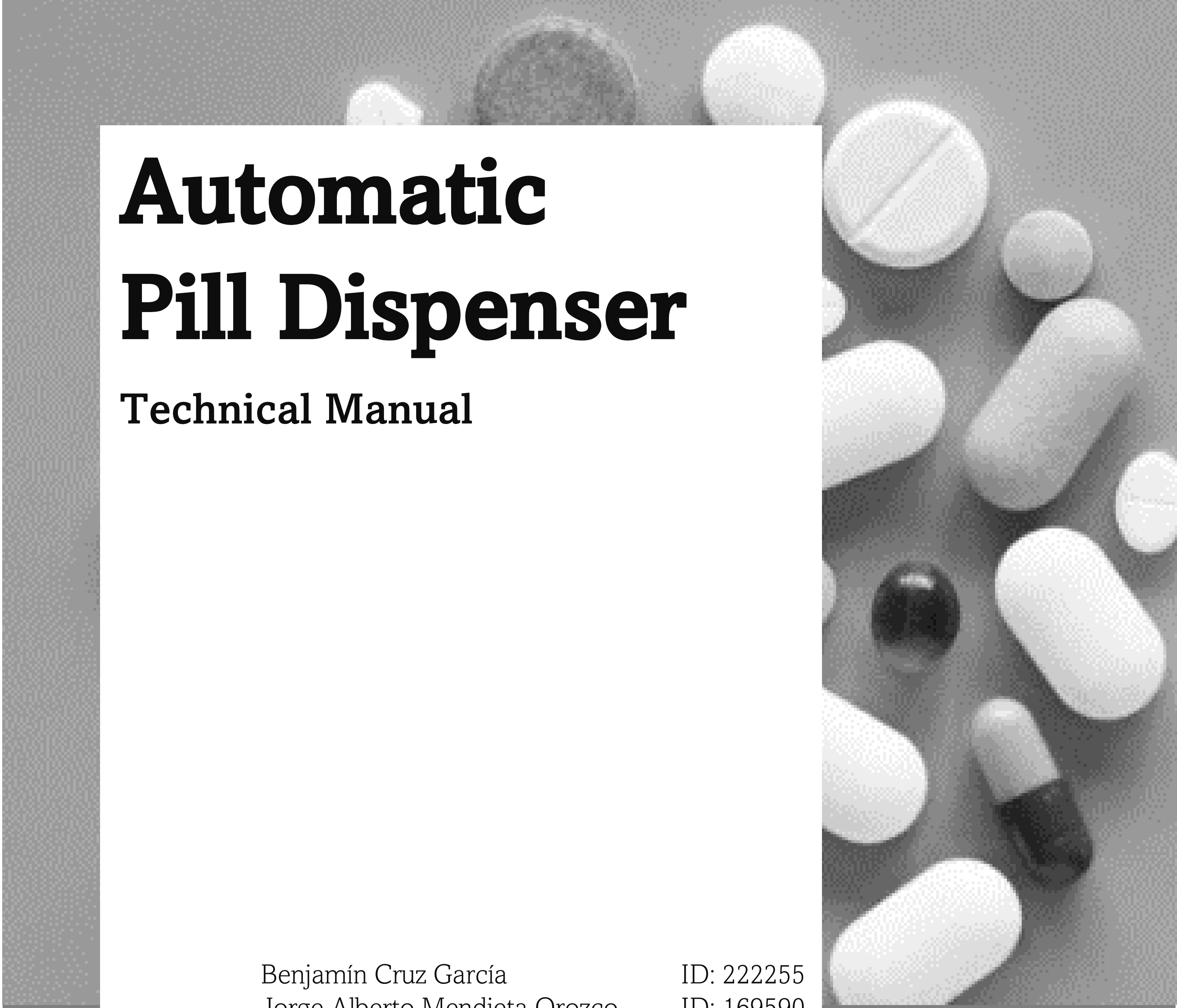


Automatic Pill Dispenser

Technical Manual

Benjamín Cruz García	ID: 222255
Jorge Alberto Mendieta Orozco	ID: 169590
Manuel Alfonzo González Moreno	ID: 204178
Alberto Chávez Pérez	ID: 222479

Ver 1.0

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

1. Index

2. Introduction

Use this document to learn about how to install, configure, and use your automatic pill dispenser.

Throughout the document, different symbols will be used to denote the following information:

- ⚠ This symbol denotes a **WARNING.** Indicates the possibility of physical injury and tells you how to avoid the problem.
- ⚠ This symbol denotes a **CAUTION** . Indicates potential hardware damage or data loss.
- ℹ This symbol denotes a **NOTE** . Indicates useful information that will help you make better use of your device.

3. Safety information

⚠ PLEASE READ – IMPORTANT SAFETY INFORMATION ⚠

- ⚠ If the power cable is found to be damaged, avoid connecting it to the dispenser and replace it.
- ⚠ Make sure that when you connect the device to the electrical outlet, the power switch is in its off position.
- ⚠ Do not open the cover while the device is in operation.

4. Precautions

- ⚠ Keep the device at a temperature between 0 °C and 40 °C.
- ⚠ Avoid moving the device while it is dispensing.
- ⚠ Avoid getting the device wet.
- ⚠ Avoid mixing different medications in the same container.
- ⚠ Avoid mixing medications with different expiration dates in the same container.
- ⚠ Avoid forcing the containers when inserting them into the tray.
- ⚠ Avoid hitting the product.
- ⚠ Avoid leaving the product within the reach of minors.
- ⚠ Avoid turning off the device while pills are being dispensed.
- ⚠ Avoid using gel pills.

5. Description

The automatic pill dispenser is made up of different electronic components, mechanical parts, and an acrylic casing.

To interact with the dispenser, there is an integrated screen and keyboard, or if desired, it can be configured in the same way through the smartphone application using Bluetooth.

To insert the pills that will be dispensed, there is a top cover that can be opened. Inside there are four containers where you can insert the pills according to size and shape.

Once configured by the user, the dispenser will dispense pills at specified intervals, where they will fall into a small container where they can be collected by the user in a small opening.

5.1 Dimensions

The automatic pill dispenser has the following physical characteristics:

Description	Worth
Dimensions (W x H x D)	180 x 180 x 310mm
Weight	2.4kg

Table 1- Physical characteristics of the dispenser

Below is a view of the different faces of the product (Figure 1):

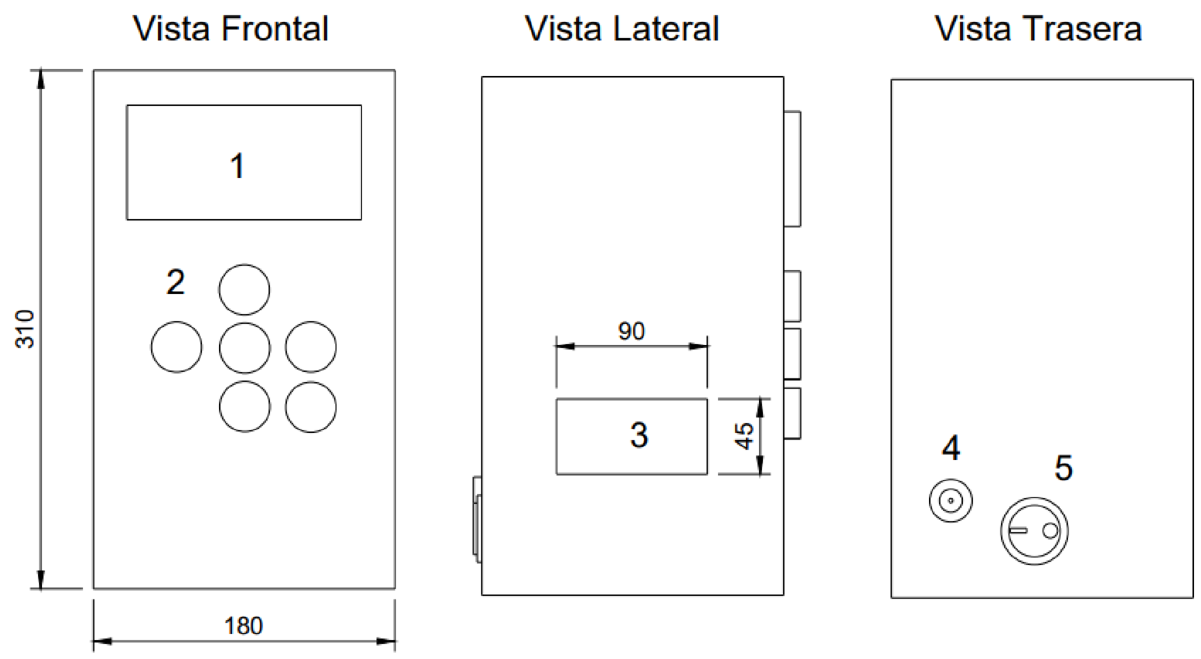


Figure 1- Front, side, and rear view of the product (measurements in mm).

5.2 System parts

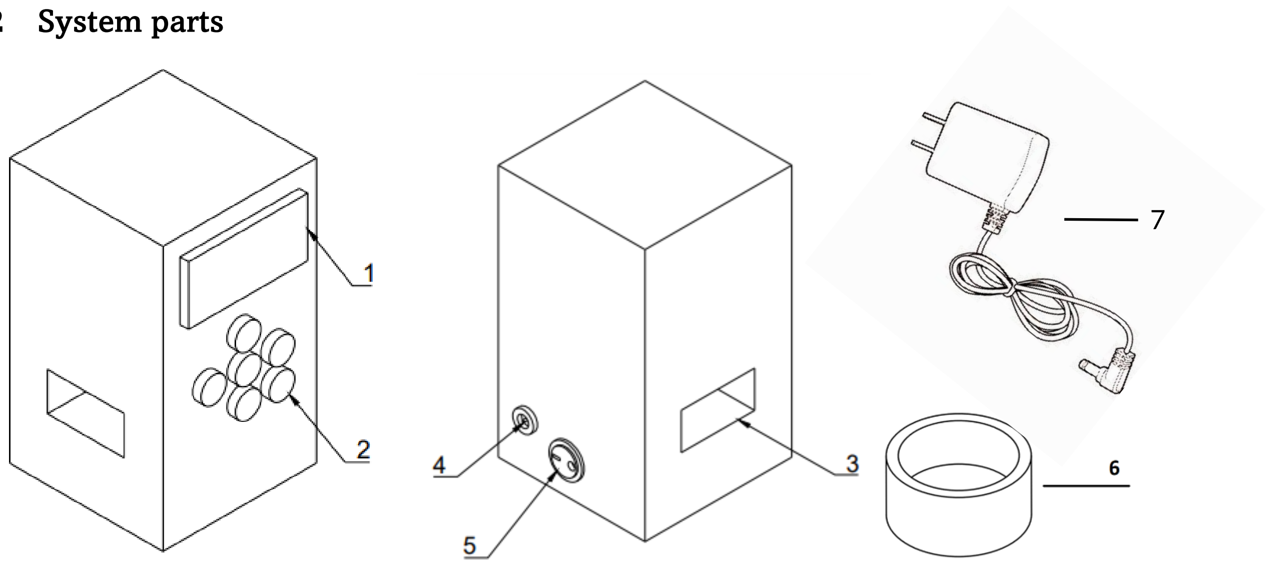
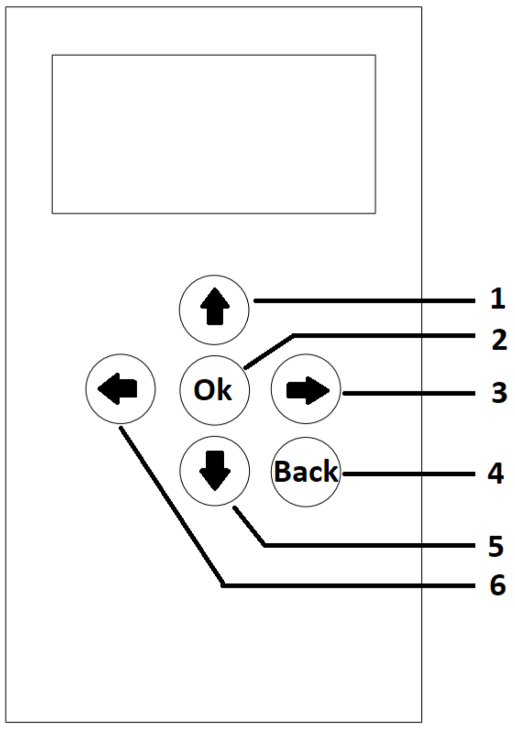


Figure 2- Isometric view of the dispenser.

The dispenser components are shown in Figure 2. These are:

1. Screen
2. Keyboard
3. pill output
4. power supply connector
5. Power button
6. Pill receiving cup
7. AC adapter

The navigation buttons of the dispenser are the following shown in Ilustración 3:



1. Up/ ↑
2. Okay
3. Right/ →
4. Back
5. Down/ ↓
6. Left/ ←

Ilustración 3 - Botones del dispensador (Vista frontal)

The dispenser has four cups to accommodate different sizes of pills shown in Figure 4

- Large circular pill holder x1
- Small circular pill container x1
- Container for pills capsules x2

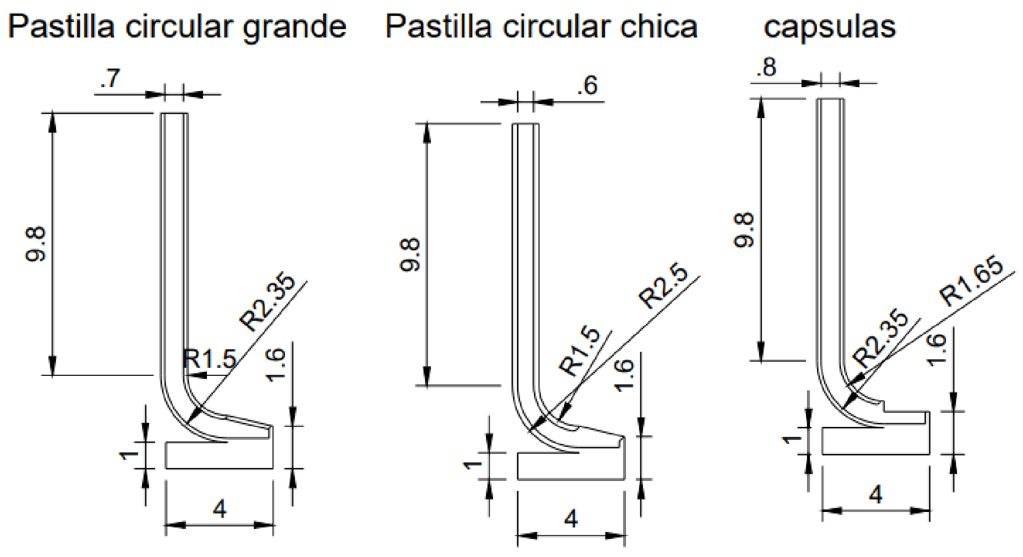


Figure 4- Containers for pills (measurements in cm).

6. Specs

Description	MIN	NOM	MAX	UNITS
Current consumption		23	65	mA
Adapter supply voltage	100	127	240	VAC rms
Dispenser supply voltage		12		VDC
Input current		2		A
operating temperature	0		40	°C
Humidity (non-condensing)	0		80	%

Table 2- Dispenser Specifications r

Using the device outside of the ranges stated in this table could damage the device or cause injury.

7. System block diagram

Figure Figure 5shows the different components and modules that make up the pill dispenser.

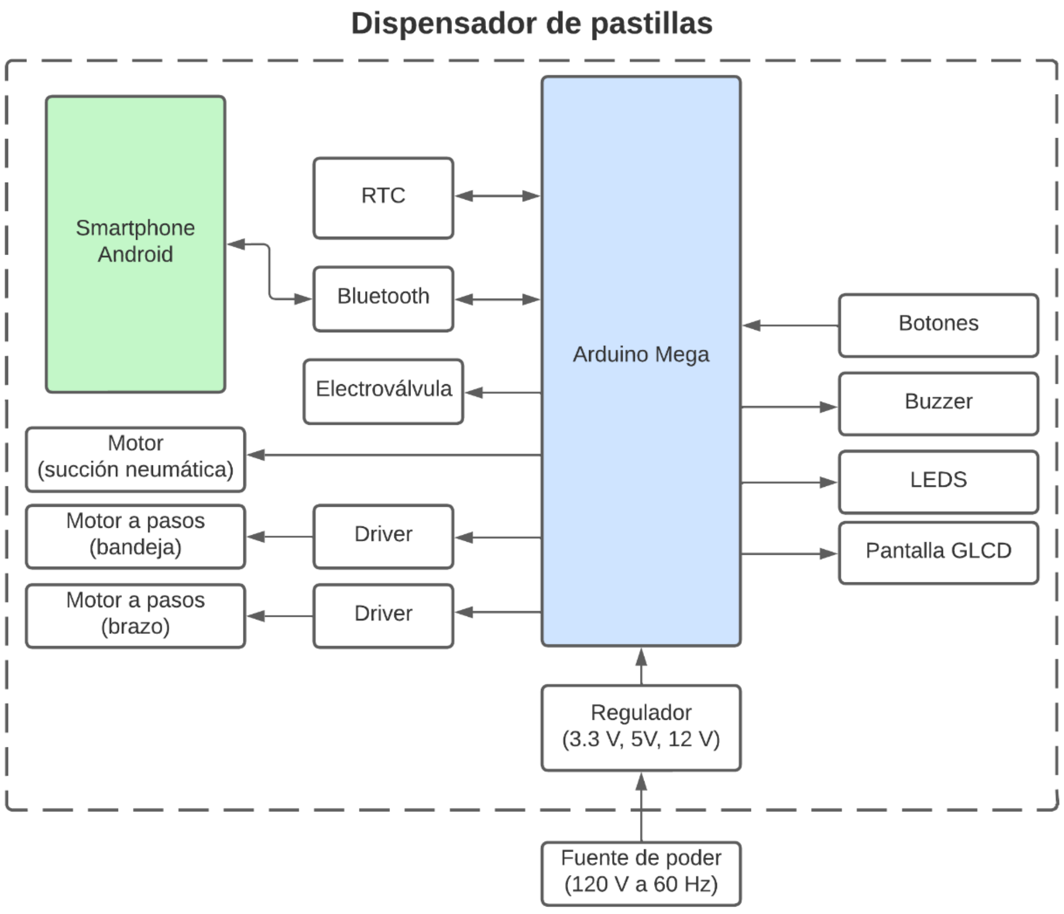


Figure 5– System block diagram

Through the application for an Android smartphone, it will be possible to establish communication with the dispenser using a Bluetooth module HC-05 in its slave configuration.

8. Electrical circuits

In this section, electrical schematics of the following printed circuits are shown:

- Three-stage voltage regulator
- Keyboard
- Motherboard.

8.1 Voltage regulator

In Figure 6is the schematic circuit of the three-stage voltage regulator. The PCB is shown in Figure 7. The regulator is powered with 12V at 2A using the power adapter included with the product.

From the 12V input, two regulators are used to obtain voltages of 5V and 3.3V.

- For the 5V stage, the KIA378R05PI regulator is used.
- For the 3.3V stage, the LM1117T-3.3 regulator is used.

It is recommended to use the included power adapter to ensure proper operation of the dispenser.

Using an adapter with a different voltage may damage the product or cause injury.

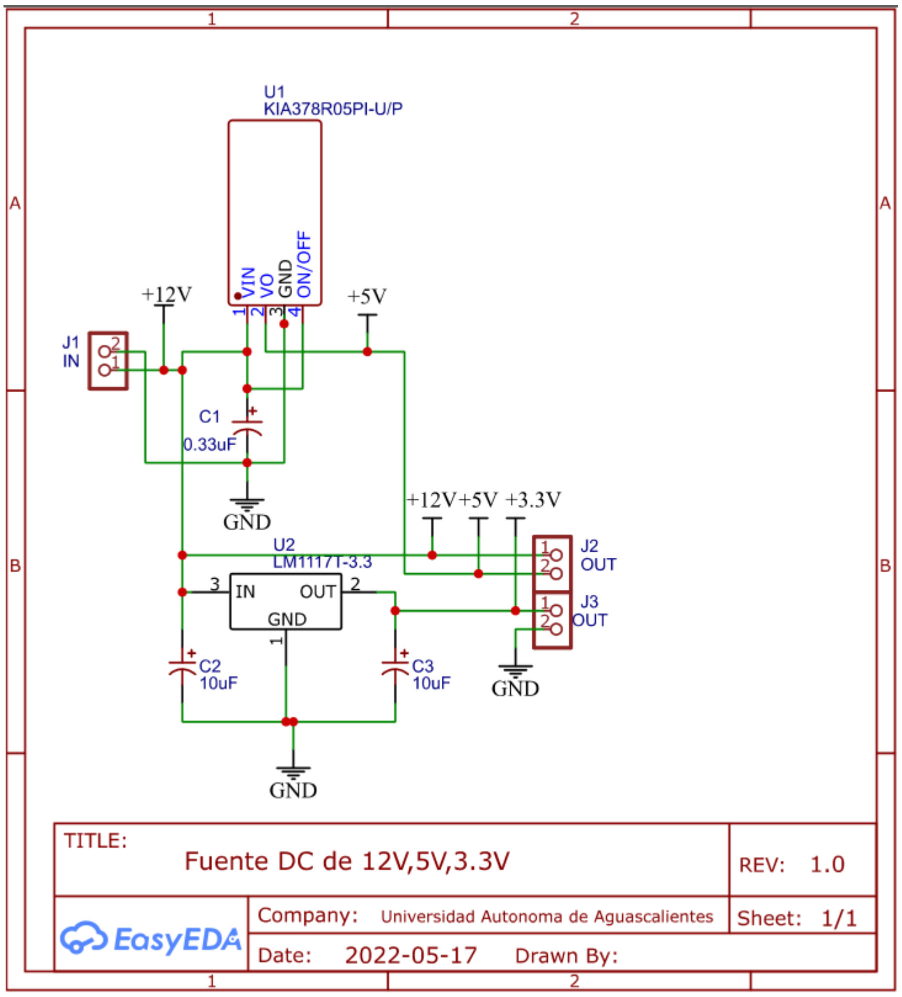


Figure 6– Schematic of the voltage regulator.

