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 – 2021/2

Technical Report
DeliveryBox

Damon N L Oliniski – damon@alunos.utfpr.edu.br
Flavia G. dos Santos – flaviasantos@alunos.utfpr.edu.br
François B. Rueckert – francois@alunos.utfpr.edu.br
Gabriel C. Guerrero – guerrero@alunos.utfpr.edu.br
Matheus V. B. Turatti – turatti@alunos.utfpr.edu.br
Murilo M. Guimarães – muriloguimaraes@alunos.utfpr.edu.br

December of 2021

Abstract

This document reports the development of the DeliveryBox, a solution for a safer method of delivering food. The proposal revolves around the rise in complains about deliveries from apps already in the market. DeliveryBox consist of a box that is locked and monitored during the transit, constructed to help each part of the equation (deliverer, restaurant and client) feels more safe when delivering food. Alongside that, a web application was developed to give to each user, the tools necessary to buy, sell, and accompany the order from the door of the restaurant all the way to the final client. This document reports the development as well as its principal results and difficulties.

1 Introduction

During the COVID-19 pandemic, the money spent on delivery apps grew by 149% [1], and that growth came with a problem: the integrity and safety of the food. The increase in the amount of delivery orders occasioned an increase in the complaints about the service itself. As you can see in [2] the increase in complaints goes by 343% in 2020, when compared to the same period of 2019. Most of the complaints come from wrong deliveries, improper billings and delivery delays, and are in those problems that the DeliveryBox intends to solve.

The solution proposed consists of a couple of modules integrated with one another: there is the physical box itself, that locks the food, takes pictures of the food once locked and sends it to the client, gives it's geolocation for tracking of the order and evaluates the driver performance. There is a web application that has three different roles: restaurant owner, deliverer and client. The web application works differently depending on the role assumed by the user, giving options to make orders, create dishes, check the wallet, check the map, lock and

unlock the box, among other things. All these functionalities are coordinated by a server deployed on Google Cloud Services. An overview of the solution is presented in Figure 1.

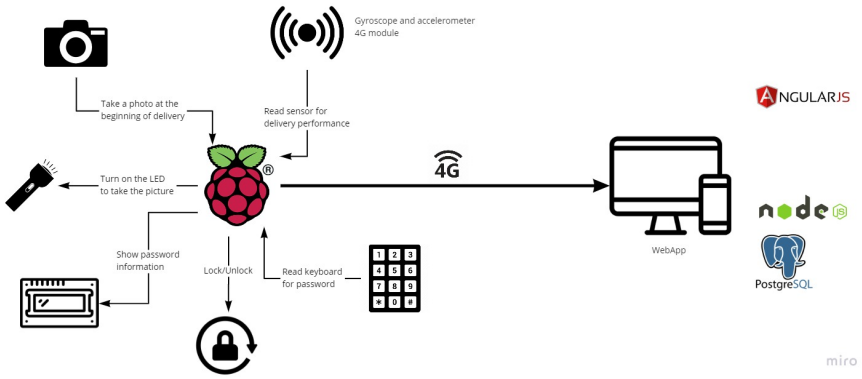


Figure 1: Block Diagram of the System

To make a transaction with DeliveryBox the first step is for the user to sign in on the web application. Upon register, some credit is given to the user. Then he will need to search for the restaurants available at the moment. He will then select one or more dishes, making an order, and select to complete the order. The server will check if the user has sufficient credit and, if he has, will create a pair of passwords and alert the restaurant. When the restaurant finishes the preparation of the order, it will press a button that will make a pool of the available deliverers, searching for the closest one. The deliverer will receive an alert and the map to the restaurant. Upon arrival, the restaurant will have two options to unlock the box: putting the code on the keyboard or clicking a button on the web application. Once unlocked, the restaurant can put the order inside, and when locked again, a picture is taken and sent to the user. After taking the picture, the deliverer and the client receives a map that shows the current location of the deliverer and the destination. Similar to the restaurant, the client can choose to unlock the box using a button or the password on the keyboard. When the deliverer reaches the destination, a timer is started. If the client does not appear by the time limit, the box is unlocked automatically. This function was planned to avoid a client causing maleffices for the deliverer. Upon unlocking the box, the server makes the confirmation of the end of the transaction and sends the credits from the user to the restaurant and deliverer. Upon ending the transaction, the performance of the deliverer is calculated using information collected through the gyroscope along the delivery.Each role and their functions and responsibilities will be explained in more detail in further sections of the technical report.

The remaining of the document is organized as follows. Sections 1.1 and 1.2 presents the main requirements of the complete project. Section 2 presents a

brief description of the technologies used in the development. Section 3 presents the complete development of the mechanical hardware (Subsection 3.1), the electronic hardware (Subsection 3.2), the firmware (Subsection 3.3), the web application (Subsection 3.4), the map and geolocation (Subsection 3.5) and the communication (Subsection 3.6). Section 4 presents the principal difficulties on the development, Section 5 the principal results and finally Section 6 presents the conclusions for the project.

1.1 Functional Requirements

The Functional Requirements of the DeliveryBox Solution can be subdivided on the Web Application and the Box. The Web Application itself can be subdivided into the three planned roles and the transaction use case.

1.1.1 Web Application

1. The system must offer a website that can be accessed in any browser, including those on smartphones (for Deliverers and Clients).
2. The website must allow three category of users (Restaurant Owners, Clients and Deliverers).
3. The website must have a registration/login/logout page for the users.
4. The website must have a wallet.

1.1.2 Web Application - Restaurant

1. The website must allow 'restaurant' users to register a store (name of the store, valid address for pick-out orders, days of the week when the store is accepting orders and time of opening/closing for each day).
2. The 'restaurant' may register items for sale.
   (a) Items for sale must have a picture, a name, a virtual price, a description and be available for sale.
3. The system must give the 'restaurant' a button to alert that the order is ready to be taken.
4. The system must give a wallet that accumulate the sums of the sells of the store.
5. The system must register in the restaurant wallet the accumulation of the sells of the store.

