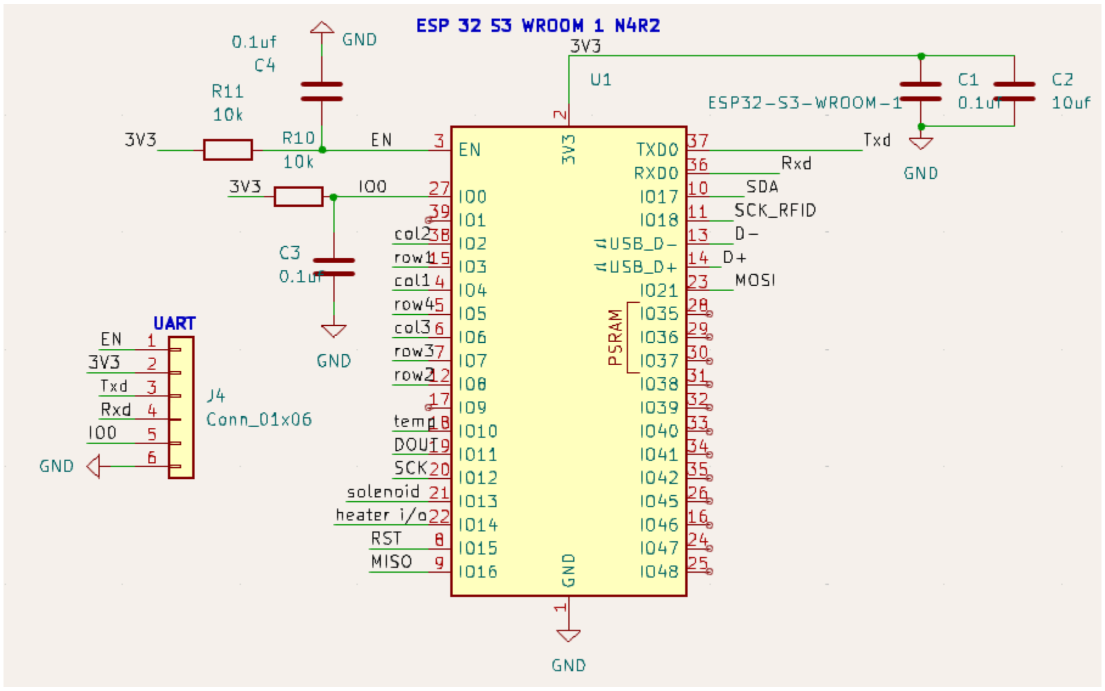


Figure 4. Control subsystem schematic

2.1.3 Authentication subsystem

The authentication subsystem is responsible for verifying if the user or delivery driver can unlock the box. The authentication subsystem contains a 3×4 keypad[7] and an RC522 RFID[8] reader. It ensures that the box is secure and does not allow unauthorized access. The driver will receive a one-time use temporary keycode to unlock the box. The temporary keycode can only be used once to ensure that the driver can not open the box multiple times. Then the user can open the box using their master keycode they set on the website, and the RFID tag given with the box. Both the master keycode and the RFID tag are needed to open the box, as 2-factor authentication was implemented to increase the security of the box. The RFID tag and master keycode can be entered in any order, but they must be entered within 30 seconds of each other.

The first part of the Authentication subsystem is the COM-1622 3x4 keypad[7]. The keypad consists of a matrix of wires and switches, with 7 connection pins, one for each row and column. When a key is pressed, it pushes a switch connecting the row and column. For example pushing number 1 connects the row 1 and column 1 traces, allowing current from the ESP32 to flow through. This matrix scanning communication method sends data to the ESP32, which can then parse that signal as a key press.
Our keypad setup stayed the same throughout the whole process. We tested that it worked using an ESP32 dev board on a breadboard, so we were confident it would work on the PCB as it did.

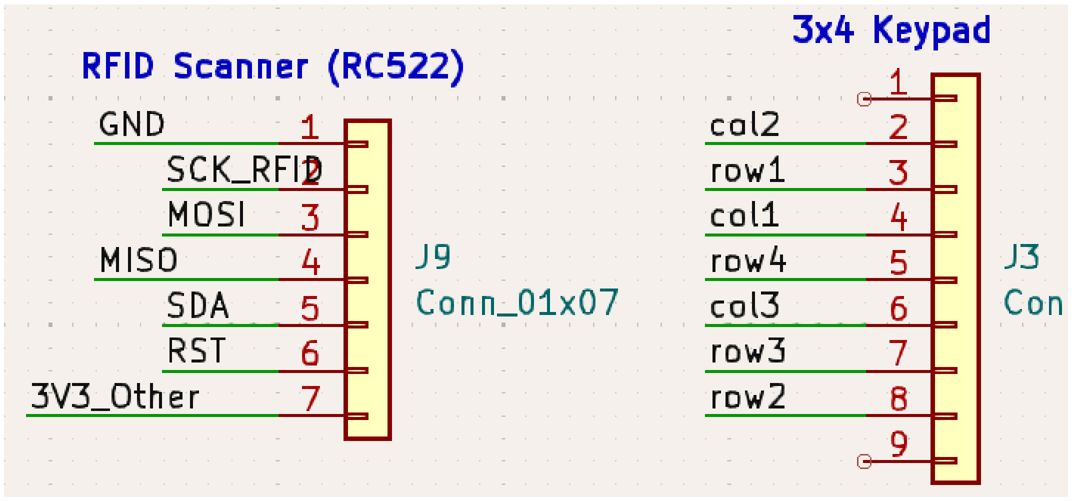


Figure 5. Authentication subsystem schematic

The second part of the Authentication subsystem is the RC522 RFID[8] scanner, which takes 3.3V. Each RFID tag has a unique ID that we could authorize in our code. When the RFID tag with an authorized ID approaches the scanner, the scanner uses radio signals to detect the tag and send the data in the serial data line (SDA). The scanner communicates with the microcontroller with the master out slave in (MOSI) and master in slave out (MISO) lines as seen in Figure 5. The scanner also has a serial clock input (SCK) from the ESP32. This communication format is known as Serial Peripheral Interface (SPI). With this setup, the ESP32 can recognize all authorized RFID tags and decline any unauthorized tags.

Initially, this subsystem was going to use a fingerprint sensor instead of the RFID reader. This was later changed to use an RFID sensor, as fingerprint sensors can sometimes reject a correct fingerprint and take many more false positives and negatives, which was an issue in our testing.

2.1.4 Box Mechanics subsystem

The Box Mechanics subsystem consists of the Lock-style Solenoid and the 0-5kg load cell. The lock-style solenoid [9] is powered by the 12V wall outlet. The electromagnets inside the lock, which move the mechanism, take a lot of current (around 500mA), so the wall outlet is used directly for power. When current is applied to the lock, the inductor inside the lock turns on a magnetic field, pulling back the deadbolt. We control the lock using a GPIO from the ESP32 and a BJT. When the lock needs to be activated, the control system sets the solenoid lock GPIO to high. This activates a 3.3V data line that heads into the base of a TIP 120[10] BJT through a resistor, closing the circuit from the power source to the lock. This was part of our original design, which we were able to test on a breadboard and then convert to the PCB. One change we made was adding a flyback diode in parallel to the lock so that no reverse kickback voltage could flow when the lock was suddenly deactivated, which could lead to sparking or burnout of the part.

