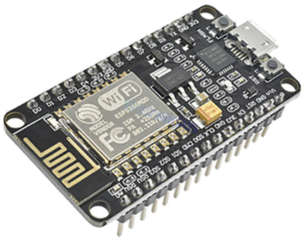
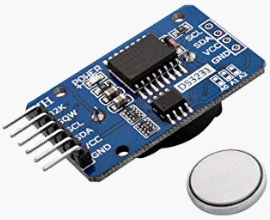
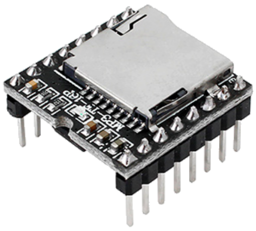
Rond-on-Air

An environmental project for makers
made by Fabrizio Calderan
on January 2020 — February 2021

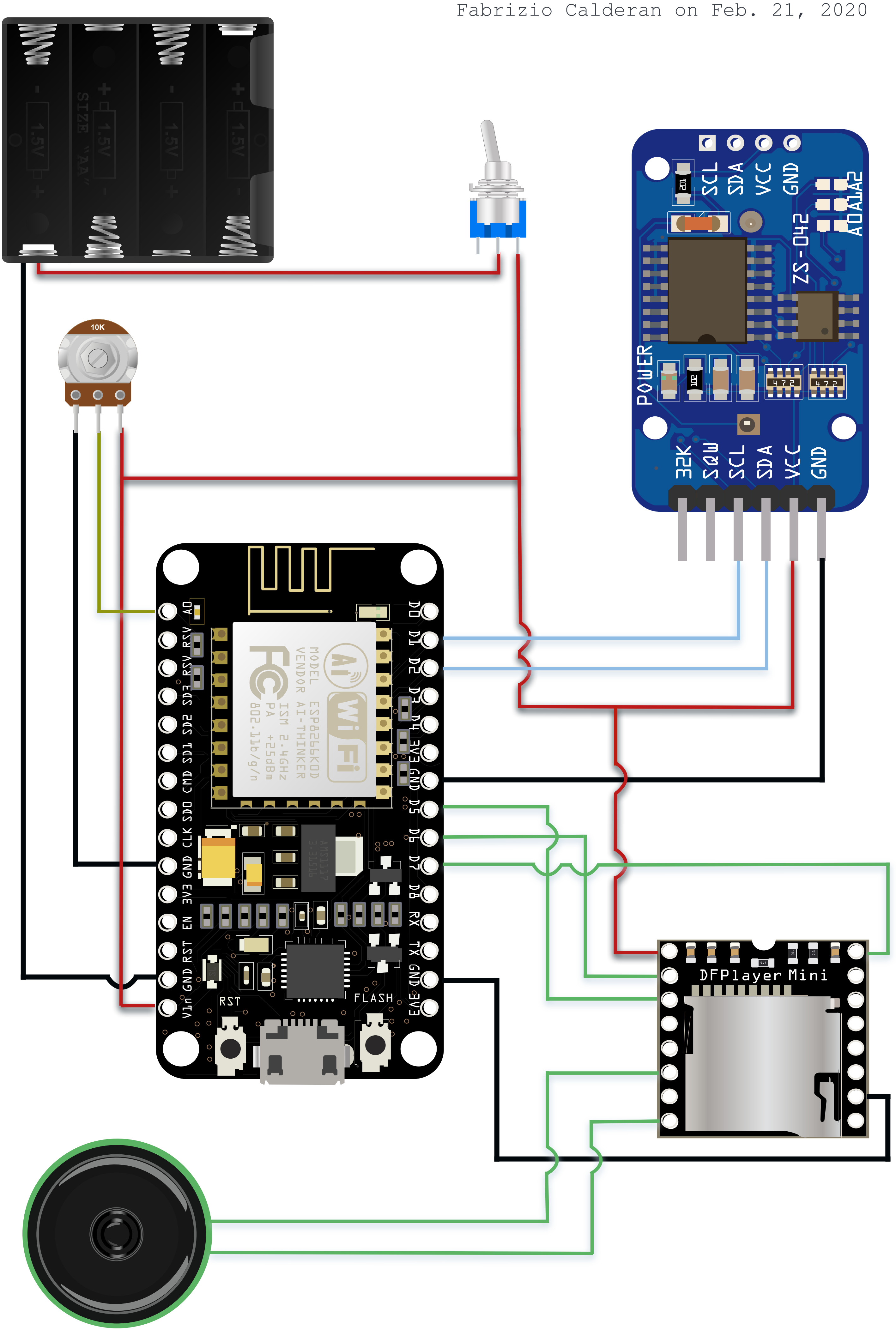
Components

Components needed for a single caller. The cost for a single component is referred to batches of 3-5 items (10 items for the switch)

