Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2020/2

Technical Report

ChickenIO - Poultry Coop Management

Gabriel Vieira Rodrigues – gabriel.vieirarodrigues@gmail.com
Gustavo Maysonnave Franck – gmaysonnavefranck@gmail.com
João Carlos Cardoso – jc.cardoso4@gmail.com
João Luiz Ithiro Sumi Borges – joaoluiz@alunos.utfpr.edu.br
Victor Feitosa Lourenço – victorfeitosalourenco@gmail.com
Vinícius Debur Bernet – vini.bernert@yahoo.com.br

April 2021

Abstract

Using as motivation the current market situation for poultry farming and availability for optimization in animal care, this report describes the development of the ChickenIO project aimed at automating a chicken coop. The proposed concept revolves around removing some of the activities which currently require constant human care in a coop, such as controlling the food given to the animals and collecting the laid eggs. Furthermore, the system aims to provide valuable information to the coop owner over the environment and each of their individual chickens, allowing for customization and control over the particularities of each animal. The prototype to which this report refers to consists of a mechanical structure, an embedded system attached to this structure, and a web application.

Keywords: poultry farming, chicken coop, automation.

1 Introduction

In the recent years, the consumption of chicken meat has risen at an alarming rate, surpassing the overall consumption of beef and pork in several countries [1, 2]. This means that poultry farming is at its current peak, demanding a large quantity of dedicated manpower and technology. At the same time, the conditions with which the animals are treated have also been publicized and criticized, given the poor quality of the work environment the workers are put through and lack of versatility when tending to a large number of animals [3].

With this in mind, developing new techniques to increase the productivity of poultry farming and optimize its procedures is a great topic of concern on this new decade. This also revolves around the conditions which not only the animals, but also humans are exposed to when working in the field. Furthermore, the average size of poultry farms and the amount of livestock — around 134 acres and 50 to 100 recommended animals per acre in the USA [4, 5] — only contributes to the difficulty of manually managing such a structure.

In this context, this project directly approaches these issues, taking the first steps towards automatizing specific segments of poultry farming, such as feeding and laid eggs

collection. Along side the mechanical functionalities aided by the project's hardware, the system incorporates a complete software structure, which enhances its capabilities, adding valuable analysis over the data that is being constantly collected.

1.1 Main Goal

Considering the current farming state just described, ChickenIO aims to introduce new concepts and innovations when it comes to monitoring and administering a poultry farm, in other words, its main goal is to facilitate the management in tending to the fowl's care in a procedural manner.

1.2 Specific Goals

The scope of the main goal of this project includes not only completely removing the necessity of human interaction when feeding the animals, but also a thorough supervision of the fowl, particular to each one's physical and biological characteristics, with the goal of ensuring a healthy environment and growth. The project involves the integration of the mechanical aspects of control with environmental monitoring through sensors and the constant identification of the fowl. This is naturally incorporated into a cloud-based system to be displayed to a user-friendly web application for tracking the system's functionalities and results.

Each animal needs to be easily identifiable through a fast and effortless procedure. This is achieved with the help of radio frequency identification (RFID), with tags attached to the shank of every hen. Strategically placed RFID readers control exactly which animal is active at a key position, which are the feeder and nest.

The feeder is the focal point of the ChickenIO, where the fowl is separated and fed individually to ensure the unique treatment each animal should receive — the use of perches is ideal for such a task, guaranteeing only one hen has access to the food recipient at a time. Upon entering the feeder, the fowl will be weighed and the food it has eaten will be logged. This information is essential, since given different hen characteristics and feeding history, its food ration might differ. A certain hen A could be required to eat 150g of food spread through 3 separate meals, whereas another hen B only needs 100g separated into 2 meals. The assurance that this is properly carried out is fundamental to treating each animal considering their particular needs. These specific characteristics can be accessed, monitored and altered via the web application.

Finally, the ChickenIO also includes a control station at the animals nest, which is where the fowl lays eggs. A second RFID reader at this location recognizes the presence of a specific hen and if it lays an egg, the same structure which would normally conduct the egg to a collection station also allows for a scale to weigh the egg and record its data to the system associated with the hen that laid it.

Meanwhile, the environment which the fowl inhabits — the coop itself — is being constantly monitored for key factors that are of extreme importance to the animals health and well-being. These are the air humidity, temperature and luminosity levels. Particular sensors are installed on the project's structure, near the feeder and nest themselves, out of reach of the animals, to measure as well as send the environment data to the server, alerting the user of any undesired conditions.

1.3 Document Structure

This document follows an organized structure to present how the project was conceptualized, planned and executed. This first Section 1 described the project's idea and sketched its main and specific goals. Section 2 investigates the viability of the project as well as discusses the functional and non-functional requirements, which should be met at the end of development. Section 3 goes over the technologies used to allow for the development of the project, including hardware and software tools and components. Section 4 serves as the core for this document, describing the development steps and how the ideas and requirements were executed — composed of mechanical, hardware and software development. Finally, Section 5 concludes the document presenting the results and with a discussion over the previous weeks of development.

2 Project Specifications

Once the goals and purpose for the project have been established, the study of how to achieve these goals follows. For this purpose, the following subsections discuss the way in which the development of the project was segmented and the requirements necessary for the advent of the poultry coop management system to be possible. This also involves a viability investigation, which briefly discusses budget, time management, organizational and operational characteristics.

This was done through conversations with a specialist of the area and studies directly related to each of the fields of the project. Thus, questions and answers were raised regarding possible key issues with the development. One essential part of the specification revolves around the functional and non-functional requirements of the system. Each of these topics is of extreme importance to the organization of the project development, allowing it to be carried out in the most secure manner possible, making the effort to avoid unexpected issues and complications.

2.1 Viability Investigation

At an early stage during planning, it is important to plainly determine in which ways does the client/user seek to gain benefit from the system and also in which ways does the system itself effectively treat these requests — and more importantly, in which ways it does not. The client looks for a way to automate a previously completely manual poultry farm, which would comport 15 animals at once — the scalability of the project is not treated in this document. The previously mentioned environment factors — temperature, humidity and light intensity — are exactly what the client deems fundamental for the monitoring of the chickens. Added to these, there should also be a way to measure the chickens weight as well as the weight of the eggs they lay. Furthermore, it is stated that a customizable interface, which is particular to each animal, should be available to the user, allowing for control over the quantity of meals per day and amount of dispensed food. The client would benefit from an overview picture of the coop which is regularly uploaded to the system — for use in further projects.

Given these functionalities, ChickenIO is set to fulfill the majority of them, with the exception of the overview picture of the coop. This functionality is partly added to the system, but with the difference that it will be leveled with the animals, rather than on the ceiling, offering an overview image. This is due to the fact that an extended connection to a camera — which would be too distant from the hardware structure of the rest of the project — would pose a superfluous adversity to the development.Nevertheless, the functionality will be present, as will the remaining discusses features, in a manner that is efficient and that brings an overall assistance and overall work relief for any individual involved in the poultry farm.

It is also relevant to mention that, since the system is autonomous, it requires no adaptation or previous knowledge of the user/client after the installation. The web application is intuitive and follows well known guidelines for web interfaces of the present time. It also includes a security layer, assuring the user that their data will be safe and stored in a reliable manner.

An additional aspect that has to be discussed before the development is initiated is the economic viability of the project. A specific maximum value was stipulated to allow

the members to securely acquire the materials and components necessary to develop the system — a quantity of R$ 50,00 per member for each month of development, totaling R$ 850,00 over three months with six members. Given that the members already had several of the necessary items for development, as well as were aided by the professors, there were no economic issues in the plan or development. It is expected that, with the reduced amount of manpower and wasted chicken food, this price will be covered overtime. Furthermore, the market for this product is, as was mentioned before, on the rise, with plenty of opportunities for further investment and incentive for advancements.

Regarding the organization of the development, nothing is more essential than good time management, especially since this project has no margin for error on this regard. The tasks executed throughout the three months range from project planning, blog creation, mechanical structure modeling and building, hardware design and organization, software and firmware development and implementation, the integration of the previous parts and the organization of the documentation — which includes this document and project presentations. With this in mind, the members discussed the abilities and experiences of each to best select which would be responsible for each task. Back-up and assistant members were also selected, as well as smaller groups of two members which would work more intensely together. The project was then split into 103 different tasks, which were organized throughout the available weeks with a well defined amount of hours that should be spent on each. During development, every week was discussed and analyzed to ensure that the members were adequately performing their tasks and that no drastic changes to the schedule could go by unnoticed and possibly hinder the project progress.

2.2 Requirements Elicitation

The process of requirement elicitation is achieved through research and definition of the prerequisites of a project or system guided towards the customers and users. It is often highlighted that this process is not a simple gathering of requirements, but rather an elicitation, since one can never be sure that all requirements were specified or even that the ones that were are correctly described. This practice requires close observation, interviews, perspective change, brainstorming and often even preliminary prototypes [6].

2.2.1 Functional and Non-Functional Requirements

Listed in the following Tables 1, 2 and 3 are the functional requirements (which are of paramount importance in defining the applicability of the system) and the non-function requirements (which better express how the system should be made) that the system must meet. These are also separated into three groups — hardware, software (web application) and firmware — to allow for a more concise and intuitive listing.

Table 1: Functional and non-functional requirements specific to the project's hardware.

Hardware Functional Requirements	
H-FR01	The hardware must be connected to an RFID reader.
H-FR02	The hardware is requested to receive signal from the camera.
H-FR03	The hardware shall read the scale which weighs the eggs.
H-FR04	The hardware must read the air humidity sensor.
H-FR05	The hardware must read the temperature sensor.
H-FR06	The hardware must read the luminosity sensor.
H-FR07	The hardware must be connected to a second RFID reader.
H-FR08	The hardware shall read the scale which weights the hens.
H-FR09	The hardware shall read the scale which weights the food recipient.
H-FR10	The hardware must control the red light indicator.
H-FR11	The hardware must control the green light indicator.
H-FR12	The hardware must control the motor that dispenses the food.
Hardware Non-Functional Requirements	
H-NFR01	The system is requested to be implemented using Raspberry Pi 3 B+.
H-NFR02	The system is requested to use the MFRC522 as the RFID reader.
H-NFR03	The system is requested to use the HX711 as the weight AD converter.
H-NFR04	The system is requested to use the DHT22 as the temperature and humidity sensor.

Table 2: Functional and non-functional requirements specific to the project's web application software.

Software Functional Requirements	
S-FR01	The web app must allow the user to register hens.
S-FR02	The web app shall allow access to registered hens' data.
S-FR03	The web app is requested to log the amount of daily meals per hen.
S-FR04	The web app should log the amount of grams of food per meal of each hen.
S-FR05	The web app must show the personal history of a hen's weight.
S-FR06	The web app shall show the personal history of a hen's number of meals.
S-FR07	The web app must show the personal history of a hen's consumed food in grams per day.
S-FR08	The web app is requested to signal if a hen's weight is going up.
S-FR09	The web app shall alert if a hen left uneaten food.
S-FR10	The web app should create reports about the weight of all the hens.
S-FR11	The web app must create reports about the amount in grams consumed by all the hens.
S-FR12	The web app must create reports about each hen's meals.
S-FR13	The web app shall show the functioning history of all the sensors.
Software Non-Functional Requirements	
S-NFR01	The web app back-end is requested to be programmed using Node.js.
S-NFR02	The web app front-end is requested to be programmed using VueJS.
S-NFR03	The web app shall store the data using a relational database.
S-NFR04	The web app shall receive the data in JSON format.
S-NFR06	The web app must be hosted in a cloud server.
S-NFR07	The web app shall show data in a dashboard.
S-NFR08	The web app shall show data in graphs.
S-NFR09	The web app shall show a form for registering hens.

Table 3: Functional and non-functional requirements specific to the project's firmware.

Firmware Functional Requirements	
F-FR01	The firmware must connect to the cloud server.
F-FR02	The firmware must send the temperature information to the cloud server every 30 seconds.
F-FR03	The firmware is requested to send the humidity information to the cloud server every 30 seconds.
F-FR04	The firmware should send the luminosity information to the cloud server every 30 seconds.
F-FR05	The firmware shall send the weight of the hen identified in the nest at the moment it is recognized.
F-FR06	The firmware shall send the weight of the hen identified on the feeder at the moment it is recognized.
F-FR07	The firmware must turn on the green light to condition the hen to wait for the food.
F-FR08	The firmware must turn on the red light to condition the hen that it can eat.
F-FR09	The firmware shall provide the exact quantity of food registered for that hen in that specific meal.
F-FR10	The firmware is requested to send a photo of the chicken coop every 30 seconds.
F-FR11	The firmware shall identify the hen using RFID.
Firmware Non-Functional Requirements	
F-NFR01	The firmware is requested to be implemented on a Raspberry Pi.
F-NFR02	The firmware must be able to send information in JSON.
F-NFR03	The firmware must be able to receive information in JSON.
F-NFR03	The firmware must be programmed using Python.