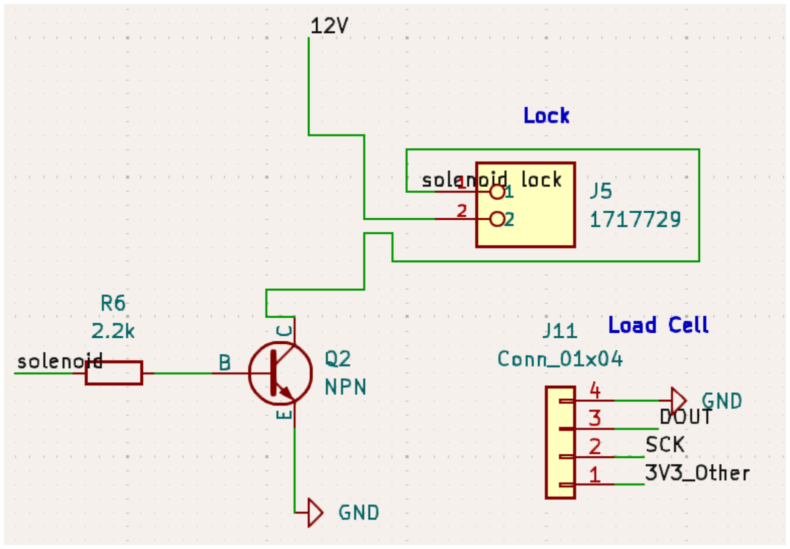


Figure 6. Box Mechanics subsystem schematic

The other portion of the box mechanics was the 0-5kg load cell [11]. The load cell is placed at the bottom of the box underneath a plate so that when food is placed on the plate, the load cell can detect food. The load cell works by a strain gauge placed on top of the thin metal bar. When a load is placed on the plate, the metal bar undergoes some strain, slightly stretching it and changing the resistance of the circuit placed above it. This change in resistance corresponds to a change in current from the 3.3V input, which can be translated to a weight measurement. We fed the output of the load cell into an HX711 analog-to-digital amplifier [12] circuit to amplify the signal. The HX711 will then output a Data Out and an SCK signal to the GPIO pins of the ESP32 for processing. Once we had the circuit set up, we tested the output with known weights and modified a calibration factor in the code to get a more accurate weight in grams. We found this value to be around -400 from our setup and testing. The outputs from our testing, once calibrated, can be seen in Figure 7.

We made many changes to the load cell from our original design. Originally, we planned to use a 0-500g load cell for its greater specificity, which fit our expected food range. However, we switched to a 0-5kg load cell for its bigger physical size in the box and wider range of detectable weights. Since our design implementation only mattered if the weight was above a certain threshold, the greater resolution in weight outputs mattered less, and a larger range was preferred. Another change we made was in the amplifier. We originally used a system involving an INA125PA chip to amplify the signal, but we then changed it to an HX711. The change was made since we found more info on the setup of the HX711 chip, as it was the recommended chip for our model of load cell.

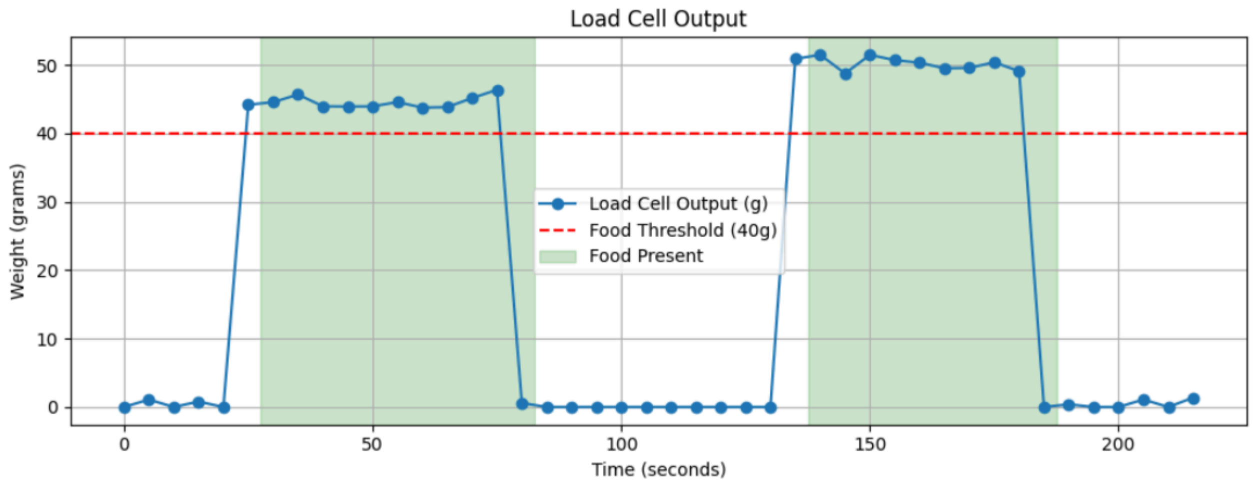


Figure 7. Load Cell reading with food present and absent

2.1.5 Heating subsystem

The heating subsystem consists of the DS18B20 temperature sensor and the polyimide heater. The DS18B20 temperature sensor[13]takes a 3.3V input and outputs the temperature back to the ESP32 through a one wire protocol. The temperature sensor outputs an accurate temperature reading within 2℃ of the thermometer at all times, often much closer when the temperature stabilizes. The temperature sensor gave the ESP32 the data needed to send the user the box temperature and allowed it to turn on and off the heater at a given temperature, unlike just using a thermometer.

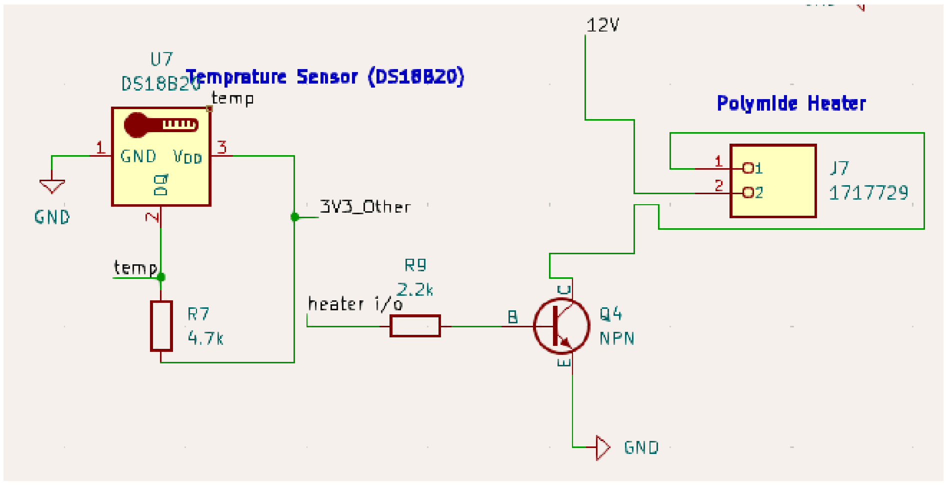


Figure 8. Heating subsystem schematic

The other part of the heating subsystem is the heater itself. The polyimide heater[14] receives 12V, 1A directly from the wall outlet for the maximum power input for the greatest maximum heating option. The heater is controlled by a TIP120 BJT [10] where user input toggles an ESP32 GPIO which feeds into the base of the BJT, either allowing the 12V to pass to the heater or not. To ensure that the heater never got too hot we installed a failsafe protocol in the code. If the heater ever got over our benchmark of 32℃ (90F), the heater would automatically shut off by removing base current from the ESP32. The user's control over the heater would be temporarily disabled until the temperature at least reached 31℃ again. This ensures that the heater can never get too hot to burn things placed inside or cook any food placed inside. Our testing to determine the functionality of this failsafe can be seen in Figure 9.

One change we made to the heating subsystem from our original design was the change from a nichrome coil. Originally, we planned to create a coil using nichrome wire as a heater for the box. However, safety concerns led us to buy our own premade polyimide heaters with their insulation. The setup still worked with a BJT, but having an exposed warm coil people could brush their hands against wasn't a safe solution.

