ESP8266-Based

Air Quality Monitoring System

Petteri Haverinen

Table of Contents

Abstract

The project aimed to design and build a system for monitoring the air quality in classrooms/rooms. The basic idea is that the air quality of a classroom can be monitored and proper breaks taken when the quality goes past a certain level. If the sensor data is displayed to the schools info-screens students can account for the temperature/CO2 levels by choosing the appropriate clothing and grabbing water thus reducing fatigue. The data can also be used for optimizing the air conditioning system.

The project was successfully executed with temperature/humidity working perfectly, but the gas sensor is uncalibrated and thus giving only rough estimates (I.E the higher value, the worse the air is). The estimated operating time of the device is around 6 days with 1Ah rechargeable batteries.

Used Components, Tools and Software

The parts used for the project are off the shelf components and 3D-printing filament. Thus the expenses were low.

Components

The parts used were as follows:

Component	**Amount**
Arduino nano	1x
ESP8266-01	1x
2x4 female pin header (2.54mm pitch)	1x
1x16 female pin header (2.54mm pitch)	2x
1k resistor	1x
2k resistor	1x
56k resistor	2x
10k resistor	2x
IRFD110	2x
LM1117-3.3	1x
10uF electrolytic capacitor	1x
3 pin 9mm micro sliding switch	1x

Side note: a length of PLA filament, 3 sticks of hot glue and a length of solder were used on top of the components mentioned above.

Tools

The tools used were a soldering station (JBC DM-4) with two cartridge tips, wide and narrow, side cutters, tweezers, screwdriver with Philips head, hot glue gun, Fluke-177, box cutter and digital caliber.