1.1.3 Web Application - Deliverer

1. The website must allow 'deliverer' user to set their availability.
2. The system must allow 'deliverer' user to connect with a box.
3. The system must show a map to the 'deliverer' current location and destination.
4. The system must credit the 'deliverer' virtual wallet when a delivery is finished.
5. The system must evaluate 'deliverer' performance when a delivery is finished.

1.1.4 Web Application - Client

1. The website must allows the registration of a 'client' with the following information: name, city, address and telephone number.
2. The website must give 'client' some credit upon the account's creation.
3. The 'client' user may browse through available stores.
4. The 'client' must only see stores from their city.
5. The system must provide a 'shopping cart' for the 'client'.
   (a) The 'shopping cart' must contain items from a single store.
   (b) The 'shopping cart' must delete all items stored when shop items from another store are inserted.
   (c) The 'shopping cart' must have a total value.
   (d) The 'client' may purchase the 'shopping cart' items.
   (e) The system must check if 'client' has more credit than the total value.
   (f) The 'shopping cart' must have a button to start the delivery
6. The System must provide a map with the 'deliverer' current position to the 'client'.

1.1.5 Web Application - Transaction

1. The transaction must send a warning to 'restaurant'.
2. The system must select the closest available 'deliverer' when the order is ready to be taken.
3. The system must provide a code to lock the box to the 'restaurant'.

4. The system must provide a code to unlock the box to the 'client'
5. The system must fail if the user does not have an address registered.
6. The website must redirect to the store in case of fail of transaction.
7. The transaction must be finished with an unlock of the box.
   (a) The 'client' may unlock with a button on the website.
   (b) The 'client' may unlock by inserting the code they receive into the keyboard.
   (c) The box may unlock itself by arriving and staying in the destination after a certain amount of time.
   (d) The system must reduce the credits of the 'client' when a transaction finishes.
   (e) The system must increase the credits of the 'restaurant' and 'deliverer' when a transaction finishes.

1.1.6 The Box

1. The box must have a lock.
2. The box must lock the food before the travel.
3. The box must have an internal light.
4. The box must have an internal camera.
5. The box must take a picture of the delivery.
6. The box must be unlockable.
   (a) The box may be unlocked with the password.
   (b) The box may be unlocked with a signal from the website.
7. The box must warn the website when a delivery is finished.
8. The box must send spacial data collected from the delivery.

1.2 Non-Functional Requirements

1. The box shall contain a Raspberry Pi connected to the following peripherals: webcam, flashlight, electronic lock, a numeric keyboard, 4G connection module, gyroscope and accelerometer.
2. The box must weigh less than 8kg.
3. The box shall have a power bank.

4. The box shall be turned on by a 'Power' push button.
5. The box may be turned off.
   (a) The box shall turn off by a 'Power' push button.
   (b) The shall turn off when it has less than 5% battery.
   (c) The Web Application shall change the driver status to unavailable.
6. The box shall send its GPS location.
   (a) The interval shall be no longer than 10 seconds.
   (b) The delivery change to 'arrived' when the box's geographical location is within a 20meter radius from the destination.
   (c) The website shall unlock the box and finish the delivery after a 10 minutes countdown of 'arrived' state.
7. The 'client' shall receive updates of the delivery.

2 Technologies

This section details the components and technologies used in the DeliveryBox solution.

2.1 Mechanical

2.1.1 AutoCAD

The design of the box was made using the AutoCAD [3] that is a proprietary software for drawing and design of mechanical parts.

2.2 Hardware

2.2.1 Raspberry Pi 3

The Raspberry Pi is a small single-board computer (SBC) developed focused on promotion of basic computer science for schools and developing countries, and also could be used as a controller for small robotics projects. For this project, we are using a Raspberry Pi 3B+ [4].

2.2.2 Camera

The camera used was a generic webcam as a simple device to capture images inside the box, when the dish is placed.

2.2.3 GSM

The SIM800L [5] is a GSM module used to enable internet connection on Raspberry Pi without using any Wi-Fi hotspot. This modules requires a SIM Card to perform this action.

2.2.4 Electromagnet

The electromagnet is used for the door lock system. When powered, the metal becomes magnetic and can hold the door by the piece of metal attached to it [6].

2.2.5 Gyroscope

The module used for the gyroscope function of the box was the MPU-6050 [7], which contains the gyroscope, the accelerometer and the temperature sensor. The gyroscope was used for evaluate the performance of the deliverer during the delivery.

2.2.6 LCD

The LCD screen is used for simple messages during the DeliveryBox usage (as for example, when the box is connected to the server, or when the password is needed to input). The model used for this project is the 1602A [8].

2.2.7 Keypad

The keypad is a matrix module that works by observing which key was pressed, line by line. When pressing any key, it tracks which row and column it is associated with, thus taking the action associated with it [9].

2.2.8 Door Sensor

For detecting the state of the door, we are using a micro switch. The micro switch is a switch operated by physical operating force, used to detect when the door is open or is it closed [10].