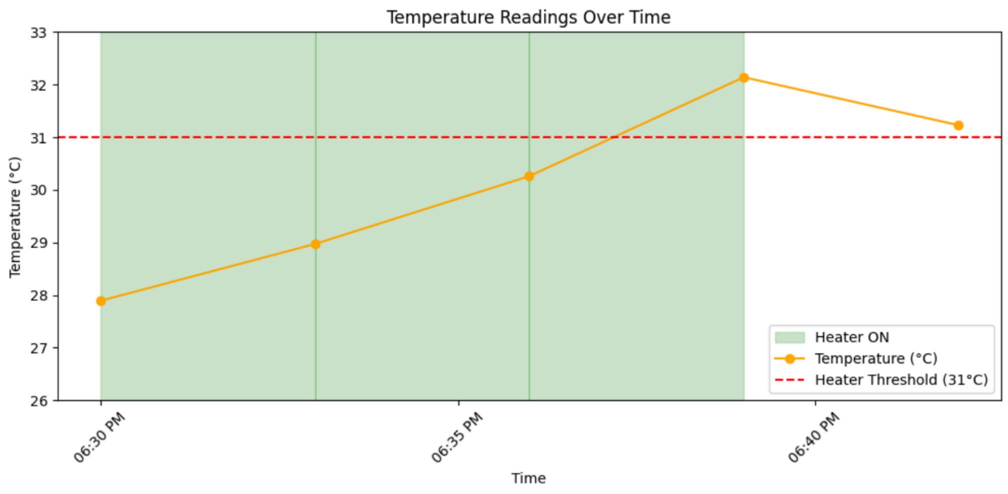
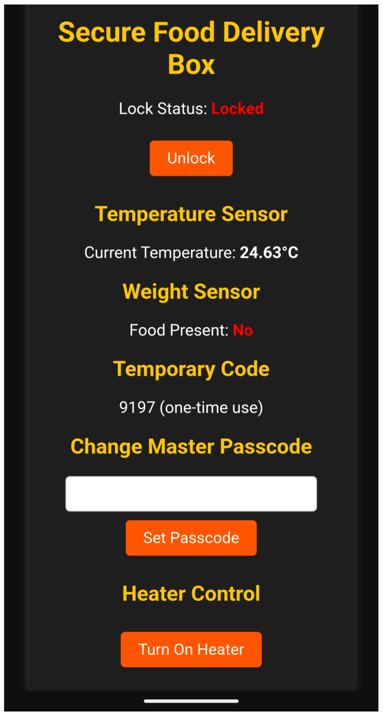


Figure 9. Temperature readings testing heater failsafe

2.1.6 User interface subsystem

The user interface subsystem is what the user sees when they use the website. It is designed to provide the user with a seamless experience when using the secure food delivery box.

Originally, the user interface was going to be a computer application that the user could use to interact with the box. The issue we found with this was that many people who use food delivery services like Uber Eats and DoorDash use it on their phones. So, in order to make the project more user-friendly, we decided to create a website instead so that users can use it on their phone, and if they want to, they can still use their computer. The user interface gives the user all the control they need over the box and updates them on the status of the box.


Secure Food Delivery Box

Lock Status: Locked

Unlock

Temperature Sensor

Current Temperature: **24.63°C**

Weight Sensor

Food Present: No

Temporary Code

9197 (one-time use)

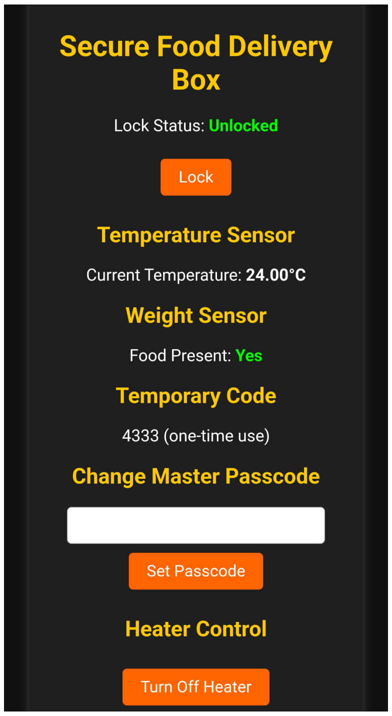
Change Master Passcode

Set Passcode

Heater Control

Turn On Heater

Figure 10.1. User Interface on Boot


Secure Food Delivery Box

Lock Status: Unlocked

Lock

Temperature Sensor

Current Temperature: **24.00°C**

Weight Sensor

Food Present: Yes

Temporary Code

4333 (one-time use)

Change Master Passcode

Set Passcode

Heater Control

Turn Off Heater

Figure 10.2 User Interface with unlock, food, heater

2.1.7 Physical Box Design

To create the actual box, we went to the ECEB Machine Shop for help. Initially, we planned to use a crate with a lid that opens up and down. After a discussion with the machine shop, we decided to use a locker for the box and then put a box on top of the locker for the PCB and all the sensors to connect to. Using a locker is much more reasonable for this project, as it makes everything compact and not bulky when compared to using a crate. The locker had some built-in holes which we used to stick the heater and temperature sensor through. The load cell was placed on the bottom of the locker with a thin aluminum plate on top of it so that weight could be placed on the load cell. The load cell wires were wired through the inside of the box in the corner and then went through a hole to the outside of the box to connect to the PCB. The same setup was done for the solenoid lock, except the lock was mounted on the latch side so that it could properly lock and unlock the box. The 3X4 keypad was mounted on top of the box, and the RFID sensor was hanging outside with wires outside the door. The RFID was meant to be mounted with the keypad on top of the box, but we ran into an issue where the Electromagnetic Interference from the PCB was interfering with the RFID reader's ability to read RFID tags. To fix this, we had to move the RFID sensor so that it was not near the PCB. Images of our box can be found in Appendix D.

3. Cost and Schedule

3.1 Cost Analysis

3.1.1 Labor estimate:

According to [15], the average yearly salary of a graduate electrical engineer from UIUC is \$87,769. From an average work week of 40 hours/week, this works out to almost \$50 per hour. We expect to work an average of 15 hours per week for 10 weeks or a total of 150 hours per person on this project.

$$\text{Labor cost per person} = \$50/\text{hour} * 2.5 * 150 \text{ hours} = \$18{,}750 \qquad (2)$$

Total Labor cost = \$18,750 * 3 people = \$56,250

3.1.2 Parts estimate:

All capacitors, resistors, BJTs, and screw terminals used in the project were acquired from the ECE shop for free. In addition, the box itself was made by the ECE machine shop for free as well. The cost of individual parts needed for the project can be seen in Appendix B for a total of \$117.53.

Total Parts cost = **\$117.53**

3.1.2 Total cost estimate:

The total cost of our project, including both the cost of project components and labor, amounts to:

$$Total\ Cost = C_{parts} + C_{labor} \tag{3}$$

Total Cost = \$117.53 + \$56,250 = \$56,367.53

The project cost \$56,367.53 in total.

3.2 Schedule

Overall, we split the work between us relatively evenly. A detailed breakdown of the week by week schedule is present in Appendix C.

4. Conclusion

4.1 Accomplishments, Future Work, and Changes

The final product of our design was fully functional and passed all of our requirements and verification tests. Our box was able to be opened with a temporary code, then a new temporary code was randomly generated immediately after. Once food was placed on the scale inside the box the website updated to correctly show the presence of food inside the box. The user was also able to turn on and off a heating element inside the box through the website and see the temperature reading from a temperature sensor inside the box on the website. Finally, the user was able to set and change a master keycode via the website and use the master keycode and the registered RFID tag to open the box. Some aspects of the final project we would have changed and might add in the future are adding insulation so that our box retains heat better and faster, adding a digital display to showcase the keycodes being inputted and visual confirmation of our RFID scan was accepted. Alternatively, we could design a version of our box with a cooling system rather than a heating system.

4.2 Ethical Considerations

Use of Open Source Projects:

The work that we did used a lot of open source libraries which were HX711 Libraries[16], MFRC522 Libraries[17], Adafruit Keypad Libraries[18], and Dallas Temperature Libraries[19]. These helped us interface with the peripheral devices that we used in the project. in accordance with the ACM code of ethics 1.5[20]. It is important to recognize those who have helped us make this project by giving them credit.

Safety:

]We used voltage regulators in this project to ensure everything gets the correct voltage thus making sure nothing gets fried on the board and the PCB and box do not overheat . We also used lab safety equipment when appropriate to uphold maximum safety standards such as multimeters, temperature sensors, and exhaust fans. The heating subsystem must not overheat to dangerous levels, so we implemented safeguards in our code to ensure the heater turns off at 32°C (90°F). Also, rigorous safety testing of all subsystems was done to ensure they worked as intended. This ensures that IEEE I.1 [21] is followed and that no one is harmed because of any errors.

References:

[1]Gignac, Rachel. “DoorDash Launches Student Membership Plan.” *CSP Daily News*, 12 Apr. 2022, www.cspdailynews.com/snacks-candy/doordash-launches-student-membership-plan. Accessed 2 May 2025.