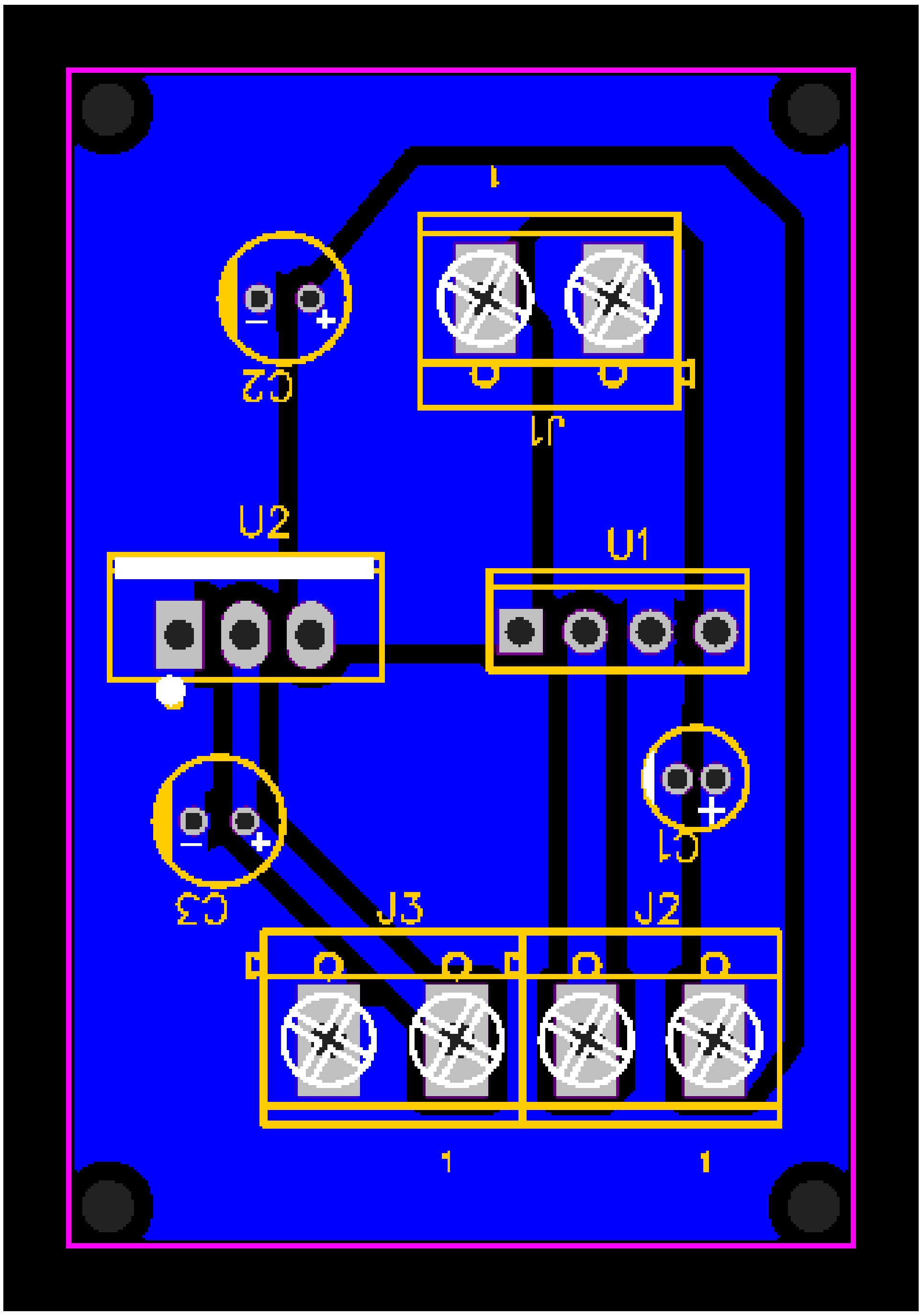


Figure 7- Three Stage Supply PCB

8.2 Keyboard

Figure 8 shows the keypad circuit schematic and Figure 9shows the PCB; it is powered by 5V supplied by the motherboard.

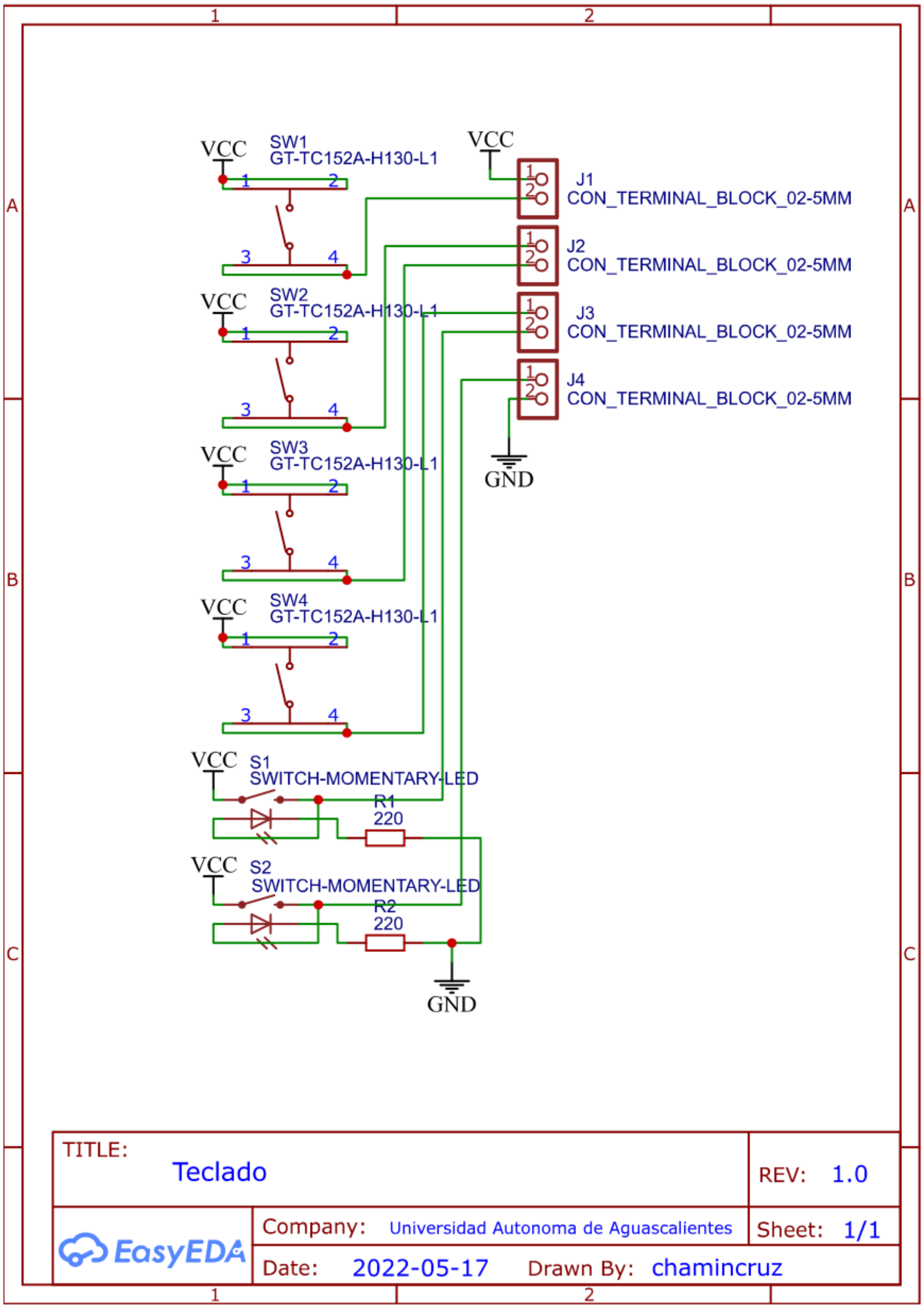


Figure 8– Electrical schematic of the keyboard.

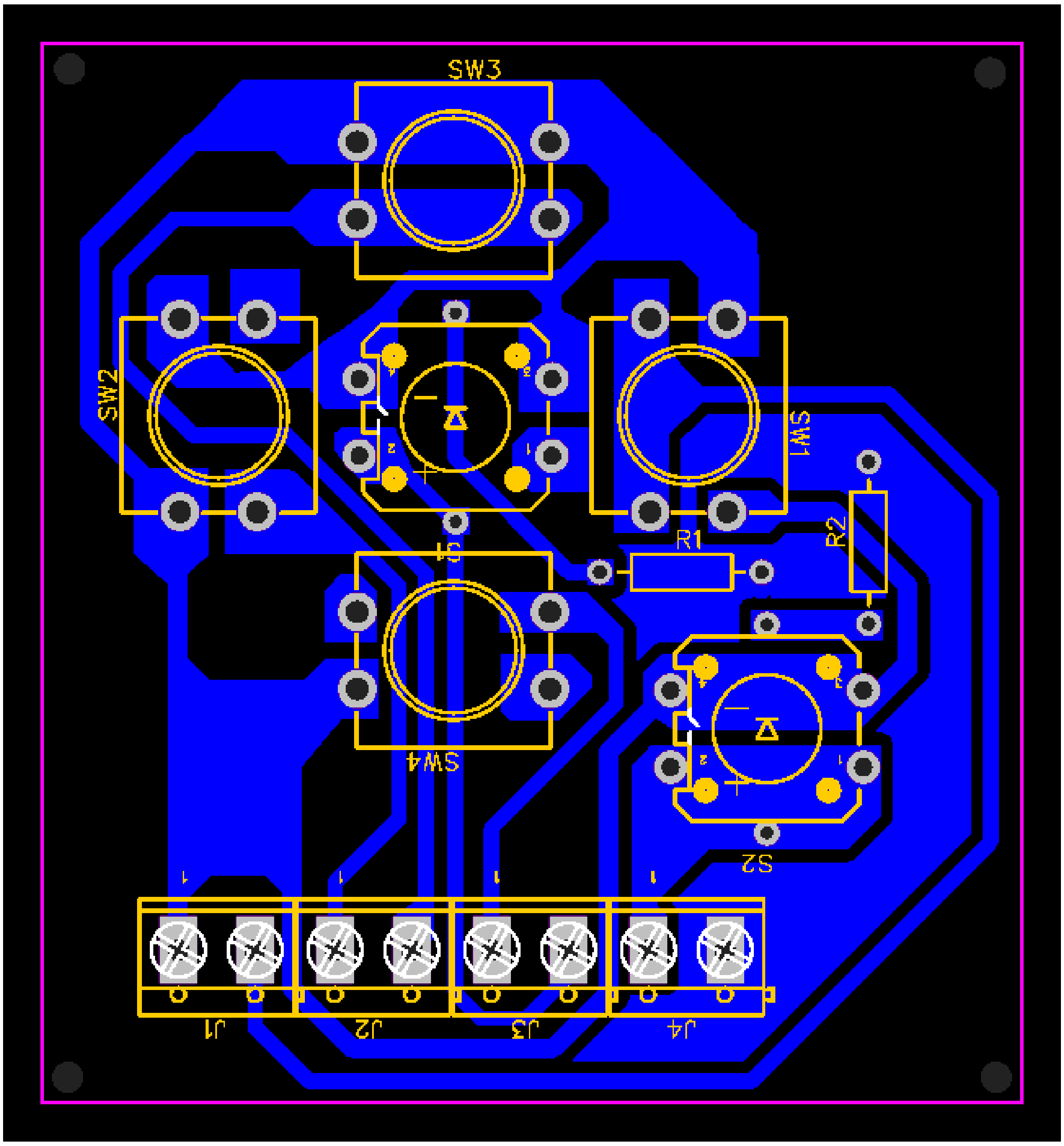


Figure 9- Keyboard PCB

8.3 Motherboard

Figure Figure 10shows the electrical schematic circuit of the motherboard. Figure Figure 11shows the PCB. This is the most important circuit in the pill dispenser as this is where the peripherals connect.

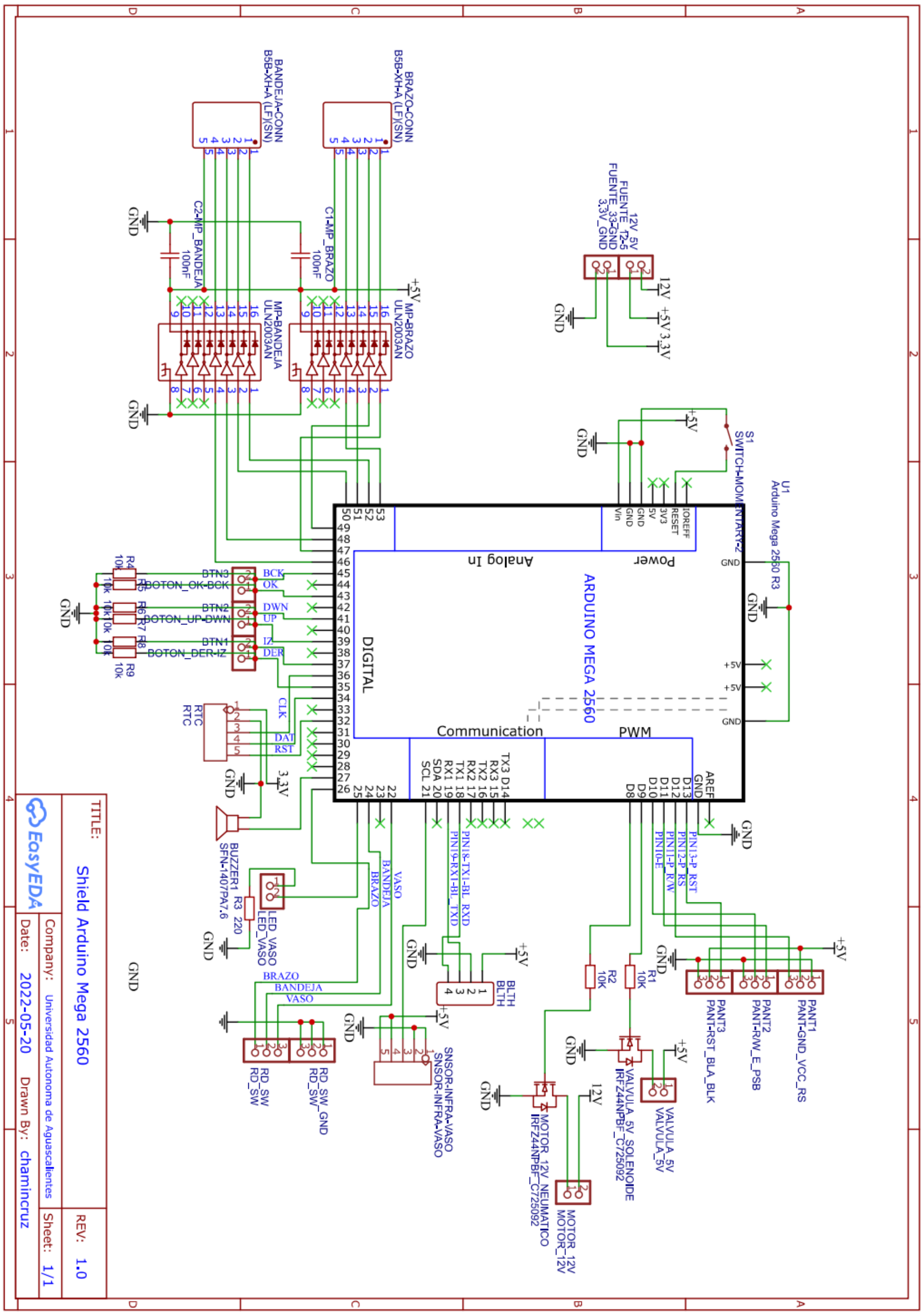


Figure 10– Electrical schematic of the motherboard

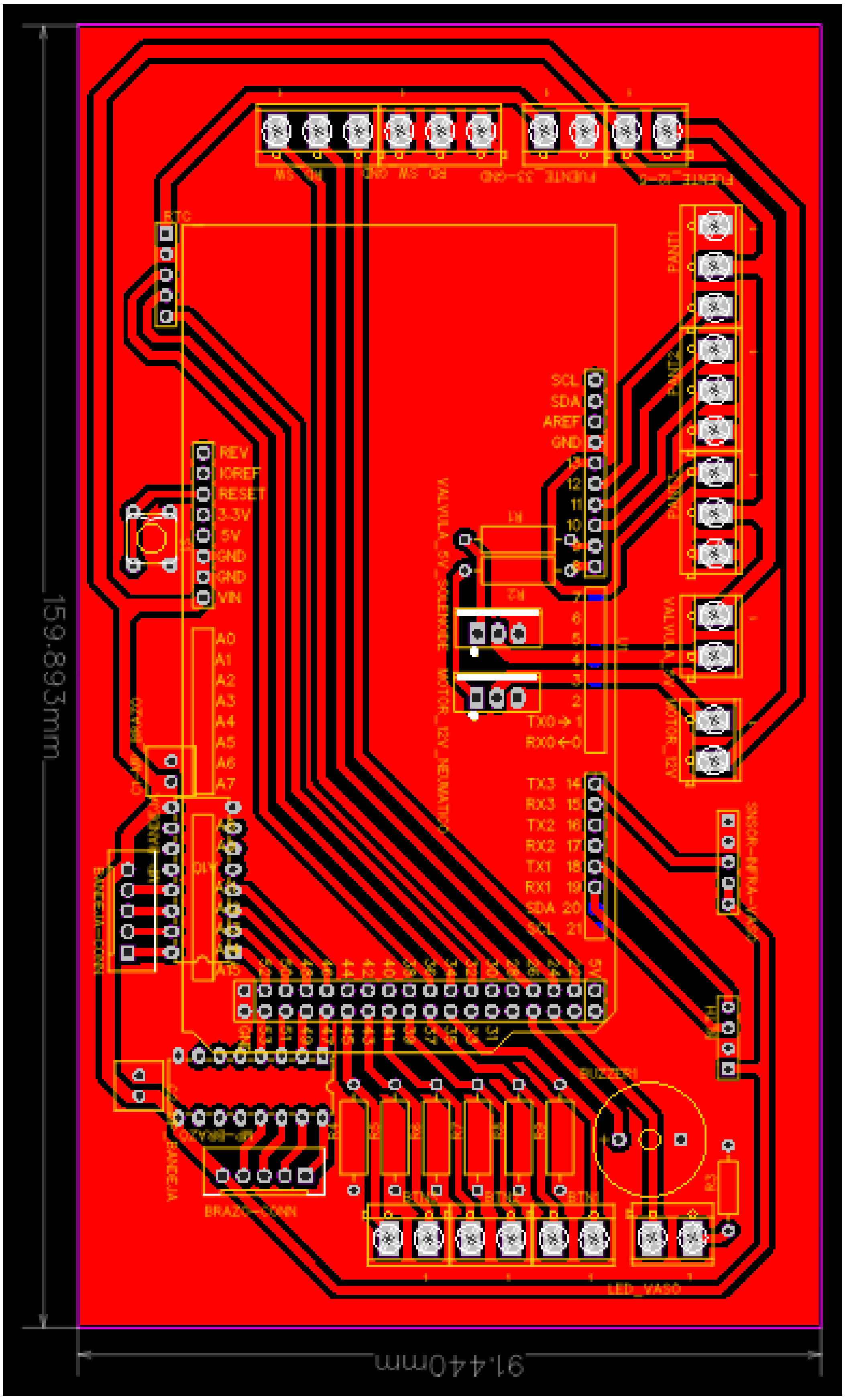


Figure 11- Motherboard PCB

9. Mechanical diagrams

9.1 Arm

The arm is a mechanism that moves only in the vertical axis, its function is to transport the suction cup and the hose to a point in one of the containers to collect a pill by suction and deposit it in the receiving cup.

The mechanical system of the arm is composed of a rack, a gear, and a base for the stepper motor. This piece is attached to the side of the dispenser to ensure that the suction cup always reaches the same position on the containers to suck up the pills.

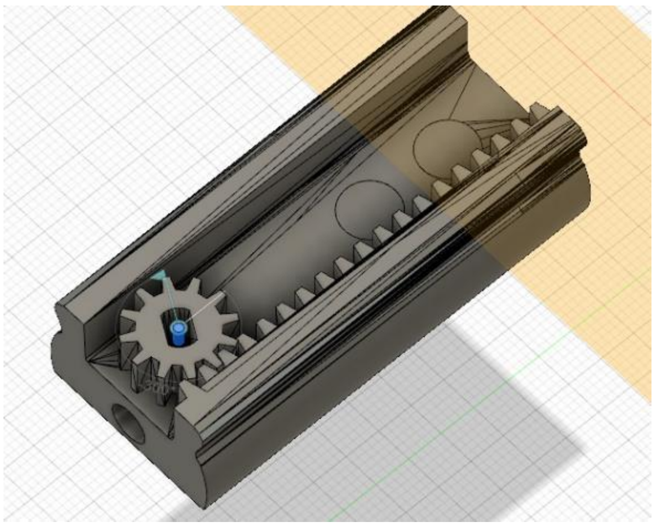


Figure 12– Mechanical arm rack and gear system

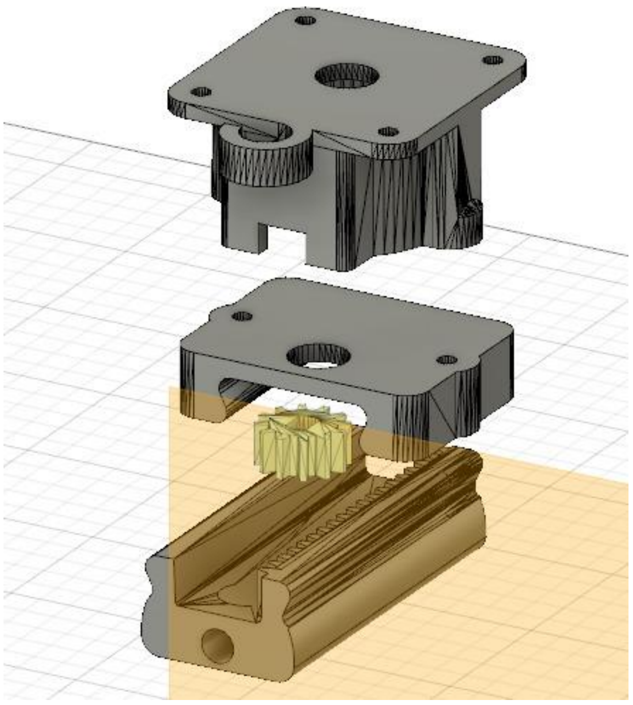


Figure 13- Rack with gear and its support for motor

9.2 Tray and containers

The tray (Figure 14) is where the four containers (Figure 15) with pills are placed. This is responsible for turning to select the desired position of the pill to be dispensed. It has a hole through which the selected pill will fall into the glass.