Item	Description	cost/item
	ESP8266/12E Microcontroller https://www.amazon.it/gp/product/B0754HWZSQ	7,50€
	DS3231 RTC Module https://www.amazon.it/gp/product/B076GP5B94	3,00€
	DFPlayer Mini https://www.amazon.it/gp/product/B07911V1NL	3,50€
	3W/4Ohm Speakers https://www.amazon.it/gp/product/B07LGHZD9R	4,50€

Item	Description	cost/item
	2-way switch https://www.amazon.it/gp/product/B077D9FRGL	0,80€
	Battery holder https://www.amazon.it/dp/B01FYUUPGE	2,50€
	Cables, wires, 10KOhm trimmer, battery snap, PCB	5,0€
	Enclosure Box 120x80x50 (IP56)	3,20€
	Total cost	**30,00€**

Connections schema created by
Fabrizio Calderan on Feb. 21, 2020



```
/*
 * Call for swallows and swifts
 * Code by Fabrizio Calderan, 02/17/2020
 * This call is set to work from mid-February to late March,
 * at sunrise and sunset. Sunrise times and sunset times (less
 * 2 hours) have been hardcoded in the functions below. They
 * are referred to the coordinates (45°N, 12°E).
 *
 *
 * Required components:
 *
 * - 1x ESP8266/12E
 * - 1x RTC Module DS3231
 * - 1x DFPlayer Mini
 * - 1x 3W/4Ω Speaker
 * - 1x On-Off switch
 * - 1x 10KΩ Trimmer
 * - 1x 120x80x50 IP56 enclosure box for outdoor use.
 */

#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"
#include <Wire.h>
#include <RtcDS3231.h>

#define DELAY_ONPLAY 5000
#define DELAY_ONPAUSE 50000
#define countof(a) (sizeof(a) / sizeof(a[0]))

// RTC Library (I2C)
RtcDS3231<TwoWire> rtcObject(Wire);

// MP3 Serial communication
SoftwareSerial mySoftwareSerial(14, 12);
DFRobotDFPlayerMini myDFPlayer;

// Variables
int volume, y;
bool IDLE = true;

RtcDateTime currentTime, compileTime, stopTime;
RtcDateTime tsBegin, tsEnd;

// Playback state get information through the BUSY pin
bool playbackState = digitalRead(13);


void setup() {

    mySoftwareSerial.begin(9600);
    // Start serial connection
    Serial.begin(115200);
    // Start I2C
    rtcObject.Begin();

    /* Date and time first configuration
     *
     * Define a date and time object that accepts the format
     * yyyy, m, d, H, M, S. Then configure the RTC with the
     * object defined.
     */
    stopTime    = RtcDateTime(1978, 4, 8, 10, 40, 0);
    compileTime = RtcDateTime(__DATE__, __TIME__);
    currentTime = rtcObject.GetDateTime();

    if (compileTime > currentTime) {
       Serial.print("Adjusting RTC datetime from ");
       Serial.print(printDateTime(currentTime));
       Serial.print(" to ");
       Serial.println(printDateTime(compileTime));
       rtcObject.SetDateTime(compileTime);
    }

    // Get the current Year
    y = currentTime.Year();

    Serial.println("Init MP3 module");
    // Use softwareSerial to begin communication with the MP3 module
    if (!myDFPlayer.begin(mySoftwareSerial, false)) {
      while(true) {
         delay(0); // ESP8266 watchdog needs this
      }
    }

    /*
     * Set the volume
     */
    setVolume();

    /*
     * Playback Test
     */
    myDFPlayer.play(1);
    delay(8000);
    myDFPlayer.stop();

    Serial.println("-------- /setup --------");
}



void loop() {

    setVolume();
    playbackState = digitalRead(13);

    // Get the time from the RTC
    currentTime = rtcObject.GetDateTime();

    Serial.print("Current datetime: ");
    Serial.println(printDateTime(currentTime));

    if (eventTS()) {

        IDLE = false;

        Serial.print("Playing until ");
        Serial.println(printDateTime(stopTime));

        /*
         *  LOW  = The MP3 module is busy
         *  HIGH = The MP3 module is available
         */

        if (playbackState == HIGH) {
            myDFPlayer.stop();
            myDFPlayer.play(1);
        }
        delay(DELAY_ONPLAY);

    }
    else {