[2]Statista, “U.S.: Users in the Online Food Delivery Market,” *Statista*, 2024. https://www.statista.com/forecasts/891084/online-food-delivery-users-by-segment-in-united-states

[3]“LM2596 SIMPLE SWITCHER ® Power Converter 150-kHz 3-A Step-Down Voltage Regulator.” Available: https://www.ti.com/lit/ds/symlink/lm2596.pdf

[4]“et 1A LDO Voltage Regulator DESCRIPTION.” Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/5011/AMS1117.pdf

[5]“ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U Datasheet 2.4 GHz Wi-Fi (802.11 b/g/n) and Bluetooth ® 5 (LE) module Built around ESP32-S3 series of SoCs, Xtensa ® dual-core 32-bit LX7 microprocessor Flash up to 16 MB, PSRAM up to 8 MB 36 GPIOs, rich set of peripherals On-board PCB antenna.” Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

[6]“Introduction to the ESP-Prog Board - - — ESP-IoT-Solution latest documentation,” *Espressif.com*, 2016. https://docs.espressif.com/projects/esp-iot-solution/en/latest/hw-reference/ESP-Prog_guide.html

[7]“Matrix Keypad Created by Kattni Rembor.” Available: https://cdn-learn.adafruit.com/downloads/pdf/matrix-keypad.pdf

[8]“MFRC522 RFID Module.” Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/5531/4411_CN0090%20other%20related%20document%20%281%29.pdf

[9]“Solenoid Lock Diagram Specifications Electrical Specifications.” Accessed: Feb. 13, 2025. [Online]. Available: https://www.farnell.com/datasheets/2865757.pdf

[10]“Technical Documentation - Design | onsemi,” *Onsemi.com*, 2022. https://www.onsemi.com/pdf/datasheet/tip120-d.pdf

[11]“Weight Sensor (Load Cell) 0-500g SKU 314990000.” Accessed: Feb. 13, 2025. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/314990000_Web.pdf

[12]Avia Semiconductor, “AVIA SEMICONDUCTOR 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales DESCRIPTION,” 2009. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[13]“General Description Benefits and Features • Unique 1-Wire ® Interface Requires Only One Port Pin for Communication • Reduce Component Count with Integrated Temperature Sensor and EEPROM • Measures Temperatures from -55°C to +125°C (-67°F to +257°F) • ±0.5°C Accuracy from -10°C to +85°C • Programmable Resolution from 9 Bits to 12 Bits • No External Components Required • Parasitic Power Mode Requires Only 2 Pins for Operation (DQ and GND) • Simplifies Distributed Temperature-Sensing Applications with Multidrop Capability • Each Device Has a Unique 64-Bit Serial Code Stored in On-Board ROM • Flexible User-Definable Nonvolatile (NV) Alarm Settings with Alarm Search Command Identifies Devices with Temperatures Outside Programmed Limits Ordering Information appears at end of data sheet,” 2019. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf

[14]“POLYIMIDE FILM INSULATED FLEXIBLE HEATERS.” Accessed: May 07, 2025. [Online]. Available: https://assets.omega.com/spec/KHRA-KHLVA-KHA-SERIES.pdf

[15]Grainger, “Salary Averages,” *Illinois.edu*, 2021. http://ece.illinois.edu/admissions/why-ece/salary-averages (accessed Mar. 05, 2025).

[16]R. Tillaart, “HX711,” *GitHub*, Apr. 17, 2023. https://github.com/RobTillaart/HX711

[17]MakerSpaceLeiden, “GitHub - MakerSpaceLeiden/rfid: Arduino RFID Library for MFRC522,” *GitHub*, 2020. https://github.com/makerspaceleiden/rfid (accessed May 07, 2025).

[18]“Adafruit Keypad Library,” *GitHub*, Jan. 18, 2023. https://github.com/adafruit/Adafruit_Keypad

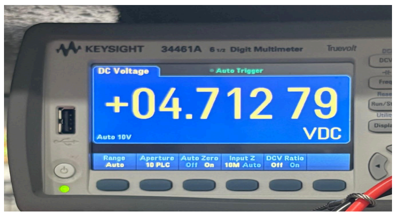
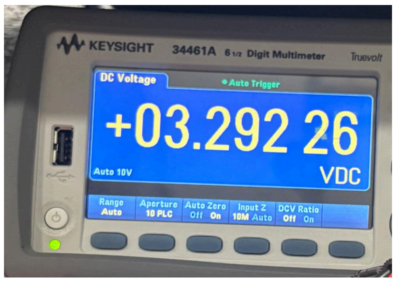
[19]M. Burton, “Arduino Library for Maxim Temperature Integrated Circuits,” *GitHub*, May 02, 2023. https://github.com/milesburton/Arduino-Temperature-Control-Library

[20]Association for Computing Machinery, “ACM code of ethics and professional conduct,” *Association for Computing Machinery*, 2018. https://www.acm.org/code-of-ethics

[21]IEEE, “IEEE Code of Ethics,” *ieee.org*, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html

Appendix A: Requirements and Verification Table

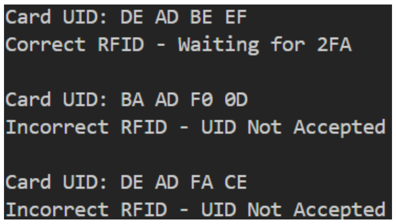
Power subsystem

Requirements	Verification	Results **(Y/N)**
1. The DC to DC buck converter must take a 12V input and output a 4.7V output	1a. Use the screw-in terminals on the 12V AC/DC converter to attach the V+ and GND to the PCB. 1b. Measure the voltage using a multimeter at the V+ and GND of the AC/DC screw terminal input and verify that it is 12 Volts. 1c. Measure the voltage across the output of the DC to DC buck converter using a multimeter and verify that it is 4.7 Volts.	1. **Y** Used a multimeter to see an output voltage of 4.7V - 4.8V consistently  KEYSIGHT 34461A 6½ Digit Multimeter DC Voltage +04.712 79 VDC
2. The AMS 1117-3.3 linear regulator must take a 4.7V input and output a 3.3V output when being powered by the barrel jack connector	2a. Use the screw-in terminals on the 12V AC/DC converter to attach the V+ and GND to the PCB. 2b. Measure the voltage using a multimeter at the V+ and GND of the AC/DC screw terminal and verify that it is 12 Volts. 2c. Measure the voltage across the AMS 1117-3.3 Vout pin and GND pin using a multimeter and verify that it is 3.3 Volts.	2. **Y** Used a multimeter to see an output voltage of 3.29226V above the minimum tolerance limit of 3.2 V and below the maximum tolerance limit of 3.4V to run necessary hardware  KEYSIGHT 34461A 6½ Digit Multimeter DC Voltage +03.292 26 VDC

Control subsystem

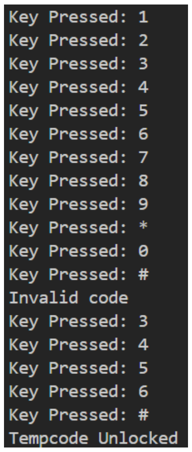
Requirements	Verification	Results **(Y/N)**
1. ESP-32 is programmable through UART bridge	1a. Connect the CP2102 UART bridge to a computer 1b. Connect the TX of the UART to RX pin of the PCB and the RX of the UART to TX of the PCB.	1. **Y** Correctly got a program to compile and upload onto the ESP on the PCB. Also got the serial terminal to output print statements

	1c. Plug in 3.3V, GND, RST, and BOOT of the UART bridge to the corresponding pins on the PCB 1b. Hold the boot button and upload a blink sketch given in Arduino using the right COM port and using the ESP 32 S3 Dev Module Board	

2. ESP32 correctly recognizes all input signals from subsystems	2a. Connect each used input pin to a high voltage of 3.3V
	2b. Check that ESP verifies input on COM6 output terminal
		2. **Y** Serial terminal recognizes every box input for keypad, correct code sequences, RFID, and heater on/off
		
		`Card UID: DE AD BE EF`
		`Correct RFID - Waiting for 2FA`
		
		`Card UID: BA AD F0 0D`
		`Incorrect RFID - UID Not Accepted`
		
		`Card UID: DE AD FA CE`
		`Incorrect RFID - UID Not Accepted`

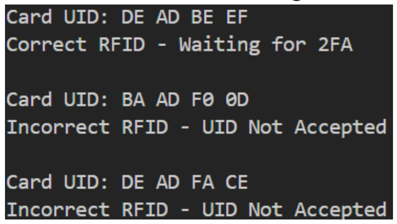
Authentication subsystem

Requirements	Verification	Results **(Y/N)**

1. Keypad recognizes a correct code	1a. Set a master code onto the ESP connected to keypad using the website
	1b. Enter the password and check that the box does not unlock
		1. **Y** Serial output on keypad shows correct output and activates lock
		