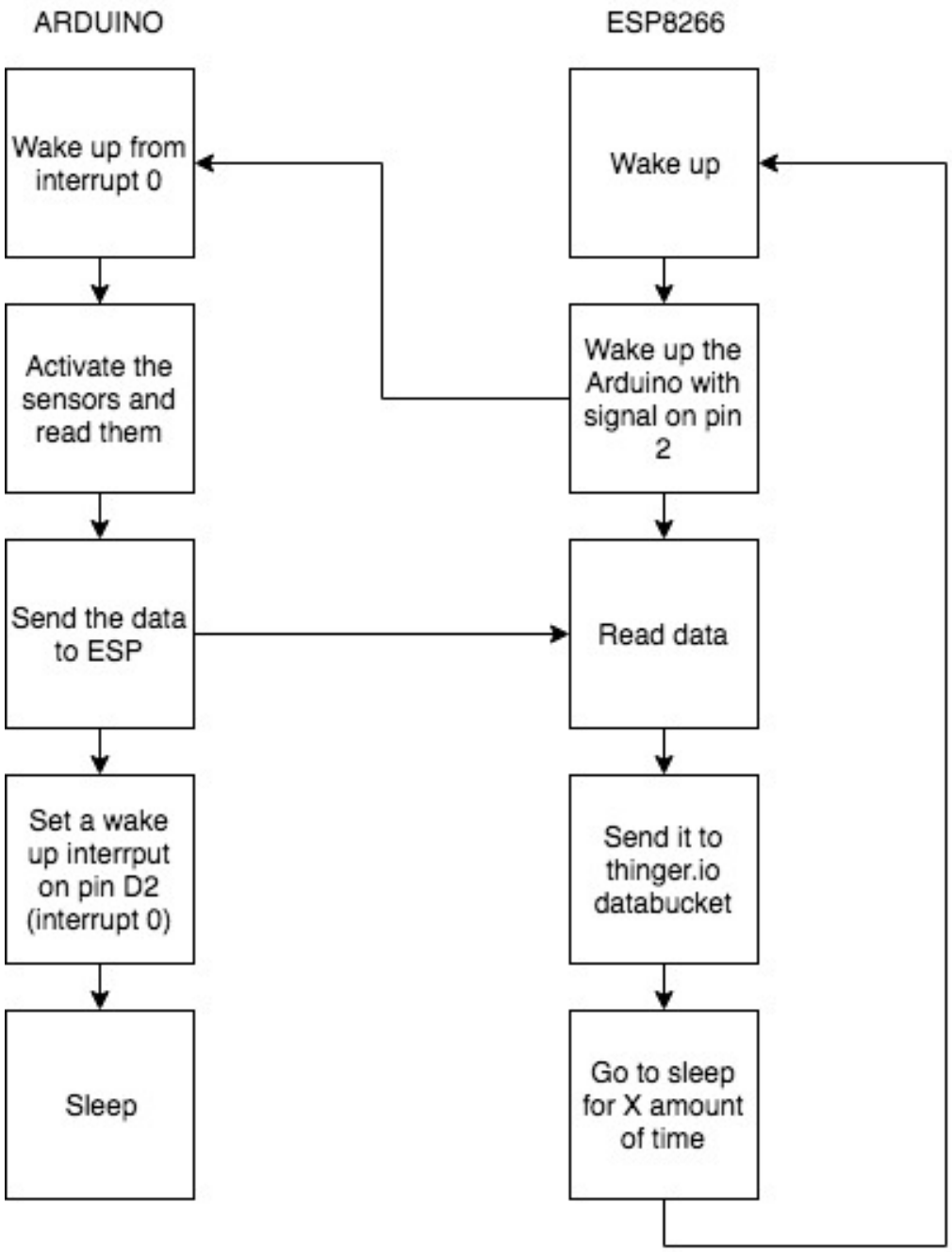
Software

The software used were: Fusion 360 for creating the CAD-models, Cura 3.4.1 for creating G-Code for 3D-printing, EasyEDA for creating schematics, Arduino IDE for programming the MCUs and thinger.io service for data processing, collecting and displaying.

Theory of Operation

Flowchart and step by step operation overview

The system is based around Arduino nano and ESP8266-01 MCUs. The following flow chart explains the rough operating principle:



Arduino

1. The Arduino wakes up with a signal to its interrupt 0.
2. The Arduino activates two MOSFETS (IRFD110) turning on the DHT11 and MQ-135 sensors.
3. The Arduino reads the data.
4. The data is sent to ESP8266 via softwareSerial connection set up on pins D3 and D4
5. The Arduino sets a wake-up interrupt on the pin D2.
6. The Arduino goes to sleep.

ESP8266

1. The ESP8266 wakes up.
2. The ESP8266 wakes up Arduino with a HIGH-signal (3.3V) on the digital pin D2.
3. The ESP8266 reads the data received from the Arduino.
4. The ESP8266 sends the data to thinger.io service on three different databuckets.
5. The ESP8266 goes to sleep for X amount of time.