2.3 Project Overview

Taking into consideration the requirements for the system and the overall concept of the project, Figure 1 serves as an initial illustration of what it is composed of, displaying the components of the ChickenIO system, as well as already indicating some of the technologies used for the development — which will be discussed in Section 3.

The complete structure revolves around two stations for the animals — the feeder and the nest — and three main elements — the embedded system, the cloud server and the web application. It is important to specify which of the items shown in Figure 1 are already a part of the chicken coop at which the system is projected to be integrated with. These are the food recipient and the motor at the feeder, along with the structure at which the animals rest in the nest to lay their eggs, which are directed downwards via a slope in the structure.

In a general manner, regarding the integral parts of the ChickenIO system, the feeder is responsible for detecting the animals presence; dispensing the correct amount of food into the bowl; and measuring the weight of the chicken as well as the food in the food bowl. The nest also detects the animal presence and measures the weight of the laid eggs. The embedded system is responsible for integrating the data sent by the feeder and nest; gathering environmental data; controlling the motor for dispensing food; and making the connection between this information and the cloud server. The cloud server itself

records and organizes the data, which will both be used by the embedded system and the web application. This application is the link between the system and the user, displaying the interface with information about each of the chickens and the environment, as well as allowing for customization and updates to each of the animals particular parameters — such as amount of meals per day, amount of food per meal and interval between meals.

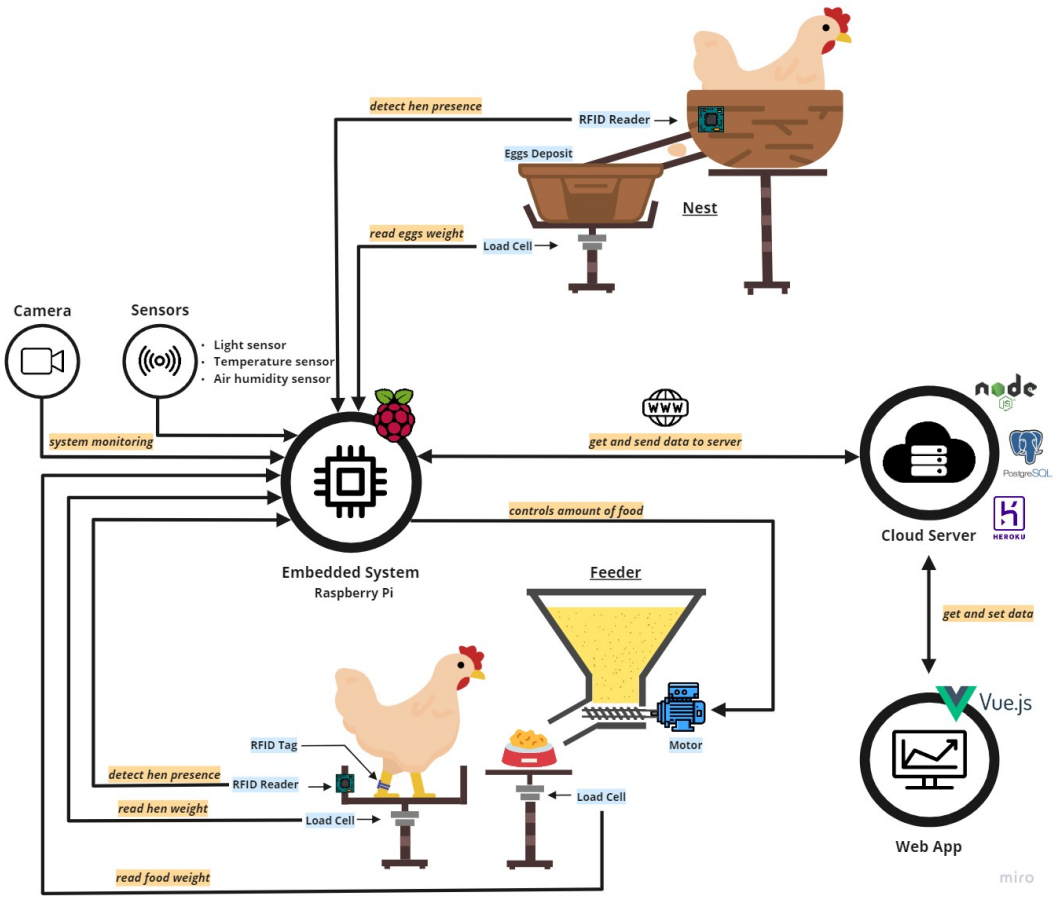


Figure 1: Project overview diagram including both the feeder and nest stations.

3 Technologies Used

This section describes the tools and components used in the development of the project, as well as their main functions in particular to the system at hand. Subsection 3.4 goes over the materials used in the mechanical structure as well as the selected interface for 3D modeling; Subsection 3.2 examines the hardware modules, components and firmware tools; and finally, Subsection 3.4 presents selected tools for software development, including the back-end database and server infrastructure, as well as the front-end of the web application.

3.1 Mechanical Structure

The mechanical structure of the project resolves around the correct modeling and execution of the design using appropriate carpentry techniques. To achieve this, different tools were considered, both for the design and the physical structure, until the optimal ones were selected.

3.1.1 3D Design Tool

The most relevant requirements when considering which 3D design tool to select are how much detail does the tool allow the user to customize and how easy it is to learn how to use it. A software such as AutoCAD would provide a large amount of customization freedom, however this means that its complexity is above what is required for the modeling of this project.

For this reason, a simpler, but still powerful tool, from the same development company — Autodesk — was selected: Tinkercad. Tinkercad is a free 3D modeling tool, which can be run on a browser, without requiring any software download. It is also integrated with a gallery of projects, allowing the users to share their designs online with one another. This is also favorable for the purpose of this project, since it allows for easy access of any individual with interest in the model.

3.1.2 Woodwork

The assembly of the mechanical structure, given the adequate 3D model, requires carpentry and woodwork. A set of wooden slabs were separated for use in the project, as well as the required tools for the manipulation of this material. These tools included nails, screws, screwdrivers, a hammer, drilling machines and saws (crosscut, hole-saw and circular saw).

3.2 Embedded System

The embedded system is composed by the system's processing unit and the input/output peripheral devices. The selection of the components for this section of the system involves the study of which would be the most adequate processing unit for the functionalities of the project, as well as which sensors would be the most convenient and accurate for this purpose.

3.2.1 Raspberry Pi

The selection of the processing unit of the system took into consideration factors such as its built in functionalities and modules, how broad were the availability of further components to be added and the community support. The Raspberry Pi was not only chosen given these characteristics, but also the familiarity of the members with the microcontroller/microcomputer.

Other than the good processing capabilities of the Raspberry Pi, it already includes several modules, such as Wi-fi and operation system availability. Furthermore, this device has a considerable amount of pins, allowing for the interface with all the remaining components of the system, as well as includes the necessary interface protocols — GPIO, I2C and two separate SPI links. This interface is also of intuitive use for most components, with various available libraries and extensive community support.

3.2.2 RFID Module

The selection of the RFID module is focused on a reader that is efficient and capable of recognizing tags from a sufficient distance. It is also important to keep in mind that the system requires two separate readers, one for the feeder and one for the nest station. The selected module was the MFRC522, which is highly used communication at a frequency of 13.56MHz. This is a low-consumption, small-size chip that uses the SPI interface and allows for contactless reading and writing on cards that follow the Mifare standard. A few distinct RFID tags that follow the correct frequency are also used for testing along with the reader itself.

3.2.3 Temperature & Humidity Module

The temperature and humidity sensor was selected given the appropriate recommendation of the field specialist, who was consulted at the beginning of project planning. The preferred component was the DHT22, which is a temperature and humidity sensor that allows temperature readings between -40 to +80 degrees Celsius and humidity between 0 to 100%, being very easy to use due to its single digital output pin. This component uses the GPIO interface for its communication.

3.2.4 Luminosity Module

Similar to the temperature and humidity sensor, the luminosity sensor was chosen based on the advice of the field specialist. The chosen component was the BH1750, which uses the I2C interface to detect the incidence of light with a light spectrum response identical to the human eye.

3.2.5 Camera

The chosen camera for takes in consideration, most of all, the availability of the component, which was borrowed from the professors for use in the project. For this component, the connection with the Raspberry Pi proves to be even more intuitive and useful, since the selected module is the Pi NoIR Camera V2, specifically designed for use with this device. The communication between the camera and the Raspberry Pi is done via

a set of reserved pins for this purpose. This component has a Sony IMX219 8-megapixel sensor, which does not employ an infrared filter, meaning that pictures can be taken not only in daylight, but also in darkness with infrared lighting.

3.2.6 Load Cells

There is not much decision to be made when it comes to the load cells, since their variety is scarce and will be calibrated individually. For this reason, the selected components were the ones already in possession of the members, which were two 20kg and one 5kg cells. These components generate a DC analog signal which needs to be converted to digital, which means the cells are connected to the Raspberry Pi via an HX711 AD converter and the GPIO interface. Since each HX711 component allows for the connection of two load cells, two distinct AD converters are used in this project.

3.2.7 LEDs

The LEDs are crucial to the conditioning of the chickens, which means that its colors should be customizable and emit enough light to influence the animals. The selected component was the WS2812B, a customizable string of very potent LEDs, which was selected to be in a ring format and contain 12 LEDs.

3.2.8 Relay

The relay module is used as the connection between the digital, low power side of the system and the high voltage analog signal that powers the DC motor for food disposal. The selection of the component was due to the availability of a 4 Relay Module, supplied by the field specialist for use in the project.

3.3 Cloud Server

The cloud system is the central part of the data handling and control of the project. Without previous knowledge on the development tools of this part of the system, the development process would surely prove to be troublesome and time consuming. For this reason, the selection of the tools was based on the familiarity of the members and their ease of use.

3.3.1 Database Management

When it comes to the database organization, PostgreSQL was selected to be relational database management system, which meets the requirements for this project. Its reliability, feature robustness, and performance are key factors which PostgreSQL offers.

3.3.2 Back-End Development Tool

The runtime environment Node.js, which executes JavaScript code, was chosen as the back-end development tool. It allows for the creation of Web servers and networking

tools using JavaScript and a collection of modules that handle various core functionalities. Due to its single-threaded nature, Node.js is very efficient at non-blocking, event-driven servers.

3.3.3 Cloud Platform

To effectively deploy the cloud server, it is necessary to use a cloud platform service, which was chosen to be Heroku for this project. Heroku is a container-based cloud platform as a service, which gives users the freedom to focus on their core product without the preoccupation of maintaining servers, hardware, or infrastructure.

3.4 Web Application

The selected tool for the web application needs to allow the developer to properly create an interface which is friendly to the user and intuitive to use.

3.4.1 Front-End Development Tool

It is important for the front-end development tool to allow for modularity and ease when adding new features to the user. Vue.js was selected for this purpose, a front-end JavaScript framework for building user interfaces and single-page applications.

4 Development

This section goes over the development stages of the project, outlining some of the difficulties and challenges throughout the process. Subsection 4.1 illustrates how the mechanical structure was modeled and how this model was actually materialized; Subsection 4.2 explains how the embedded system was developed, including the integration between the hardware and firmware; Subsection 4.3 clarifies the steps in software development regarding the web application of the project.

4.1 Mechanical Structure

Before the mechanical structure could be properly developed, steps in 3D modelling and specifying dimensions must be taken. Attention was given to aspects, such as how high should the perch be from the ground so that the chickens can reach it; how broad should this perch be to allow only one of the animals to be on it at a time; how far away should the food bowl be from the perch to allow the animals to bend their bodies and eat. These and other measurements were confirmed with the field specialist and a 3D model of the structure was developed with the use of *Tinkercad*, as discussed in Subsection 3.1.

Additionally, the components used in the project were also included in this model, making it clear how they would be arranged and integrated. For this, the *datasheets* of each was consulted regarding their dimensions. Finally, when put together, the overall 3D can be seen in Figure 2.