		`Key Pressed: 1`
		`Key Pressed: 2`
		`Key Pressed: 3`
		`Key Pressed: 4`
		`Key Pressed: 5`
		`Key Pressed: 6`
		`Key Pressed: 7`
		`Key Pressed: 8`
		`Key Pressed: 9`
		`Key Pressed: *`
		`Key Pressed: 0`
		`Key Pressed: #`
		`Invalid code`
		`Key Pressed: 3`
		`Key Pressed: 4`
		`Key Pressed: 5`
		`Key Pressed: 6`
		`Key Pressed: #`
		`Tempcode Unlocked`

2. Keypad declines all incorrect codes	2a. Set a master code onto the ESP connected to the keypad. 2b. Enter a password that is not the current master code or temp	2. **Y** Keypad does not open if wrong code is pressed before enter (#)

	code and check that the box does not open	

3. RFID scanner recognizes correct RFID tag	3a. Box only opens when both master keycode and the correct RFID tag are scanned	3. **Y** RFID scanner recognizes a hard coded authorized tag  `Card UID: DE AD BE EF`
		`Correct RFID - Waiting for 2FA`
		
		`Card UID: BA AD F0 0D`
		`Incorrect RFID - UID Not Accepted`
		
		`Card UID: DE AD FA CE`
		`Incorrect RFID - UID Not Accepted`

4. RFID declines all unauthorized RFID tags	4a. Box does not open when an incorrect RFID tag is scanned even with the correct master keycode. 4b. Verify this operation through COM6 serial monitor on Arduino	4. **Y** RFID scanner doesn't recognize an unauthorized tag
5. Dual-factor authentication involving master keycode and RFID works	5a. Scan the correct RFID tag 5b. Enter the master code within 30 seconds of scanning the RFID tag 5c. Verify that the lock unlocks	5. **Y** Dual factor authentication won't unlock the box unless both the master keycode and correct RFID tag are entered

Box Mechanics subsystem

Requirements	Verification	Results **(Y/N)**
1. Solenoid lock activates and deactivates when the lock and unlock button is pressed within 15 seconds	1a. Run the program and press the unlock button and verify that is unlock 1b. Press the lock button and verify that the lock locks.	1. **Y** Lock activates and deactivates when button is pressed 100% of the time within 5 seconds
2. Weight sensor should update the website that there is food in the box within 20 seconds	2a. Place a weight above 40 grams on the bottom of the box. 2b. Check that the food present reading on the website says "yes" 2c. Remove weight from the box 2b. Check that the food present reading on the website now says "no"	2. **Y** Weight sensor recognizes a weight placed on the box and can see the difference between a weight above or below the threshold of 40g (See Figure 7)

Heating subsystem

Requirements	Verification	Results **(Y/N)**
1. The heater will turn off within 30 seconds after the temperature inside the box is 32 degrees Celsius (90F) or more and will not be allowed to be turned on until the temperature is 31 degrees Celsius	1a. Turn the heater on using the enable heater button on the website 1b. Wait for the temperature sensor reading on the website to be 32 degrees Celsius or more. 1c. Once it is 32 degrees Celsius or more, check that the heater enable button is off and verify by viewing the heater directly 1d. Try to turn the heater on when the temperature is over 31 degrees Celsius and verify that it does not turn on until it is below 31 degrees Celsius	1. **Y** The heater turns off above 32°C within 15 seconds and wont turn on until 31°C as defined (See Figure 9)
2. The temperature sensor displays the correct temperature	2a. Place the temp sensor in a high temperature environment with a thermometer. 2b. Verify that the temperature displayed by the sensor on the website matches the thermometer. 2c. Repeat for a cold environment	2. **Y** Temperature sensor displays a temperature within 2°C of the thermometer at all times
3. The temperature displayed on the website is updated every 30 seconds	2a. Turn on the heater to change the temperature 2b. Wait for the temperature on the website to update 2c. As soon as it has updated start a timer for 30 seconds 2d. Once the 30 seconds are finished, look to see if the temperature has updated inside the box	3. **Y** Temperature sensor reading refreshes with each run of the loop, around 15s

User Interface system

Requirements	Verification	Results **(Y/N)**
1. Heater turns on within 30 seconds of pressing the heater enable button on the website	1a. Press the heater enable button on the website 1b. Wait up to 30 seconds for the heater to turn on 1c. Check the temp sensor on website to see if the temperature inside the box is increasing 1d. Verify correct usage through repeated observation	1. **Y** Heater turned on almost immediately after the button was pressed
2. User receives notification of food arrival when weight sensor is activated within 20 seconds	2a. Send input on weight sensor by putting food in box 2b. Website shows food present	2. **Y** Weight sensor updates in less than 10 seconds after food is placed
3. User receives new randomly generated temp keycodes within 30 seconds	3a. Use temporary keycode to unlock box 3b Website should show a new temporary keycode 3c. Previous temporary keycode does not unlock the box anymore	3. **Y** The temporary code on the website updates almost immediately after it is used. The previous keycode doesn't work anymore afterwards
4. User can create a new master key code	3a. Enter a new 4 digit master keycode in the master key code box on the website and press enter 3b. Wait 1 minute 20 seconds 3c. Enter the new master keycode in the keypad and enter your fingerprint to unlock the box	4. **Y** The input box on the user interface allows the user to choose their own master keycode which works consistently

Appendix B: Cost of Parts Table

Part	Manufacturer	Quantity	Cost/Unit
ESP32-S3-WROOM-1-N4R2	Espressif Systems	2	$3.10
10 pcs AMS1117-3.3	Adafruit	1	$6.49
ALITOVE DC 12V 5A Power Supply Adapter Converter Transformer AC 100-240V Input	ALITOVE	1	$11.99
Tegg 1PC 3x4 Keypad MCU Board Matrix Array Switch Tactile Keypad 12 Button Phone-Style Matrix Keypad for Arduino Raspberry Pi	Tegg	1	$9.99
Waterproof 1-Wire DS18B20 Digital temperature sensor	Adafruit	1	$9.95
Lock-style Solenoid - 12VDC	Adafruit	1	$14.95
0-5kg load cell	Sparkfun	1	$17.81
15pcs 10mmx93mm Film Heater Plate Adhesive Pad, PI Heating Elements Film 12V 12W Strip Heater Adhesive	XIITIA	1	$12.99
LM2596 Buck Breakout Board	Mouser	1	4.99
AMS1117-3.3V Breakout Board	HiLetgo	2	1.99
ESP PROG	Espressif	1	14

Appendix C: Detailed schedule

WEEK	**TASKS**	**PERSON**
3/3	Design Document	Everyone
	Continue buying parts	Rohan
	Start breadboard assembly	Rohan
3/10	Continue breadboard assembly	Everyone
	Assemble PCB and test	Taniah
	Work on and tests power subsystems	Work
	Breadboard demo - Wednesday	Everyone
	Redesign PCB	Dhruva
	Second PCB order - Thursday	Dhruva
	Talk with machine shop and give parts received - Friday	Rohan
3/17 (spring break)		
3/24	Assemble new PCB and test	Taniah
	Modify the schematic if needed	Everyone
	Work on and test authentication subsystem	Rohan
	Work on and test control subsystems	Dhruva
	Work on user interface	Taniah
	Redesign PCB	Dhruva
3/31	Third PCB order - Monday	Dhruva
	Work on and test box mechanics subsystem	Rohan
	Assemble new PCB and test	Taniah
	Individual progress reports due - Wednesday	Everyone
4/7	Fourth PCB order - Monday	Dhruva
	Work on and test Heating subsystem	Dhruva
	Assemble new PCB and test	Rohan

4/14	Team Contract assignment - Friday	Everyone
	Finalize all components of the box	Everyone
	Prepare for mock demo	Everyone
4/21	Mock Demo with TA - Tuesday	Everyone
	Prepare for final demo	Everyone
4/28	Final Demo	Everyone
	Mock Presentation	Everyone
	Prepare for final presentation	Everyone
	Work on final paper	Everyone
5/5	Final Presentation	Everyone
	Final Paper due - Wednesday	Everyone