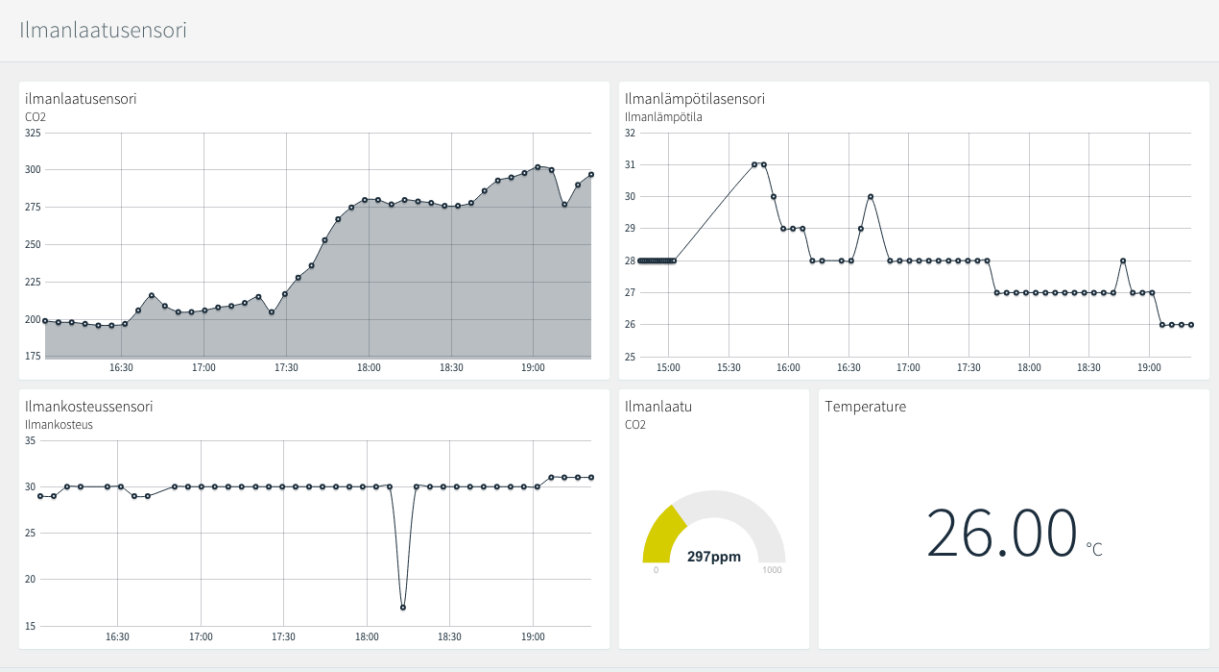
Data Processing and Display

The data is processed and displayed in an online service called thinger.io. The data is displayed on a custom dashboard, that has time-value graphs for humidity, temperature and air quality and two meters for displaying the temperature and air quality.

The data is stored in three CSV files in the server that are read by the widgets in the dashboard. The CSV files hold each one element of the transmitted data, temperature, humidity and air quality.

The data is stored in these files by calling a command: *thing.write_bucket("Quality", "qlit");* in the code. The command holds the databucket id (in the example: “Quality”) and the resource that carries the data (“qlit”). The resource is declared in the void setup() by:

thing["qlit"] >> outputValue(a); in which “qlit” is the name of the resource and “a” is a variable. These commands need the thinger.io library.



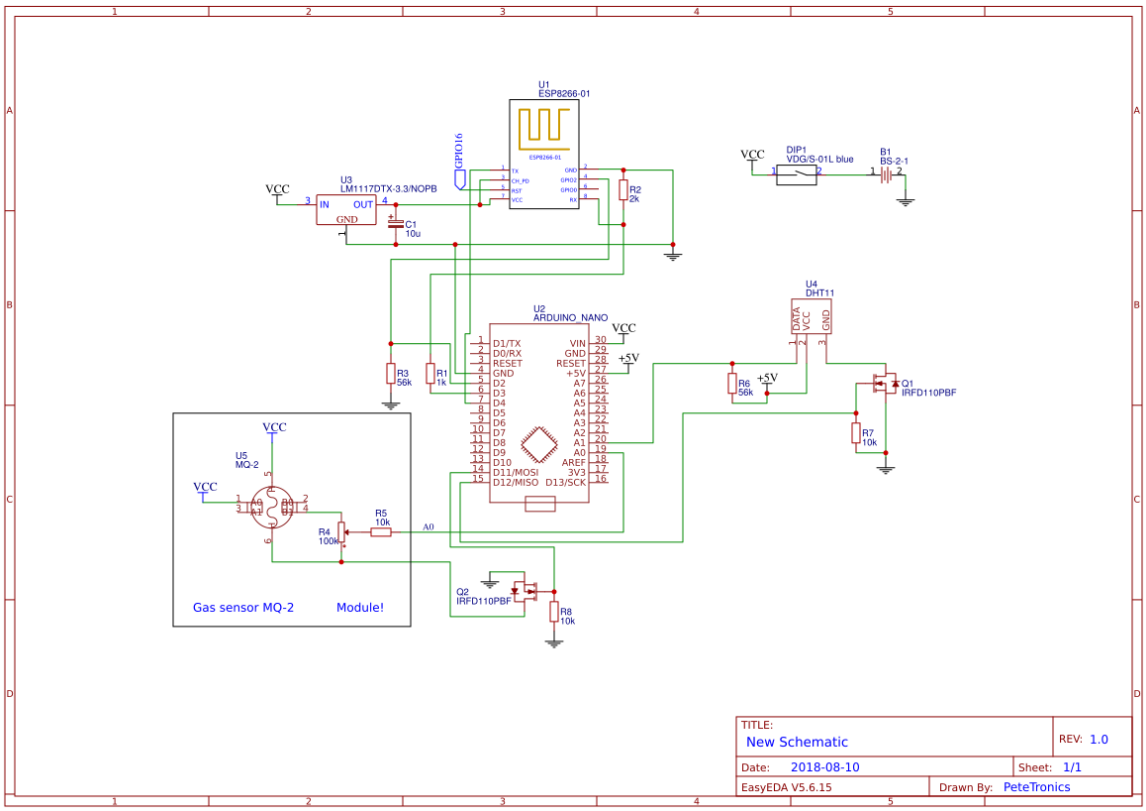
picture 2. How the data is displayed in the dashboard.