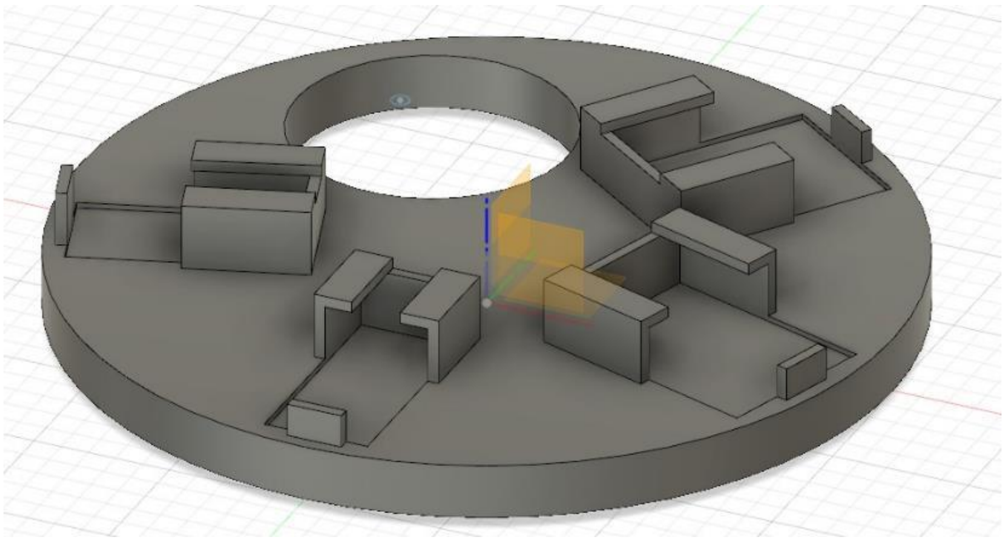


Figure 14- Tray

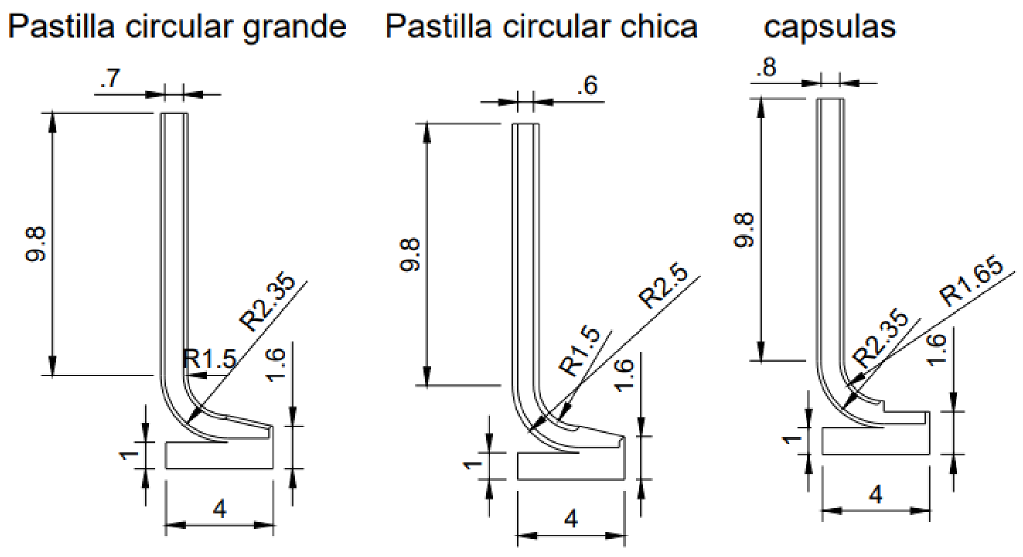


Figure 15- Pill Containers

9.3 Base

The base (Figure 16) is the part that supports the tray with a stepper motor. This piece is fixed inside the pill dispenser to prevent the pill containers from moving and to ensure that the pill always falls into the receiving cup.

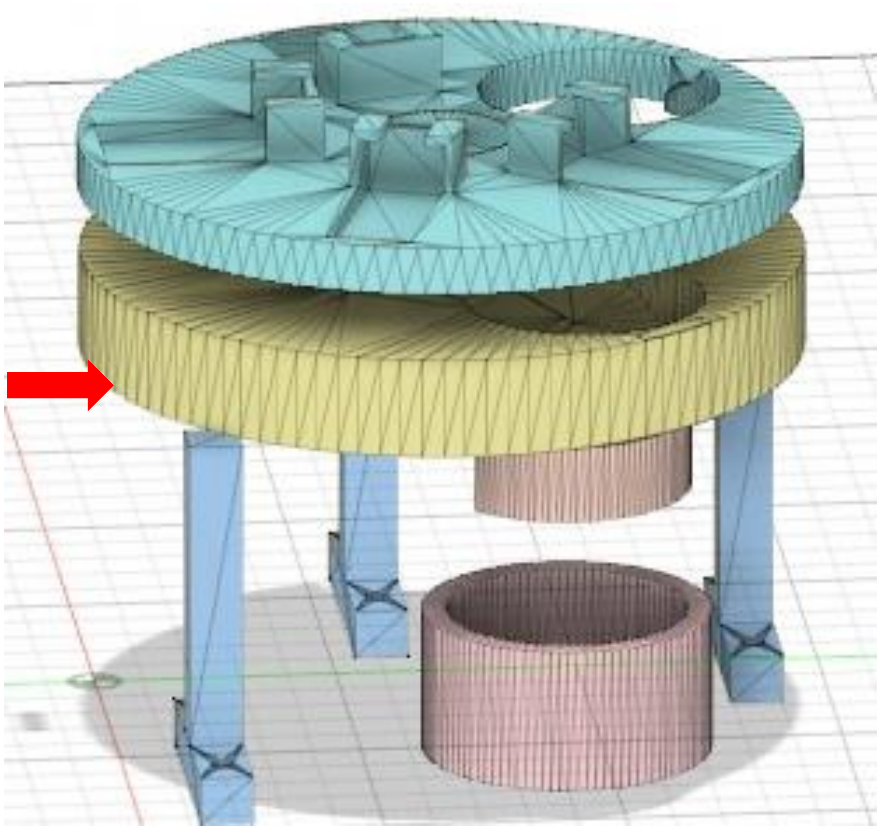


Figure 16- Tray Base

10. Assembly/disassembly procedure

10.1 Assembly

To assemble the pill dispenser, it is necessary to follow the steps below.

Do not force the parts at any time to avoid damage to the product.

1. It starts with the casing assembly, joining the faces with screws and nuts in the upper and lower corners (Figure 17).

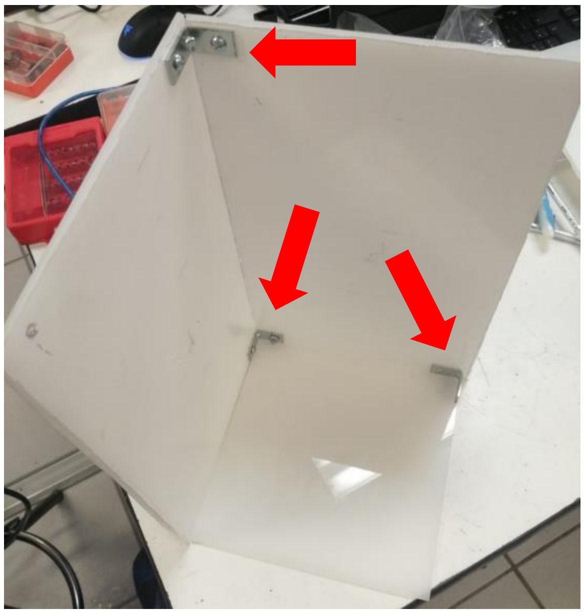


Figure 17– First parts assembled

2. Subsequently, the screen (1) and the keyboard (2) shown in Figure 18.

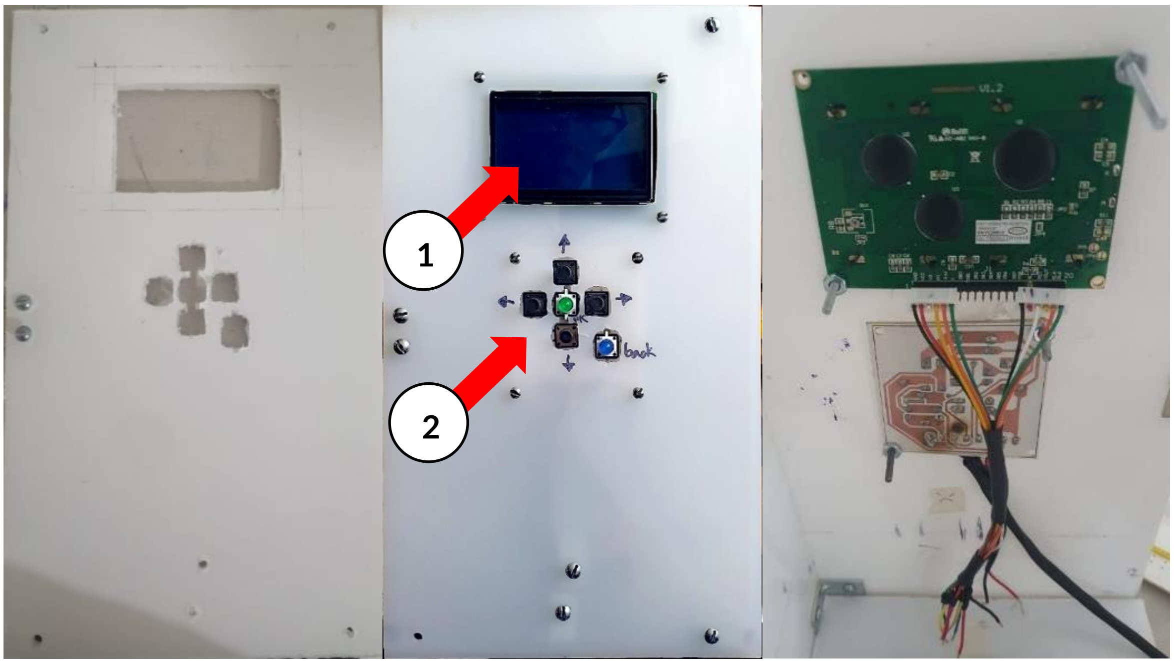


Figure 18– Front before and after adding the screen (1) and the keyboard (2) and interior view.

3. In the lower part of the rear face, the voltage regulator (Figure 19) is added, as well as the power supply connector and the power switch (Figure 20).

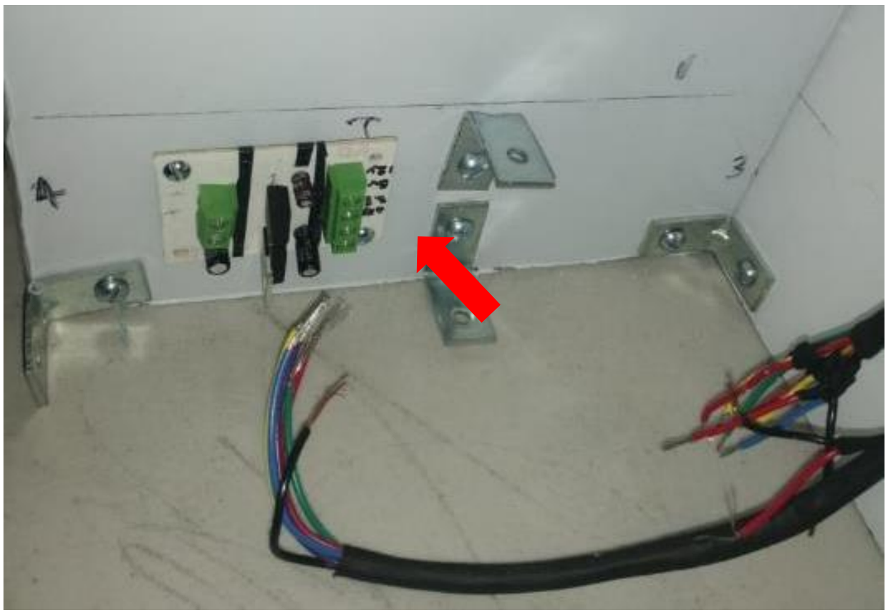


Figure 19- Voltage Regulator Added to Sidewall

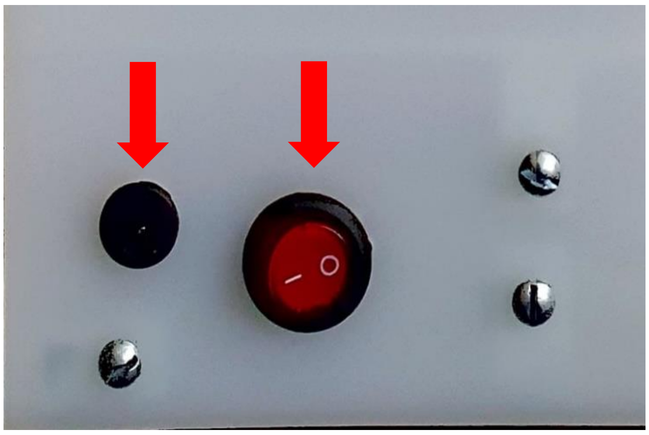


Figure 20- Back of Dispenser with Power Connector and Switch

⚠ Care must be taken that the voltage regulator is properly isolated.

4. Add the motherboard to the Arduino Mega 2560 and connect the RTC (1) and the Bluetooth module (2) shown in Figure 21.

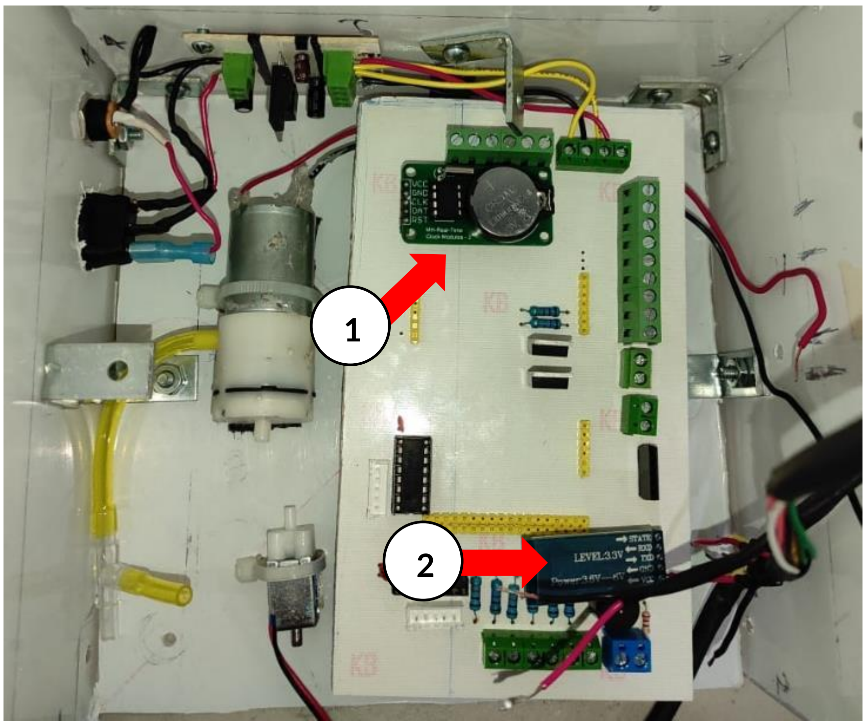


Figure 21- Motherboard with components installed

5. Connect the keyboard and screen to the motherboard based on the electrical schematic.

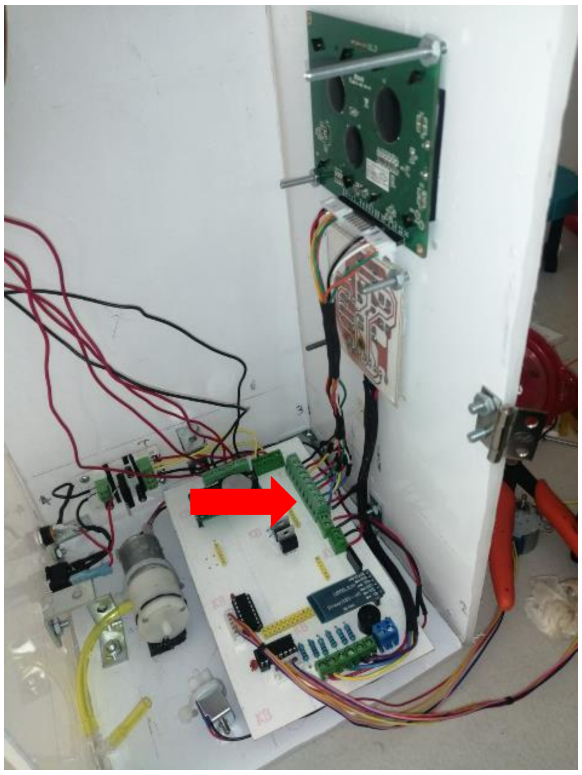


Figure 22– Electrical section fully connected.

6. Join the hose to the pneumatic motor (1) and to the solenoid valve (2) with a T-joint (3) shown in Figure 23.

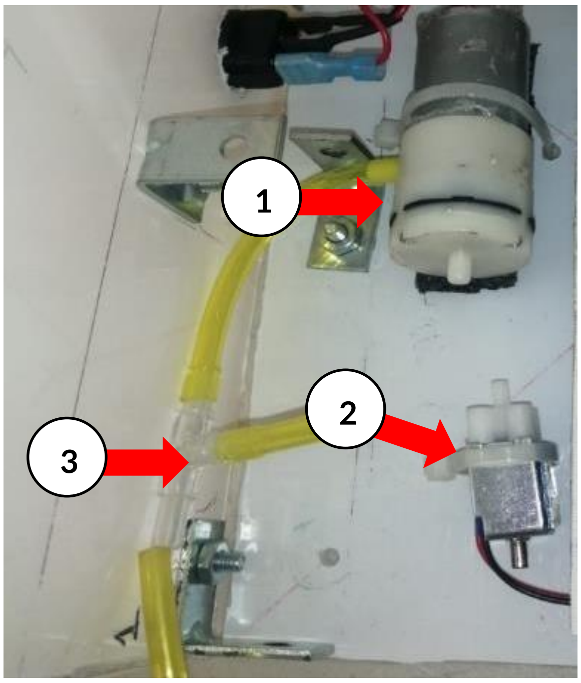


Figure 23– Connection of the hose with the pneumatic motor.

7. The intermediate division is added that will separate the base with the tray from the electrical part (Figure 24).

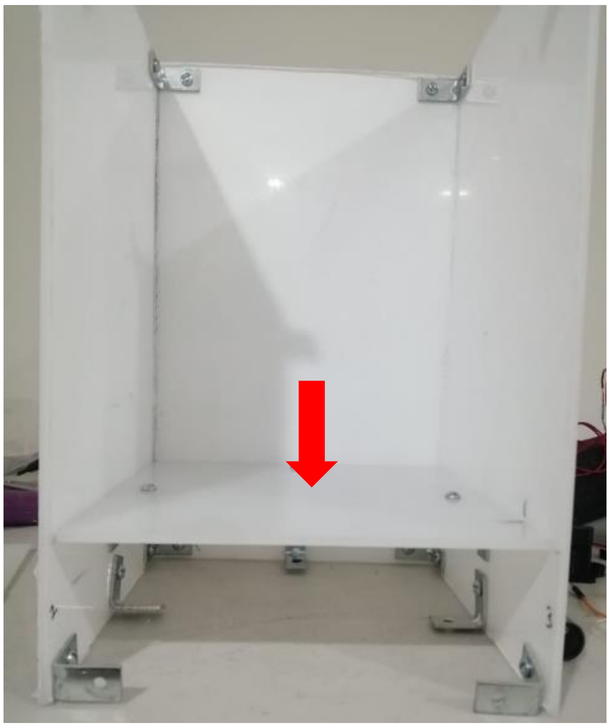


Figure 24– Casing with the division where the base will rest

8. The reed switch of the arm (1), the reed switch of the tray (2), reed switch of the cup (3) and the hose for the pneumatic system (4) shown in the figure are passed through the holes of the intermediate division. Figure 25.

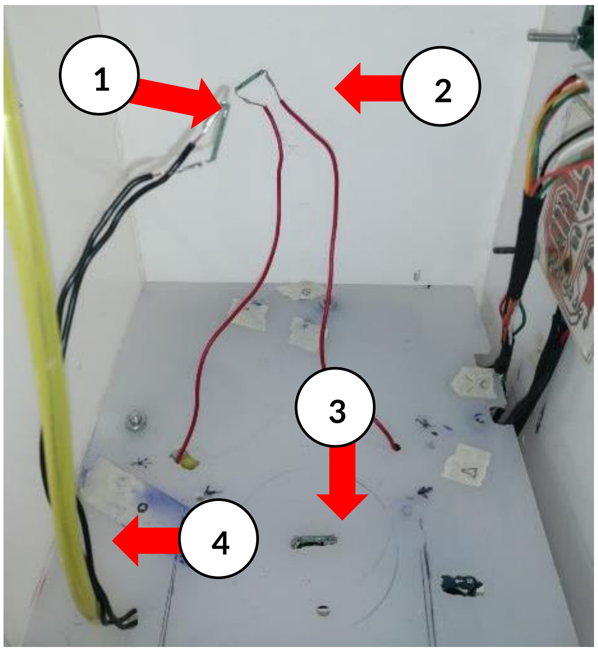


Figure 25– Arm Reed Switch (1), Tray Reed Switch (2), Hose (3)

Base assembly

9. The base is screwed to the intermediate division (1) and a stepper motor is inserted into the hole (2) shown in Figure 26.