Appendix D: Supporting Images

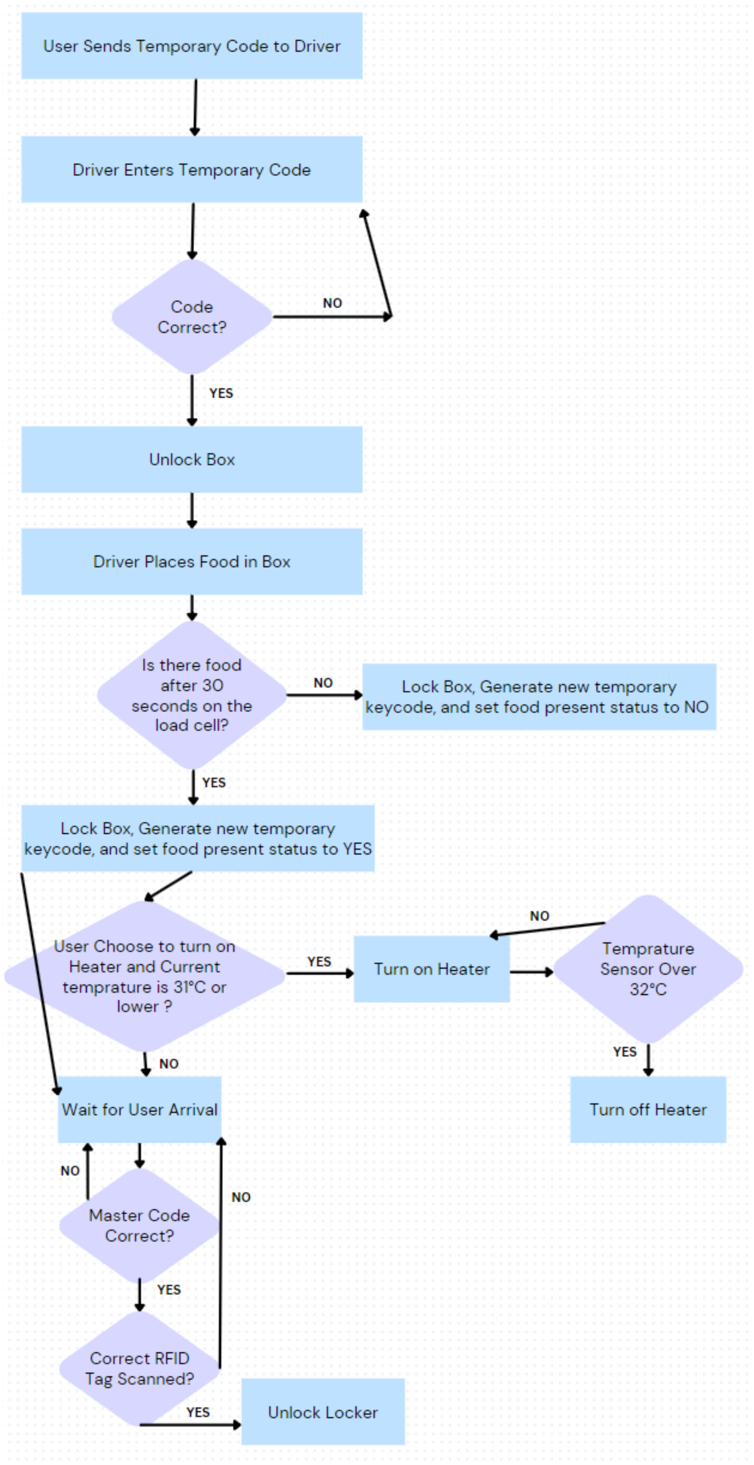

User Sends Temporary Code to Driver
Driver Enters Temporary Code
Code Correct?
NO
YES
Unlock Box
Driver Places Food in Box
Is there food after 30 seconds on the load cell?
NO
Lock Box, Generate new temporary keycode, and set food present status to NO
YES
Lock Box, Generate new temporary keycode, and set food present status to YES
User Choose to turn on Heater and Current temprature is 31°C or lower ?
YES
Turn on Heater
Temprature Sensor Over 32°C
NO
YES
Turn off Heater
NO
Wait for User Arrival
NO
Master Code Correct?
NO
YES
Correct RFID Tag Scanned?
YES
Unlock Locker

Figure 11. Flow Chart of Final Code

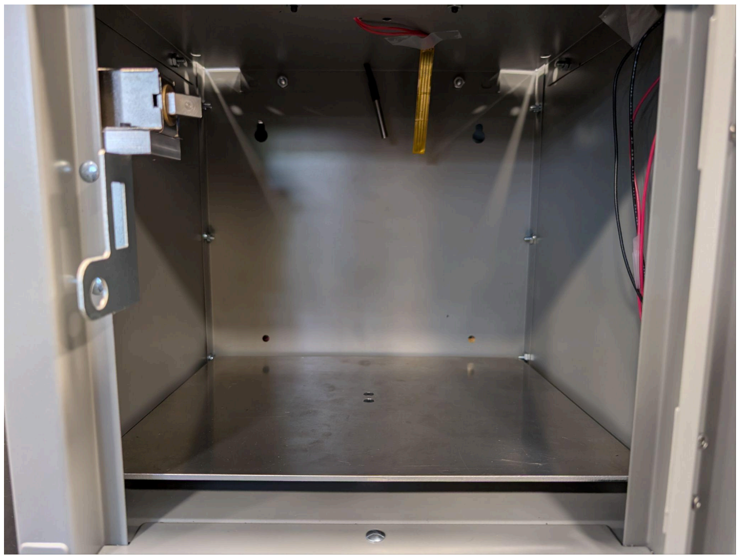


Figure 12: Inside of Secure Food Dropbox with Temperature Sensor, Heater, Load Cell, and Lock in View