2.2.9 Power Supply

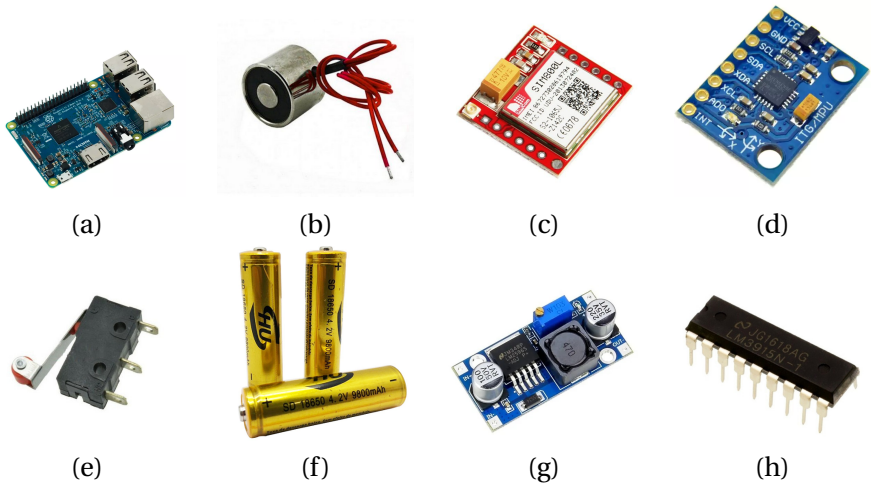
For the power supply of the system, we are using three rechargeable batteries 4.2V 2600mAh [11], which can provide up to 12V to the entire system.

2.2.10 Voltage Regulator

The LM2596[12] is a voltage regulator which can supply the current necessary for each part of the project. We use three modules, which one is for the Raspberry Pi supply, one is for the GSM module and the last one is for the Battery Level, LED and Electromagnet supply.

2.2.11 Battery Level

For measuring the battery level of the system, was used a VU Meter (LM3915)[13], which a bar of LEDs is connected, and one of this LED are used as a indicator for low battery. To communicate with the Raspberry, this LED is pointed to a LDR Module, which this sends a data to one of the ports of Raspberry Pi.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 2: Key components of DeliveryBox hardware - (a) Raspberry Pi 3B+, (b) Electromagnet, (c) GSM Module (SIM800L), (d) Gyroscope Module (MPU-6050), (e) Micro switch, (f) Batteries, (g) Voltage Regulator (LM2596), (h) VU Meter (LM3915)

2.3 Software

2.3.1 Angular

Angular [14] is a development platform built on TypeScript. Angular is a component-based framework for building scalable web applications. All the front-end (interfaces with the user) of the DeliveryBox web application was developed using Angular.

2.3.2 Node.js

Node.js [15] is a JavaScript runtime built on Chrome's V8 JavaScript engine. As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. All the back-end (functionalities) of the DeliveryBox web application was developed using Node.js.

2.3.3 Heroku Postgres

Heroku [16] is a cloud plataform for build, deliver, monitor and scale apps. The Heroku Postgres is an open-source service for developing data-driven apps where

the operation, building, security and validation relies on the Heroku itself. All the database of the DeliveryBox was developed inside Heroku Postgres. The PostgreSQL solution of the Heroku is free of charge.

2.3.4 Google Cloud Platform

The Google Cloud Platform [17] is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products. In the DeliveryBox solution, it was used the Geocoding API, to calculate the latitude and longitude for a given address; the Places API, to help locating a given place on the map; the Directions API, to calculate the route between to given places; the Maps Javascript API, to display, update and control the map of the web application; The Google Cloud Compute Engine to create a Virtual Machine that hosted the solution and the Google Cloud Storage for management and storage of the pictures taken by the box.

2.3.5 WebSocket

WebSockets [18] is a advanced technology that makes possible to open a communication section between the browser and a server. With the WebSocket you can send messages to a server and receive event oriented answers without the need of consulting the server. It is a bidirectional TCP (Transmission Control Protocol) connection that stays open, giving the freedom to exchange more information and modification if needed, without the necessity of making a new connection every time. The communication between the parts of the DeliveryBox solution where made using the WebSocket.

2.3.6 JWT

JavaScript Object Notation (JSON) Web Token [19] is an industry standard for data creation with optional signing and/or cryptography. The tokens are signed using a pair public/private key between the server and the client. The payloads that were send between the parts where encrypted using JWT.

2.3.7 Nginx

Nginx [20] is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. The software was created by Igor Sysoev and publicly released in 2004. Nginx is free and open-source software

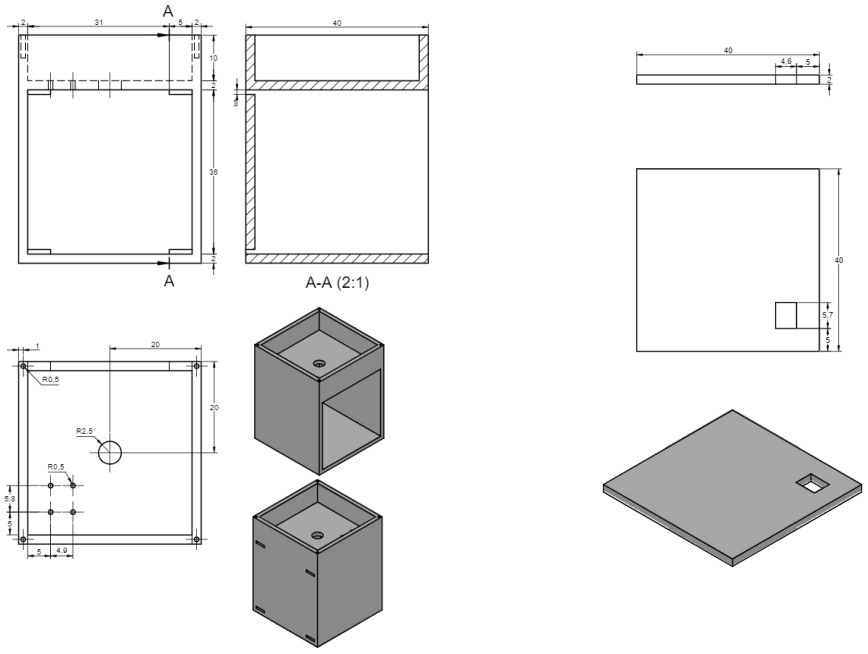
3 Development

This Section details the development of the project. The section explains the steps taken as well as some of the difficulties that were encountered by the team.

The development blog of the DeliveryBox can be found in [21].