	Bucket	Name	Description	State	Enabled
☐	**Temperature**	Temperature	temperature	Normal	Enabled
☐	**Humidity**	Humidity	Humidity	Normal	Enabled
☐	**Quality**	Quality	quality	Normal	Enabled

Picture 3. The list of databuckets

The Circuit Overview

The circuit is built onto a piece of perfboard. It uses Arduino nano and ESP8266 as the main MCUs and DHT11 & MQ-135 sensors.

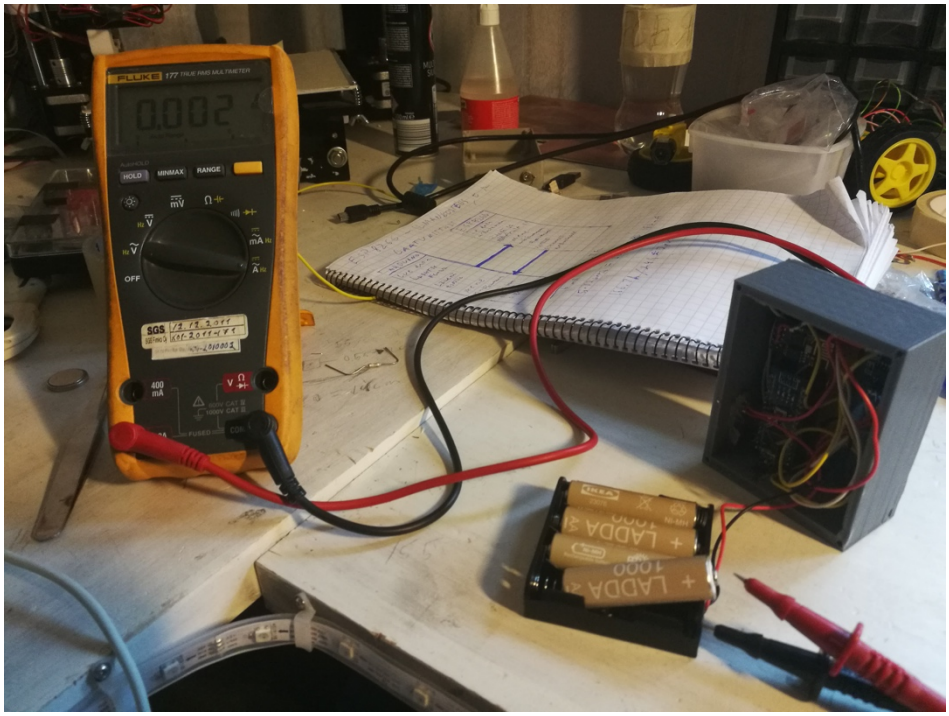


The MQ-135 sensor is a module. The DHT11 is not on a module, so it requires a resistor (5.6K in this case, the recommendation is 4.7K) between pins VCC and data. The regulator supplying the ESP8266 3.3V is LM1117-3.3. The circuit uses 6V in form of 4xAA batteries in series.

The MOSFETS for the Arduino are used for switching power on/off to the sensors in order to conserve energy in sleep mode. The pin 02 of the ESP8266 is connected to the pin D2 of the Arduino for the same reason. The Arduino nano has no watchdog timer and it can't wake up from deep sleep on its own. This is why we use an interrupt and wake it up by using the ESP8266 as "alarm clock". This means that when the ESP, that has a watchdog timer, wakes up from deep sleep and gives HIGH signal to the Arduino, of which D2 is pulled low via 5.6K resistor. The Arduino can read this as a HIGH signal, though it is 3.3V logic HIGH signal and Arduino nano operates on 5V TTL logic level.