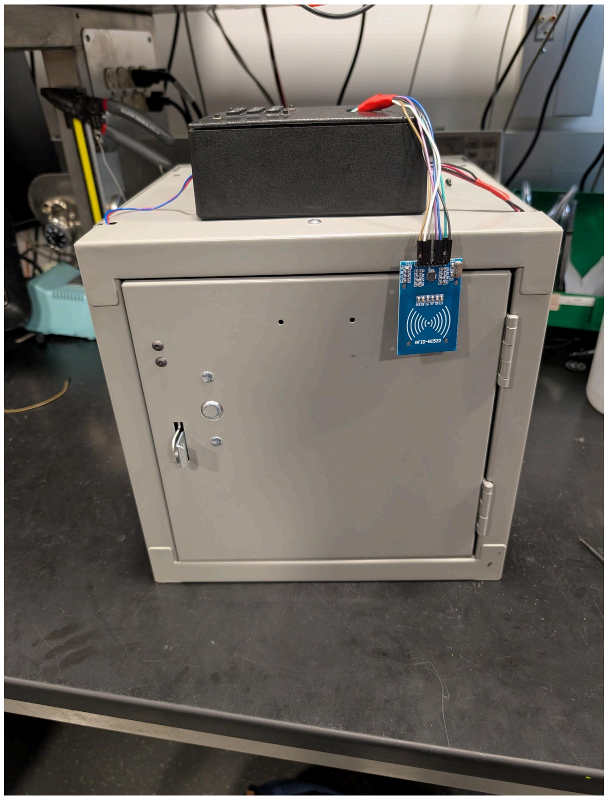


Figure 13: Outside of Secure Food Delivery Dropbox with RFID in View

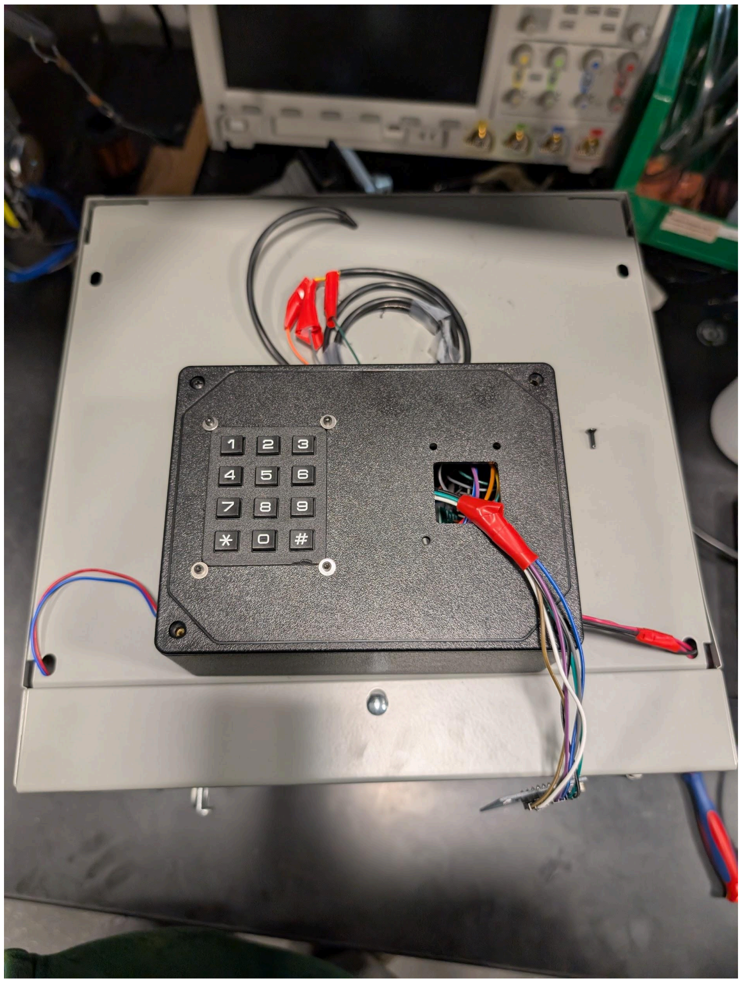


Figure 14: Top of Secure Food Delivery Dropbox with Keypad in View

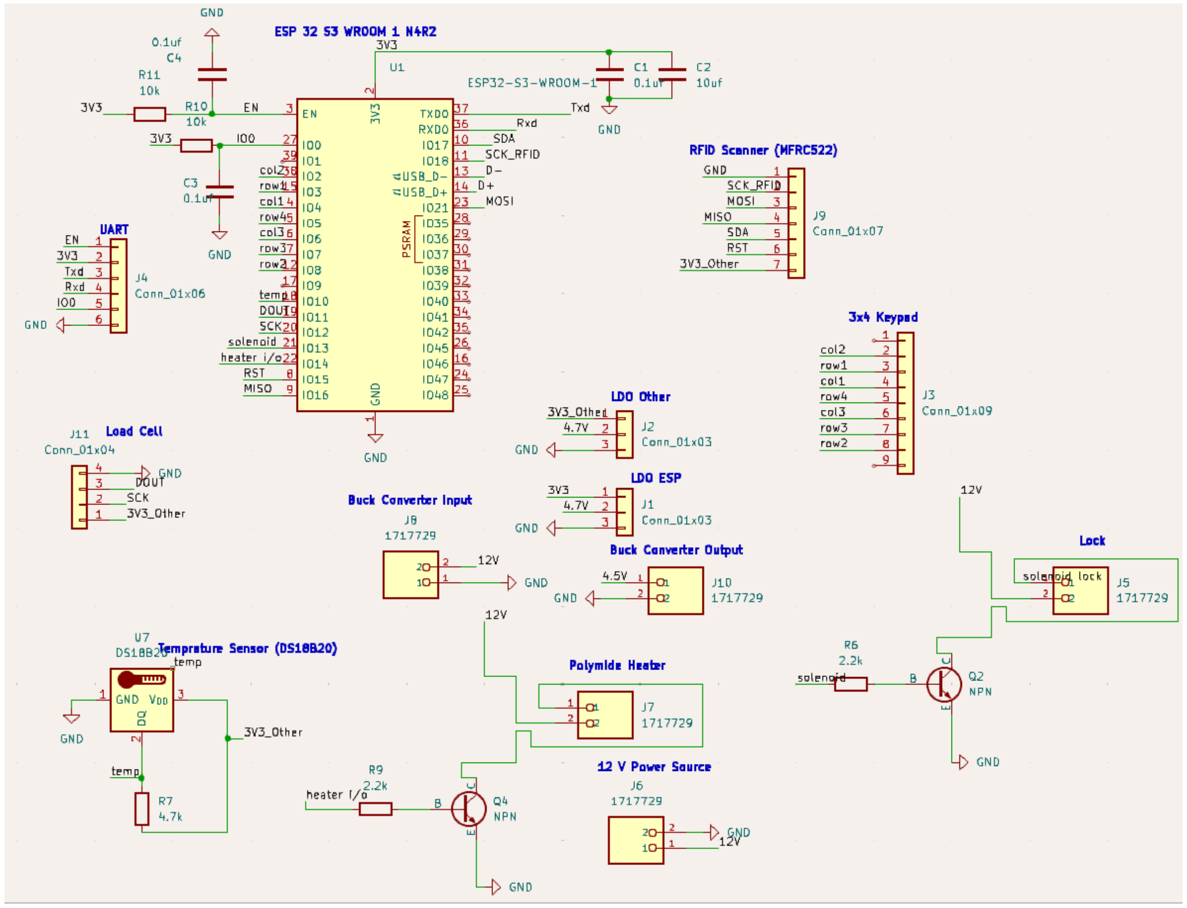


Figure 15: Entire Schematic