3.1 Mechanical

The starting point in the mechanical development and design was made using AutoCAD. With this design tool the box parts were designed as shown in Figure 3. The design was made so that the box has a compartment for the electronics at the top, and an open space for the food.



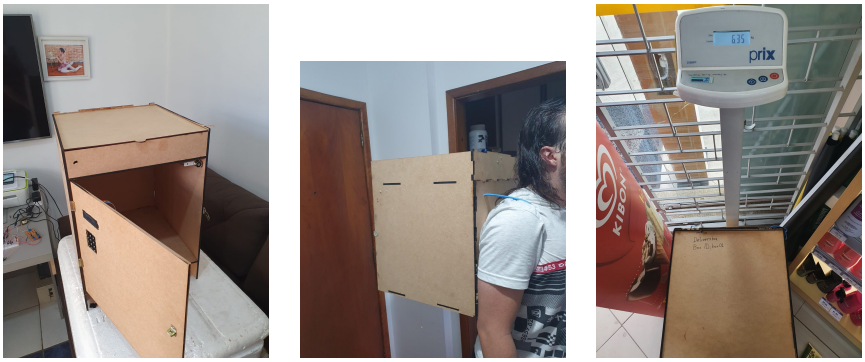
(a) (b)

Figure 3: (a) Box View, (b) Door View

The more relevant aspect of this initial development was the delay caused by a third party that was contracted to build the box. For schedule reasons, the group needed to go around and build the box by themselves. Another relevant aspect that was mentioned on Section 1.2 was the weight of the box. As a object that needs to be carried on the back of a person and on a motorcycle, the box itself could not weight more than a certain amount. The threshold stipulated by the team was 8 kilograms, and as shown in Figure 4.c the final DeliveryBox Box weight 6.35 kilograms. Figures 4.a and 4.b shows a little view of the finalized box.

3.2 Hardware

The hardware project was developed using Proteus [22] to elaborate the schematic of the system. Several PCBs (Printed Circuit Boards) were made to facilitate the mounting of some modules, and also the connections between them. The PCBs can be seen in the Figure 6. For the power supply three batteries were used,



(a) (b) (c)

Figure 4: (a) The Box: Front View, (b) The Box: Side View, (c) Weight of the Box

which provides 12V to three voltage regulators. Each one is used for a part of the circuit (one is used for Raspberry Pi and some modules, another is used for the GPS module and the third one is used for the electromagnet, LEDs and the battery level).

3.3 Firmware

The Firmware was developed using a Raspberry Pi 3B+ with Python 3.7. The initial development was planned to be an hybrid of Node.js and Python, but for the lack of processing power that will be discussed on Section 4, the plan was changed for a pure Python approach.

The first part in the Firmware development was to build a state diagram, that can be seen in Figure 7. The diagram show the expected results of the physical box. The box starts in idle until the deliverer connect to it.

From this moment on, the box waits for an order, that will be sent from the server by a WebSocket. The box then enters the main stage of it's functioning, the macro state of 'delivering'. At this point, the box awaits for another information that it arrived at the location. Then the behavior of the firmware depends on what traveling was chosen, if it is the traveling for the restaurant, after receiving the password and locking the box again, a photo is taken and sends to the server. If is traveling for the client, upon reaching destination a timer starts, if the client does not input the password before the end of the timer, the box unlocks itself automatically. After unlocking at the client site, the box ends the transaction and sends the evaluation of the deliverer to the server. If the power button is pressed, or if the box hits the 5% battery gauge, the box alerts the server and cancel the order, shutting itself down afterwards.

The code for the Firmware can be subdivided in two fundamental parts: the WebSocket communication, responsible for the exchange of messages with the server, and the drivers, responsible for control and communication of the electronic components.

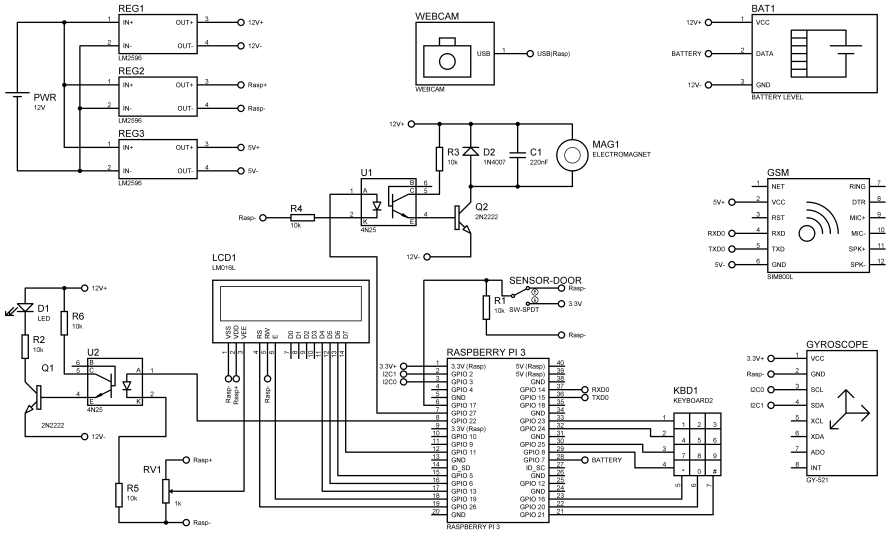


Figure 5: Final schematic of the project

The development of the drivers was made in a modular way, with each script working independently from the others, in order to make sure that all the components works. This method was chosen to make sure that a malfunctioning in one script does not flood the others in a way that everything just stops working.

The heuristic for the performance of the deliverer is calculated in by the following metric:

We start with a evaluation of 5.0 and we reduce in a punitive way. We collect three dimensional data from both gyroscope and accelerometer at intervals of 0.5 seconds. We average 10 data points (5 seconds) and put it to persist on disk on a new line from a .csv file. When the deliverer arrives we stop the collection of metrics and read the file, comparing each of the 6 parameters from 2 subsequent collections. We defined that if the difference between them exceeds a certain amount we reduce in 0.05 from the current evaluation, that originally started with 5.0. If it goes to negative we send 0 as a evaluation for the performance. This heuristic it's pretty simple, for future project we would need to collect a lot of performance data and then use a bag-of-words' strategy [23].