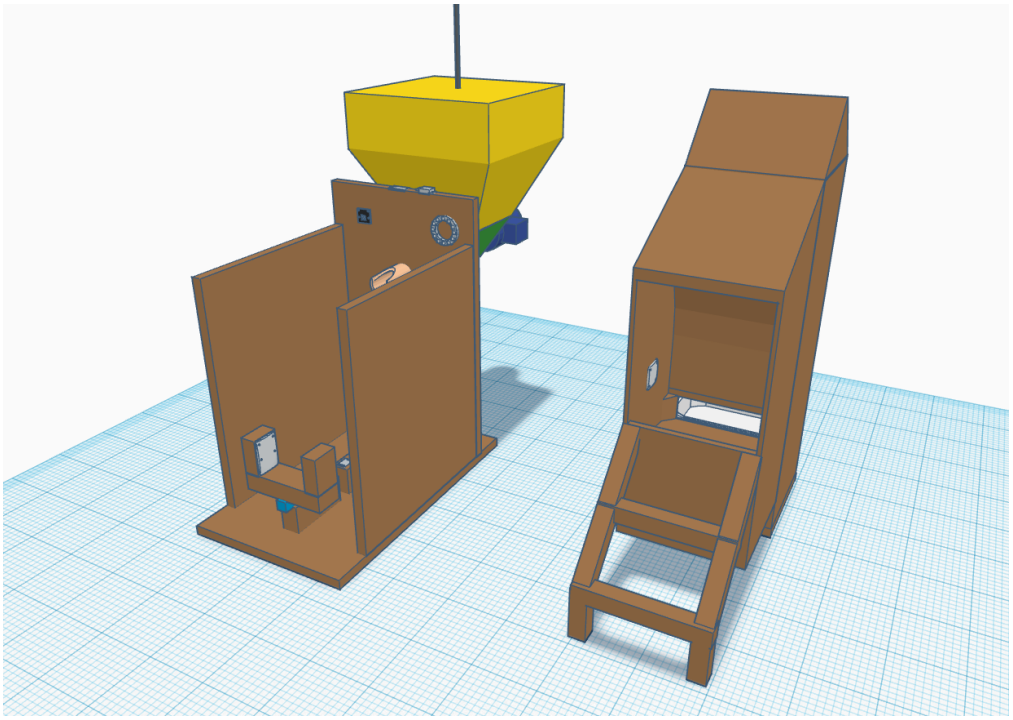


Figure 2: Mechanical structure 3D model overview including both the feeder and nest stations.

To understand better what Figure 2 is composed of, Figures 3, 4 and 5 respectively illustrate the modeled components, the feeder station and the nest station. It is also important to mention that not all structures present in this model were effectively built for the project. This is due to the fact that some of these elements are already an integral part of the chicken coop that the project was developed to. Therefore, the product of this development acts as an attachment to the already existing coop, facilitating its automation.

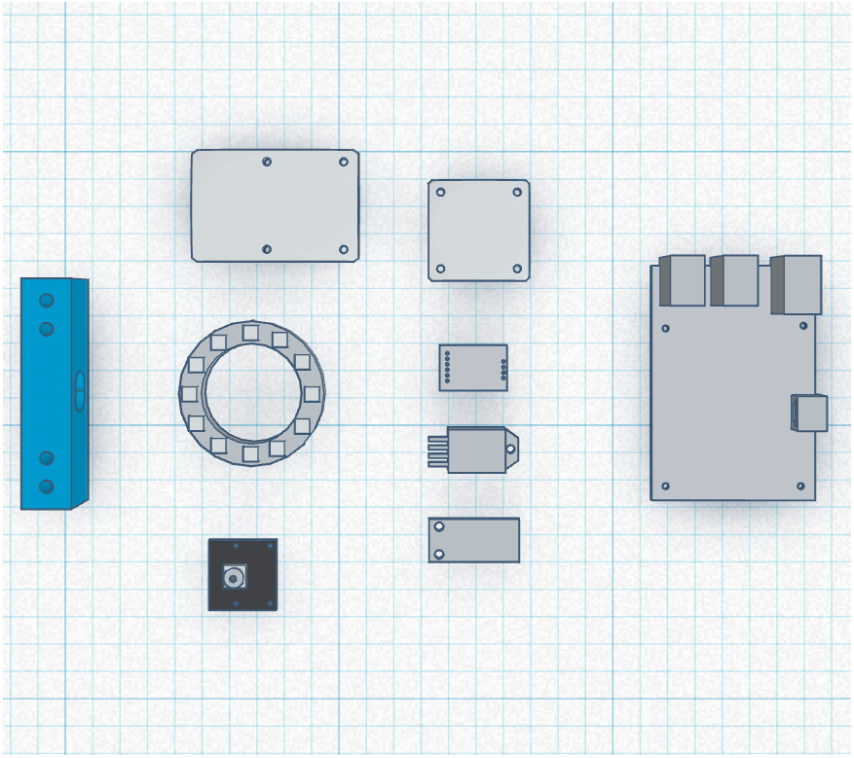
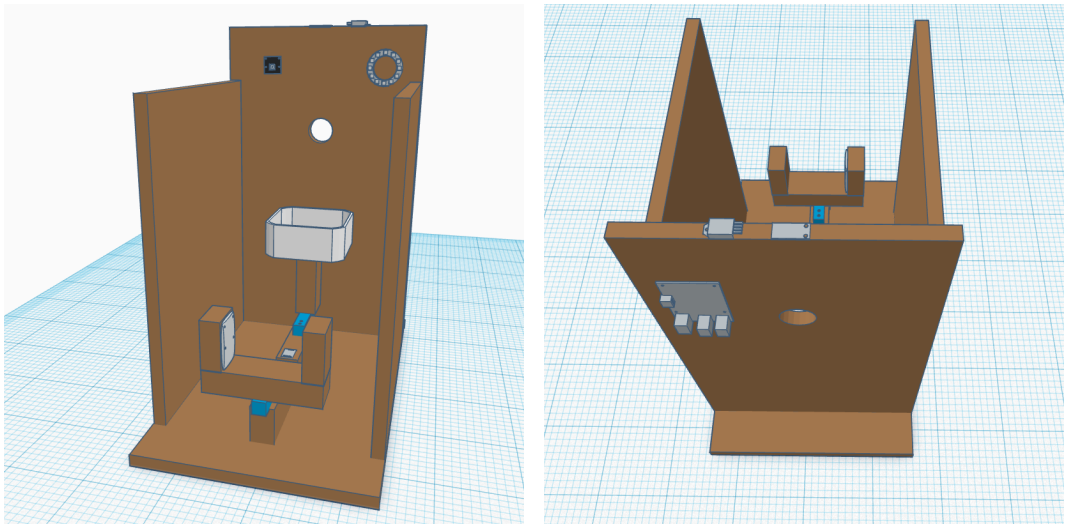


Figure 3: 3D model of the components included in the mechanical structure.

From top left to bottom right, Figure 3 displays the load cell, one model of the RFID, the LED ring, the camera, a second model of the RFID reader, the luminosity sensor, the humidity and temperature sensor, the AD converter and the Raspberry Pi. These are only simple representations used to identify each component and guide the overall assembly of the structure.

Figure 4 isolates the feeder station from the rest of the structure, as well as only displays the elements which were built in this project. The missing element from this figure, in comparison with Figure 3, is the DC motor and its food pipe, which is inserted in the hollow opening at the top center of the back division board. These elements are part of the chicken coop and are integrated with the structure in Figure 4 via a relay, electronically isolating both parts of the structure. It is also possible to note through this figure that the Raspberry Pi will be attached to the top back part of the structure, close to most of the components and the DC motor.



(a) Front view of the feeder. (b) Back view of the feeder.

Figure 4: 3D model of the feeder station including only elements assembled in this project.

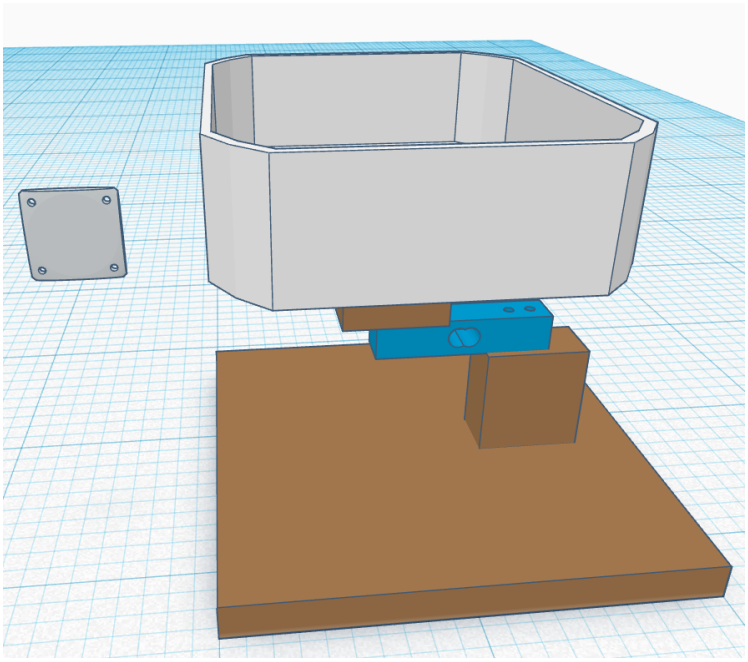


Figure 5: 3D model of the nest station including only elements assembled in this project.

The nest station in Figure 5 is actually a much simpler structure than what can be seen in Figure 2. That is because the nest itself, where the chickens will rest and lay their eggs, is already a part of the coop and was not built for this project. This element ensures that the eggs can slowly roll downwards via a very gentle slope, guiding these to the reservoir in Figure 5, which weighs the eggs. An RFID reader is also part of this station

and is used to identify which chicken just laid the egg that was collected and measured by the reservoir. The complete project is available for inspection at this link.

With the 3D model finished and understood, the physical assembly of the mechanical structure could begin. This involved the tools and techniques discussed in Subsection 3.1, as well as cooperation between members, given that this is a practical heavy work that most of the time requires more than one person to perform certain tasks. Figure 6 shows the final result of the assembly of the mechanical structure. Small adjustments were still made to this to allow for the hardware integration (small holes to allow cables to pass through more easily and to attach components).

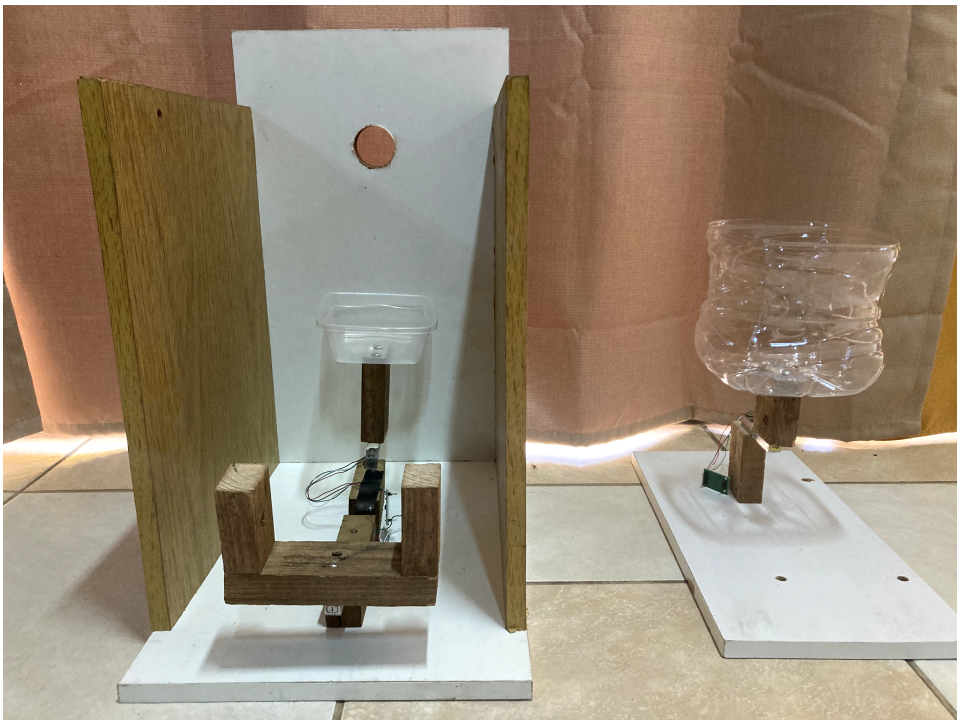


Figure 6: Finalized mechanical structure.

4.2 Embedded System

The embedded system is comprised both of the hardware and firmware elements of the project. Although both of these were designed and implemented separately before integration, the functionalities of each was always taken into account when working on the other.

4.2.1 Hardware

In combination with the firmware, the components for the hardware were gathered individually and their functioning was tested. At first, the Raspberry Pi needed to be set up and a Raspberry Pi OS (*Raspbian*) was installed in the system. Afterwards, with the data collected from each component's *datasheet*, the functioning of the modules was tested. This also involved welding cables to some of the items in preparation for the inte-

gration process to come afterwards — this was done using Ethernet cables and Dupont reed 2.54 connectors.