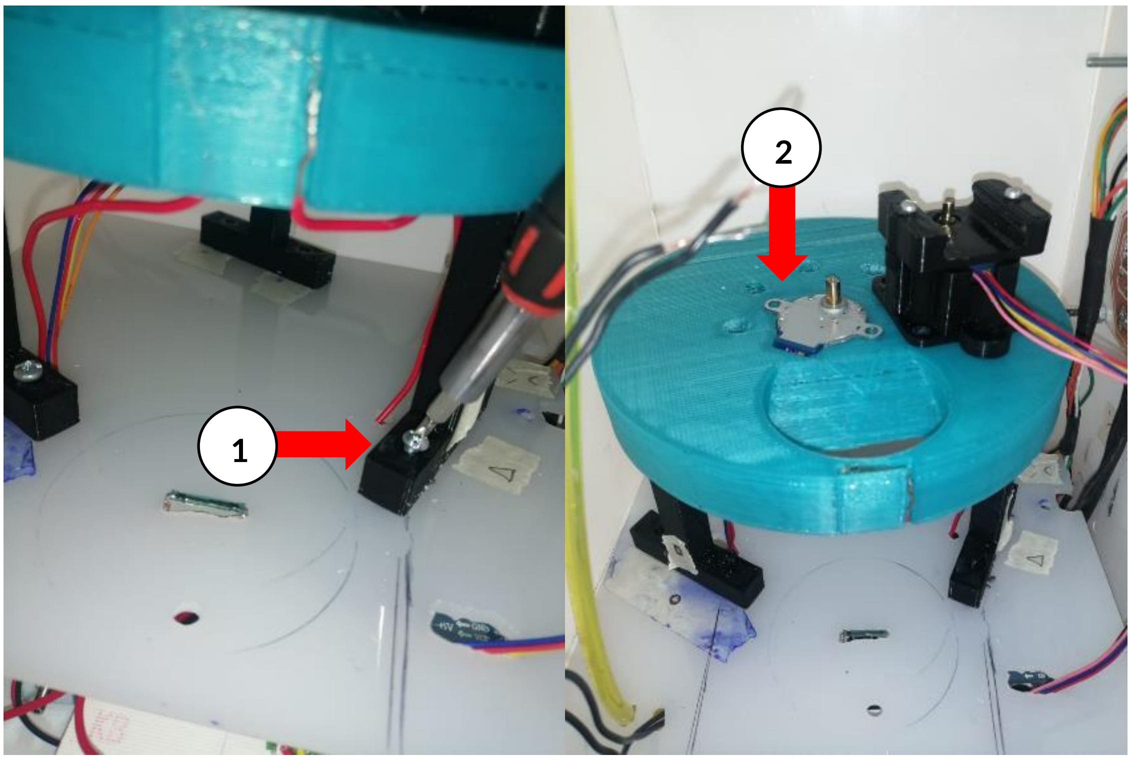


Figure 26- Base Secured to Intermediate Divider (1) and Stepper Motor Attached (2)

10. The six pellets are attached to the base to add greater stability (Figure 27).

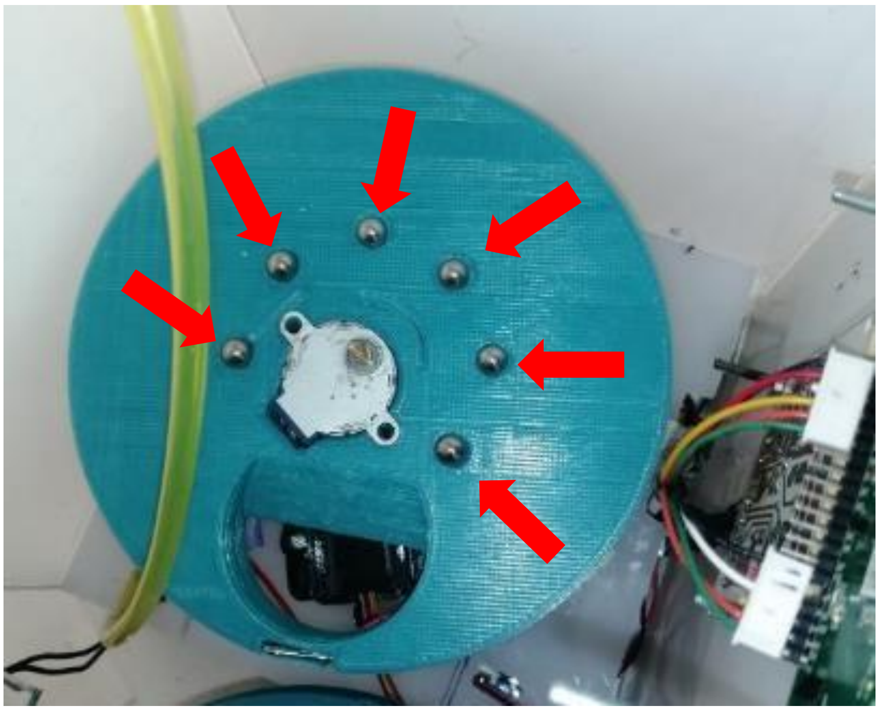


Figure 27– Bearings placed in the base

11. Place the tray and secure it to the motor previously placed on the base (Figure 28).

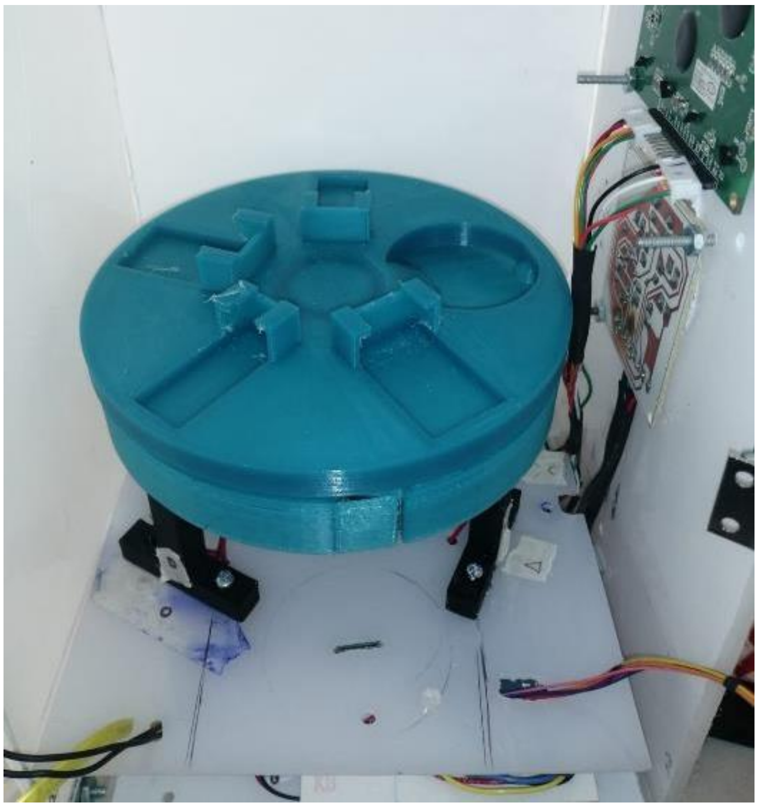


Figure 28– Base with turntable attached

Arm assembly

12. Place the stepper motor at the base of the arm (Figure 29).

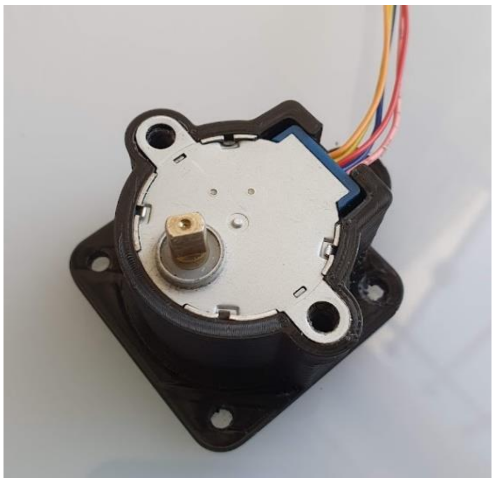


Figure 29- Stepper Motor at the Base of the Arm

13. Insert the arm reed switch into the joint shown in Figure 30.

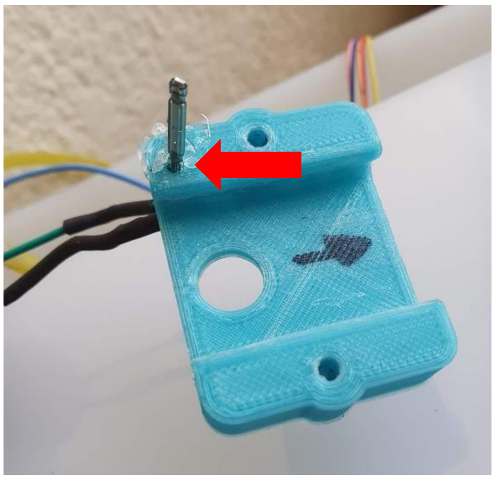


Figure 30- Reed switch placed at the junction

14. Screw the union to the motor base (1) and place the gear to the motor (2) shown in Figure 31.

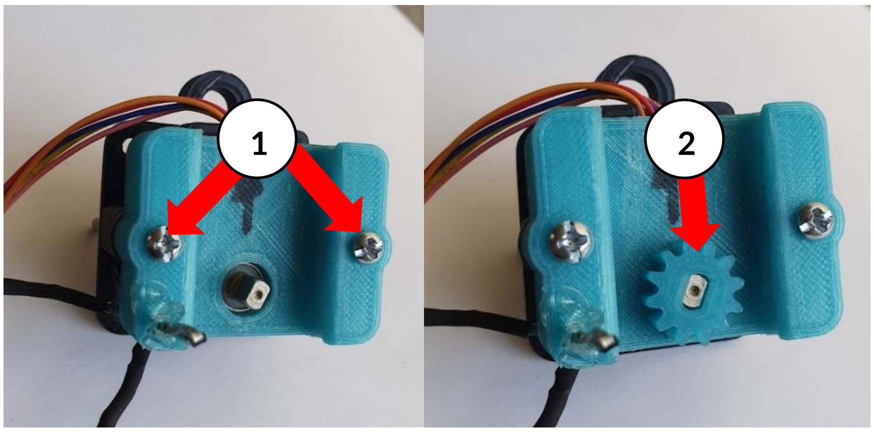


Figure 31- Bolted to Base (1) and Engaged Gear (2)

15. Insert the hose into the rack through the hole indicated by the arrow and add the bellows suction cup (Figure 32).

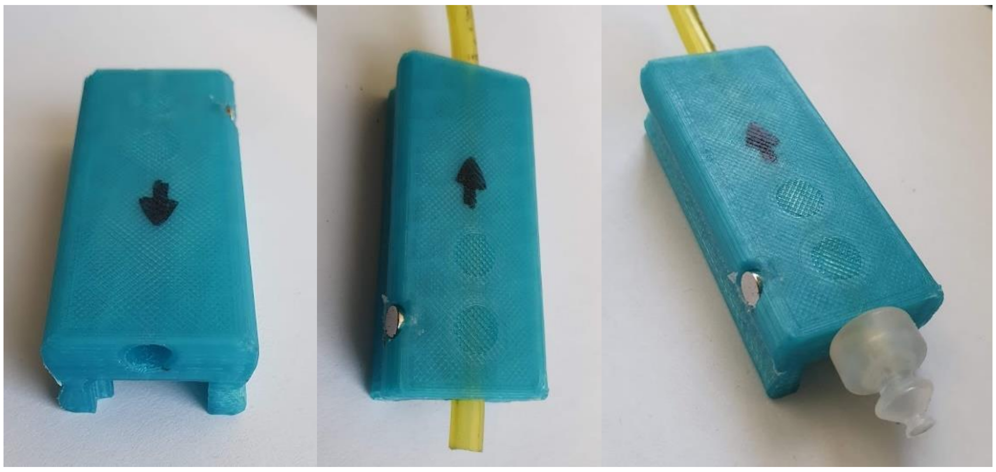


Figure 32- Hose Attached to Suction Cup Rail

16. Slide the zipper to the joint at the base of the arm (Figure 33)

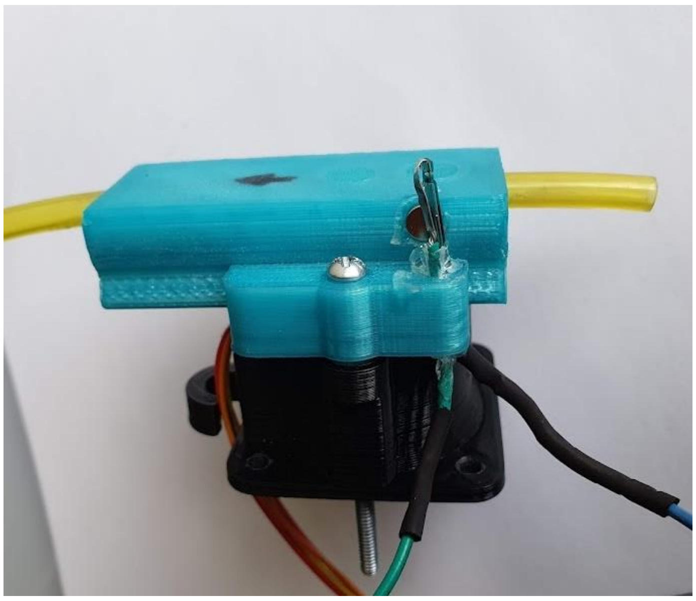


Figure 33- Rack Attached to the Base of the Arm

⚠ Be careful not to damage the gear when sliding the rail into the joint.

17. Screw the side face to the dispenser with the hinge
18. Screw the top cover to the dispenser with the hinge

19. Screw the assembled arm to the inside of the side face of the dispenser (Figure 34

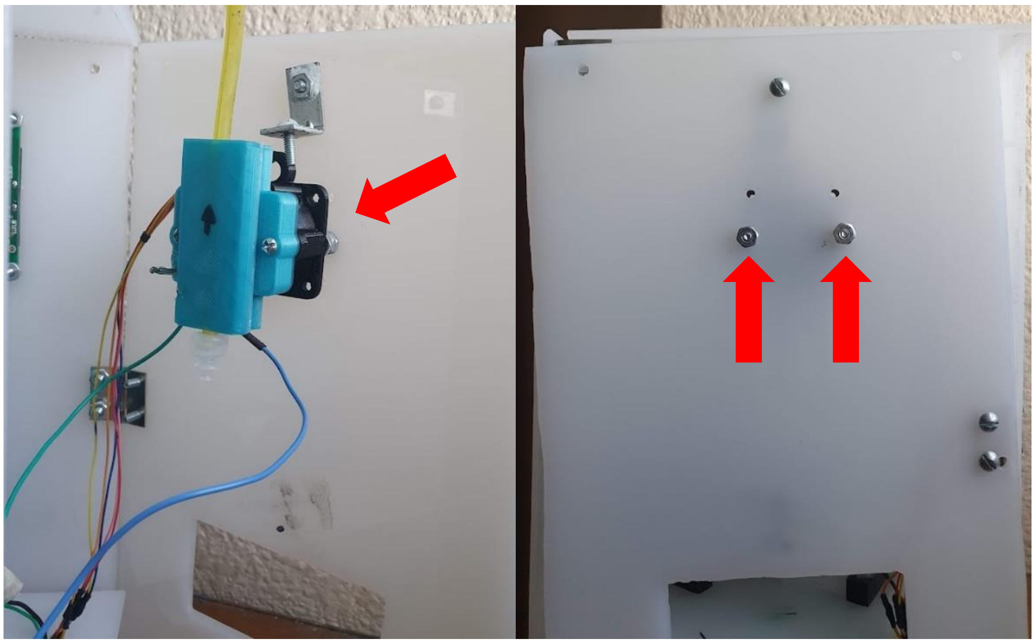


Figure 34- Arm Bolted to Dispenser

20. Connect the stepper motors according to the motherboard schematic (Figure 10) as shown in Figure 35.

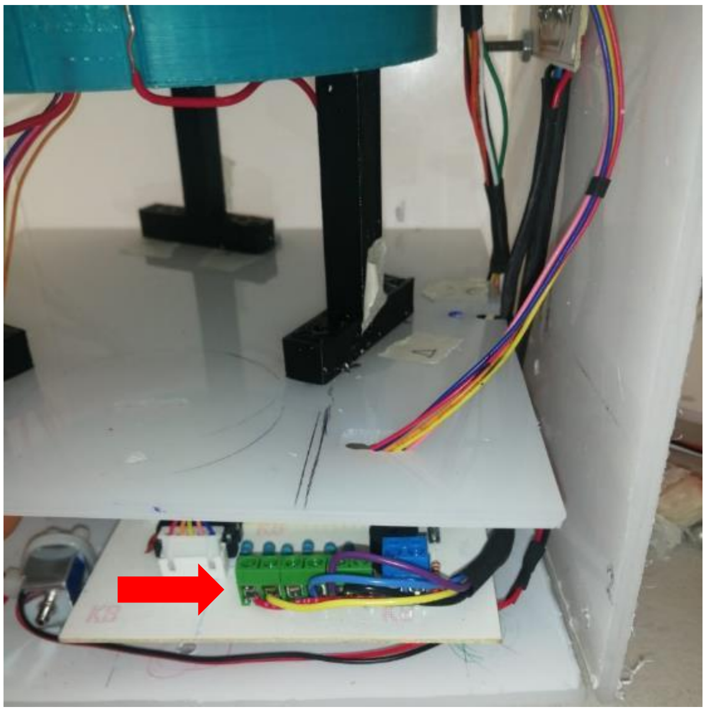


Figure 35– Connection of the motors in the electrical part.

21. Fix the lateral face to the dispenser with the remaining screws (Figure 36)

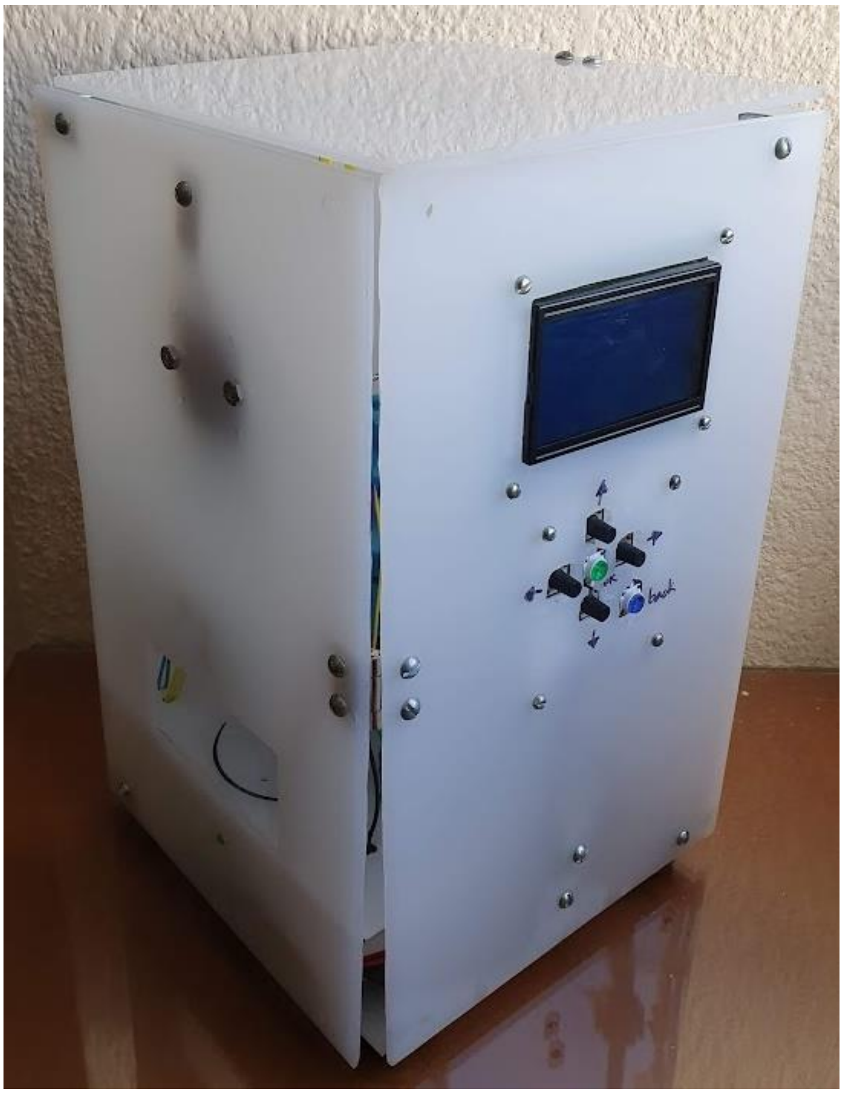


Figure 36- Assembled Pill Dispenser

22. Assembly of the automatic pill dispenser is complete.

✓

10.2 Disassembly

For disassembly, follow steps 1-21 in reverse order.

11. Tests to check the operation of the system

11.1 Voltage test

The 12V to 2A adapter included with the dispenser must first be connected to the power supply connector (Refer to Figure 2). Once connected, the different voltage stages of the internal regulator should be verified to be 3.3V, 5V and 12V with a multimeter (Figure 37).

Avoid using an adapter with a voltage greater than 12V to avoid damaging the device or causing injury.

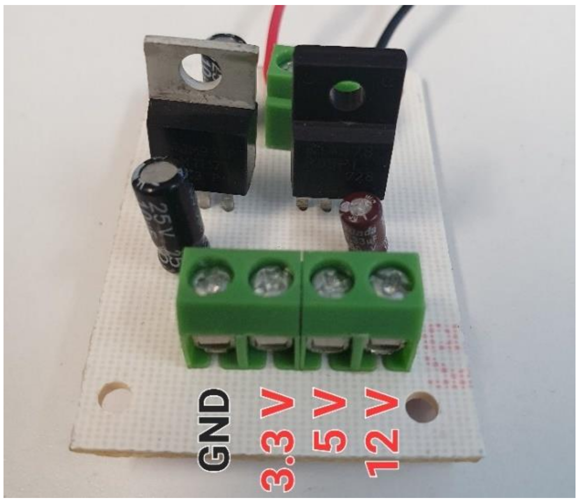


Figure 37- Internal Regulator Test Points

11.2 Initial setup

Once the regulator voltages are verified to be correct, the system should be turned on using the ignition switch. The tray must be rotated automatically, and the arm moved to its initial positions using the magnets together with the reed switches (Figure 38).

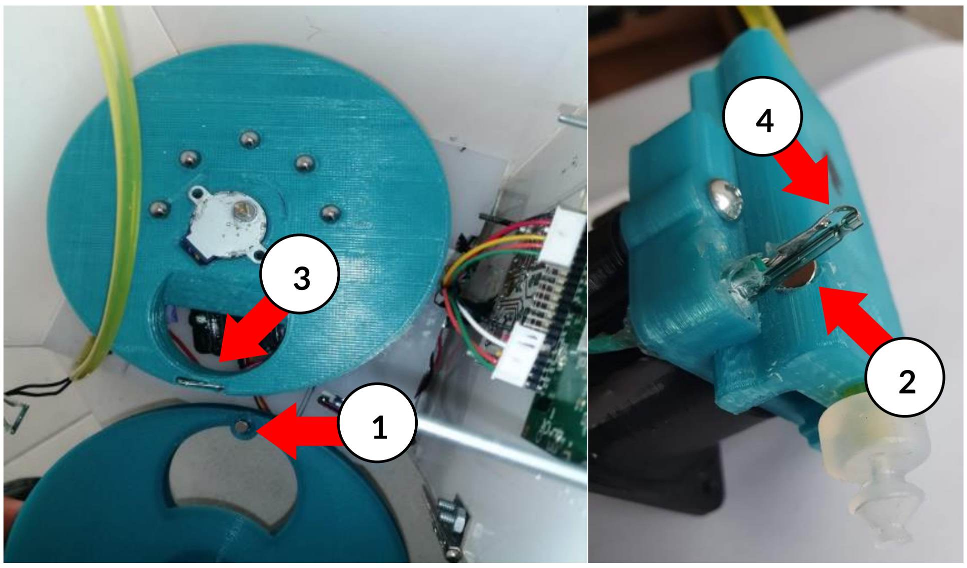


Figure 38– Magnet (1,3) and reed switch (2,4) of the tray and arm.

Once in its initial position, the screen should show the main menu and the clock should be updated every second (Figure 39).

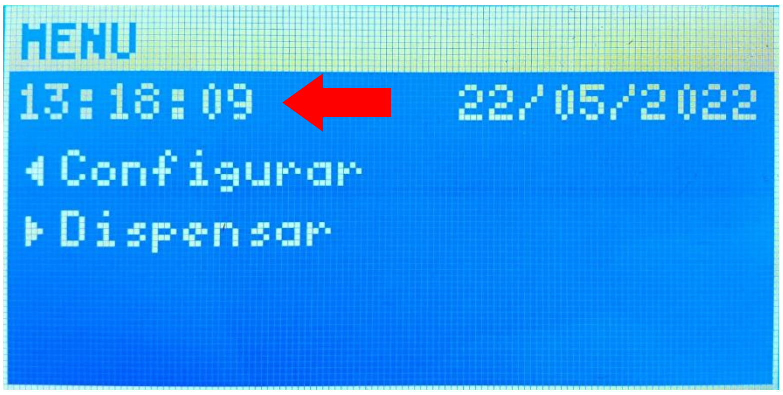


Figure 39- Initial Screen

11.3 Other tests

Device does not turn on

- Check the connection of the power adapter to the dispenser
- Verify that the voltages supplied by the regulator are 3.3V, 5V and 12V

stepper motors not turning

- Check that your space is not obstructed.
- Check that they are well connected to the motherboard

Rail does not move

- Check that there is nothing obstructing the gear path and that it is firmly seated on the stepper motor.