3.4 Web Application

The first important concern in developing the Web Application was to make it in a way that the user could access all the functionalities by himself. With this in mind, the first step in the development was to route all the most important aspects and functionalities, for it to be called from the front-end. Figure 11 presents the route design.

Both the front-end and the back-end were developed using Javascript. On

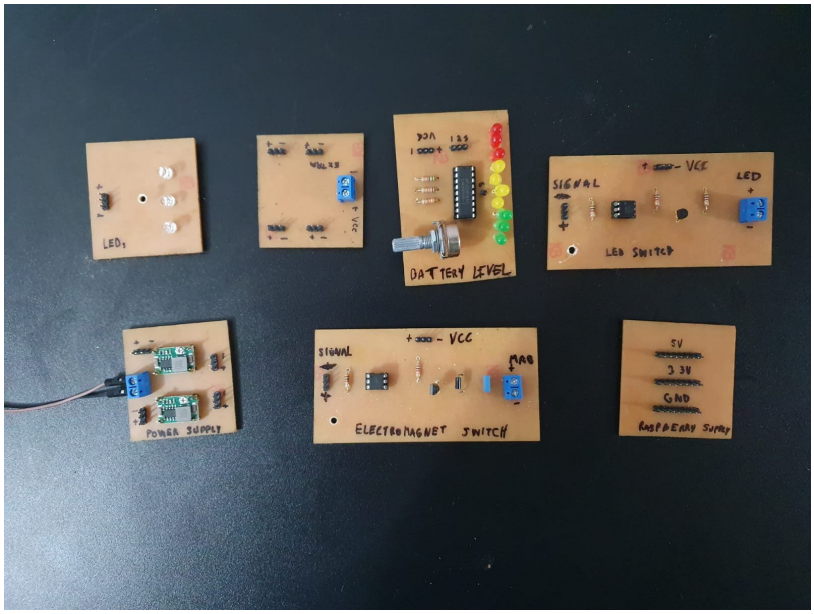


Figure 6: Printed Circuit Boards

the back-end it was used the Node.js. On the front-end it was used the Angular. The Angular was a powerfull framework and, for being modular, helped to speed up the development and testing of each part separately.

The functionalities developed in the Web Application were: a page for create user (Figure 8.a) ; a page for login (Figure 8.b); a page with the logs of finalized deliveries (Figure 8.c); a page for each actor to update its profile (Figure 8.d); a page for the restaurant to add a dish; a page for the client to search for restaurants (Figure 8.e); a page for the client for search the dishes; a shopping cart (Figure 8.f) and a wallet. A video exploring all the pages can be found in [24]. The Web Application is currently hosted on the Google Compute Engine, and can be accessed through the URL in [25].

For modeling the database, it was used the Workbench [26]. The Workbench is a tool for modeling the Entity Relationship Diagram that allows the export of SQL of the model that was designed. The database was projected in a way that the roles are descendants of the entity User. The reason behind that was because the Web Application has a great number of pages that are identical between the three roles, the only thing that really changes is the content displayed. In other cases, it is not the content, but the order. Hence, if all the user can access the same backbone, the code gets leaner. Figure 9 presents the Entity Relationship Diagram of the project.