In particular, the load cells themselves needed to be integrated with the mechanical structure to be tested accordingly. This can already be seen in Figure 6. Several particularities of each component were recorded, such as the precision of the temperature and luminosity sensors (checking the temperature with a mercury thermometer and forcing the temperature to rise by pressing it with the finger, as well as enforcing darkness by enclosing the luminosity sensor). One other particularity involves the maximum distance at which the RFID reader can still recognize the tags, which was recorded to be approximately five centimeters.

The components and the Raspberry Pi were gradually added together to a protoboard as a prototype of the entire hardware system. After most of the parts were integrated, it became clear that a printed circuit board (PCB) would be necessary to properly integrate the components and not lead to disorder and confusion in the integration of the parts. Figure 7 displays how the hardware was arranged when using the protoboard for prototyping.

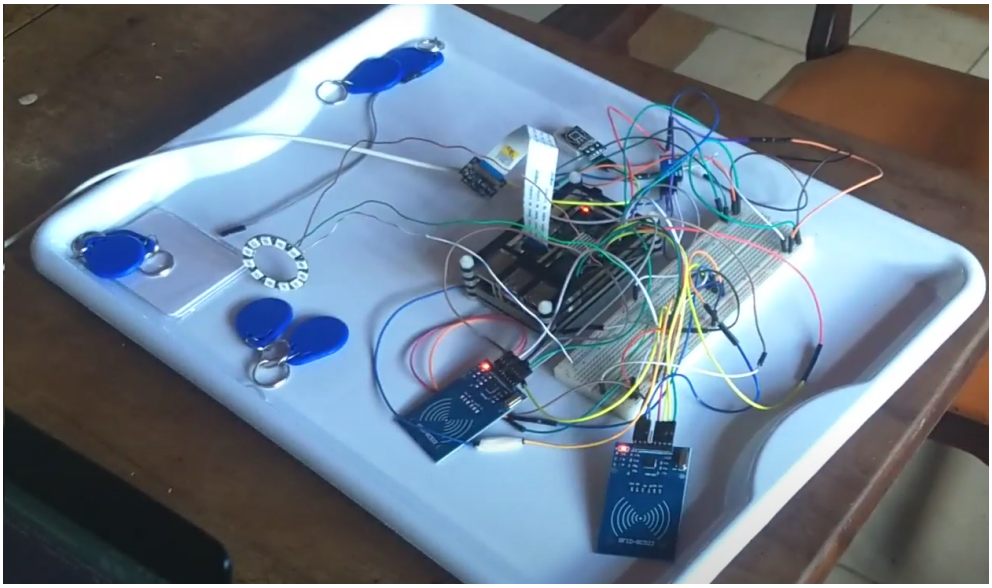


Figure 7: Initial distribution of the hardware components using a protoboard.

It is worth mentioning that the design and homemade development of the PCB was not a planned activity in the schedule of the project and lead to complications and delays in its execution — this will be better discussed in Subsection 5.4. To start the creation task of this board, the schematic was designed using the EAGLE software, followed by the routing of the PCB in a board sized 8x10cm. It was important to take into account the available space in the mechanical structure and the size of the Raspberry Pi when designing this board. The circuit was printed in a photographic paper and the ink was transferred to the board with the use of a common household flatiron. This process requested 9 tries until the result was satisfactory, as well as two failed attempts at corroding the routed PCB. The board was finally adequately corroded, drilled and the pin bars were welded to in the according positions.

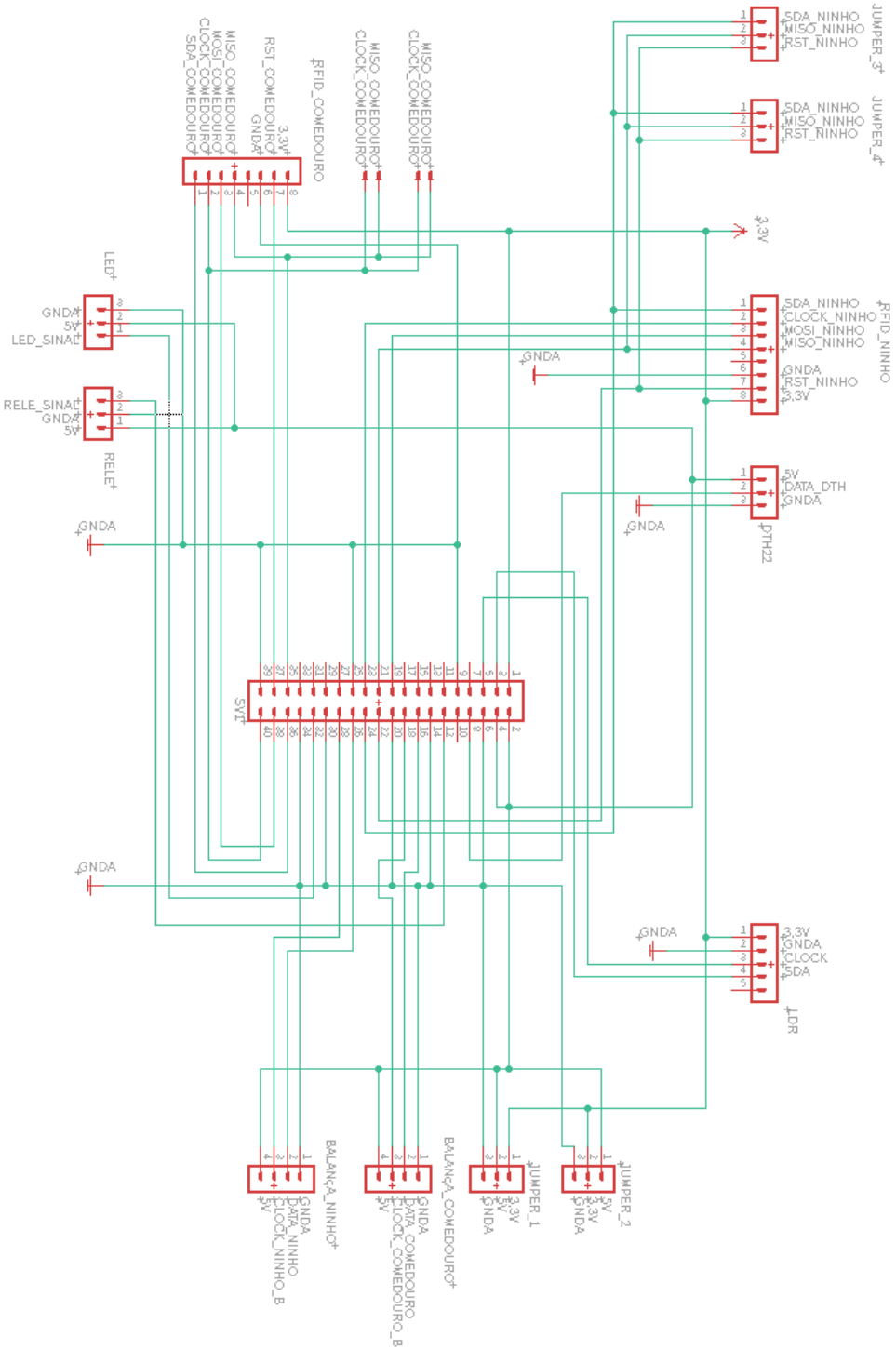


Figure 8: PCB schematic.

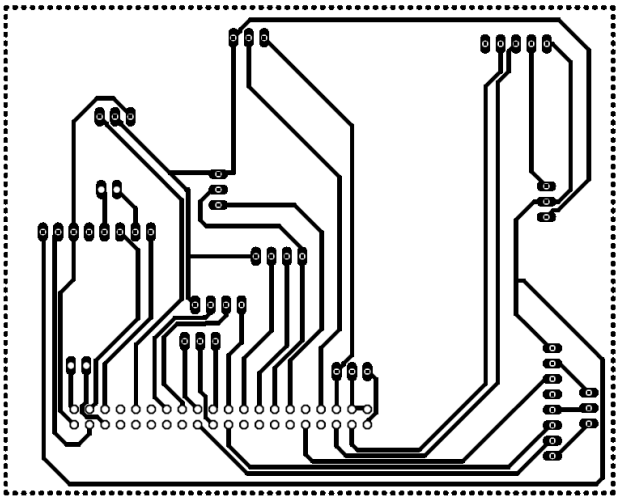


Figure 9: Routed PCB.

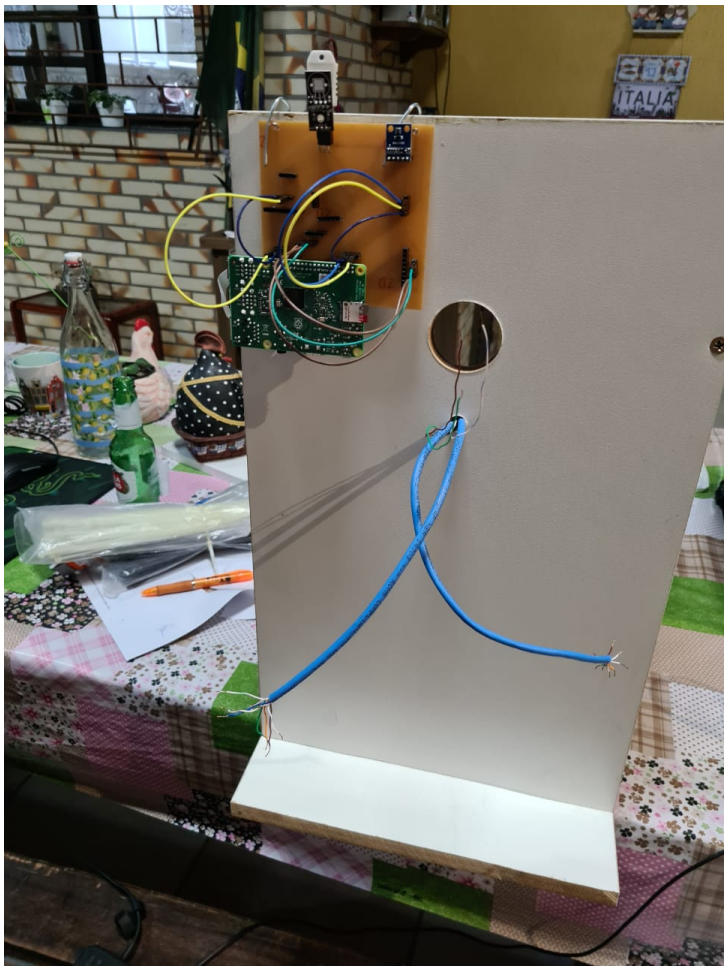


Figure 10: Finalized PCB integrated with mechanical structure.

4.2.2 Firmware

To develop the firmware, a Raspberry Pi 3b+ was chosen for the main control unit, and the Python programming language for its ease of use, integration and development speed (as was discussed in Subsection 3.2). Given the heavy reliance on libraries to handle the peripherals, Python version 2 was chosen instead of the newer, not-backwards-compatible, version 3, despite its approaching end-of-life to cut back significantly on development time. The decision paid off as only a few of the devices had Python 3 libraries available.

Before the effective implementation of the firmware could be started, a flowchart draft was created to visualize and define the program's operation, all based on the project's ideas and objective. After some revisions with the help of the field specialist and sponsors, the diagram on Figure 11 was finalized. There are some details given by the field specialist that guide the implementation of this part of the project. First, the chickens have an improved vision over the humans, with over one hundred times the amount of rod cells in their eyes. For this reason, LED lights will be used in their conditioning — whether they are allowed to eat (indicated with a red light) or not (indicated with a blue light). Furthermore, following the principle of "least discomfort" a chicken would not stay on the perch if it does not need to or does not have food to eat. It also holds food in their gizzard, which means that it does not immediately ingest the food it collects, but also reserves some for later. Finally, it is important to note that some of the behavior of the animals is actually object of study, which the project will serve for, helping understand better how the animals react when coexisting with a system such as ChickenIO.

According to the chart in Figure 11, the program functions by constantly checking if an RFID tag was detected near the feeder by doing a series of checks to ensure a chicken is really on the feeder waiting for its food. This constant check is essential, since the behavior of the animal can be unpredictable. Given that a chicken has rested on the perch and was correctly detected, the system tests the connection to the back-end REST API. If the chicken could not be correctly identified, the system returns to the beginning state and increases a fail counter. Once the fail counter is high enough, it engages a default operation sequence which simply disperses a standard amount of food. This is done as a security measure to prevent the animals from starving in case of a problems in the system.