        if (IDLE == false) {
            myDFPlayer.pause();
            myDFPlayer.stop();
            Serial.println("End of playback");
        }

        IDLE = true;
        delay(DELAY_ONPAUSE);
    }

    Serial.println("-------- /loop --------");

}



bool eventTS() {

    if (currentTime < stopTime) {
        return true;
    }

    return (sunriseTS() || sunsetTS());
}


void setVolume() {

    // Get the value of potentiometer in the range of [3..30]
    volume = map(analogRead(A0), 0, 1023, 2, 30);
    volume = constrain(volume, 3, 30);

    // Set the player volume
    myDFPlayer.volume(volume);
    Serial.print("Volume level: ");
    Serial.println(String(volume));
}



bool _TS(int mth_, int day_, int h_, int m_) {

    tsBegin = RtcDateTime(y, mth_, day_, h_, m_, 0);
    tsEnd   = RtcDateTime(y, mth_, day_, h_ + 2, m_, 0);

    if (currentTime >= tsBegin && currentTime <= tsEnd) {
        stopTime = tsEnd;
        return true;
    }
    else {
        return false;
    }
}



bool sunriseTS() {

    /* Sunrise datetimes from February 20 to March 27 */
    return (_TS(2, 20, 7, 8)  ||
        _TS(2, 21, 7, 6)  ||
        _TS(2, 22, 7, 4)  ||
        _TS(2, 23, 7, 3)  ||
        _TS(2, 24, 6, 1)  ||
        _TS(2, 25, 6, 59) ||
        _TS(2, 26, 6, 57) ||
        _TS(2, 27, 6, 56) ||
        _TS(2, 28, 6, 54) ||
        _TS(2, 29, 6, 52) ||
        _TS(3, 1, 6, 49)  ||
        _TS(3, 2, 6, 47)  ||
        _TS(3, 3, 6, 45)  ||
        _TS(3, 4, 6, 43)  ||
        _TS(3, 5, 6, 41)  ||
        _TS(3, 6, 6, 40)  ||
        _TS(3, 7, 6, 38)  ||
        _TS(3, 8, 6, 36)  ||
        _TS(3, 9, 6, 34)  ||
        _TS(3, 10, 6, 32) ||
        _TS(3, 11, 6, 30) ||
        _TS(3, 12, 6, 28) ||
        _TS(3, 13, 6, 27) ||
        _TS(3, 14, 6, 25) ||
        _TS(3, 15, 6, 23) ||
        _TS(3, 16, 6, 21) ||
        _TS(3, 17, 6, 19) ||
        _TS(3, 18, 6, 17) ||
        _TS(3, 19, 6, 15) ||
        _TS(3, 20, 6, 13) ||
        _TS(3, 21, 6, 11) ||
        _TS(3, 22, 6, 9)  ||
        _TS(3, 23, 6, 7)  ||
        _TS(3, 24, 6, 6)  ||
        _TS(3, 25, 6, 4)  ||
        _TS(3, 26, 6, 2)  ||
        _TS(3, 27, 6, 1));
}



bool sunsetTS() {

    /* Sunset datetimes from February 20 to March 27 (minus 2 hours) */
    return (_TS(2, 20, 15, 41) ||
        _TS(2, 21, 15, 42) ||
        _TS(2, 22, 15, 44) ||
        _TS(2, 23, 15, 45) ||
        _TS(2, 24, 15, 47) ||
        _TS(2, 25, 15, 48) ||
        _TS(2, 26, 15, 49) ||
        _TS(2, 27, 15, 51) ||
        _TS(2, 28, 15, 52) ||
        _TS(2, 29, 15, 54) ||
        _TS(3, 1, 15, 56)  ||
        _TS(3, 2, 15, 58)  ||
        _TS(3, 3, 15, 59)  ||
        _TS(3, 4, 16, 1)   ||
        _TS(3, 5, 16, 2)   ||
        _TS(3, 6, 16, 3)   ||
        _TS(3, 7, 16, 5)   ||
        _TS(3, 8, 16, 6)   ||
        _TS(3, 9, 16, 8)   ||
        _TS(3, 10, 16, 9)  ||
        _TS(3, 11, 16, 10) ||
        _TS(3, 12, 16, 12) ||
        _TS(3, 13, 16, 13) ||
        _TS(3, 14, 16, 14) ||
        _TS(3, 15, 16, 16) ||
        _TS(3, 16, 16, 17) ||
        _TS(3, 17, 16, 18) ||
        _TS(3, 18, 16, 20) ||
        _TS(3, 19, 16, 21) ||
        _TS(3, 20, 16, 22) ||
        _TS(3, 21, 16, 24) ||
        _TS(3, 22, 16, 25) ||
        _TS(3, 23, 16, 26) ||
        _TS(3, 24, 16, 28) ||
        _TS(3, 25, 16, 29) ||
        _TS(3, 26, 16, 30) ||
        _TS(3, 27, 16, 31));
}



String printDateTime(const RtcDateTime& dt) {
    char datestring[20];

    snprintf_P(datestring,countof(datestring),
            PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
            dt.Month(), dt.Day(), dt.Year(),
            dt.Hour(), dt.Minute(), dt.Second());

    return datestring;
}
```