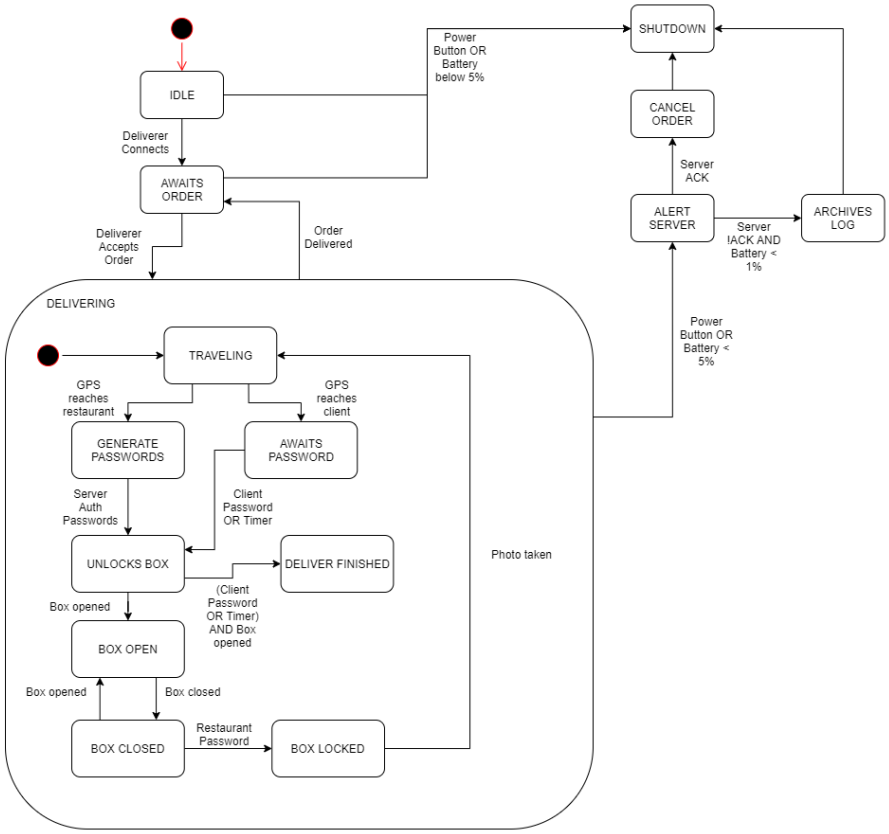


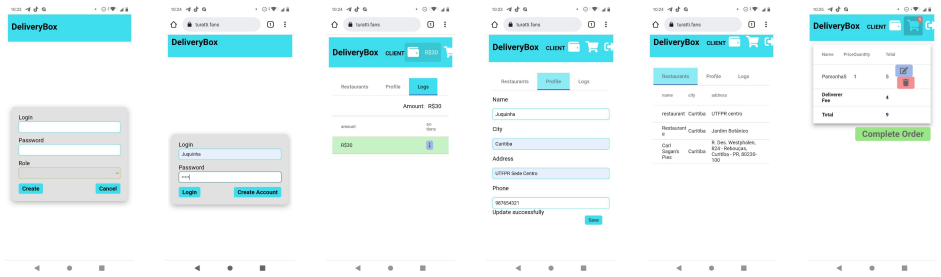
Figure 7: Firmware State Chart

3.5 Map and Geolocation

An important aspect of the software development revolves around the Google Cloud Platform. The display and development of the map was vital for a solution that purposes to be a safer food delivery solution. For this reasons, the studies of the platform started early in the developing of the project.

The choice for the Google Cloud Platform was made because of the possibility of using the Google Maps Javascript API. The API serves the map, with possibilities for drawing routes, adding and removing markers, as well as changing the zoom and the center.

To receive the positions and addresses of the different users, a couple of methods were used. For the restaurant and the client, it was given a camp that asks for an address. The restaurant and the client write their current address, and the system calls the Geocoding API. The Geocoding API uses the Google Maps search engine to find the address given, and returns the information of the latitude and longitude of the given address. For the deliverer, the method was to use the API Geolocation. This API is located in most of the recent internet



(a) (b) (c) (d) (e) (f)

Figure 8: (a) Create Profile, (b) Login Page, (c) Logs Page, (d) Update Profile, (e) Select Restaurant Page, (f) Shopping Cart

browsers and, after the user authorizes, gives the actual latitude and longitude of the device. The method that the team used was the 'watchPosition' that returns a new pair of latitude and longitude every time the device moves. Figure 10 shows the final version of the deliverer map.

With the information of the locations both from the deliverer and the destination (either the restaurant or the client) the system can call for the Directions API. This API calculates the route that needs to be followed by the deliverer. The Directions API can be configured with the same parameters as the Google Maps app, for the DeliveryBox, the parameters chosen were to estimate realistically, using vehicles, and driving through all types of roads. After a certain amount of time or distance was travel by the deliverer, the Directions API recalculates the route, updating the map in the process.

3.6 Communication

The project needed a communication system between different actors (consumer, deliverer, restaurant and the physical box) that depending on the action of each actor changes the interface of the others. With the final objective of creating a transaction of a product that interacts with the box. With this in mind it was researched which technology could supply the demand.

In order to build the real time communication system needed, WebSockets were chosen. The reason behind the choice can be seen on the explanation of the WebSocket on the Section 2.3.5.

The communication protocol was created after mapping the communication needed for a successful transaction, as well as its possible problems. The protocol can be resumed as 37 different JSON, these packages are mapping from the re-connection to the different points of the transaction. Some examples include a package for the shopping cart button to "complete order", another one for choosing the deliverer, another one to alert that the order is on the way.

Through the information present in the different JSONs that were created for the DeliveryBox, it was build the logic behind the interface of the Web Ap-

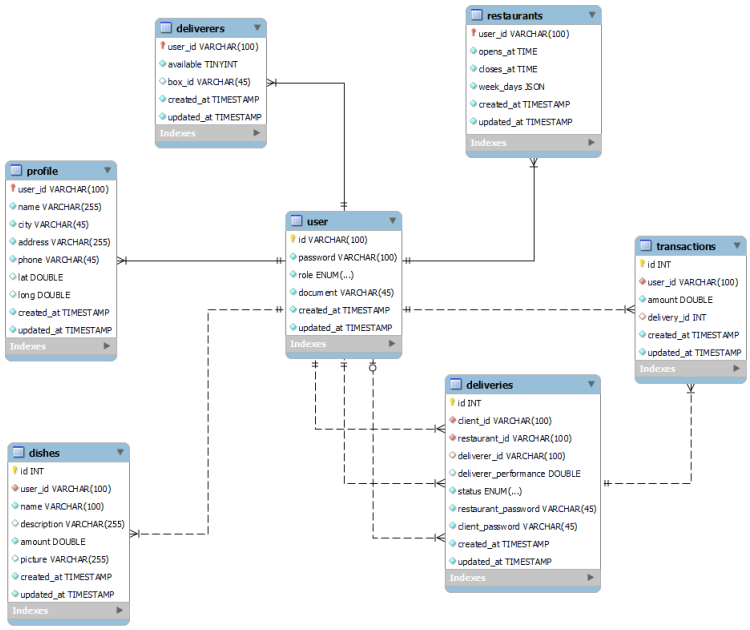


Figure 9: Entity Relationship Diagram

plication. This interface can dynamically change the content seen by each type of user in a given point of the transaction. At the same time, these information are used to control the internal state of the physical box. That is important because the routines of password checking and sensor uses are blockers. That makes necessary that the use of each routine is dynamical through the Python files, each one with only one function, at the same time that a main script is in charge of the communication with the server.

An important aspect to be taken into account for scaling the project is that there is a possible problem of the server turning stateful. That is possible considering the architecture of exchanging messages between the actors. In the current architecture, the only possible to grow is the scale up, and any modern architecture focuses on being stateless and focuses on scaling dynamically by client demand. To resolve this problem in a continuation of the project would be necessary to create a new layer with a RAM Database that would be responsible to store the state of the transactions, and the WebSocket would be responsible for listening and sending messages after looking in the RAM Database.