Following a successful recognition, the fail counter is reset, and a LED light is turned on. In the feeder, these lights are used not only to show the system's current major state, but also to condition the chicken into a desirable behaviour while interacting with the system. While the light colors were chosen in the diagram for identification of state and ease of testing, the final implementation makes sure that these colors are replaced by the red and blue lights only — which the chickens more easily respond to, as mentioned before.

After having its RFID detected and the connection test to the back-end is successful, the feeder asks the server if that specific chicken can actually eat at that time. If not, it returns to its original state. If yes, it then starts to measure the weight of the chicken by doing several readings of its scale. If the weight is found to be too inconsistent between these measures (which can be cause by the chicken leaving or moving too much on top of the perch), the system is returned to its initial state. If the weight is sufficiently consistent,

it does a pair of operations with the database: posting the chicken's info and getting the correct amount of food to dispense for that chicken. Ultimately, the food is dispensed into the bowl until its scale measures the correct amount. The system then waits for the chicken to eat and leave by reading its RFID tag until it is no longer detected. It then saves the leftover amount to the database and returns to its original state.

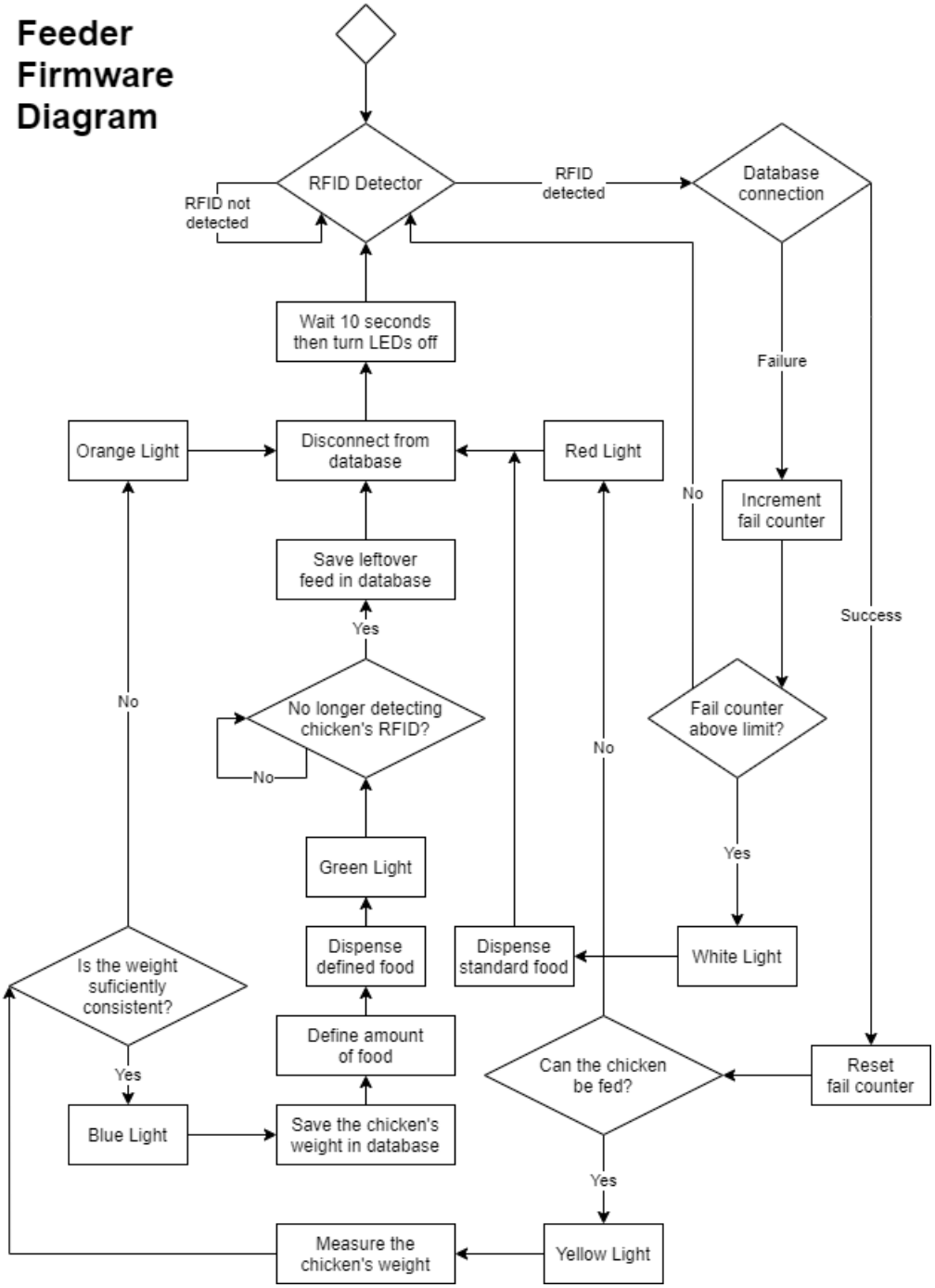


Figure 11: Diagram for the firmware's feeder operation flowchart.

The operation of the ambient controller is much simpler, and revolves around reading environmental data periodically and sending it to the database. To not overload the Raspberry Pi, the system alternates between doing so and taking pictures of the location for surveillance, as requested by the client.

Finally, the nest works similarly to the feeder, as it reads an RFID tag, waits for the chicken to leave by checking to see if that RFID tag is still present. At the same time, the difference between the starting and end weights in its scale are also being constantly measure. If changes are detected to this weight, it means that the lastly detected chicken just laid an egg, and its weight is saved to the database. This functioning is displayed in a diagram in Figure 12.

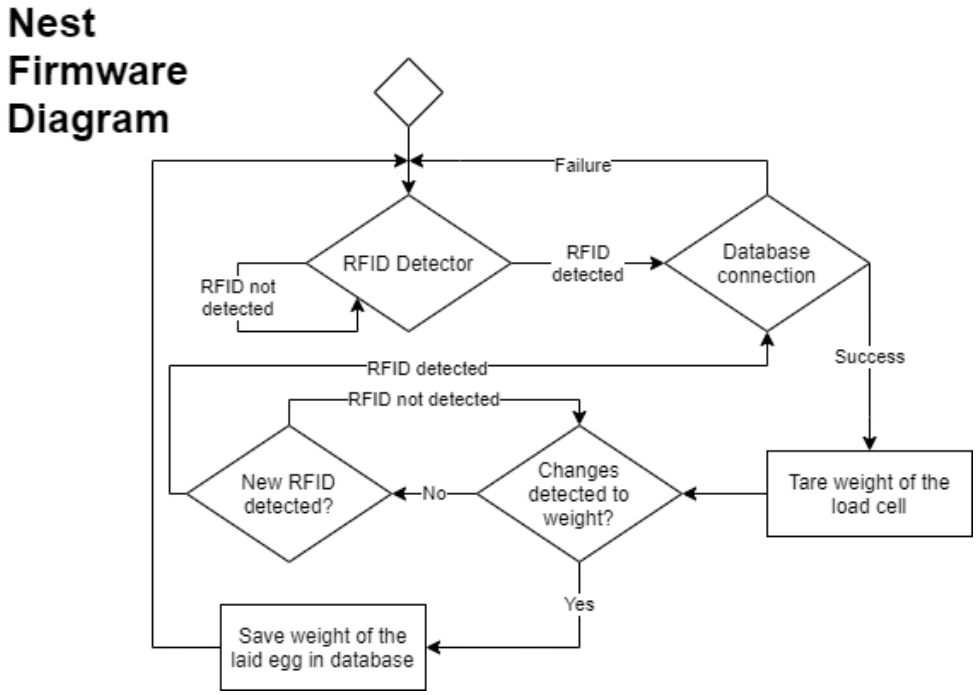


Figure 12: Diagram for the firmware's nest operation flowchart.

The actual work on the firmware's code was initiated after most of the peripherals were gathered and assembled on the Raspberry Pi. The adequate Python libraries for these components was acquired, downloaded and tested for the particularities of the project, assuring their functionality. After several tests and adjustments to the libraries, each individual piece was running and enable the following step: coding the program's logic.

The basic functioning of the firmware was completed first with blank, empty function calls that returned placeholder values. After testing for logic and consistency, the actual implementation with the real peripheral functions from the libraries was started. This proved to be a challenge, as heavy modifications and adaptations to the libraries had to be made to account for changes in the assembly and specially the pin-out of the Raspberry Pi. After much time spent on testing, refining and modifying the code for each iteration of the hardware, the finished, fully functional product was achieved.

4.3 Web Application

Similar to the embedded system, the web application is divided into two parts, the back-end and the front-end. Both of these were developed separately, but, as mentioned before, required a strong regard for the other when implementing their functions. Furthermore, the back-end element is also linked to the embedded system (as seen in Figure 1), which needed to be kept in mind to allow for a proper implementation and make the integration process was intuitive.

4.3.1 Back-end

The back-end was developed using *Node.js* and uses the rest API architecture. When the API receives a JSON with an existing address that may or may not have parameters, it makes an HTTP request for the server which gives an answer based on the parameters and the address provided. The address dictates what is being requested by the client and the parameters are used to specify the content of that request. When the server receives the HTTP request and the parameters, it uses that to build a query and execute it on the database and send as an HTTP response the result of that query. The API then take that result and send to the client as a JSON.

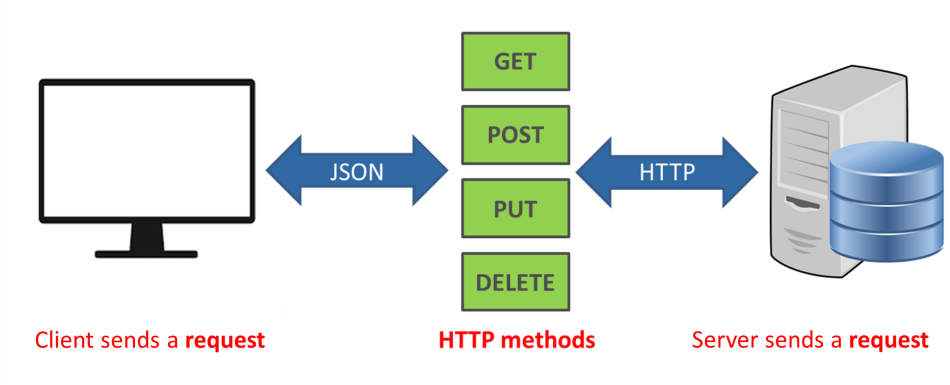


Figure 13: Diagram for rest API architecture.

The requests the client can make are used to read, create, update and delete information stored on the database. Each one of those methods have their own way of being called. To read information the GET method is called passing the path and the parameters is they are needed. All the methods work similarly, DELETE method to delete, PUT method to update and POST to create.

The request is made by choosing the correct method and passing the path with the parameters in case the path accepts any. The following image represents the entire flow. We need to send a JSON using the DELETE method and pass the path as /users/destroy/id, ID being the parameter.

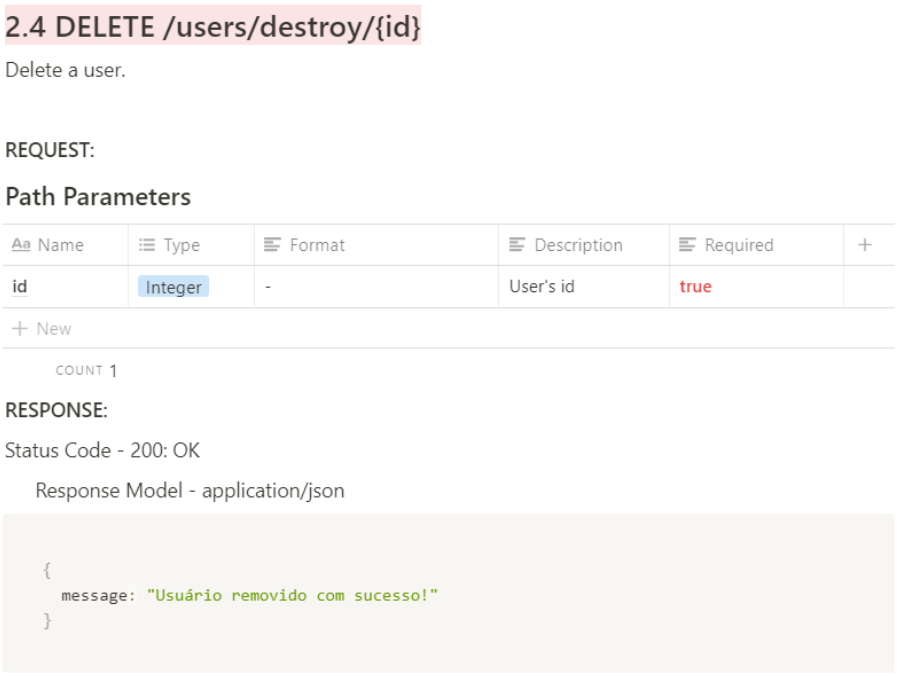


Figure 14: Example of a request.