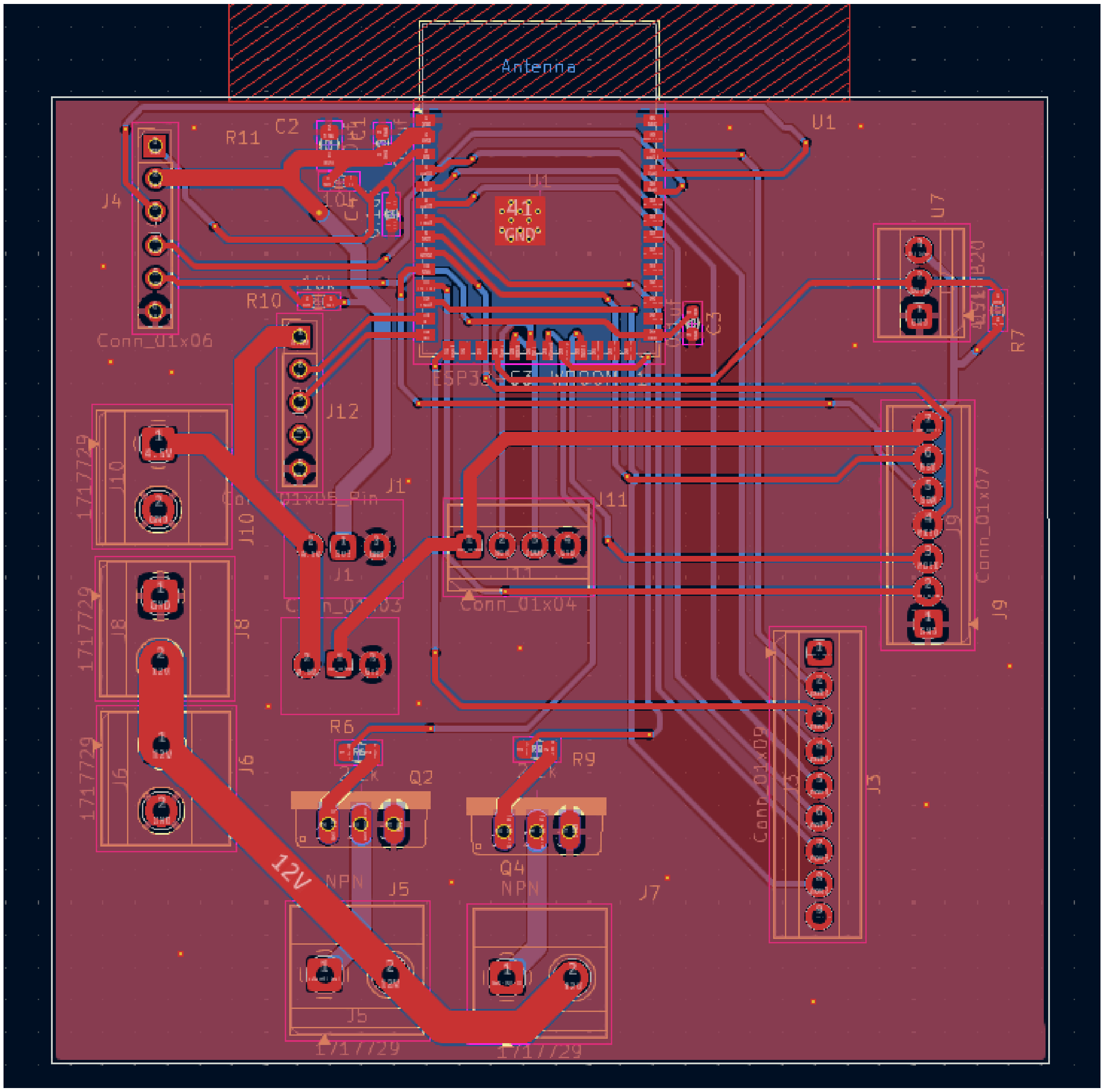


Figure 16: Entire PCB Design

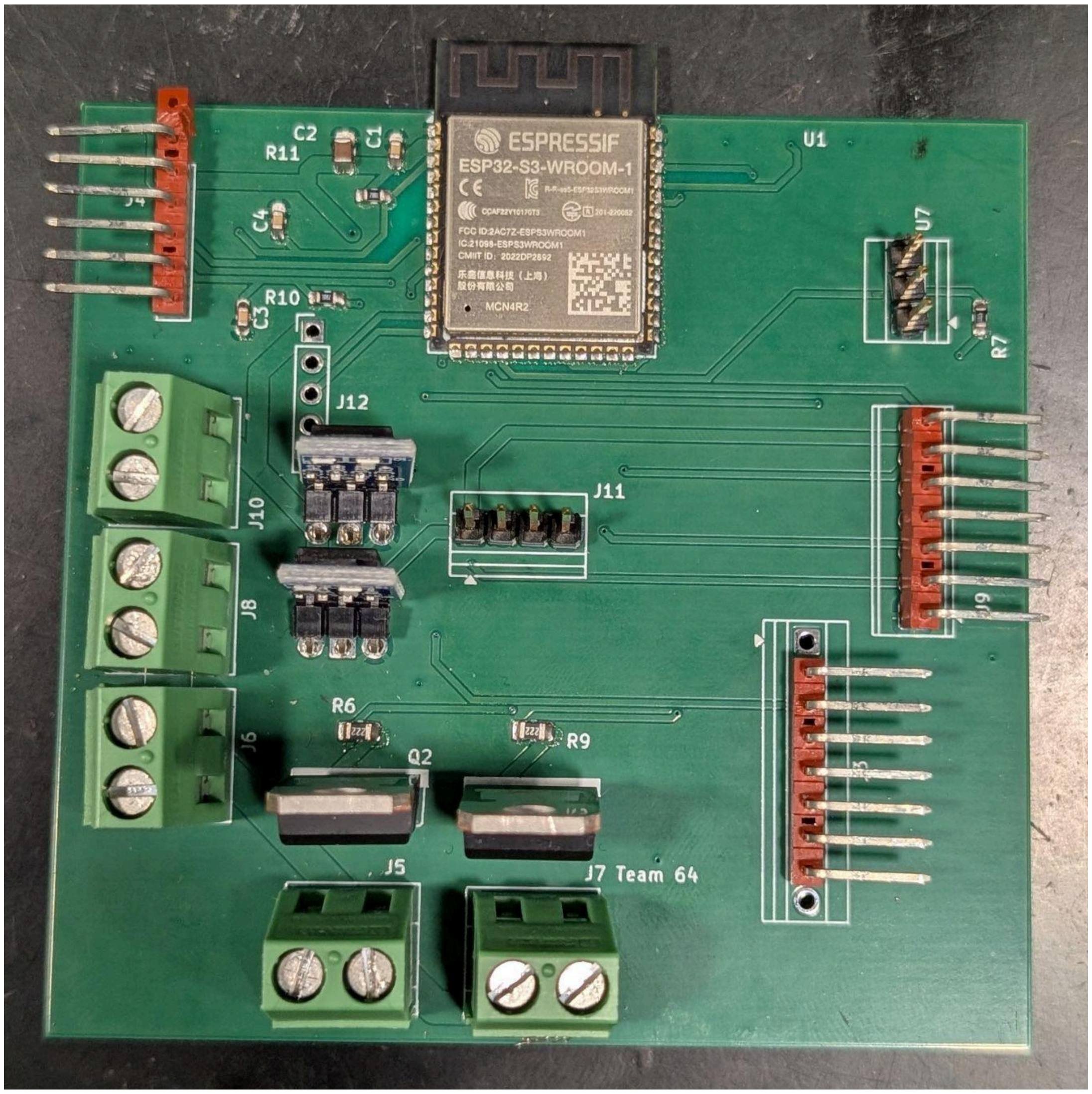


Figure 17: Physical PCB with Components Including AMS1117-3.3V Breakout Boards

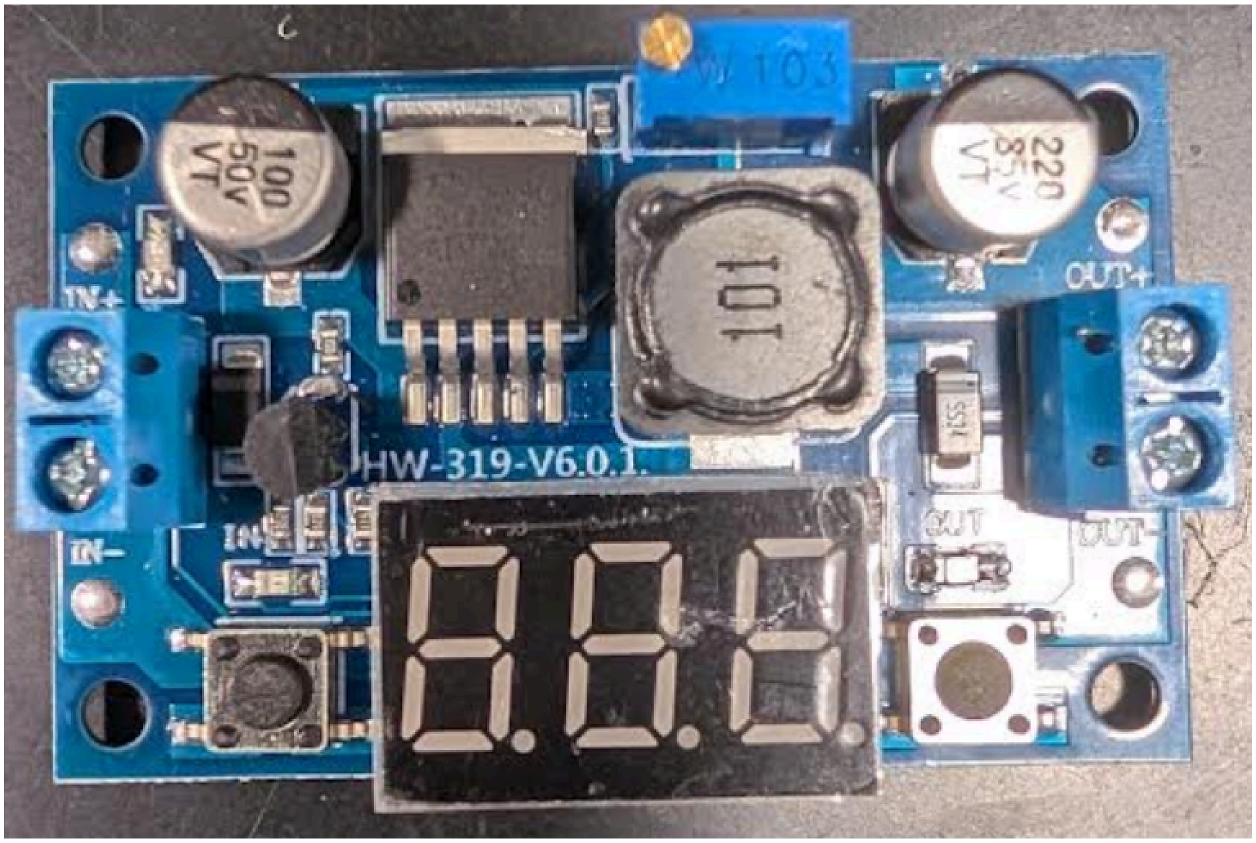


Figure 18: LM2596 Breakout Board