4 Difficulties

The first important difficulty to be addressed is the COVID-19 pandemics. The pandemic that started last year of 2020 resulted in a new form of semester completely online. The online environment makes it harder to communicate within

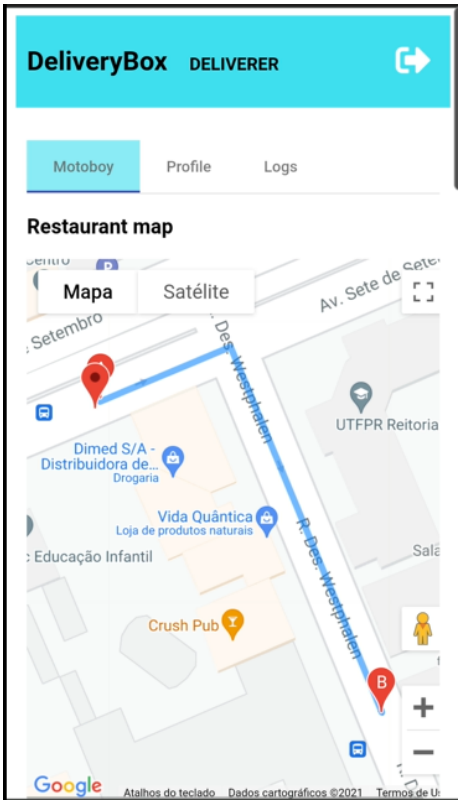


Figure 10: Deliverer's Map

the team and coordinate activities. One member of the DeliveryBox team, for instance, is currently living in São Paulo, more than 350 kilometers away from the rest of the team. The distance makes all the more important to communicate efficiently and with clarity with the tools that are given, being that e-mails, Discord or Whatsapp.

Another aspect that made an impact on the development was the length of the semester. The year of 2021 needed to be, at the same time, the semester of 2020/2, 2021/1 and 2021/2, to compensate the semester that did not happen last year. For it to happen, the semesters were cut short and with that, the time for development. The semester that this project was made was the shorter semester in the history of the UTFPR, and the amount of time that was cut was a challenge by itself.

The first two are external aspects that influenced on the project, but there are internal ones. The first one being the already mentioned problem with the processing power of the Raspberry Pi. The initial idea of the firmware was to make all the controlling and communication on Node.js and only the usage of the electronic components in Python. Test were made on the PCs (personal computers) of the team to make the fork from the Node.js to a Python script, and receive its

output information, and it worked well. But the problem is, when this idea was transferred to the Raspberry Pi 3B+, the problems started to arise.

To make the problem more clear, the team detected this malfunction on the keyboard. The delay between pressing the key on the keyboard and the Node.js code receiving that information was longer than two minutes. And the script in Python was only getting the input and outputting it for the Node.js. Hence the time that would take to get only the password, testing it, and unlocking the door in another Python script, the project was doomed to fail. The team decided to rewrite the code entirely in Python for this particular reason.

Another important aspect that caused difficulty for the team was the WebSocket. As it was explained on the Section 3.6 the WebSocket was a massive layer of the DeliveryBox project. The amount of work needed to develop all the JSON messages, as well as, to connect all the parts was hugely underestimated part of the project by the team. And the difficulty aggravated because neither of the members used the WebSocket before, being a completely new technology.

The last part that caused problems and difficulties in the development was the GPS module. Originally, the box itself would acquire and transmit its geographical localization to the server. During the test phase of the project, it was discovered that the antenna of the GPS module that was bought for the project was not strong enough to receive the signal. The tests were made in Curitiba, firstly in the center and after in a suburban area. The team borrowed another module to make sure that the problem was not factory defect, but the results were the same. Because this problem was foretold in the risk analysis planning, it was decided to remove the GPS module from the project.

5 Results

5.1 Proof of Concept

Even with the adversities cited in the Section 4, we could implement all the functionalities planned for the project. We developed a box that had a lock system, a gyroscope functionality to evaluate the quality of the transport, a screen and a keypad for interaction with the deliverer, customer and the restaurant, a camera that could take pictures, and all of that could communicate with the server and the devices from the customer, the restaurant and the deliverer via WebSocket.

A video with the prototype working can be seen on the Youtube in [24].

5.2 Schedule

The Table 1 represents the summary of the schedule of the project. It is important to note that there are a range of differences between the planned with safety and the actual executed amount. For the project it was established an error margin of 30% over the initial hour estimation, that margin was called safety.That can be explained by some areas ending up easier than previously

thought, and other areas, especially the WebSocket, ending up way harder than initially planned. For a more detailed schedule, there is a Google Docs at the blog of the project [21]. The amount of actually worked hours was 37% more than initially projected.

Table 1: Summary of Schedule.

Deliverable	Planned	Safety	Total	Executed	(E/P)%
Project Charter	26	7.8	33.8	18	-30.7%
Follow-up blog	17	5.1	22.1	15	-11.7%
Mechanical project	38	11.4	49.4	38	0.0%
Hardware project	81	24.3	105.3	64	-21.0%
Software project	149	44.7	193.7	258	+73.1%
HW & SW integration	51	15.3	66.3	81	+58.8%
Overall integration	43	12.9	55.9	88	+104.7%
Technical report	31	9.3	40.3	38	+22.6%
Final Sum	**436**	**130.8**	**566.8**	**600**	**+37.6%**

5.3 Budget

The budget of the project was within the boundaries stipulated by the discipline. The total cost for the team was lower than the presented on Table 2 because the group already had an Raspberry Pi 3. Considering this factor, the actual cost of the project development was R$ 531.00, or R$ 88.50 per member. The APIs and services used in the development were either on the free tier, or on the test period.

6 Conclusions

The project was a tough challenge for all the group members, in which we faced some new technologies, such as web sockets, in which none of the members had any prior knowledge about. During the development, we learned about some mistakes we made while planning, which ended up affecting the project's progress, but all this in time to be able to proceed with the development.