4.3.2 Front-end

The front-end of the Web Application was developed using *Vue.js*, a framework built using Javascript language that is used for developing single page applications. During development it was noted that an interface in the form of a dashboard was the most appropriate template for the objectives of the project.

The Dashboard is where the user is able to monitor data obtained from the physical elements of the system present in the farm, that is stored in a cloud server and can be accessed from anywhere. The user can also register new chickens and delete chickens from the database. All these tasks are performed by a single user, a use case diagram can be seen in Figure 15.

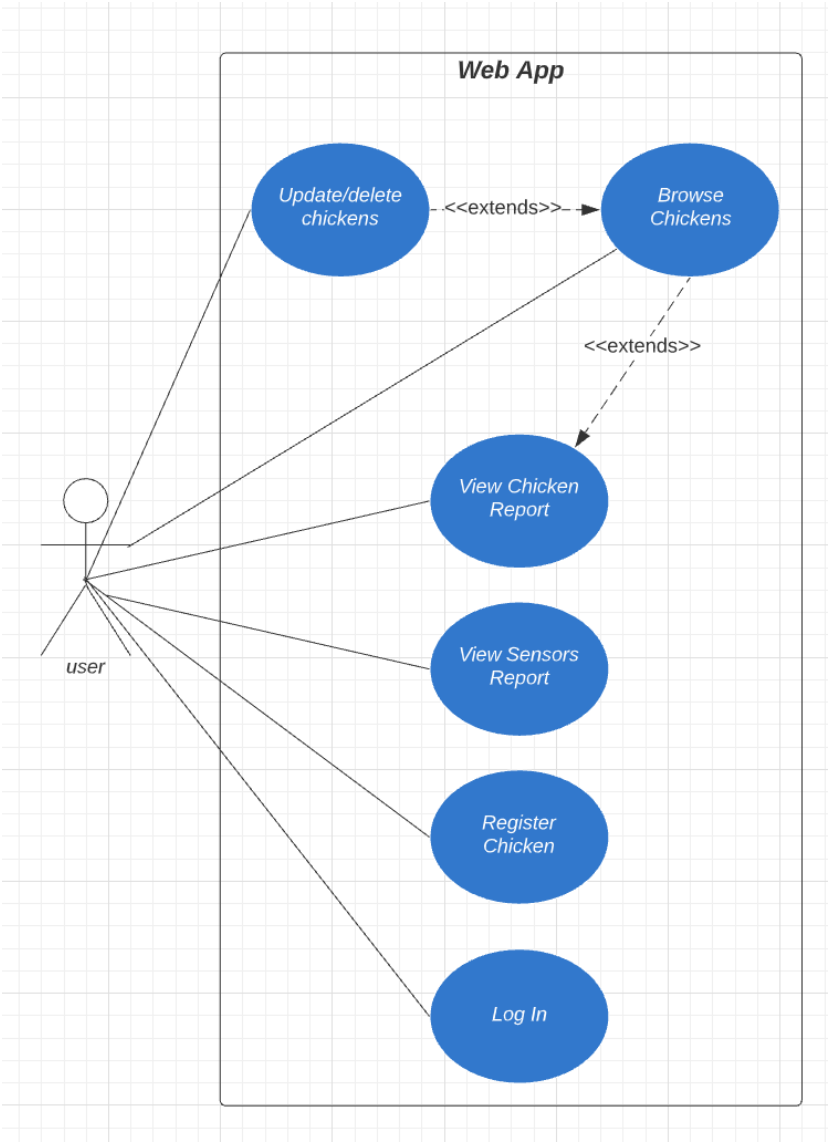


Figure 15: Use case diagram of the web app front-end

For the sensors report, the information that can be visualized includes: data obtained from temperature, air humidity and luminosity sensors, as well as weight readings from the meal and chicken scales. The interface also presents detailed reports for each chicken including their current weight, weight reading history, how many meals per day, how much grams of feed each meal should have for the specific chicken, and their RFID tag number.

A Vue.js app is made up of Vue Components, inside each component, in the same file, it is possible to define its layout, also known as template, written in HTML, its functionality (scripts, queries, data, etc), written in Javascript, and its style (alignments, colors, fonts, etc) written in CSS. These components can be assembled in a modular and hierarchical way, where a single component can contain multiple components. A parent component that can be routed through the application is called a view, or page.

In this project the components were organized in a structure composed of a main view with a side bar with links for sub-views nested inside the main view, where more components present the data or can be interacted with by the user. The routing through these views is performed by a client-side router and its structure is presented in Figure 16, where the structure of the components inside the sub-views can also be seen.

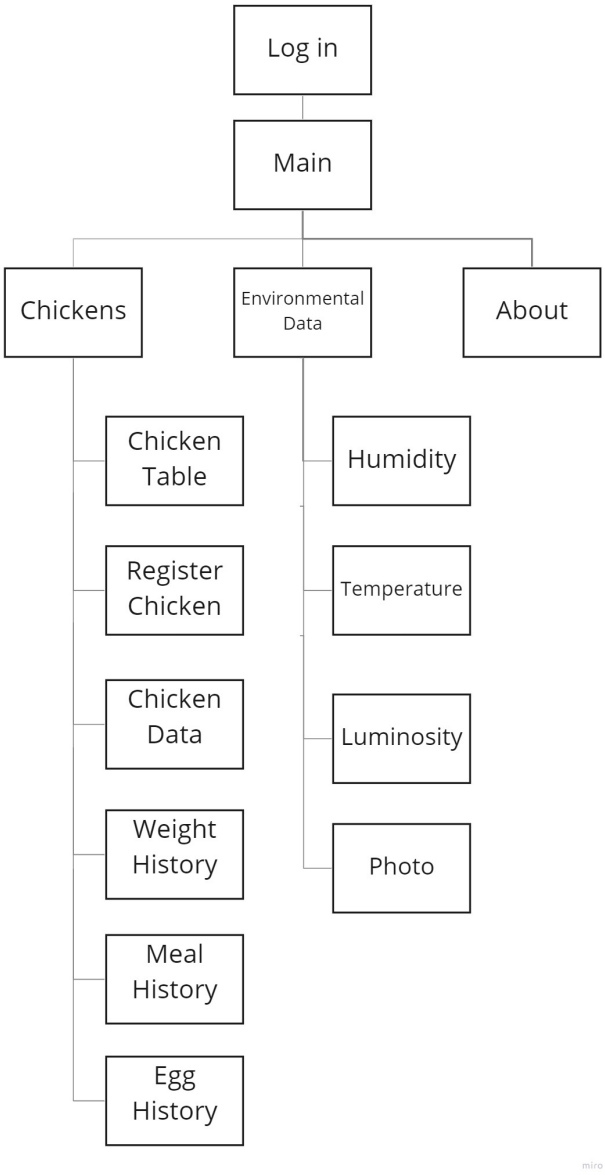


Figure 16: Sitemap diagram of the web app front-end

Inside each component are the methods for the functionalities of the app, such as making a request to the API for sending or receiving data. For the specification of the structure of the app, each of these components were treated as an entity. The sequence diagrams of all the actions the user can perform and how the application behaves are presented in Figures 17 through 23.

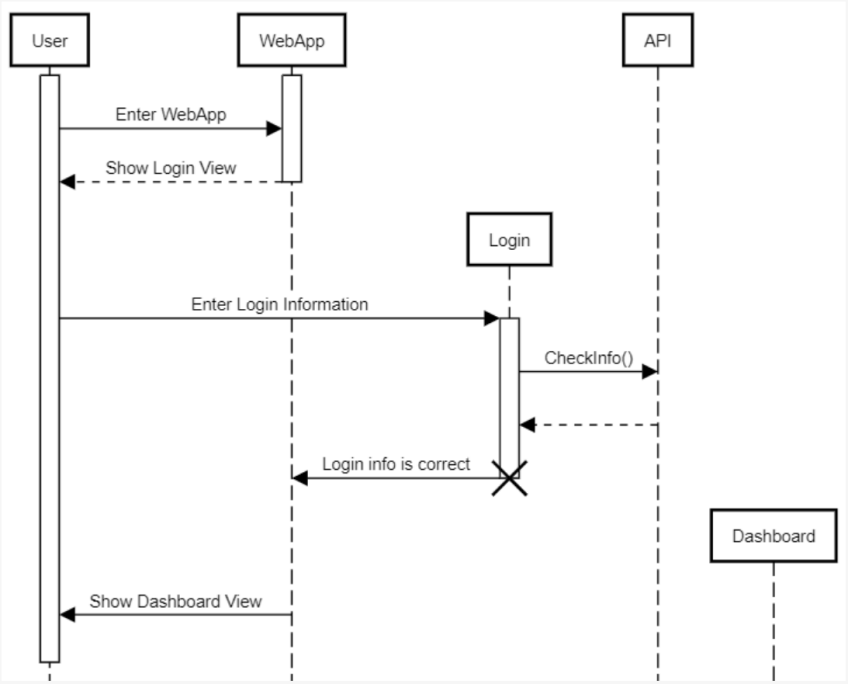


Figure 17: Sequence diagram for when the user logs in.

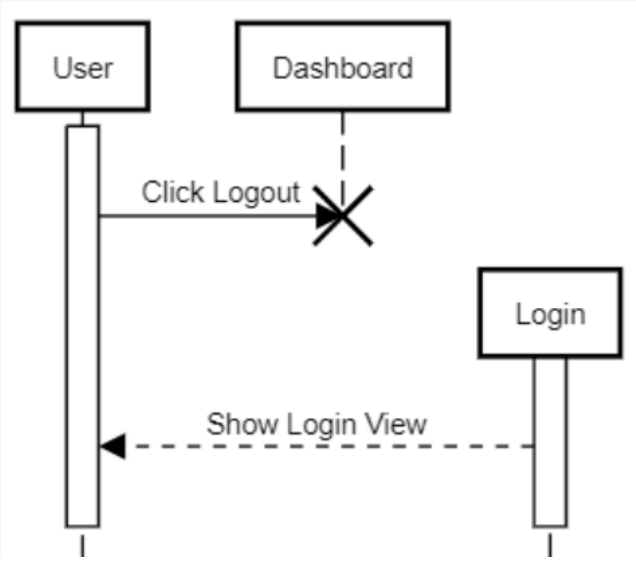


Figure 18: Sequence diagram for when the user logs out.

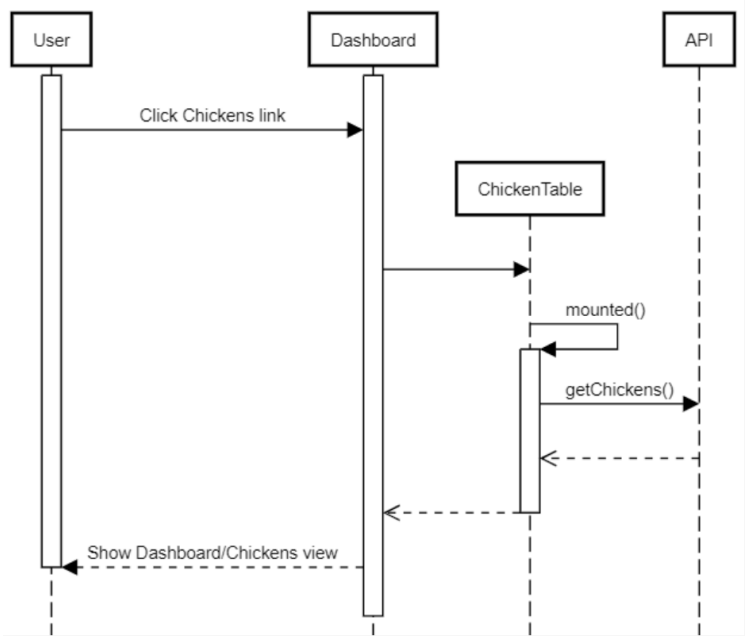


Figure 19: Sequence diagram for when the user selects the chickens view.