Power Consumption and Usage Time

The power consumption was measured using Fluke-177 true RMS multimeter. The setup was following:



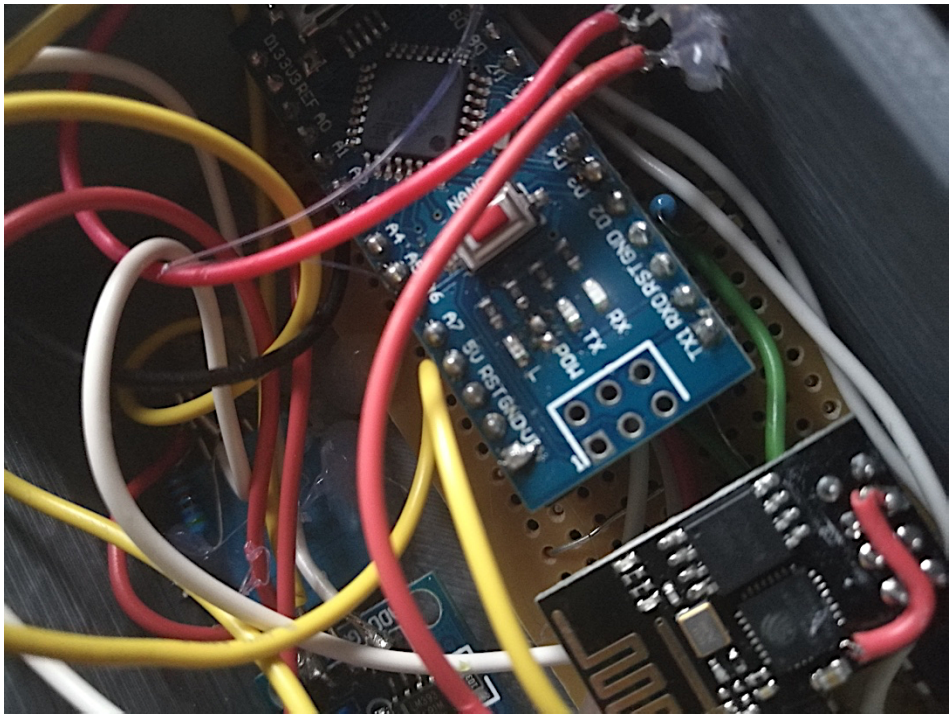
The multimeter was connected in series with the battery pack and the system. The average value during operation was 180mA and 6mA in sleep mode. The update is done once every 5 minutes, thus the operating time could be prolonged by choosing longer update times.

Note: the measurements were done in A range instead of mA range due to a blown fuse in the meter.

Given these power values, it is estimated that the system can operate for around 6 days with 5 minute data update intervals using 1000mAh Ni-MH batteries. The batteries are rechargeable so the system is not wasteful.

Reducing the power usage

The power usage was limited by using sleep modes and switching the power to the sensors only when transmitting data. The power consumption thus goes to 8mA (in sleep mode). It was however brought down to 6mA by removing the power indicator LEDs from the Arduino nano and ESP8266.