Screen does not turn on properly

- Verify that the power cables are well connected to the motherboard

Arm and/or tray do not return to their initial positions

- Check that the magnets make contact with the reed switches correctly
- Check continuity of the reed switches by bringing a magnet closer

Pills are not sucked

- Check that there are no obstructions both in the suction cup and in the hose
- Check for air leaks
- Check that the solenoid valve is properly connected
- Verify that the voltage received by the solenoid valve is 5V
- Check that the air motor is properly connected
- Verify that the voltage received by the pneumatic motor is 12V

If the problem persists, please contact support_tecnico@laresistencia.com by sending an email detailing the problem or by calling (+52) 449 123 4567.

Office hours: Monday to Friday from 9:00 am to 6:00 pm.

12. Codes with comments

12.1 Dispenser code

The source code for the dispenser is in the appendix, section A.

The program for the pill dispenser was made in C++11 using Microsoft's Visual Studio Code IDE together with the PlatformIO extension, which allows direct programming of the microcontroller without using the Arduino IDE.

12.2 Android app code

The application code blocks are shown in the appendix, section B.

The home screen (Figure 40) is made up of 3 buttons (Configure, Instructions, Options)

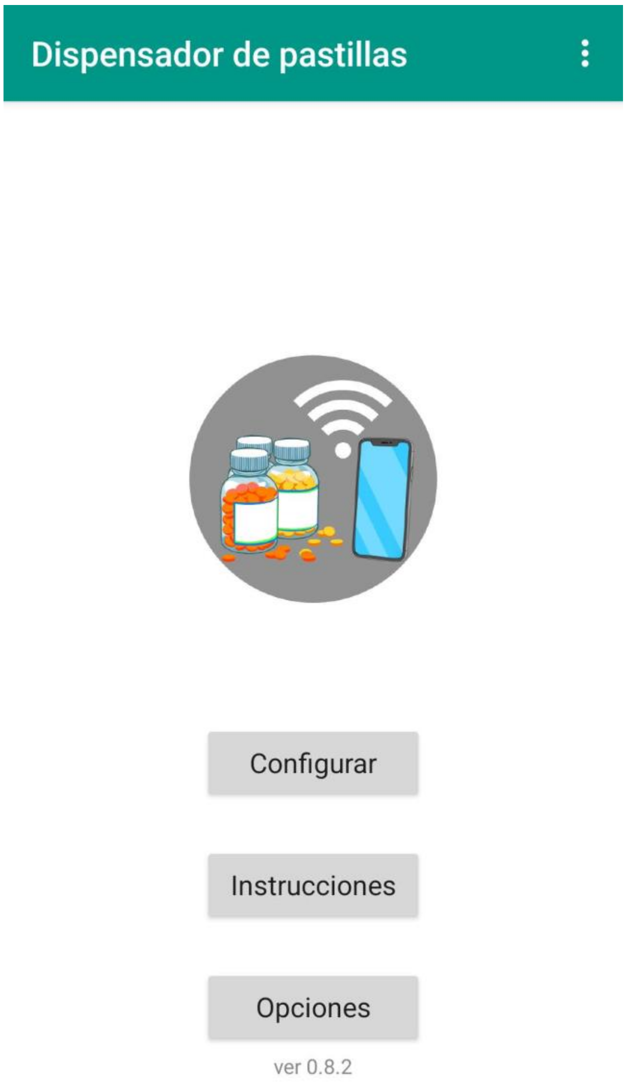
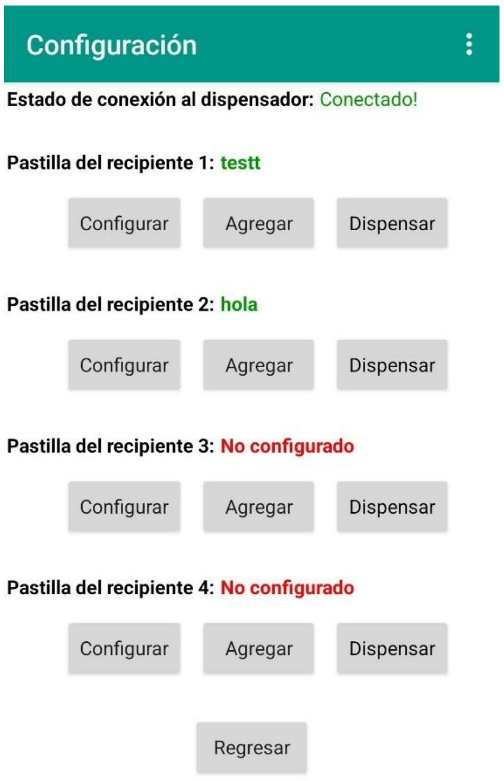


Figure 40– Home Screen

The setup screen (Figure 41) is where pill information is saved along with the treatment, as well as an option to add pills and dispense.



Estado de conexión al dispensador: Conectado!

Pastilla del recipiente 1: testt

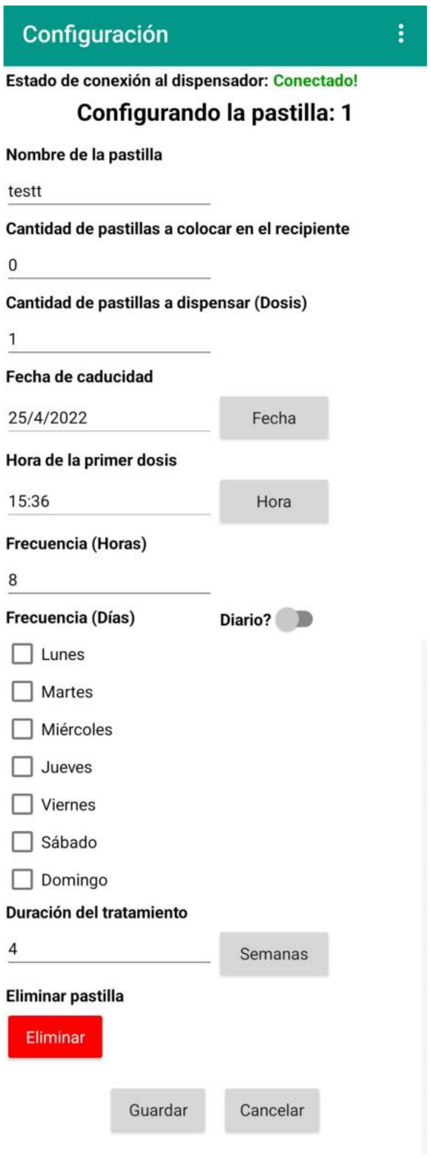
Pastilla del recipiente 2: hola

Pastilla del recipiente 3: No configurado

Pastilla del recipiente 4: No configurado

Figure 41 – Pickup Configuration Screen

The configuration screen for the selected tablet is where the data to be saved in the dispenser is entered (Figure 42).



Configuración

Estado de conexión al dispensador: Conectado!

Configurando la pastilla: 1

Nombre de la pastilla: testt

Cantidad de pastillas a colocar en el recipiente: 0

Cantidad de pastillas a dispensar (Dosis): 1

Fecha de caducidad: 25/4/2022 [Fecha]

Hora de la primer dosis: 15:36 [Hora]

Frecuencia (Horas): 8

Frecuencia (Días) Diario?

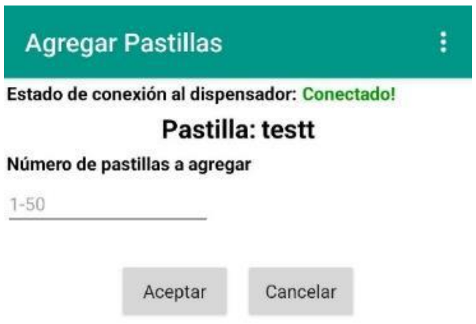
- ☐ Lunes
- ☐ Martes
- ☐ Miércoles
- ☐ Jueves
- ☐ Viernes
- ☐ Sábado
- ☐ Domingo

Duración del tratamiento: 4 [Semanas]

Eliminar pastilla

Figure 42– Pickup Configuration Screen

The add pills screen (Figure 43) consists of a simple text box to enter the number of pills.



Agregar Pastillas

Estado de conexión al dispensador: Conectado!

Pastilla: testt

Número de pastillas a agregar: 1-50

Figure 43– Screen to add pills

Finally, the options screen (Figure 44) is where you can connect to the dispenser via Bluetooth and set the time for the dispenser.

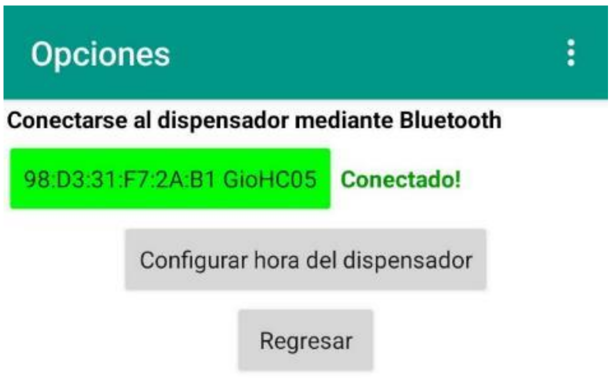

Opciones
Conectarse al dispensador mediante Bluetooth
98:D3:31:F7:2A:B1 GioHC05
Conectado!
Configurar hora del dispensador
Regresar

Figure 44– Options screen

13. Material's list

Table Table 3shows the materials used in the pill dispenser along with their part codes.

#	Part	part code	Quantity
1	Front face	(DISP-C01)	1
2	Left side face	(DISP-C02)	1
3	Right side face	(DISP-C03)	1
4	Rear face	(DISP-C04)	1
5	Upper face	(DISP-C05)	1
6	Underside	(DISP-C06)	1
7	Middle face	(DISP-C07)	1
9	Turntable	(DISP-B01)	1
10	Base (Tray)	(DISP-B02)	1
11	Zipper	(DISP-A01)	1
12	Base (Arm)	(DISP-A02)	1
13	Hinges	(DISP-D01)	2
14	Straps	(DISP-D02)	2
15	Mesh	(DISP-A03)	1
16	3mm diameter hose	(DISP-D03)	50cm
17	Corbels	(DISP-D04)	20
18	Rail (Arm)	(DISP-A04)	1
19	Screws	(DISP-D05)	40
20	Nuts	(DISP-D06)	40
21	Bellows Suction Cup (3mm)	(DISP-N01)	1

Table 3- Materials

14. List of electronic components

Table Table 4shows the electronic components used.

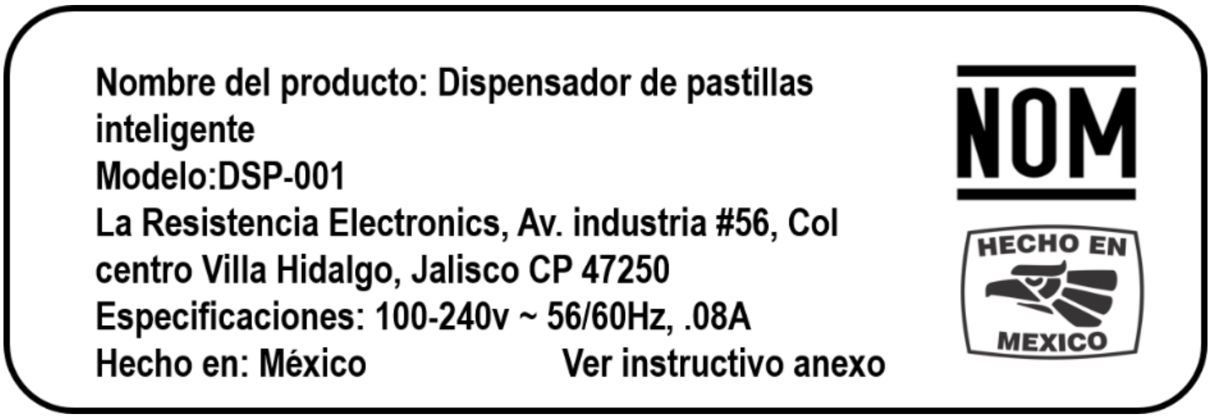
#	Name	part number	Quantity	Link
1	Arduino	Mega 2560 R3	1	link
2	Bluetooth module	HC-05	1	link
3	10 µF electrolytic capacitor 50V	MAL213650221E3	2	link
4	100 nF ceramic capacitor 50V	K104K15X7RF5UL2	2	link
5	0.33µF ceramic capacitor 50V	K334K20X7RF5TH5	1	link
6	Stepper motor driver	ULN2003AN	2	link
7	Air solenoid valve	FA0520E	1	link
8	Source 12VDC at 2A	PSD1202D	1	link
9	Green LED	C503B-GCN-CA0B0782	1	link
10	N-channel MOSFET	IRFZ44NPBF_C725092	2	link
11	Engine by steps	28BYJ-48	2	link
12	Pneumatic motor	ZR370-02PM	1	link
13	LCD screen 128x64	LCD-128H064A	1	link
14	Push-button	GT-TC152A-H130-L1	4	link
15	Reed-switch	MDSR-4-22-33	3	link
16	3.3V regulator	LM1117T-ADJ/NOPB	1	link
17	5V regulator	KA378R05TU	1	link
18	Real Time Clock (RTC)	DS1302	1	link
19	Resistor 10kΩ ¼W	MFR1WSFTE52-10K	8	link
20	Resistor 220Ω ¼W	279-CBT25J220R	3	link
21	Infrared sensor	2167	1	link
22	Power switch	471NS04268540	1	link
23	Buzzer	CMI-1210-5-95T	1	link

Table 4- Electronic Components

15. Norms and standards that it complies with

The pill dispenser complies with the following norms and standards:

- IEEE 802.15 standard
- NOM-003-SCFI-2000
- NOM-001-SCFI-1993
- NOM-024-SCFI-2013 (Figure 45)



Nombre del producto: Dispensador de pastillas inteligente
Modelo:DSP-001
La Resistencia Electronics, Av. industria #56, Col centro Villa Hidalgo, Jalisco CP 47250
Especificaciones: 100-240v ~ 56/60Hz, .08A
Hecho en: México Ver instructivo anexo

NOM
HECHO EN
MEXICO

Figure 45- Product labeling

16. Appendices

16.1 A – Dispenser Code

Below are the source files that were used to program the ATmega2560 microcontroller.

Main.cpp

```
#include <Arduino.h>
#include "Dispensador.h"
// Defines

// Constantes

// Variables
Dispensador dispensador;

void setup()
{
  dispensador.setup();
}

void loop()
{
  dispensador.loop();
}
```

Dispensador.cpp

```
#include <Arduino.h>
#include "Dispensador.h"
#include "EEPROM.h"
#include "DebugUtils.h"

/***************************************************************************
*****************/
// ASIGNACION DE PINES
/***************************************************************************
*****************/
// Pines del dispensador
#define PIN_SENSOR_PASTILLA      21  // Pin del Arduino Mega capaz de generar
interrupciones
#define PINS_TECLADO             35, 37, 39, 41, 43, 45 // r, l, u, d, o, b
#define PIN_BUZZER               27
#define PIN_LED_VASO             25
#define PIN_REED_VASO            22

// Pines del brazo
#define PINS_MOTOR_BRAZO         47, 49, 51, 53
#define PIN_MOTOR_NEUM           8
#define PIN_VALVULA              9
```

```
#define PIN_REED_BRAZO          24

// Pines de la bandeja
#define PINS_MOTOR_BAND         52, 50, 48, 46
#define PIN_REED_BAND           26

// Pines del módulo BT HC-05
// PIN_RX                       TX1
// PIN_TX                       RX1

// Pines del módulo RTC (DS1302)
#define PINS_RTC                36, 34, 32 // SCLK, IO, CE

// Pines para la pantalla
#define CLK_PIN                 10
#define DATA_PIN                11
#define CS_PIN                  12
#define RESET_PIN               13

/*****************************************************************************
*****************/
// CONSTANTES
/*****************************************************************************
*****************/
#define NOTIF_FREC_MINS         3
#define NUM_MAX_INTENTOS        4

/*****************************************************************************
*****************/
// VARIABLES GLOBALES
/*****************************************************************************
*****************/
static volatile bool pastillaDetectada;

// Constructor
// Inicializar objetos
Dispensador::Dispensador() :
    brazo(PINS_MOTOR_BRAZO, PIN_MOTOR_NEUM, PIN_VALVULA, PIN_REED_BRAZO),
    bandeja(PINS_MOTOR_BAND, PIN_REED_BAND),
    hc05(/*RX1, TX1*/),
    teclado(PINS_TECLADO),
    reloj(PINS_RTC),
    pantalla(CLK_PIN, DATA_PIN, CS_PIN, RESET_PIN)
{
    pastADispensar[0] = false;
    pastADispensar[1] = false;
    pastADispensar[2] = false;
    pastADispensar[3] = false;
    isFinishedDispensing = false;
    pastillaDetectada = false;
    playOnce = true;
}

// Destructor
Dispensador::~Dispensador()
{
}

// ------------------------- Implementación -------------------------
void Dispensador::setup()
{
#ifdef DEBUG
    // Init serial port
    Serial.begin(57600);
    // Wait until serial port is ready
    while (!Serial);
#endif

    DEBUG_PRINTLN("--------------------------------");
    DEBUG_PRINTLN("Inicializando...");
    // Sensor para detectar pastillas dispensadas
    pinMode(PIN_SENSOR_PASTILLA, INPUT_PULLUP);
    // Vincular una interrupcion de cambio de flanco negativo al pin
    attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_PASTILLA),
Dispensador::pastillaDetectadaISR, FALLING);
    pinMode(PIN_REED_VASO, INPUT_PULLUP);
    pinMode(PIN_LED_VASO, OUTPUT);
    DEBUG_PRINTLN("- Pines configurados!");
    pantalla.setup();
    DEBUG_PRINTLN("- Pantalla inicializada!");
    brazo.retornarPosInicial();
    DEBUG_PRINTLN("- Brazo en posicion inicial!");
    bandeja.retornarPosInicial();
    DEBUG_PRINTLN("- Bandeja en posición inicial!");

    int eeAddress = 100;
    for (int i = 0; i < 4; i++)
    {
        EEPROM.get(eeAddress*(i+1), pastilla[i]);
        pastilla[i].getPillInfo();
    }
    DEBUG_PRINTLN("- Pastillas recuperadas de la EEPROM!");

    DEBUG_PRINTLN("Dispensador inicializado!");
    reloj.updateTime();
    DEBUG_PRINT("Hora: ");
    DEBUG_PRINT(reloj.hours); DEBUG_PRINT(":"); DEBUG_PRINT(reloj.minutes);
DEBUG_PRINT(":"); DEBUG_PRINTLN(reloj.seconds);
    DEBUG_PRINTLN("--------------------------------\n");
}

void Dispensador::loop()
{
    reloj.updateTime();

    char* com;
    pantalla.botonPantalla = teclado.getTecla();
    pantalla.loop();

    pantalla.setDate(reloj.dayofmonth, reloj.month, reloj.year);
    pantalla.setTime(reloj.hours, reloj.minutes, reloj.seconds);
    pantalla.setRemainingPills(pastilla[0].pastillasRestantes,
pastilla[1].pastillasRestantes, pastilla[2].pastillasRestantes,
pastilla[3].pastillasRestantes);

    verificarHoraPastilla();

    com = hc05.getCommand();
    if(strncmp(com, "", 1) != 0)
    {
        DEBUG_PRINT("\n[BT]: ");
        DEBUG_PRINTLN(com);

        executeCommand(com);
    }

    com = pantalla.getCommand();
    if(strncmp(com, "", 1) != 0)
    {
        DEBUG_PRINT("\n[PANT]: ");
        DEBUG_PRINTLN(com);

        executeCommand(com);
    }

    verificarVaso();
}

// -------------------------- Implementación --------------------------
void Dispensador::dispensarPastilla(uint8_t recipiente)
{
    bandeja.seleccionarPastilla(recipiente);
    brazo.succionPastilla();
    bandeja.retornarPosInicial();
    brazo.soltar();

    delay(1000); // Tiempo para que la pastilla caiga al vaso
}

void Dispensador::dispensarPastilla(uint8_t recipiente, uint8_t cantidad)
{
    for (int i = 0; i < cantidad; i++)
    {
        dispensarPastilla(recipiente);
    }
}

/*
    Pasar direccion de memoria de pastilla para conservar estado
*/
void Dispensador::dispensarPastilla(Pastilla &pastilla)
{
    isFinishedDispensing = false;
    pastillaDetectada = false;

    if (pastilla.getRecipiente() == 0)
    {
        DEBUG_PRINTLN("*Error: No existe la pastilla!");
        return;
    }

    if (pastilla.verificarCantidad())
    {
        DEBUG_PRINT("Dispensando ");
        DEBUG_PRINT(pastilla.getDosis());
        DEBUG_PRINT(" pastillas de ");
        DEBUG_PRINT(pastilla.getNombre());
        DEBUG_PRINT(" (");
        DEBUG_PRINT(pastilla.pastillasRestantes);
        DEBUG_PRINT(" restantes) ");

        // Seguir intentando dispensar pastilla hasta que se detecte que cayó
        for (int intento = 0; intento < NUM_MAX_INTENTOS; intento++)
        {
            dispensarPastilla(pastilla.getRecipiente(), pastilla.getDosis());

            if(pastillaDetectada == true)
            {
                pastilla.pastillasRestantes -= pastilla.getDosis();
                break;
            }
        }

        if (pastillaDetectada == false)
        {
            isFinishedDispensing = false;

            DEBUG_PRINTLN("*Error: Nunca se detectó caer la pastilla!");
            return;
        }

        isFinishedDispensing = true;
        playOnce = true;

        int eeAddress = 100 * pastilla.getRecipiente();
        EEPROM.put(eeAddress, pastilla);
    }
    else
    {
        DEBUG_PRINTLN("Ya no quedan suficientes pastillas!");
        DEBUG_PRINT("(Quedan: ");
        DEBUG_PRINT(pastilla.pastillasRestantes);
        DEBUG_PRINT(", se necesitan: ");
        DEBUG_PRINT(pastilla.getDosis());
        DEBUG_PRINTLN(")");
        isFinishedDispensing = false;
    }
}

/*
    Pasar direccion de memoria de pastilla para conservar estado
*/
void Dispensador::eliminarPastilla(Pastilla &pastilla)
{
    if(pastilla.getRecipiente() == 0)
    {
        DEBUG_PRINTLN("*Error: Pastilla ya estaba sin configurar!");
        return;
    }
    pastilla.eliminar();

    int eeAddress = 100 * pastilla.getRecipiente();
    EEPROM.put(eeAddress, pastilla);

    DEBUG_PRINTLN("Pastilla eliminada!");
}

void Dispensador::eliminarPastilla(uint8_t recipiente)
{
    eliminarPastilla(pastilla[recipiente]);
}

/*
    Pasar direccion de memoria de pastilla para conservar estado
*/
void Dispensador::agregarPastillas(Pastilla &pastilla, uint8_t cantidad)
{
    if(pastilla.getRecipiente() == 0)
    {
        DEBUG_PRINTLN("*Error: Pastilla no configurada!");
        return;
    }
    // agregarPastillas(pastilla.getRecipiente(), cantidad);
    pastilla.pastillasRestantes += cantidad;

    int eeAddress = 100 * pastilla.getRecipiente();
    EEPROM.put(eeAddress, pastilla);

    DEBUG_PRINT("Agregadas ");
    DEBUG_PRINT(cantidad);
    DEBUG_PRINT(" pastillas de ");
    DEBUG_PRINT(pastilla.getNombre());
    DEBUG_PRINT(" (");
    DEBUG_PRINT(pastilla.pastillasRestantes);
    DEBUG_PRINTLN(" en total)");
}

void Dispensador::agregarPastillas(uint8_t recipiente, uint8_t cantidad)
{
}

// Interrupt Service Routine (ISR)
void Dispensador::pastillaDetectadaISR()
{
    pastillaDetectada = true;
    DEBUG_PRINT("*");
}

/*
    Ejecuta el comando (char array) recibido
*/
void Dispensador::executeCommand(char* command)
{
    if(strncmp(command, "", 1) != 0)
    {
        // Verifica el opcode para ejecutar la función correspondiente
        // Configura la pastilla
        if(strncmp(command, "SET", 3) == 0)
        {
            char opCode[4];
            int recipiente;     // Recipiente en la bandeja
            char nombre[20];    // Nombre de la pastilla
            int dosis;          // Numero de pastillas a dispensar
            int caducidad[3];   // dd, mm, aaaa
            int frecuencia;     // Cada cuánto tiempo se tiene que dispensar
            int primerDosis[2]; // hh, mm
            int duracionTratamiento;
            int duracionTiempo; // Días/Semanas/Meses/Años
            int diario;         // Opcion para marcar si es diario o no
            int dias[7];        // Días a dispensar

            sscanf(command, "%s %i %s %i %i %i %i %i %i %i %i %i %i %i %i %i
%i %i %i %i",
                opCode, &recipiente, nombre,
                &dosis,
                &caducidad[0], &caducidad[1], &caducidad[2],
                &frecuencia,
                &primerDosis[0], &primerDosis[1],
                &duracionTratamiento, &duracionTiempo,
                &diario, &dias[0], &dias[1], &dias[2], &dias[3], &dias[4],
&dias[5], &dias[6]
                );

            // Map int to bool
            bool diarioAux = !!diario;
            bool diasAux[7];
            for (int i = 0; i < 7; i++)
            {
                diasAux[i] = !!dias[i];
            }

            pastilla[recipiente-1].setup(/*nombre,*/ dosis, recipiente,
frecuencia, caducidad, primerDosis, duracionTratamiento, duracionTiempo,
diarioAux, diasAux);

            int eeAddress = 100 * recipiente;
            EEPROM.put(eeAddress, pastilla[recipiente-1]);

            pastilla[recipiente-1].getPillInfo();
        }
        // Agrega pastillas a una pastilla ya configurada previamente
        else if (strncmp(command, "ADD", 3) == 0)
        {
            int recipiente, cantidad;
            char opCode[4];
            sscanf(command, "%s %i %i", opCode, &recipiente, &cantidad);

            agregarPastillas(pastilla[recipiente-1], cantidad);
        }
        // Elimina una pastilla configurada
        else if (strncmp(command, "DEL", 3) == 0)
        {
            int recipiente;
            char opCode[4];
            sscanf(command, "%s %i", opCode, &recipiente);

            eliminarPastilla(pastilla[recipiente-1]);
        }
        // Intenta dispensar una pastilla configurada
        else if (strncmp(command, "DIS", 3) == 0)
        {

            int recipiente;
            char opCode[4];
            sscanf(command, "%s %i", opCode, &recipiente);

            dispensarPastilla(pastilla[recipiente-1]);
        }
        // Prueba para medir la corriente del dispensador
        else if (strncmp(command, "TST", 3) == 0)
        {
            testCorriente();
            DEBUG_PRINTLN("Test para medir corriente iniciado!");
        }
        // Configura pastilla mediante el dispensador
        else if (strncmp(command, "PST", 3) == 0)
        {
            int recipiente;
            char opCode[4];

            sscanf(command, "%s %i", opCode, &recipiente);

            // Asigna pastilla auxiliar usando copy constructor
            pastilla[recipiente-1] = pantalla.getPastilla();

            int eeAddress = 100 * recipiente;
            EEPROM.put(eeAddress, pastilla[recipiente-1]);

            pastilla[recipiente-1].getPillInfo();
        }
        // Configura el RTC mediante la hora y fecha del teléfono
        else if (strncmp(command, "TIM", 3) == 0)
        {
            char opCode[4];
            int date[3]; // dd/MM/yyyy
            int time[3]; // hh:mm:ss

            sscanf(command, "%s %i %i %i %i %i %i", opCode, &date[0],
&date[1], &date[2], &time[0], &time[1], &time[2]);

            reloj.setDS1302Time(time[2], time[1], time[0], 1, date[0],
date[1], date[2]);
            DEBUG_PRINTLN("Hora configurada!");
            DEBUG_PRINT("Hora: ");
            DEBUG_PRINT(reloj.hours); DEBUG_PRINT(":");
DEBUG_PRINT(reloj.minutes); DEBUG_PRINT(":"); DEBUG_PRINTLN(reloj.seconds);
        }
    }

    free(command);
}

void Dispensador::testCorriente()
{
    brazo.testCorriente();
    bandeja.testCorriente();
}

void Dispensador::playMelody(int opcion)
{

    // if(opcion == 1)
    // {
        int melody[8] = {NOTE_C5, NOTE_E5, NOTE_C6, NOTE_B5, NOTE_G5, 0, 0,
0};
        int noteDurations[8] = {4, 4, 4, 2, 2, 4, 4, 4};
    // }
    // else
    // {
    //     int melody[8] = {NOTE_C5, NOTE_E5, NOTE_C6, NOTE_B5, NOTE_G5, 0, 0,
0};
    //     int noteDurations[8] = {4, 4, 4, 2, 2, 4, 4, 4};
    // }

    for (int thisNote = 0; thisNote < 8; thisNote++) {
        // to calculate the note duration, take one second divided by the note
type.
        // e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
        int noteDuration = 1000 / noteDurations[thisNote];
        tone(PIN_BUZZER, melody[thisNote], noteDuration);
        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        int pauseBetweenNotes = noteDuration * 1.20;
        delay(pauseBetweenNotes);

        // stop the tone playing:
        noTone(PIN_BUZZER);
    }
}

void Dispensador::ledNotification()
{
    digitalWrite(PIN_LED_VASO, HIGH);
}

void Dispensador::verificarHoraPastilla()
{
    // Verificar para todas las pastillas
    for (int i = 0; i < 4; i++)
    {
        if(pastilla[i].getRecipiente() != 0)
        {
            uint8_t frecuenciaAux   = pastilla[i].getFrecuencia();
            int *primerDosisAux     = pastilla[i].getPrimerDosis();

            // Verifica que haya pasado las horas de la frecuencia desde la
primer toma
            // TODO: Cambiar a horas y minutos
            // int temp = (reloj.hours - primerDosisAux[0]) % frecuenciaAux;
            // if(temp == 0 && (reloj.minutes == primerDosisAux[1]))

            int temp = (reloj.minutes - primerDosisAux[0]) % frecuenciaAux;
            if(temp == 0 && (reloj.seconds == primerDosisAux[1]))
            {
                pastADispensar[i] = true;
            }
        }
    }

    dispensarPastillas();
}

void Dispensador::dispensarPastillas()
{
    for (int i = 0; i < 4; i++)
    {
        if (pastADispensar[i] == true)
        {
            DEBUG_PRINT("\n[TIME ");
            DEBUG_PRINT(reloj.hours); DEBUG_PRINT(":");
DEBUG_PRINT(reloj.minutes); DEBUG_PRINT(":"); DEBUG_PRINT(reloj.seconds);
            DEBUG_PRINT("]: ");
            DEBUG_PRINT("Es hora de tomar la pastilla "); DEBUG_PRINTLN(i+1);

            dispensarPastilla(pastilla[i]);
            delay(1000);
            pastADispensar[i] = false;
        }
    }
}

/**
 * Funcion que se encarga de verificar si el vaso con las pastillas ha sido
retirado
 */
void Dispensador::verificarVaso()
{
    if(isFinishedDispensing)
    {
        if(playOnce)
        {
            ledNotification();
            playMelody(1);
            playOnce = false;
        }

        // HIGH = Imán alejado, LOW = Imán pegado
        if(digitalRead(PIN_REED_VASO) == HIGH)
        {
            // Apagar luz cuando se retira el vaso
            digitalWrite(PIN_LED_VASO, LOW);
            isFinishedDispensing = false;
        }

        // Notificar cada cierto tiempo
        if((reloj.minutes % NOTIF_FREC_MINS) == 0 && (reloj.seconds == 0))
        {
            DEBUG_PRINT("\n[TIME ");
            DEBUG_PRINT(reloj.hours); DEBUG_PRINT(":");
DEBUG_PRINT(reloj.minutes); DEBUG_PRINT(":"); DEBUG_PRINT(reloj.seconds);
            DEBUG_PRINT("]: ");
            DEBUG_PRINTLN("Pastillas no retiradas!!");

            playMelody(1);
        }
    }
```

```
}
```