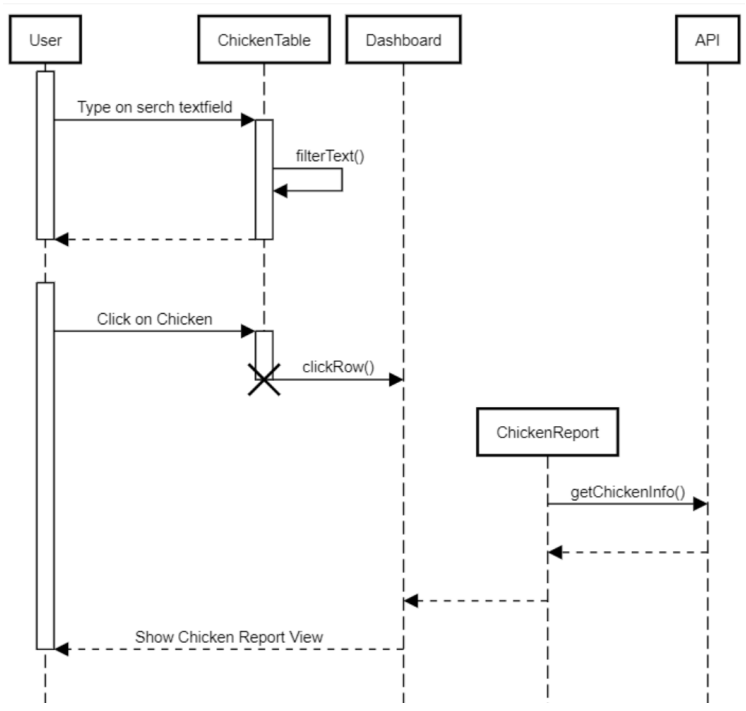


Figure 20: Sequence diagram for when the user selects a chicken from the chicken table.

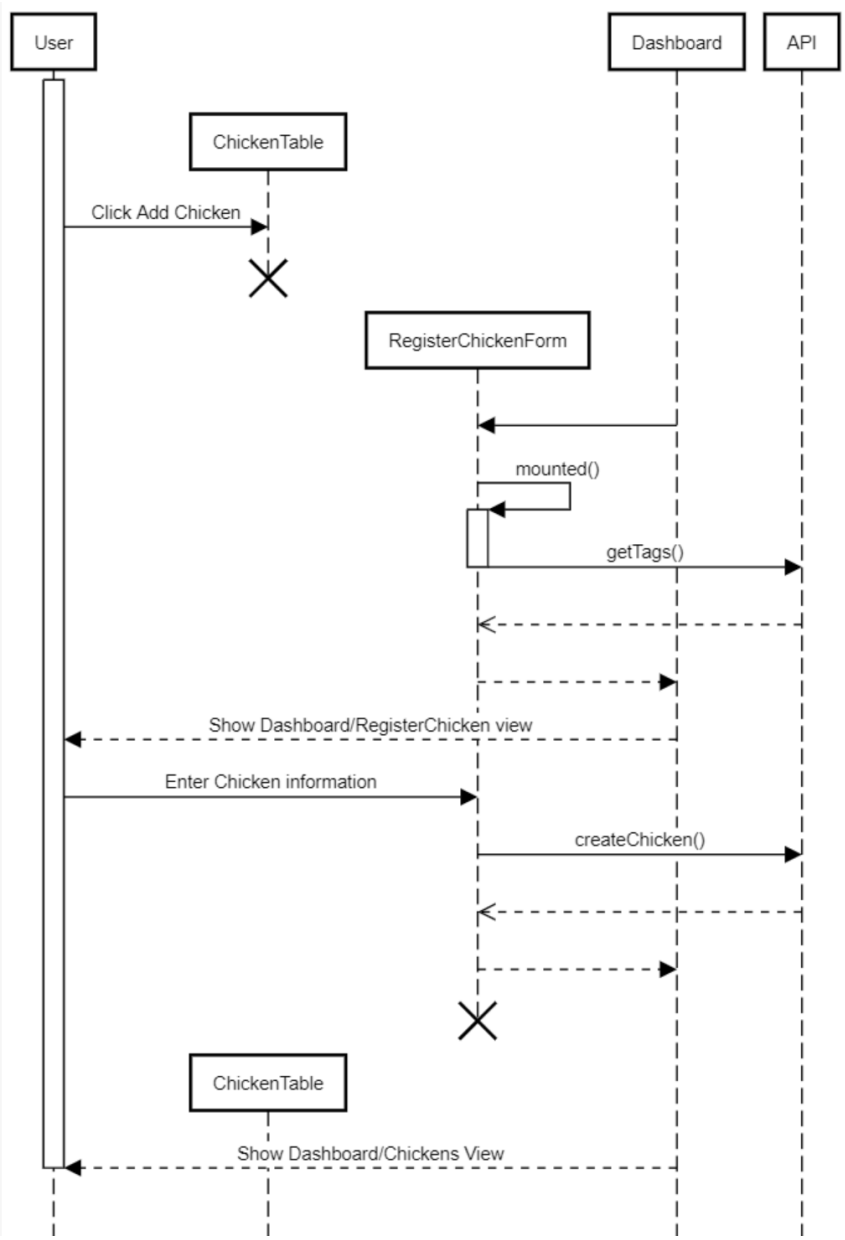


Figure 21: Sequence diagram for adding a chicken.

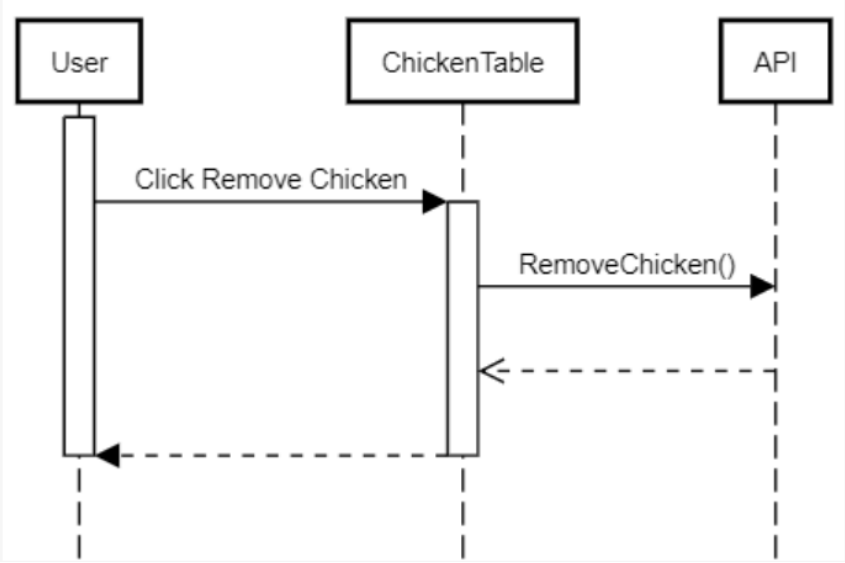


Figure 22: Sequence diagram for removing a chicken.

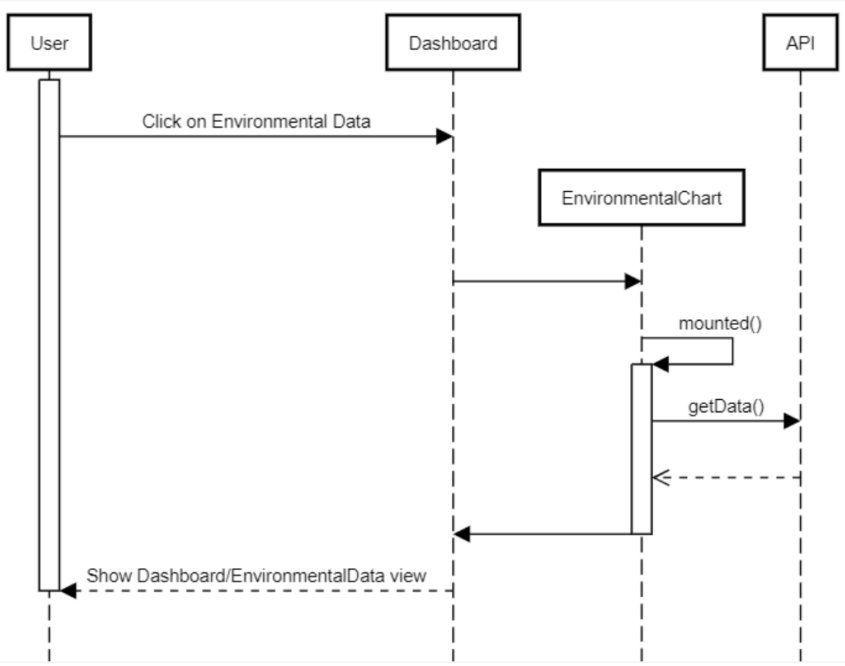


Figure 23: Sequence diagram for when the user selects the environmental data view.

After planning and implementing the platform locally, the Web Application was hosted on a cloud server, using the *Heroku* provider (as mentioned in Subsection 3.4). The results of the development can be seen in Figures 24 through 27, where the different screens available to the user are displayed.

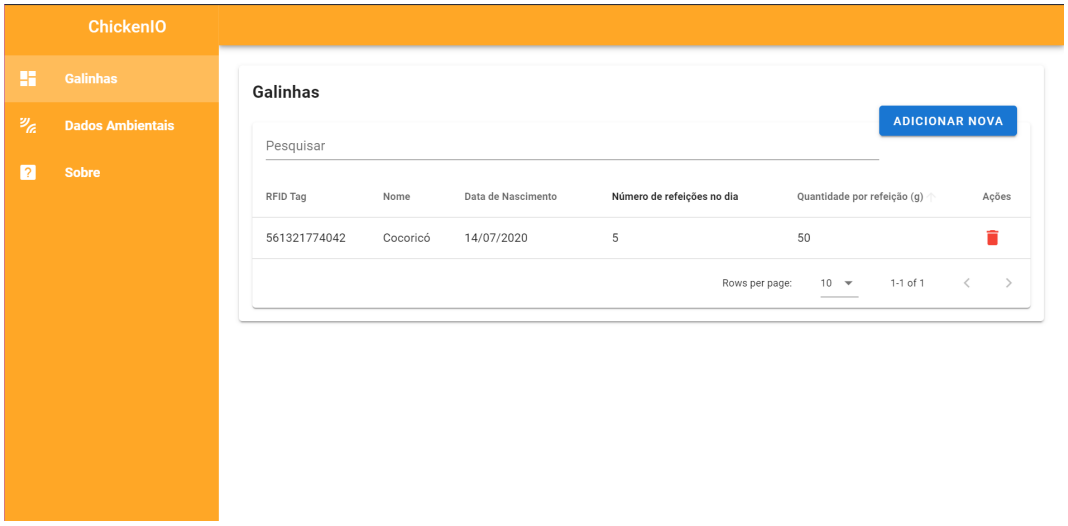


Figure 24: Chickens view.

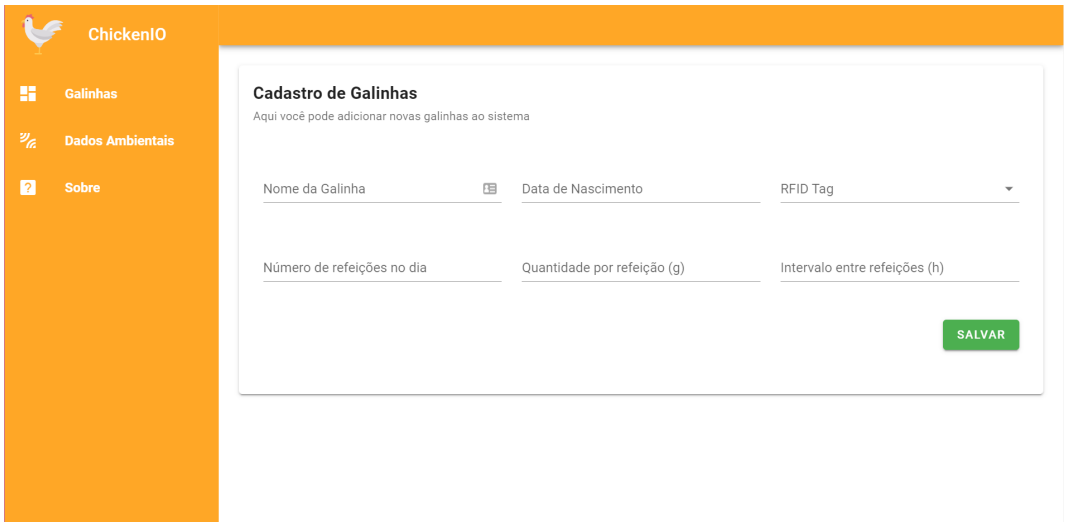


Figure 25: Register chicken view.