Sunrise and sunset times

These times have been hardcoded in the ESP microcontroller in order to run at the sunrise and 2 hours before the sunset.

20/2 - 29/2

Date	Sr	Ss	Daylight
G 20/02/2020	7:08	17:41	(10h 33m)
V 21/02/2020	7:06	17:42	(10h 36m)
S 22/02/2020	7:04	17:44	(10h 40m)
D 23/02/2020	7:03	17:45	(10h 42m)
L 24/02/2020	7:01	17:47	(10h 46m)
M 25/02/2020	6:59	17:48	(10h 49m)
M 26/02/2020	6:57	17:49	(10h 52m)
G 27/02/2020	6:56	17:51	(10h 55m)
V 28/02/2020	6:54	17:52	(10h 58m)
S 29/02/2020	6:52	17:54	(11h 02m)

1/3 - 10/3

Date	Sr	Ss	Daylight
D 01/03/2020	6:49	17:56	(11h 07m)
L 02/03/2020	6:47	17:58	(11h 11m)
M 03/03/2020	6:45	17:59	(11h 14m)
M 04/03/2020	6:43	18:01	(11h 18m)
G 05/03/2020	6:41	18:02	(11h 21m)
V 06/03/2020	6:40	18:03	(11h 23m)
S 07/03/2020	6:38	18:05	(11h 27m)
D 08/03/2020	6:36	18:06	(11h 30m)

Date	Sr	Ss	Daylight
L 09/03/2020	6:34	18:08	(11h 34m)
M 10/03/2020	6:32	18:09	(11h 37m)

11/3 - 20/3

Date	Sr	Ss	Daylight
M 11/03/2020	6:30	18:10	(11h 40m)
G 12/03/2020	6:28	18:12	(11h 44m)
V 13/03/2020	6:27	18:13	(11h 46m)
S 14/03/2020	6:25	18:14	(11h 49m)
D 15/03/2020	6:23	18:16	(11h 53m)
L 16/03/2020	6:21	18:17	(11h 56m)
M 17/03/2020	6:19	18:18	(11h 59m)
M 18/03/2020	6:17	18:20	(12h 03m)
G 19/03/2020	6:15	18:21	(12h 06m)
V 20/03/2020	6:13	18:22	(12h 09m)

21/3 - 28/3

Date	Sr	Ss	Daylight
S 21/03/2020	6:11	18:24	(12h 13m)
D 22/03/2020	6:09	18:25	(12h 16m)
L 23/03/2020	6:07	18:26	(12h 19m)
M 24/03/2020	6:06	18:28	(12h 22m)
M 25/03/2020	6:04	18:29	(12h 25m)
G 26/03/2020	6:02	18:30	(12h 28m)
V 27/03/2020	6:00	18:31	(12h 31m)
S 28/03/2020	5:58	18:33	(12h 35m)

Labels for Acetone transfer