Dispensador.h

```
#ifndef Dispensador_h
#define Dispensador_h

#include <Arduino.h>
#include "Brazo.h"
#include "Bandeja.h"
#include "Pastilla.h"
#include "HC05.h"
#include "virtuabotixRTC.h"
#include "Pantalla.h"
#include "Teclado.h"
#include "Pitches.h"

#include "DebugUtils.h"

#define NUM_PASTILLAS   4

class Dispensador
{
private:
    // Instanciar objetos
    Brazo       brazo;
    Bandeja     bandeja;
    Pastilla    pastilla[NUM_PASTILLAS];
    HC05        hc05;
    Teclado     teclado;
    virtuabotixRTC reloj;
    Pantalla    pantalla;

    bool pastADispensar[NUM_PASTILLAS];
    bool playOnce;
    bool isFinishedDispensing;

public:
    Dispensador();
    ~Dispensador();

    void setup();
    void loop();

    void eliminarPastilla(Pastilla &pastilla);
    void eliminarPastilla(uint8_t recipiente);

    void agregarPastillas(Pastilla &pastilla, uint8_t cantidad);
    void agregarPastillas(uint8_t recipiente, uint8_t cantidad);

    void dispensarPastilla(uint8_t recipiente);
    void dispensarPastilla(uint8_t recipiente, uint8_t cantidad);
    void dispensarPastilla(Pastilla &pastilla);
    void dispensarPastillas();

    void verificarHoraPastilla();

    void playMelody(int option);
    void ledNotification();

    void verificarVaso();
    static void pastillaDetectadaISR();

    void executeCommand(char* command);

    void testCorriente();
};

#endif
```

Bandeja.cpp

```
#include "Arduino.h"
#include "Bandeja.h"


/*
    Total de pasos -> 512 = 360° (una vuelta)
    Se tienen 5 recipientes -> 360°/5 = 72° por recipiente

    PASOS_RECIPIENTE = 72°*512 pasos/360°
    PASOS_RECIPIENTE = 102.4 pasos
*/
#define PASOS_RECIPIENTE 103
#define GRADOS_RECIPIENTE (uint16_t)72

#define DERECHA 1
#define IZQUIERDA 0

// Constructor
// Inicializar objeto de tipo MotorPasos con "Initialization List"
Bandeja::Bandeja(int in1, int in2, int in3, int in4, int pinReed) :
    motorBandeja(in1, in2, in3, in4)
{
    this->pinReed = pinReed;
    // Reed switch para detección de posición de la bandeja
    pinMode(pinReed, INPUT_PULLUP);
}

// Destructor
Bandeja::~Bandeja()
{
}

// ------------------------- Implementación -------------------------
void Bandeja::retornarPosInicial()
{
    // HIGH = Imán alejado, LOW = Imán pegado
    while (digitalRead(pinReed) == HIGH)
    {
        motorBandeja.girar(DERECHA, 1);
    }
    motorBandeja.apagar();
}

void Bandeja::seleccionarPastilla(uint8_t pastilla)
{
    motorBandeja.girar(DERECHA, pastilla * PASOS_RECIPIENTE);
    motorBandeja.apagar();
}

void Bandeja::vibrar()
{
    motorBandeja.vibrar();
    motorBandeja.apagar();
}

void Bandeja::testCorriente()
{
    motorBandeja.derecha();
}
```

Bandeja.h

```
#ifndef Bandeja_h
#define Bandeja_h

#include <Arduino.h>
#include "MotorPasos.h"
#include "DebugUtils.h"

class Bandeja
{
private:
    // Instanciar objeto de tipo MotorPasos para el motor de la bandeja
    MotorPasos motorBandeja;

    int pinReed;

public:
    Bandeja(int in1, int in2, int in3, int in4, int pinReed);
    ~Bandeja();

    void retornarPosInicial();
    void seleccionarPastilla(uint8_t pastilla);
    void vibrar();

    void testCorriente();
};

#endif
```

Brazo.cpp

```
#include "Arduino.h"
#include "Brazo.h"

/*
    Total de pasos -> 512 = 360° (una vuelta)
    El brazo necesita recorrer 70° para llegar al fondo del recipiente

    PASOS_FONDO = 70°*512 pasos/360°
    PASOS_FONDO = 115 pasos
*/
#define PASOS_FONDO 610
#define GRADOS_FONDO (uint8_t)70

#define ABAJO 1
#define ARRIBA 0

// Constructor
// Inicializar objeto MotorPasos con "Initialization List"
Brazo::Brazo(int in1, int in2, int in3, int in4, int pinMotorNeum, int
pinValvula, int pinReed) :
    motorBrazo(in1, in2, in3, in4)
{
    this->pinMotorNeum = pinMotorNeum;
    this->pinValvula = pinValvula;
    this->pinReed = pinReed;

    // Motor neumático para levantar pastillas
    pinMode(pinMotorNeum, OUTPUT);
    // Valvula con solenoide electrico
    pinMode(pinValvula, OUTPUT);
    // Reed switch para detección de posición del brazo
    pinMode(pinReed, INPUT_PULLUP);
}

// Destructor
Brazo::~Brazo()
{
}

// -------------------------- Implementación --------------------------
void Brazo::retornarPosInicial()
{
    // HIGH = Imán alejado, LOW = Imán pegado
    while (digitalRead(pinReed) == HIGH)
    {
        motorBrazo.girar(ARRIBA, 1);
    }
    motorBrazo.apagar();
}

void Brazo::succionPastilla()
{
    retornarPosInicial();
    motorBrazo.girar(ABAJO, PASOS_FONDO);
    motorBrazo.apagar();
    comenzarSuccion(1000);
    retornarPosInicial();
}

void Brazo::comenzarSuccion(int ms)
{
    digitalWrite(pinMotorNeum, HIGH);
    digitalWrite(pinValvula, LOW);  // Cierra válvula para crear vacío
    delay(ms);
}

void Brazo::soltar()
{
    // motorBrazo.girar(ABAJO, 500);
    // motorBrazo.apagar();
    detenerSuccion();
    retornarPosInicial();
}

void Brazo::detenerSuccion()
{
    digitalWrite(pinMotorNeum, LOW);
    digitalWrite(pinValvula, HIGH); // Abre válvula para ecualizar presión
    delay(100);
    digitalWrite(pinValvula, LOW);  // Regresa al estado bajo lógico

}

void Brazo::testCorriente()
{
    motorBrazo.derecha();
    digitalWrite(pinMotorNeum, HIGH);
    digitalWrite(pinValvula, HIGH);
}
```

Brazo.h

```
#ifndef Brazo_h
#define Brazo_h

#include <Arduino.h>
#include "MotorPasos.h"
#include "DebugUtils.h"

class Brazo
{
private:
    // Instanciar objeto de tipo MotorPasos para el motor del brazo
    MotorPasos motorBrazo;

    int pinMotorNeum;
    int pinValvula;
    int pinReed;

public:
    Brazo(int in1, int in2, int in3, int in4, int pinMotorNeum, int
pinValvula, int pinReed);
    ~Brazo();

    void retornarPosInicial();
    void succionPastilla();
    void comenzarSuccion(int ms);
    void detenerSuccion();
    void soltar();

    void testCorriente();
};

#endif
```

Boton.cpp

```
#include "Boton.h"
#include "DebugUtils.h"

// Constructor
Boton::Boton()
{
  lastButtonState = 0;
  buttonState = 0;
}

// Destructor
Boton::~Boton()
{
}

// -------------------------- Implementación --------------------------
void Boton::setup(uint8_t pBoton, char boton)
{
  this->pBoton = pBoton;
  this->boton = boton;
  pinMode(pBoton, INPUT);
}

char Boton::getBoton()
{
  char aux = '\0';
  buttonState = digitalRead(pBoton);
  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      // if the current state is HIGH then the button went from off to on:

    } else {
      // if the current state is LOW then the button went from on to off:
      aux = boton;
    }
    // Delay a little bit to avoid bouncing
    delay(20);
  }
  // save the current state as the last state, for next time through the loop
  lastButtonState = buttonState;

  // Si el botón no se presionó, retorna el caracter nulo '\0'
  return aux;
}
```

Boton.h

```
#ifndef Boton_h
#define Boton_h

#include <Arduino.h>

class Boton
{
private:
    uint8_t pBoton;  // Pin del boton
    char boton; // char correspondiente al boton presionado para identificarlo
    int buttonState, lastButtonState;

public:
    Boton();
    ~Boton();

    void setup(uint8_t pBoton, char boton);
    char getBoton(); // Retorna boton presionado
};

#endif // Boton_h
```

Teclado.cpp

```
#include "Teclado.h"
#include "DebugUtils.h"

#define NUM_BOTONES 6

Teclado::Teclado(int right, int left, int up, int down, int ok, int back) :
    boton({}) // Inicializa vector de botones
{
    /*
    BUTTON  VALUE
    right   [r]
    left    [l]
    up      [u]
    down    [d]
    ok      [o]
    back    [b]
    */

    boton[0].setup(right, 'r');
    boton[1].setup(left, 'l');
    boton[2].setup(up, 'u');
    boton[3].setup(down, 'd');
    boton[4].setup(ok, 'o');
    boton[5].setup(back, 'b');

}

Teclado::~Teclado()
{
}

// Verifica con un ciclo que algún botón haya sido presionado
char Teclado::getTecla()
{
    char aux = '\0';
    for (int i = 0; i < NUM_BOTONES; i++)
    {
        // Recorre el vector de botones y obtiene su estado (Presionado/Sin 
presionar)
        aux = boton[i].getBoton();
        if(aux != '\0')
        {
            // Si algún botón es presionado, rompe el ciclo y retorna el valor
            // DEBUG_PRINT("[KB]: ");
            // DEBUG_PRINTLN(aux);
            break;
        }
        aux = '\0';
    }

    return aux;
}
```

Teclado.h

```
#ifndef TECLADO_H_
#define TECLADO_H_

#include "Boton.h"

#define NUM_BOTONES 6

class Teclado
{
private:
    // El teclado contiene 6 botones
    Boton boton[NUM_BOTONES];

public:
    Teclado(int right, int left, int up, int down, int ok, int back);
    ~Teclado();
    char getTecla();
};

#endif // TECLADO_H_
```

Motorpasos.cpp

```
#include "Arduino.h"
#include "MotorPasos.h"
#include "DebugUtils.h"

#define RETARDO_MOTOR (uint8_t)2
#define NUM_PASOS (uint16_t)512

// Constructor
MotorPasos::MotorPasos(uint8_t pDriver0, uint8_t pDriver1, uint8_t pDriver2,
uint8_t pDriver3)
{
  IN1 = pDriver0;
  IN2 = pDriver1;
  IN3 = pDriver2;
  IN4 = pDriver3;

  setup();
}

// Destructor
MotorPasos::~MotorPasos()
{
}

// -------------------------- Implementación --------------------------
void MotorPasos::setup()
{
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}

void MotorPasos::apagar()
{
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
```

```
void MotorPasos::girar(uint8_t dir, int pasos)
{
  for (int i = 0; i < pasos; i++)
  {
    if (dir == 1)
    {
      derecha();
    }
    else
    {
      izquierda();
    }
  }
}

void MotorPasos::vibrar()
{
  for (int i = 0; i < 20; i++)
  {
    for (int j = 0; j < 10; j++)
    {
      derecha();
      derecha();
    }
    for (int j = 0; j < 10; j++)
    {
      izquierda();
      izquierda();
    }
  }
}

void MotorPasos::derecha()
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delay(RETARDO_MOTOR);
}

void MotorPasos::izquierda()
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  delay(RETARDO_MOTOR);

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delay(RETARDO_MOTOR);
}
```

Motorpasos.h

```
#ifndef MotorPasos_h
#define MotorPasos_h

#include <Arduino.h>

class MotorPasos
{
```

```
private:
  // Pines para el driver del motor
  uint8_t IN1;
  uint8_t IN2;
  uint8_t IN3;
  uint8_t IN4;

public:
  MotorPasos(uint8_t pDriver0, uint8_t pDriver1, uint8_t pDriver2, uint8_t
pDriver3);
  ~MotorPasos();

  void setup();
  void apagar();
  void derecha();
  void izquierda();
  void girar(uint8_t dir, int pasos);
  void vibrar();
};
#endif
```