The DeliveryBox team made all the tests and all worked as intended.

Even with some adversities, it was possible to conclude the project with success. All the planning (schedule, risk analysis) were very important to the project, and also the collaboration of every member from this group were crucial to making the DeliveryBox real.

Table 2: Project Costs.

Product	Specification	Quantity	Price (R$)	Total (R$)
Raspberry Pi 3	Microcontroller	1	300	300
Battery 4.2V	Power Supply	3	15	45
Battery Charger	Power Supply	1	15	15
Sim800L	GSM	1	60	60
LM2596	Voltage Regulator	3	10	30
Electromagnet	Lock	1	50	50
Webcam	Camera	1	40	40
GY-521	Gyroscope	1	15	15
LCD	Box Screen	1	15	15
Numpad	Box Keypad	1	25	25
PCBs	Circuits Board	7	15	105
LM3915	Battery Level	1	15	15
LDR Module	Battery Level	1	8	8
Jumper	Connection	2	20	40
LEDs	Illumination	4	5	20
Micro Switch	Door sensor	1	5	5
Screws	Box	8	0.25	2
Metal Corner	Box	4	0.25	1
Box	DeliveryBox	1	40	40
			TOTAL	**831.00**

References

[1] DINO. Com distanciamento social, cresce o consumo de deliverys. https://www.terra.com.br/noticias/dino/com-distanciamento-social-cresce-o-consumo-de-deliverys,292202e02560f2fbb79b087a310d66c1qerdutal.html.

[2] Marilia Almeida. Reclamações sobre apps de entrega disparam até 343% na pandemia. https://invest.exame.com/mf/reclamacoes-sobre-apps-de-entrega-disparam-na-pandemia.

[3] AutoDESK. Autocad software. https://www.autodesk.com/products/autocad/overview.

[4] Raspberry Pi Foundation. Raspberry pi 3 model b+ product brief. https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[5] SIM Tech. Sim800l hardware design. https://components101.com/sites/default/files/component_datasheet/SIM800L-Datasheet.pdf.

[6] 5v electromagnet - 10kg holding force. https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/3874_Web.pdf.

[7] InvenSense Inc. Mpu-6050 product specification. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[8] Specification for lcd module 1602a. https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf.

[9] Keypad general specification. https://www.hobbytronics.co.za/Content/external/573/AK-XXXX.pdf.

[10] Honeywell. Micro switch™ standard subminiature snap-action z series. http://www.farnell.com/datasheets/1676940.pdf.

[11] Lithium-ion battery data sheet. https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf.

[12] Texas Instrument. Lm2596 simple switcher power converter. https://www.ti.com/lit/ds/symlink/lm2596.pdf.

[13] Texas Instrument. Lm3915 dot/bar display driver. https://www.mouser.com/datasheet/2/405/lm3915-443929.pdf.

[14] Angular. Angular: The modern web developer's platform. https://angular.io/.

[15] Node.js. https://angular.io/.

[16] Heroku: Cloud application platform. https://www.heroku.com/.

[17] Google. Cloud computing services. https://cloud.google.com/.

[18] MDN. Websockets - apis da web. https://developer.mozilla.org/pt-BR/docs/Web/API/WebSockets_API.

[19] JWT. Json web token. https://jwt.io/.

[20] NGINX. Advanced load balancer, web server, reverse proxy. https://www.nginx.com/.

[21] DeliveryBox Team. Deliverybox blog. https://marmalade-alloy-3e1.notion.site/DeliveryBox-826c58b1cc6d45e0a001f10c3f3ae9bd.

[22] Labcenter Electronics Ltd. Proteus. https://www.labsis.com.br/index.php/produtos/proteus.

[23] Manuel Ricardo Carlos, Luis C. González, Johan Wahlström, Graciela Ramírez, Fernando Martínez, and George Runger. How smartphone accelerometers reveal aggressive driving behavior?—the key is the representation. *IEEE Transactions on Intelligent Transportation Systems*, 21(8):3377–3387, 2020.

[24] DeliveryBox Team. Deliverybox final video. https://youtu.be/fQGR6bmGnvM.

[25] DeliveryBox Team. Deliverybox. https://turatti.fans/.

[26] MySQL. Mysql workbench. https://www.mysql.com/products/workbench/.

Appendix

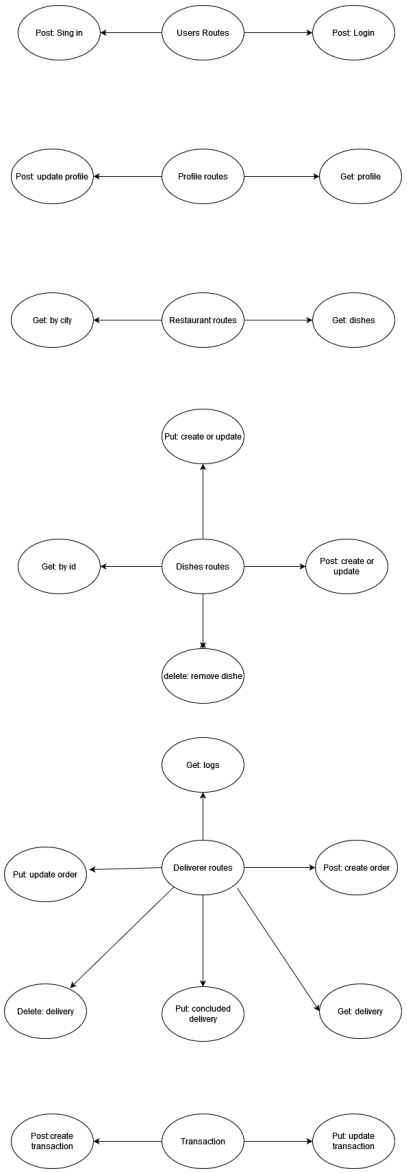


Figure 11: Routes