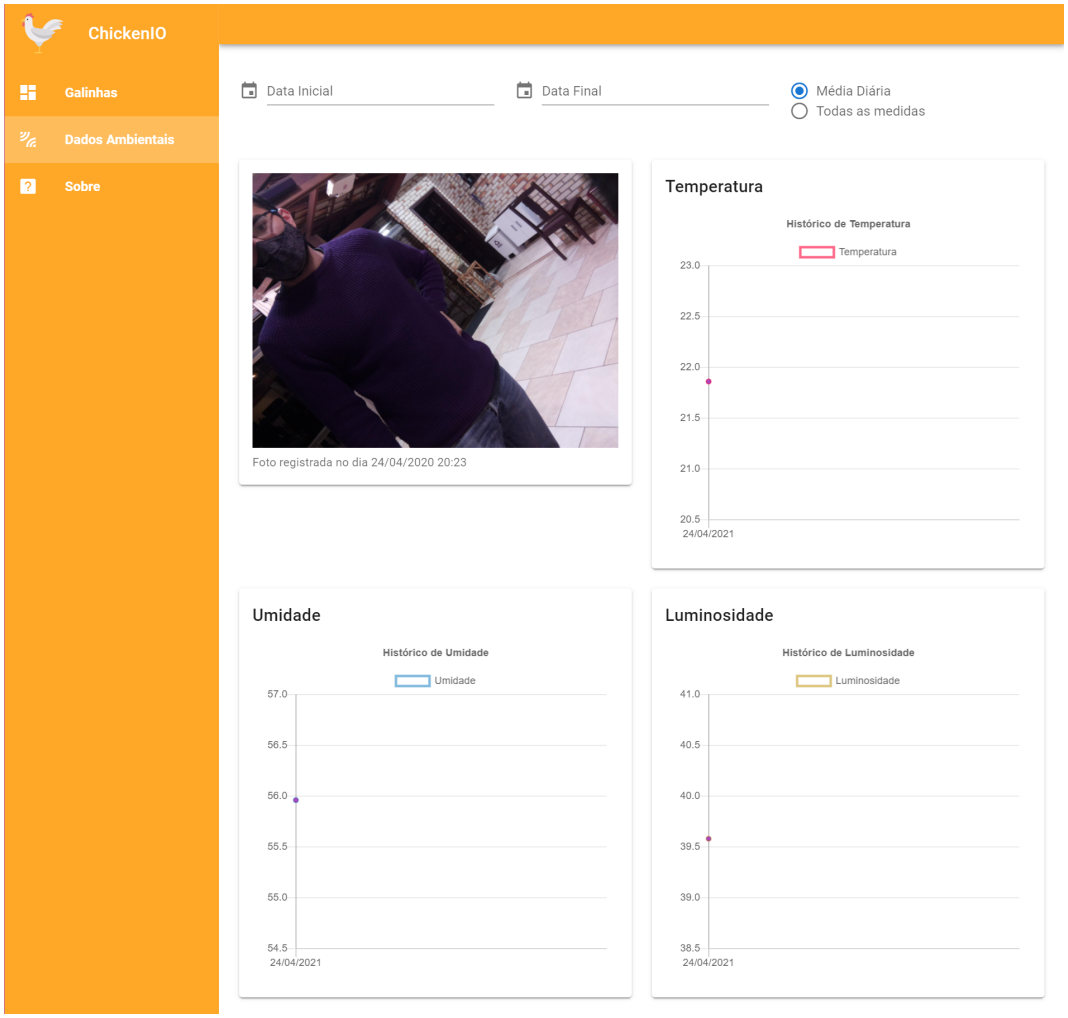


Figure 26: Environmental data view.

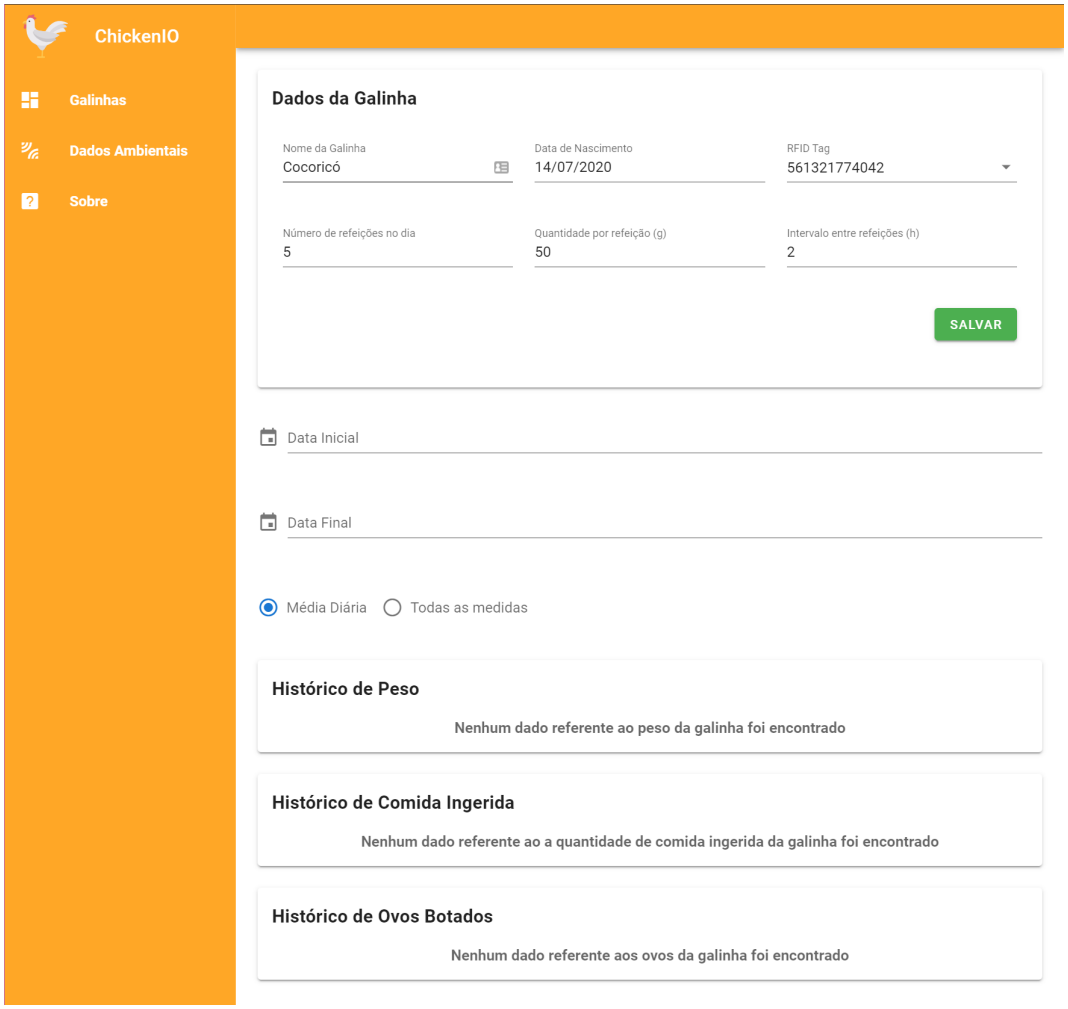


Figure 27: Chicken Report view.

5 Results and Conclusions

This section gives an overview of the results of the project, discussing the conclusions and outcomes of activities planned in the first stages and how they evolved throughout development. Subsection 5.1 outlines the tests and demonstrations of the project; Subsection 5.2 briefly discusses the budget of the project; Subsection 5.3 analysed the projected schedule over what was in fact executed; Subsection 5.4 highlights the difficulties in project development; and finally Subsection 5.5 concludes the project.

5.1 Tests and Results

All tests proved the system to be fully functional and integrated. The front-end could indeed interface and communicate with the back-end and vice versa. The web API server could be put online and still interact with the other physical systems over the internet through an URL. Its worth noting that the system was not tested on site and with real chickens, so more testing and adjustments may be needed for the system to work in its full potential.

The following videos were made as demonstrations of the integration of the system as well as testing its complete functionality.

[Preliminary test of the feeder]

[Final demonstration of the complete system with front-end interface]

5.2 Budget

A brief overview of the projected budget and the actual value of the components can be seen in Figure 28.

Component	Estimated Value	Actual Value	Acquired	Buyer
Raspberry Pi 3	R$329.00	R$270.00	☑	Gabriel(already had)
DHT22 Humidity and temperature sensor	R$60.00	R$54.60	☑	Gustavo
HX711 AD Converter (x2)	R$50.00	R$60.00	☑	Gustavo(already had)
MRFC522 RFID Reader (x2)	R$60.00	R$55.00	☑	Gabriel
Physical Structure (Wood)	R$50.00	R$43.00	☑	Vinicius(already had)
Pi NoIR Camera V2	R$110.00	R$58.00	☑	Borrowed (Prof. Fabro)
LED	R$40.00	R$43.68	☑	Gustavo
Luminosity sensor	R$8.90	R$20.90	☑	Gustavo
Scale (x3)	R$75.00	R$88.25	☑	Gustavo(already had 2)
Basic Tools	R$40.00	R$55.00	☑	All
Basic Materials	R$20.00	R$25.00	☑	All
COUNT 11	SUM R$842.90	SUM R$773.43		

Figure 28: Budget overview

5.3 Schedule

Table 4 presents a summary of the schedule for the project. Most importantly, this table aims to compare how the development was planned and the number of expected

hours with how the development actually happened and the number of effective executed hours. Refer to this link to the complete schedule of all tasks executed throughout the weeks.

Table 4: Summary of schedule

Section	**Planned**			**Executed Hours**
	Scheduled Hours	**Safety Margin**	**Total**	
Project Plan	38h	9h30	47h30	44h
Blog/Site	10h	3h	13h	12h45
Mechanical Structure	19h	5h	24h	36h15
Hardware	57h	14h45	71h45	98h75
Software	147h	37h10	184h10	144h25
Hardware Software-Integration	29h	7h45	36h74	28h30
Remaining Integration	11h	3h20	14h20	19h
Adjustments & Documentation	55h	14h20	69h20h	75h45
Presentations	25h	7h	32h	?
		Total:	**492h30**	**433h20**

5.4 Problems and Challenges

The main, most prevalent challenge throughout the entire development process was the COVID-19 pandemic severely limiting the ability to meetup and work together in person to assemble, test and discuss the project. Other problems included failures in converting our physical prototype to the final model, specially from the protoboard to the printed circuit board.

5.4.1 Printed Circuit Board

Most notably in the project, the PCB creation proved to be troublesome. All members had long not gone through the process of preparing a homemade PCB, which led to mistakes and problems. Before the final board was finalized, eight failed attempts at transferring the ink to the board were scrapped — one of which can be seen in Figure 29 —, including two attempts at corroding the board itself and not reaching satisfactory results.

This task was already something not planned for in the schedule of the project, which already is an issue in it of itself. Furthermore, given the recurrent errors, the task also took several hours more than the expected, leading to significant delays in the schedule. To prevent impairment of the project's functionalities, the workload of the members was increased during development to accommodate this tasks extra hours.

Figure 29: Failed attempt to transfer ink to copper board.

5.5 Final Considerations

In the end, the project proved to be successful and fully functional, as seen in the demonstration videos. It is worth mentioning the effectiveness with which the project was integrated, since this was something the members prepared for since the beginning and worked together to facilitate collective work.

Much about how to develop a real-life project was learned, mixing concepts of not only the members's area of expertise, but also being apply to apply the concepts and knowledge to a completely separate domain of work. Additionally, overcoming the many challenges that come with making a system evolve from a theoretical and simulated environment to physical and practical was a key point in development.

References

[1] J. C. Buzby and H. A. Farah, "Chicken consumption continues longrun rise," tech. rep., 2006. Available at https://ageconsearch.umn.edu/record/126587.

[2] S. Begley, "The rise of chicken," TIME, vol. August 08, 2016, 2016. Available at https://time.com/4428006/the-rise-of-chicken/.

[3] A. Hamid, A. Ahmad, and N. Khan, "Respiratory and other health risks among poultry-farm workers and evaluation of management practices in poultry farms," Brazilian Journal of Poultry Science, vol. 20, no. 1, pp. 111–118, 2018. Available at http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-635X2018000100111&lng=en&tlng=en.

[4] J. Salatin, Pastured Poultry Profits. Polyface Titles Series, Polyface, 1993. Available at https://books.google.com.br/books?id=vPEJAAAACAAJ.

[5] J. Perry, D. E. Banker, and R. Green, "Poultry production in the united states," tech. rep., 1999. Available at https://www.ers.usda.gov/publications/pub-details/?pubid=42211.

[6] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, pp. 35–46, 2000.