Pastilla.cpp

```
#include "Arduino.h"
#include "Pastilla.h"
#include "DebugUtils.h"

// Constructor
Pastilla::Pastilla()
{
    // nombre = '\0';
    dosis = 0;
    recipiente = 0;
    frecuencia = 0;
    caducidad[0] = 0;
    caducidad[1] = 0;
    caducidad[2] = 0;
    primerDosis[0] = 0;
    primerDosis[1] = 0;
    duracionTratamiento = 0;
    duracionTiempo = 0;
    diario = 0;
    dias[0] = false;
    dias[1] = false;
    dias[2] = false;
    dias[3] = false;
    dias[4] = false;
    dias[5] = false;
    dias[6] = false;
    pastillasRestantes = 0;
}

// Copy constructor
Pastilla::Pastilla(const Pastilla &p)
{
    // nombre = p.nombre;
    dosis = p.dosis;
    recipiente = p.recipiente;
    frecuencia = p.frecuencia;
    memcpy(caducidad, p.caducidad, 3*sizeof(int));
    memcpy(primerDosis, p.primerDosis, 2*sizeof(int));
    duracionTratamiento = p.duracionTratamiento;
    duracionTiempo = p.duracionTiempo;
    diario = p.diario;
    memcpy(dias, p.dias, 7*sizeof(bool));
    pastillasRestantes = 0;
}

// Destructor
Pastilla::~Pastilla()
{
}

void Pastilla::setup(
        // char* nombre,            // Nombre de la pastilla
        int dosis,          // Numero de pastillas a dispensar
        int recipiente,     // Recipiente en la bandeja
        int frecuencia,     // Cada cuánto tiempo se tiene que dispensar
        int caducidad[3],       //dd, mm, aaaa
        int primerDosis[2],     //hh, mm
        int duracionTratamiento,
        int duracionTiempo, // Días/Semanas/Meses/Años
        bool diario,            // Opcion para marcar si es diario o no
        bool dias[7]            // Días a dispensar
    )
{
    // this->nombre = nombre;
    this->dosis = dosis;
    this->recipiente = recipiente;
    this->frecuencia = frecuencia;
    memcpy(this->caducidad, caducidad, 3*sizeof(int));
    memcpy(this->primerDosis, primerDosis, 2*sizeof(int));
    this->duracionTratamiento = duracionTratamiento;
    this->duracionTiempo = duracionTiempo;
    this->diario = diario;
    memcpy(this->dias, dias, 7*sizeof(bool));
    pastillasRestantes = 0;
}

const char* Pastilla::getNombre()
{
    // return nombre;
}

uint8_t Pastilla::getDosis()
{
    return dosis;
}

uint8_t Pastilla::getFrecuencia()
{
    return frecuencia;
}

uint8_t Pastilla::getDuracionTratamiento()
{
    if(duracionTiempo == 1)         // Dias
    {
        return duracionTratamiento;
    }
    else if(duracionTiempo == 2)    // Semanas
    {
        return duracionTratamiento*7;
    }
    else if(duracionTiempo == 3)    // Meses
    {
        return duracionTratamiento*30;
    }
    else
    {
        return 0;
    }
}

int* Pastilla::getPrimerDosis()
{
    return primerDosis;
}

uint8_t Pastilla::getRecipiente()
{
    return recipiente;
}
```

```
bool Pastilla::verificarCantidad()
{
    return pastillasRestantes >= dosis;
}

void Pastilla::agregarPastillas(uint8_t cantidad)
{
    pastillasRestantes += cantidad;
}

void Pastilla::eliminar()
{
    // nombre = '\0';
    dosis = 0;
    recipiente = 0;
    frecuencia = 0;
    caducidad[0] = 0;
    caducidad[1] = 0;
    caducidad[2] = 0;
    primerDosis[0] = 0;
    primerDosis[1] = 0;
    duracionTratamiento = 0;
    duracionTiempo = 0;
    diario = 0;
    dias[0] = false;
    dias[1] = false;
    dias[2] = false;
    dias[3] = false;
    dias[4] = false;
    dias[5] = false;
    dias[6] = false;
    pastillasRestantes = 0;
}

void Pastilla::getPillInfo()
{
    DEBUG_PRINTLN("\n--- Informacion de la pastilla ---")
    // DEBUG_PRINT("Nombre: "); DEBUG_PRINTLN(nombre);
    DEBUG_PRINT("Recipiente: "); DEBUG_PRINTLN(recipiente);
    DEBUG_PRINT("Dosis: "); DEBUG_PRINTLN(dosis);
    DEBUG_PRINT("Frecuencia: "); DEBUG_PRINTLN(frecuencia);
    DEBUG_PRINT("Caducidad (dd/MM/aaaa): "); DEBUG_PRINT(caducidad[0]);
DEBUG_PRINT("/"); DEBUG_PRINT(caducidad[1]); DEBUG_PRINT("/");
DEBUG_PRINTLN(caducidad[2]);
    DEBUG_PRINT("Primera toma (hh:mm): "); DEBUG_PRINT(primerDosis[0]);
DEBUG_PRINT(":"); DEBUG_PRINTLN(primerDosis[1]);
    DEBUG_PRINT("Pastillas restantes: "); DEBUG_PRINTLN(pastillasRestantes);
    DEBUG_PRINT("Duracion del tratamiento: "); DEBUG_PRINT(duracionTiempo);
    if(diario)
    {
        DEBUG_PRINTLN(" dias");
    }
    else
    {
        DEBUG_PRINTLN(" semanas");
        for (int i = 0; i < 7; i++)
        {
            DEBUG_PRINT("Dia "); DEBUG_PRINT(i+1);

            if (dias[i])
            {
                DEBUG_PRINTLN(" Sí")
            }
            else
            {
                DEBUG_PRINTLN(" No")
            }
        }
    }
}
```

Pastilla.h

```
#ifndef Pastilla_h
#define Pastilla_h

#include <Arduino.h>

class Pastilla
{
private:
    const char* nombre; // Nombre de la pastilla
    uint8_t dosis;      // Numero de pastillas a dispensar
    uint8_t recipiente; // Recipiente en la bandeja
    uint8_t frecuencia; // Cada cuánto tiempo se tiene que dispensar
    int caducidad[3];   //dd, mm, aaaa
    int primerDosis[2]; //hh, mm
    int duracionTratamiento;
    uint8_t duracionTiempo; // 1=dias, 2=semanas, 3=meses, 4=años
    bool diario;        // Opcion para marcar si es diario o no
    bool dias[7];       // Días a dispensar

    // Tamaño total: 18 Bytes + nombre (variable)

public:
    uint8_t pastillasRestantes;
    Pastilla();
    Pastilla(const Pastilla &p);
    ~Pastilla();

    void setup(
        // char* nombre,            // Nombre de la pastilla
        int dosis,                  // Numero de pastillas a dispensar
        int recipiente,             // Recipiente en la bandeja
        int frecuencia,             // Cada cuánto tiempo se tiene que dispensar
(horas)
        int caducidad[3],           //dd, mm, aaaa
        int primerDosis[2],         //hh, mm
        int duracionTratamiento,
        int duracionTiempo,         // 1=dias, 2=semanas, 3=meses, 4=años
        bool diario,                // Opcion para marcar si es diario o no
        bool dias[7]                // Días a dispensar
    );

    const char* getNombre();
    uint8_t getDosis();
    uint8_t getRecipiente();
    uint8_t getFrecuencia();
    uint8_t getDuracionTratamiento();
    int* getPrimerDosis();

    void eliminar();
    bool verificarCantidad();
    void agregarPastillas(uint8_t cantidad);

    void getPillInfo();
};

#endif
```

HC05.cpp

```
#include "HC05.h"

// Name:    GioHC05
// Pin:     1234
// Role:    0 (Slave)
// UART:    9600,1,2

HC05::HC05()
{
    ch = '\0';
    commandStr = "";
    Serial1.begin(9600);
}

HC05::~HC05()
{
}

char* HC05::getCommand()
{
    if(Serial1.available())
    {
        ch = Serial1.read();
        commandStr += ch;

        if(ch == '*') // '*' marks the end of the string
        {
            int tempSize = commandStr.length() + 1;
            int allocSize = tempSize*sizeof(char);
            char* temp = (char*) malloc(allocSize); // Allocate memory and
cast it to char*
            strcpy(temp, commandStr.c_str());
            commandStr = "";   // Clear string so it doesn't accumulate
characters
            ch = '\0';

            return temp;
        }
    }
    return "";
}
```

HC05.h

```
#ifndef HC05_H_
#define HC05_H_

#include <Arduino.h>
#include "DebugUtils.h"

class HC05
{
private:
    char ch;
    String commandStr;

public:
    HC05();
    ~HC05();

    char* getCommand();
};

#endif // HC05_H_
```

Pantalla.cpp

```
/**
 * @file Pantalla.cpp
 * @author your name (you@domain.com)
 * @brief Archivo relacionado con las funciones para dibujar las páginas en la
pantalla LCD
 * @version 0.5
 * @date 2022-05-08
 *
 * @copyright Copyright (c) 2022
 *
 */

#include "Pantalla.h"
#include "DebugUtils.h"

// Constructor
Pantalla::Pantalla(int pinClk, int pinData, int pinCS, int pinReset) :
    u8g2(U8G2_R2, pinClk, pinData, pinCS, pinReset)
{
    this->pinClk = pinClk;
    this->pinData = pinData;
    this->pinCS = pinCS;
    this->pinReset = pinReset;

    botonPantalla = '\0';
    pageIndex = 1;

    horaAux  = false;
    fechaAux = false;


    resetVars();
}

// Destructor
Pantalla::~Pantalla()
{
}

// -------------------------- Implementación --------------------------
```

```
void Pantalla::setup()
{
    // pinMode(pinReset, OUTPUT);
    u8g2.begin();
    // digitalWrite(pinReset, LOW);
    // delay(100);
    // digitalWrite(pinReset, HIGH);

    // +-------------------------------------------------------------------------
------+
    // | Font: <u8g2> _ <font_5x8> _ <m>
<f>                                         |
    // +-------------------------------------------------------------------------
------+
    // | <prefix> '_' <name> '_' <purpose> <char
set>                                |
    // +-------------------------------------------------------------------------
------+
    // | <name>     font_5x8  - 6 pixel height
font                                      |
    // | <purpose>  m         - All glyphs have common height and width
(monospace) |
    // | <char set> f         - The font includes up to 256
glyphs                       |
    // +-------------------------------------------------------------------------
------+

    u8g2.setFont(u8g2_font_5x8_mr);
    u8g2.setFontMode(/* transparent = */ true);
}

/****************************************************************************
*****************/
// UTILIDADES
/****************************************************************************
*****************/
void Pantalla::printHora()
{
    u8g2.setCursor(2, 19);
    u8g2.print(u8x8_u8toa(hh, 2));
    u8g2.print(":");
    u8g2.print(u8x8_u8toa(mm, 2));
    u8g2.print(":");
    u8g2.print(u8x8_u8toa(ss, 2));
}

void Pantalla::printFecha()
{
    u8g2.setCursor(75, 19);
    u8g2.print(u8x8_u8toa(dd, 2));
    u8g2.print("/");
    u8g2.print(u8x8_u8toa(MM, 2));
    u8g2.print("/");
    u8g2.print(u8x8_u16toa(aaaa, 4));
}

// Imprime el titulo y botones de navegacion
void Pantalla::setTitle(const char* title, int botones)
{
    u8g2.drawFrame(0, 0, screenWidth, screenHeight);
    u8g2.drawBox(0,0,screenWidth, 11);
    u8g2.setDrawColor(2);
    u8g2.setCursor(3,9);
    u8g2.print(title);
    navButtons(botones);
    u8g2.setDrawColor(1);
}

// Imprime un subtitulo
void Pantalla::setSubtitle(const char* subtitle)
{
    u8g2.drawStr(3, 19, subtitle);
}

void Pantalla::backButton()
{
    u8g2.drawButtonUTF8(81, 8, U8G2_BTN_HCENTER|U8G2_BTN_BW1, 24, 2, 0,
"Back");
}

void Pantalla::okButton()
{
    u8g2.drawButtonUTF8(112, 8, U8G2_BTN_HCENTER|U8G2_BTN_BW1, 24, 2, 0,
"Ok");
}

void Pantalla::navButtons(int opcion)
{
    switch (opcion)
    {
    case 1:
        backButton();
        break;

    case 2:
        okButton();
        break;

    case 3:
        backButton();
        okButton();
        break;

    default:
        break;
    }
}

void Pantalla::variableFlechas(int x, int y, char direccion)
{
    int h = 10;
    int w = 36;

    u8g2.drawFrame(x, y, w, h-1);
    u8g2.drawVLine(x+(w/2), y, h-1);

    u8g2.setCursor(x+3, y+(h-2)-1);
    u8g2.print("-    +");

    if (direccion == 'h')
    {
        int yoffset  = y+2;
        int hoffset  = x+12;
        int hSpacing = hoffset+10;

        // Triangulo izquierda
        u8g2.drawVLine(hoffset+2, yoffset+0, 5);
        u8g2.drawVLine(hoffset+1, yoffset+1, 3);
        u8g2.drawVLine(hoffset+0, yoffset+2, 1);

        // Triangulo derecha
        u8g2.drawVLine(hSpacing+0, yoffset+0, 5);
        u8g2.drawVLine(hSpacing+1, yoffset+1, 3);
        u8g2.drawVLine(hSpacing+2, yoffset+2, 1);
    }
    else
    {
        int yoffset  = y+3;
        int hoffset  = x+11;
        int hSpacing = hoffset+10;

        // Triangulo arriba
        u8g2.drawHLine(hoffset+0, yoffset+0, 5);
        u8g2.drawHLine(hoffset+1, yoffset+1, 3);
        u8g2.drawHLine(hoffset+2, yoffset+2, 1);

        // Triangulo abajo
        u8g2.drawHLine(hSpacing+0, yoffset+2, 5);
        u8g2.drawHLine(hSpacing+1, yoffset+1, 3);
        u8g2.drawHLine(hSpacing+2, yoffset+0, 1);
    }
}

void Pantalla::mostrarVariable(int x, int y, const char nombre[5], int var,
char direccion)
{
    u8g2.setCursor(x, y);
    u8g2.print(nombre);
    u8g2.print(" = ");
    u8g2.print(u8x8_u8toa(var, 2));

    variableFlechas(x+47, y-8, direccion);
}

void Pantalla::mostrarVariable(int x, int y, const char nombre[5], char
direccion)
{
    u8g2.setCursor(x, y);
    u8g2.print(nombre);

    variableFlechas(x+47, y-8, direccion);
}

/*
    Opcion 1 = Tecla izquierda
    Opcion 2 = Tecla derecha
    Opcion 3 = Tecla abajo
    Opcion 4 = Tecla arriba
*/
void Pantalla::opcion(const char* opcionNombre, int numOpcion)
{
    int xoff = 3;
    int yoff = (numOpcion*10)+20;

    u8g2.setCursor(10, yoff);
    u8g2.print(opcionNombre);

    int xoff2 = xoff+1;
    int yoff2 = yoff-5;

    switch (numOpcion)
    {
```

```
    case 1:
        // Triangulo izquierda
        u8g2.drawVLine(xoff2+2, yoff2+0, 5);
        u8g2.drawVLine(xoff2+1, yoff2+1, 3);
        u8g2.drawVLine(xoff2+0, yoff2+2, 1);
        break;

    case 2:
        // Triangulo derecha
        u8g2.drawVLine(xoff2+0, yoff2+0, 5);
        u8g2.drawVLine(xoff2+1, yoff2+1, 3);
        u8g2.drawVLine(xoff2+2, yoff2+2, 1);
        break;

    case 3:
        // Triangulo arriba
        u8g2.drawHLine(xoff+0, yoff2+0, 5);
        u8g2.drawHLine(xoff+1, yoff2+1, 3);
        u8g2.drawHLine(xoff+2, yoff2+2, 1);
        break;

    case 4:
        // Triangulo abajo
        u8g2.drawHLine(xoff+0, yoff2+2, 5);
        u8g2.drawHLine(xoff+1, yoff2+1, 3);
        u8g2.drawHLine(xoff+2, yoff2+0, 1);
        break;

    default:
        break;
    }

}

void Pantalla::resetVars()
{
    dosis = 1;
    recipiente = 0;
    frec = 1;
    // cad = {1, 1, 2022};
    cad[0] = 1;
    cad[1] = 1;
    cad[2] = 2022;
    // primToma = {0, 0};
    primToma[0] = 0;
    primToma[1] = 0;
    durTrat = 1;
    durTiempo = 0;
    selectorDia = 1;
    // diasSemana = {false};
    diasSemana[0] = false;
    diasSemana[1] = false;
    diasSemana[2] = false;
    diasSemana[3] = false;
    diasSemana[4] = false;
    diasSemana[5] = false;
    diasSemana[6] = false;
    numSemanas = 0;
    numDias = 0;
    diario = false;

    cantidadAAgregar = 1;

    commandStr = "";
}

/*****************************************************************************
*****************/
// PAGINAS A MOSTRAR
/*****************************************************************************
*****************/
void Pantalla::p1menu()
{
    u8g2.firstPage();
    do
    {
        setTitle("MENU", NO_BTN);

        printFecha();
        printHora();

        opcion("Configurar", 1);
        opcion("Dispensar", 2);
    } while (u8g2.nextPage());

    menuBackend();
}

void Pantalla::menuBackend()
{
    resetVars();
    if(botonPantalla == 'l'){pageIndex++;}
    if(botonPantalla == 'r'){pageIndex = 10;}
}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p2selecPastilla()
{
    u8g2.firstPage();
    do
    {
        setTitle("PASTILLA", BACK_BTN);
        setSubtitle("Seleccionar pastilla");

        opcion("1", 1);
        opcion("2", 2);
        opcion("3", 3);
        opcion("4", 4);
    } while (u8g2.nextPage());

    selecPastillaBackend();
}

void Pantalla::selecPastillaBackend()
{
    if(botonPantalla == 'l'){recipiente = 1; pageIndex = 11;}
    if(botonPantalla == 'r'){recipiente = 2; pageIndex = 11;}
    if(botonPantalla == 'd'){recipiente = 3; pageIndex = 11;}
    if(botonPantalla == 'u'){recipiente = 4; pageIndex = 11;}

    if(botonPantalla == 'b'){pageIndex--;}
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p3frecuencia()
{
    u8g2.firstPage();
    do
    {
        setTitle("FRECUENCIA", BACK_BTN);
        setSubtitle("Seleccionar");

        opcion("Diaria", 1);
        opcion("Dias x semana", 2);
    } while (u8g2.nextPage());

    frecuenciaBackend();
}

void Pantalla::frecuenciaBackend()
{
    if(botonPantalla == 'l')
    {
        diario = true;
        durTiempo = 1;
        pageIndex = pageIndex + 2;
    }

    if(botonPantalla == 'r')
    {
        diario = false;
        durTiempo = 2;
        pageIndex++;
    }

    if(botonPantalla == 'b'){pageIndex--;}
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p4dias()
{
    u8g2.firstPage();
    do
    {
        int xoff = 60;

        setTitle("DIAS", BACK_OK_BTN);

        opcion("Borr" , 1);
        opcion("Selec", 2);
        opcion("Bajar", 3);
        opcion("Subir", 4);

        u8g2.drawStr(xoff, 16, "Lunes");
        u8g2.drawStr(xoff, 24, "Martes");
        u8g2.drawStr(xoff, 32, "Miercoles");
        u8g2.drawStr(xoff, 40, "Jueves");
        u8g2.drawStr(xoff, 48, "Viernes");
        u8g2.drawStr(xoff, 56, "Sabado");
        u8g2.drawStr(xoff, 64, "Domingo");

        // SELECTOR
        u8g2.setDrawColor(2);
        u8g2.drawBox(xoff-1, (selectorDia*8)+1, 46, 8);

        for (int i = 0; i < 7; i++)
        {
            if(diasSemana[i] == true)
```

```
                u8g2.drawStr(xoff-8, (i+2)*8, "*");
        }
    } while (u8g2.nextPage());

    diasBackend();
}

void Pantalla::diasBackend()
{
    if(botonPantalla == 'u' && selectorDia > 1){selectorDia--;}
    if(botonPantalla == 'd' && selectorDia < 7){selectorDia++;}

    if(botonPantalla == 'r'){diasSemana[selectorDia-1] = true;}
    if(botonPantalla == 'l'){diasSemana[selectorDia-1] = false;}

    if(botonPantalla == 'b'){pageIndex--;}
    if(botonPantalla == 'o'){pageIndex++;}
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p5duracionTratamiento()
{
    u8g2.firstPage();
    do
    {
        setTitle("DURACION", BACK_OK_BTN);
        setSubtitle("Duracion y frec (hrs)");

        if(diario)
        {
            mostrarVariable(3, 45, "Dias", durTrat, 'h');
        }
        else
        {
            mostrarVariable(3, 45, "Sem ", durTrat, 'h');
        }

        mostrarVariable(3, 55, "Frec", frec, 'v');
    } while (u8g2.nextPage());

    duracionTratamientoBackend();
}

void Pantalla::duracionTratamientoBackend()
{
    if (botonPantalla == 'r' && durTrat < 99){durTrat++;}
    if (botonPantalla == 'l' && durTrat >  1){durTrat--;}
    if (botonPantalla == 'u' && frec < 24){frec++;}
    if (botonPantalla == 'd' && frec >  1){frec--;}

    if(botonPantalla == 'b'){pageIndex = pageIndex - 2;}
    if(botonPantalla == 'o'){pageIndex++;}
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p6primerToma()
{
    u8g2.firstPage();
    do
    {
        setTitle("PRIMER TOMA", BACK_OK_BTN);
        setSubtitle("Seleccionar");

        u8g2.setCursor(36, 30);
        u8g2.print("Hora ");
        u8g2.print(u8x8_u8toa(primToma[0], 2));
        u8g2.print(":");
        u8g2.print(u8x8_u8toa(primToma[1], 2));
        mostrarVariable(3, 45, "Hora", 'h');
        mostrarVariable(3, 55, "Min ", 'v');
    } while (u8g2.nextPage());

    primerTomaBackend();
}

void Pantalla::primerTomaBackend()
{
    if(horaAux == false)
    {
        // Initialize time for user's convenience only once when entering the
screen
        // (Perform only once to allow modifying the time)
        primToma[0] = hh;
        primToma[1] = mm;

        horaAux = true;
    }

    if (botonPantalla == 'l' && primToma[0] >  0){primToma[0]--;}
    if (botonPantalla == 'r' && primToma[0] < 23){primToma[0]++;}

    if (botonPantalla == 'd' && primToma[1] >  4){primToma[1] = primToma[1] -
5;}
```

```
    if (botonPantalla == 'u' && primToma[1] < 55){primToma[1] = primToma[1] +
5;}

    if(botonPantalla == 'b'){pageIndex--; horaAux = false;}
    if(botonPantalla == 'o'){pageIndex++; horaAux = false;}
}


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p7dosis()
{
    u8g2.firstPage();
    do
    {
        setTitle("DOSIS", BACK_OK_BTN);
        setSubtitle("Num de past a dispensar");
        mostrarVariable(3, 45, "Cant", dosis, 'h');
    } while (u8g2.nextPage());

    dosisBackend();
}

void Pantalla::dosisBackend()
{
    if (botonPantalla == 'l' && dosis >  1){dosis--;}
    if (botonPantalla == 'r' && dosis <  4){dosis++;}

    if(botonPantalla == 'b'){pageIndex--;}
    if(botonPantalla == 'o'){pageIndex++;}
}


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++
void Pantalla::p8caducidad()
{
    u8g2.firstPage();
    do
    {
        setTitle("CADUCIDAD", BACK_OK_BTN);
        setSubtitle("Seleccionar");

        u8g2.setCursor(34, 30);
        u8g2.print("Fecha ");
        u8g2.print(u8x8_u8toa(cad[1], 2));
        u8g2.print("/");
        u8g2.print(u8x8_u16toa(cad[2], 4));
        mostrarVariable(3, 45, "Mes ", 'h');
        mostrarVariable(3, 55, "Ano ", 'v');
        u8g2.drawHLine(8, 49, 3); // Para la Ñ
    } while (u8g2.nextPage());

    caducidadBackend();
}

void Pantalla::caducidadBackend()
{
    if(fechaAux == false)
    {
        // Initialize date for user's convenience only once when entering the
screen
        // (Perform only once to allow modifying the time)
        cad[1] = MM;
        cad[2] = aaaa;

        fechaAux = true;
    }

    if (botonPantalla == 'l' && cad[1] >   1){cad[1]--;}
    if (botonPantalla == 'r' && cad[1] <  12){cad[1]++;}

    if (botonPantalla == 'd' && cad[2] > 2022){cad[2]--;}
    if (botonPantalla == 'u' && cad[2] < 2099){cad[2]++;}

    if(botonPantalla == 'b'){pageIndex--; fechaAux = false;}
    if(botonPantalla == 'o'){pageIndex++; fechaAux = false;}
}
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p9resultado()
{
    u8g2.firstPage();
    do
    {
        setTitle("RESUMEN", BACK_OK_BTN);

        int vOffsetpx = 12;
        int vspacing  = 8;

        u8g2.setCursor(3, vOffsetpx+vspacing*1);
        u8g2.print("Pastilla #");
        u8g2.print(recipiente);

        u8g2.setCursor(3, vOffsetpx+vspacing*2);
        u8g2.print("Past a dispensar: ");
        u8g2.print(dosis);

        u8g2.setCursor(3, vOffsetpx+vspacing*3);
        u8g2.print("Cada: ");
        u8g2.print(frec);
        u8g2.print(" horas");

        u8g2.setCursor(3, vOffsetpx+vspacing*4);
        u8g2.print("Duracion: ");
        u8g2.print(durTrat);
        if(durTiempo == 1)
        {
            u8g2.print(" dias");
        }
        if(durTiempo == 2){
            u8g2.print(" sem");
        }

        u8g2.setCursor(3, vOffsetpx+vspacing*5);
        u8g2.print("Caducidad: ");
        u8g2.print(u8x8_u8toa(cad[1], 2));
        u8g2.print("/");
        u8g2.print(u8x8_u16toa(cad[2], 4));

    } while (u8g2.nextPage());

    resultadoBackend();
}

void Pantalla::resultadoBackend()
{
    if(botonPantalla == 'b'){pageIndex--;}
    if(botonPantalla == 'o')
    {
        pageIndex = 12; // Saltar a pantalla para agregar pastillas
        setPastilla();

        String temp = "PST ";
        commandStr = temp + recipiente;
    }
}
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p10dispensar()
{
    u8g2.firstPage();
    do
    {
        setTitle("DISPENSAR", BACK_BTN);
        setSubtitle("Seleccionar");

        opcion("Past 1", 1);
        opcion("Past 2", 2);
        opcion("Past 3", 3);
        opcion("Past 4", 4);

        u8g2.setCursor(50, 30);
        u8g2.print("(");
        u8g2.print(remainingPills[0]);
        u8g2.print(" restantes)");

        u8g2.setCursor(50, 40);
        u8g2.print("(");
        u8g2.print(remainingPills[1]);
        u8g2.print(" restantes)");

        u8g2.setCursor(50, 50);
        u8g2.print("(");
        u8g2.print(remainingPills[2]);
        u8g2.print(" restantes)");

        u8g2.setCursor(50, 60);
        u8g2.print("(");
        u8g2.print(remainingPills[3]);
        u8g2.print(" restantes)");

    } while (u8g2.nextPage());

    dispensarBackend();
}

void Pantalla::dispensarBackend()
{
    if (botonPantalla == 'l'){commandStr = "DIS 1";}
    if (botonPantalla == 'r'){commandStr = "DIS 2";}

    if (botonPantalla == 'd'){commandStr = "DIS 3";}
    if (botonPantalla == 'u'){commandStr = "DIS 4";}

    if(botonPantalla == 'b'){pageIndex = 1; /*commandStr = "";*/}
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p11opciones()
{
    u8g2.firstPage();
    do
    {
        setTitle("OPCIONES", BACK_BTN);
        setSubtitle("Seleccionar");

        opcion("Configurar", 1);
        opcion("Agregar", 2);
        opcion("Eliminar", 3);
    } while (u8g2.nextPage());

    opcionesBackend();
}

void Pantalla::opcionesBackend()
{
    if (botonPantalla == 'l'){pageIndex = 3;}   // Configurar pastillas
    if (botonPantalla == 'r'){pageIndex = 12;}  // Agregar pastillas
    if (botonPantalla == 'd'){pageIndex = 13;}// Eliminar pastilla

    if(botonPantalla == 'b'){pageIndex = 2;}
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p12agregar()
{
    u8g2.firstPage();
    do
    {
        setTitle("AGREGAR", BACK_OK_BTN);
        setSubtitle("Seleccionar cantidad");

        mostrarVariable(3, 45, "Cant", cantidadAAgregar, 'h');
    } while (u8g2.nextPage());

    agregarBackend();
}

void Pantalla::agregarBackend()
{
    if (botonPantalla == 'r' && cantidadAAgregar < 50){cantidadAAgregar++;}
    if (botonPantalla == 'l' && cantidadAAgregar >  1){cantidadAAgregar--;}

    if(botonPantalla == 'o')
    {
        String temp = "ADD ";
        commandStr = temp + recipiente + " " + cantidadAAgregar;
        pageIndex = 1;
    }

    if(botonPantalla == 'b'){pageIndex = 11;}
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::p13eliminar()
{
    u8g2.firstPage();
    do
    {
        setTitle("ELIMINAR", NO_BTN);
        setSubtitle("Seguro?");

        opcion("Si", 1);
        opcion("No", 2);
    } while (u8g2.nextPage());

    eliminarBackend();
}

void Pantalla::eliminarBackend()
{
    if (botonPantalla == 'l')
    {
        String temp = "DEL ";
        commandStr = temp + recipiente;
        pageIndex = 1;
    }
    if(botonPantalla == 'r'){pageIndex = 11;}

    if(botonPantalla == 'b'){pageIndex = 1;}
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++
void Pantalla::loop()
{
    switch (pageIndex)
    {
    case 1:
        p1menu();
        break;

    case 2:
        p2selecPastilla();
        break;

    case 3:
        p3frecuencia();
        break;

    case 4:
        p4dias();
        break;

    case 5:
        p5duracionTratamiento();
        break;

    case 6:
        p6primerToma();
        break;

    case 7:
        p7dosis();
        break;

    case 8:
        p8caducidad();
        break;

    case 9:
        p9resultado();
        break;

    case 10:
        p10dispensar();
        break;

    case 11:
        p11opciones();
        break;

    case 12:
        p12agregar();
        break;

    case 13:
        p13eliminar();
        break;

    default:
        break;
    }
}

void Pantalla::setPastilla()
{
    pastillaAux.setup(
        /*" ",*/
        dosis,
        recipiente,
        frec,
        cad,
        primToma,
        durTrat,
        durTiempo,
        diario,
        diasSemana
    );
}

Pastilla Pantalla::getPastilla()
{
    return pastillaAux;
}

char* Pantalla::getCommand()
{
    if(commandStr != "")
    {
        int tempSize = commandStr.length() + 1;
        int allocSize = tempSize*sizeof(char);
        char* temp = (char*) malloc(allocSize); // Allocate memory and cast it
to char*
        strcpy(temp, commandStr.c_str());
        commandStr = "";   // Clear string so it doesn't accumulate characters

        return temp;
    }

    return "";
}

void Pantalla::setTime(int hh, int mm, int ss)
{
    this->hh = hh;
    this->mm = mm;
    this->ss = ss;
}
```