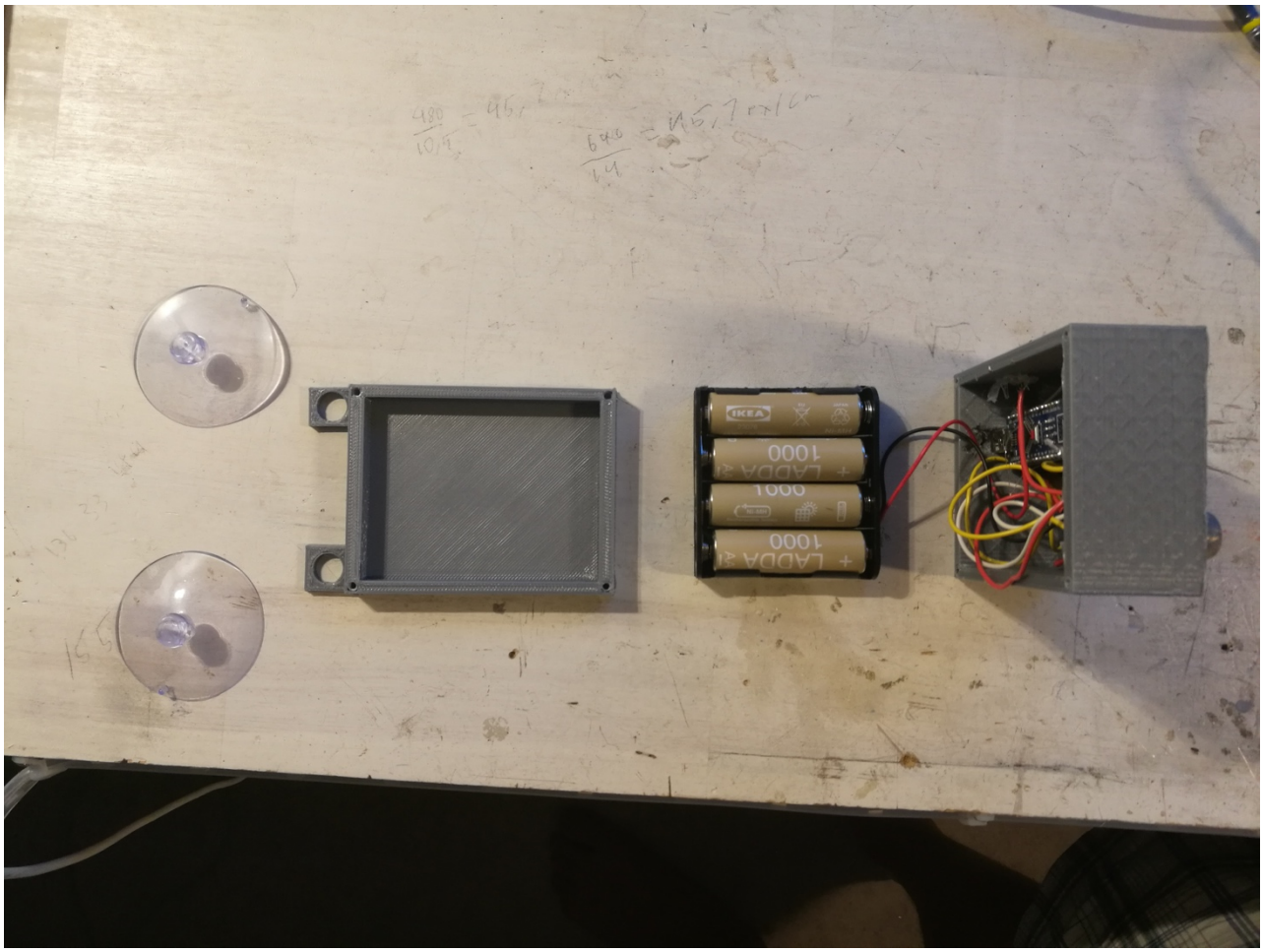
Picture 6. The removed LED

Enclosure & Mechanical build

The enclosure is 3D-printed in two parts using Prusa i3 printer (clone) from PLA-plastic. The print speeds were 90mm/s print, 120mm/s travel and 0.3mm layer height. The enclosure has holes for the sensors, switch and Arduino nanos USB-port. The hole for the USB-port is in too thick wall, thus requiring custom USB-connector if the Arduino is wanted to be programmed while in the enclosure.

The enclosure has 2x50mm suction cups for attaching the sensor to walls or other vertical surfaces. They have proven to be working perfectly but need moisture when attaching in order to reach maximum suction.

The enclosure is held together by screws (2.2x30mm plastic screws).



Picture 7. The enclosure dissassembled

Further Development Plans

The device is planned to be used in a school and thus it is crucial to actually carry out a long period field testing. The most apparent problem is making the batteries last longer. The solution for this would be to

1. Reducing the power usage by replacing the requlators with switch mode PSU and/or more efficient regulators.
2. Reduce the sampling rate by atleast 50%
3. Use higher capacity batteries or switch to high capacity Li-Ion cells instead of alkaline batteries or Ni-MH
4. Use wall adapters instead of batteries

The data feed (dashpanel) could be integrated into info screens at the facilities they would be used in.

The device could be integrated into a bigger IoT-system and have them control air conditioning thus optimizing them and bringing operation cost down and studying comfort up.

Code and Attachments

Arduino nano code:

https://drive.google.com/open?id=1mNFGaUfruMqWe5BxYxLuMGveSVsUgHG9

ESP8266 code:

https://drive.google.com/open?id=1-zfdXaLZoOwdtqaNNVlNY2Lmjh9Epnxa

Schematic:

https://drive.google.com/open?id=1uVexD2Cp8ttwnQg4tt4Vy09LQJ_cj58L

Cad Files:

https://drive.google.com/open?id=10rstp591m9TYZ-fM5fsg_HNG3pRlmQEH

Flowchart in PDF:

https://drive.google.com/open?id=1Q9oVbcaan1C74IzgsBaI-xwJvIA706kU