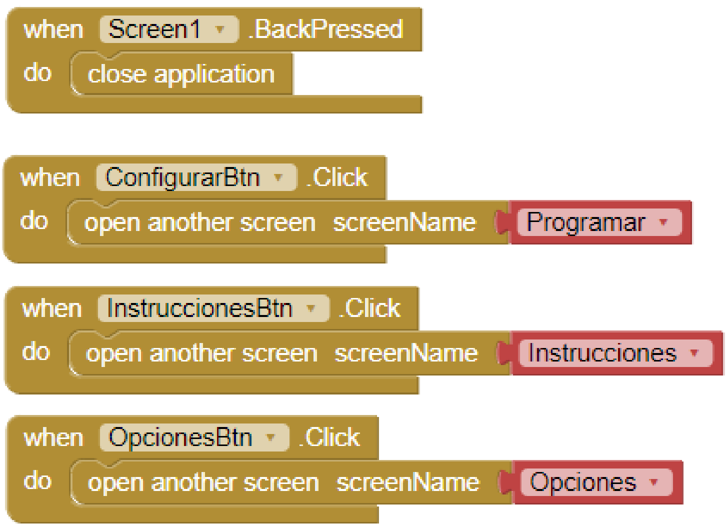
```
void Pantalla::setDate(int dd, int MM, int aaaa)
{
    this->dd = dd;
    this->MM = MM;
    this->aaaa = aaaa;
}

void Pantalla::setRemainingPills(int p1, int p2, int p3, int p4)
{
    remainingPills[0] = p1;
    remainingPills[1] = p2;
    remainingPills[2] = p3;
    remainingPills[3] = p4;
}
```

Pantalla.h

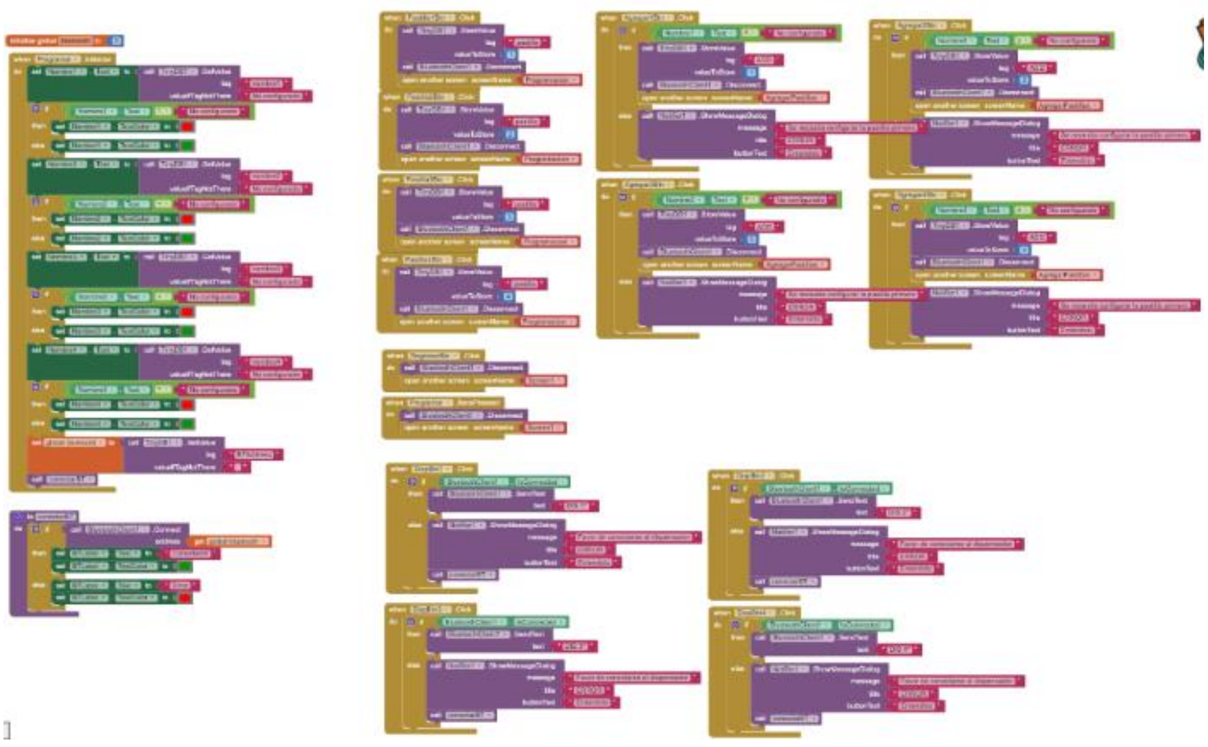
```
#ifndef Pantalla_H_
#define Pantalla_H_

#include <Arduino.h>
#include "U8g2lib.h"
#include "Pastilla.h"

#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

#define NO_BTN 0
#define BACK_BTN 1
#define OK_BTN 2
#define BACK_OK_BTN 3

class Pantalla
{
private:
// Variables
  // +-----------------------------------------------------+
  // | Constructor name                                    |
  // +-----------------------------------------------------+
  // | 1  Prefix                U8G2                       |
  // | 2  Display Controller    ST7920                     |
  // | 3  Display Name        128X64                       |
  // | 4  Buffer Size           1, 2 or F (full frame buffer)|
  // | 5  Communication       SW_SPI                       |
  // +----------------------------------------------------------+
  U8G2_ST7920_128X64_1_SW_SPI u8g2;

  int pinClk;
  int pinData;
  int pinCS;
  int pinReset;

  uint8_t pageIndex;
  String commandStr;

// Constantes
  const uint8_t screenWidth = 128;
  const uint8_t screenHeight = 64;

// Variables para la pastilla
  Pastilla pastillaAux;
  uint8_t dosis;
  uint8_t recipiente;
  uint8_t frec;
  int cad[3];
  int primToma[2];
  uint8_t durTrat;
  uint8_t durTiempo;
  uint8_t selectorDia;
  bool diasSemana[7];
  uint8_t numSemanas;
  uint8_t numDias;
  bool diario;

  uint8_t cantidadAAgregar;

  uint8_t remainingPills[4];

// Variables para el reloj
  int hh, mm, ss;
  int dd, MM, aaaa;
  bool horaAux, fechaAux;


// Funciones relacionadas a la impresión de pantallas
  void setTitle(const char* title, int botones);
  void setSubtitle(const char* subtitle);
  void printHora();
  void printFecha();
  void backButton();
  void okButton();
  void navButtons(int opcion);
  void opcion(const char* opcionNombre, int numOpcion);
  void variableFlechas(int x, int y, char direccion);
  void mostrarVariable(int x, int y, const char nombre[5], int var, char
direccion);
  void mostrarVariable(int x, int y, const char nombre[5], char direccion);

// Funciones del backend de cada pantalla
  void menuBackend();
  void selecPastillaBackend();
  void frecuenciaBackend();
  void diasBackend();
  void semanasBackend();
  void duracionTratamientoBackend();
  void primerTomaBackend();
  void dosisBackend();
  void caducidadBackend();
  void resultadoBackend();
  void dispensarBackend();
  void opcionesBackend();
  void agregarBackend();
  void eliminarBackend();

public:
  char botonPantalla;

  Pantalla(int pinClk, int pinData, int pinCS, int pinReset);
  ~Pantalla();

  void setup();
  void loop();

// Páginas de la pantalla a mostrar
  void p1menu();
  void p2selecPastilla();
  void p3frecuencia();
  void p4dias();
  void p5duracionTratamiento();
  void p6primerToma();
  void p7dosis();
  void p8caducidad();
  void p9resultado();
  void p10dispensar();
  void p11opciones();
  void p12agregar();
  void p13eliminar();

// Funciones para el control con la pantalla
  void resetVars();
  void setPastilla();
  Pastilla getPastilla();
  char* getCommand();

  void setTime(int hh, int mm, int ss);
  void setDate(int dd, int MM, int aaaa);

  void setRemainingPills(int p1, int p2, int p3, int p4);
};

#endif // Pantalla_H_
```

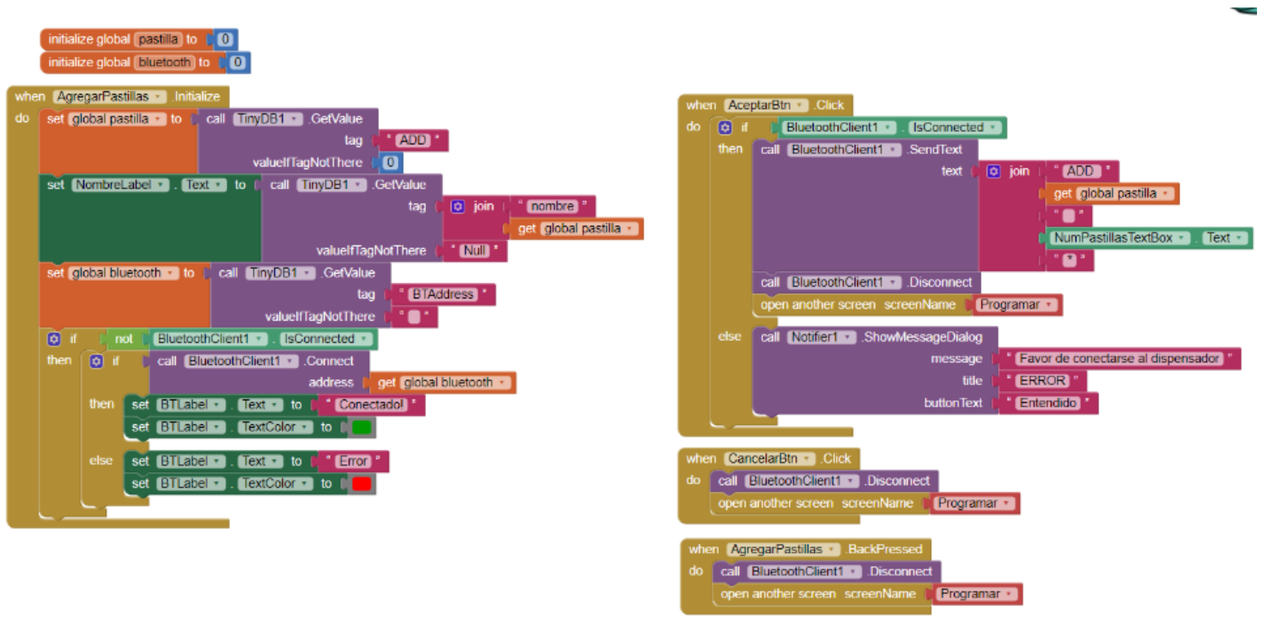
16.2 B – Application code

Below are the MIT App Inventor 2 code blocks for programming Android app.

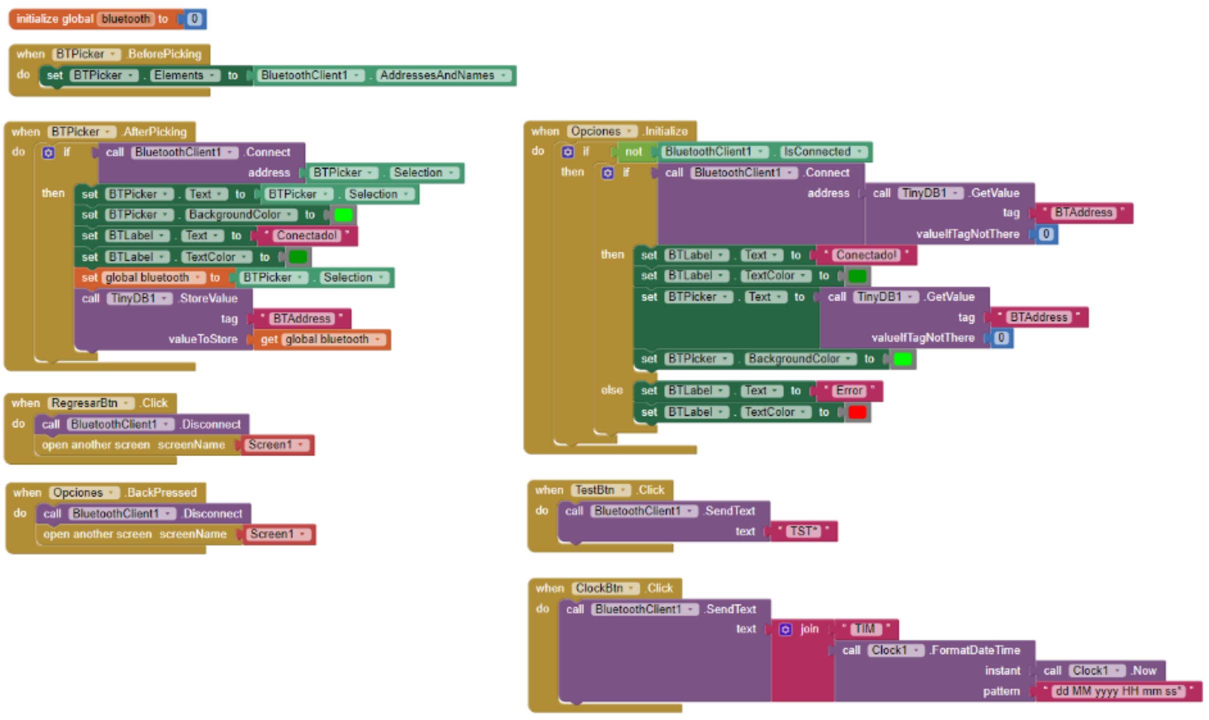


```
when Screen1 .BackPressed
do   close application

when ConfigurarBtn .Click
do   open another screen  screenName  Programar

when InstruccionesBtn .Click
do   open another screen  screenName  Instrucciones

when OpcionesBtn .Click
do   open another screen  screenName  Opciones
```





```
initialize global pastilla to 0
initialize global bluetooth to 0

when AgregarPastillas .Initialize
do  set global pastilla to  call TinyDB1 .GetValue
                                tag "ADD"
                                valueIfTagNotThere 0
    set NombreLabel . Text to  call TinyDB1 .GetValue
                                   tag  join "nombre"
                                             get global pastilla
                                   valueIfTagNotThere "Null"
    set global bluetooth to  call TinyDB1 .GetValue
                                 tag "BTAddress"
                                 valueIfTagNotThere " "
    if  not  BluetoothClient1 . IsConnected
    then  if  call BluetoothClient1 .Connect
                   address  get global bluetooth
          then  set BTLabel . Text to "Conectado!"
                set BTLabel . TextColor to 🟩
          else  set BTLabel . Text to "Error"
                set BTLabel . TextColor to 🟥

when AceptarBtn .Click
do  if  BluetoothClient1 . IsConnected
    then  call BluetoothClient1 .SendText
              text  join "ADD"
                         get global pastilla
                         " "
                         NumPastillasTextBox . Text
                         " "
          call BluetoothClient1 .Disconnect
          open another screen  screenName "Programar"
    else  call Notifier1 .ShowMessageDialog
              message "Favor de conectarse al dispensador"
              title "ERROR"
              buttonText "Entendido"

when CancelarBtn .Click
do  call BluetoothClient1 .Disconnect
    open another screen  screenName "Programar"

when AgregarPastillas .BackPressed
do  call BluetoothClient1 .Disconnect
    open another screen  screenName "Programar"
```



```
initialize global bluetooth to 0

when BTPicker .BeforePicking
do  set BTPicker . Elements to BluetoothClient1 . AddressesAndNames

when BTPicker .AfterPicking
do  if   call BluetoothClient1 .Connect
              address BTPicker . Selection
    then set BTPicker . Text to BTPicker . Selection
         set BTPicker . BackgroundColor to 🟩
         set BTLabel . Text to " Conectado! "
         set BTLabel . TextColor to 🟩
         set global bluetooth to BTPicker . Selection
         call TinyDB1 .StoreValue
                     tag " BTAddress "
                     valueToStore get global bluetooth

when RegresarBtn .Click
do  call BluetoothClient1 .Disconnect
    open another screen  screenName Screen1

when Opciones .BackPressed
do  call BluetoothClient1 .Disconnect
    open another screen  screenName Screen1

when Opciones .Initialize
do  if   not BluetoothClient1 . IsConnected
    then if   call BluetoothClient1 .Connect
                   address call TinyDB1 .GetValue
                                       tag " BTAddress "
                                       valueIfTagNotThere 0
         then set BTLabel . Text to " Conectado! "
              set BTLabel . TextColor to 🟩
              set BTPicker . Text to call TinyDB1 .GetValue
                                                 tag " BTAddress "
                                                 valueIfTagNotThere 0
              set BTPicker . BackgroundColor to 🟩
         else set BTLabel . Text to " Error "
              set BTLabel . TextColor to 🟥

when TestBtn .Click
do  call BluetoothClient1 .SendText
                         text " TST* "

when ClockBtn .Click
do  call BluetoothClient1 .SendText
                         text join " TIM "
                                   call Clock1 .FormatDateTime
                                                   instant call Clock1 .Now
                                                   pattern " dd MM yyyy HH mm ss* "
```