PONTIFICAL CATHOLIC UNIVERSITY OF CAMPINAS

POLYTECHNIC SCHOOL

FINAL DELIVERY

NETWORKS AND COMMUNICATION SYSTEMS WITH IOT PROJECT + PHOTOVOTAIC ENERGY

STUDENTS: CHARLES DE SOUZA **RA:** 21931233

PATRICK GERALDI **RA:** 21959390

EMERSON MAFALDA **RA:** 21004538

COURSE: CONTROL AND AUTOMATION ENGINEERING

Campinas
2023

Project title

"IoT-based humidity and temperature monitoring system".

Summary

In an increasingly connected world, networks play a key role in enabling the transfer of information from sensors, which may be in remote locations, to central processing centers, where the data is analyzed and made available to users. In this project, the focus is on understanding how a network and communication infrastructure works, the aim of which is to deliver the data collected by the sensors to the cloud in real time and accurately via Wi-Fi. The choice, therefore, was for a humidity and temperature monitoring system, since these are crucial parameters in applications such as agriculture, the food industry, air conditioning, among others.

Introduction

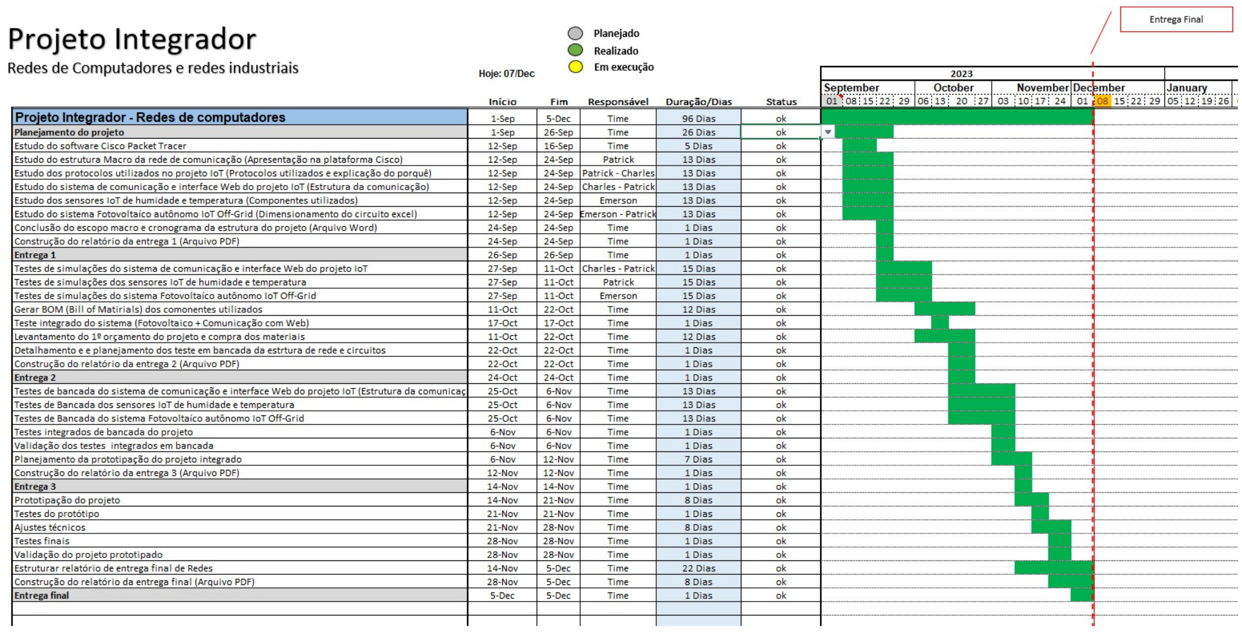
The project consists of implementing a sensor signal monitoring system that will capture temperature and relative humidity levels and transmit this data remotely via WI-FI to a cloud database, where it will be stored. Once the data has been processed, it will be displayed in graphical indicators via a web interface using the Ubidots platform, which users will be able to view and interact with. It will also have a photovoltaic energy supply system with battery(s), ensuring that the system is completely autonomous. The main parts of this project are as follows:

- Photovoltaic energy storage system with batteries; Sensor system to
- capture humidity and temperature signals;
- Microcontrolled hardware module for handling input signals and Wi-Fi
- transmission of output signals;
- Network systems that will ensure data communication between devices;
- Database for storing collected data; Web interface for data
- indicators.

Objective of the work

The aim of this work is to deepen the concepts of the subject of Communication Networks and Systems through the construction and development of a project that integrates the various subjects taught in the Control and Automation Engineering course offered by PUC-Campinas and that covers real market demands, seeking practical applications for a everyday problem.

Timetable



Projeto Integrador

Redes de Computadores e redes industriais

Hoje: 07/Dec

● Planejado
🟩 Realizado
● Em execução

	Início	Fim	Responsável	Duração/Dias	Status
Projeto Integrador - Redes de computadores	1-Sep	5-Dec	Time	96 Dias	ok
Planejamento do projeto	1-Sep	26-Sep	Time	26 Dias	ok
Estudo do software Cisco Packet Tracer	12-Sep	16-Sep	Time	5 Dias	ok
Estudo do estrutura Macro da rede de comunicação (Apresentação na plataforma Cisco)	12-Sep	24-Sep	Patrick	13 Dias	ok
Estudo dos protocolos utilizados no projeto IoT (Protocolos utilizados e explicação do porquê)	12-Sep	24-Sep	Patrick - Charles	13 Dias	ok
Estudo do sistema de comunicação e interface Web do projeto IoT (Estrutura da comunicação)	12-Sep	24-Sep	Charles - Patrick	13 Dias	ok
Estudo dos sensores IoT de humidade e temperatura (Componentes utilizados)	12-Sep	24-Sep	Emerson	13 Dias	ok
Estudo do sistema Fotovoltaíco autônomo IoT Off-Grid (Dimensionamento do circuito excel)	12-Sep	24-Sep	Emerson - Patrick	13 Dias	ok
Conclusão do escopo macro e cronograma da estrutura do projeto (Arquivo Word)	24-Sep	24-Sep	Time	1 Dias	ok
Construção do relatório da entrega 1 (Arquivo PDF)	24-Sep	24-Sep	Time	1 Dias	ok
Entrega 1	26-Sep	26-Sep	TIme	1 Dias	ok
Testes de simulações do sistema de comunicação e interface Web do projeto IoT	27-Sep	11-Oct	Charles - Patrick	15 Dias	ok
Testes de simulações dos sensores IoT de humidade e temperatura	27-Sep	11-Oct	Patrick	15 Dias	ok
Testes de simulações do sistema Fotovoltaíco autônomo IoT Off-Grid	27-Sep	11-Oct	Emerson	15 Dias	ok
Gerar BOM (Bill of Matirials) dos comonentes utilizados	11-Oct	22-Oct	Time	12 Dias	ok
Teste integrado do sistema (Fotovoltaico + Comunicação com Web)	17-Oct	17-Oct	Time	1 Dias	ok
Levantamento do 1º orçamento do projeto e compra dos materiais	11-Oct	22-Oct	Time	12 Dias	ok
Detalhamento e e planejamento dos teste em bancada da estrtura de rede e circuitos	22-Oct	22-Oct	Time	1 Dias	ok
Construção do relatório da entrega 2 (Arquivo PDF)	22-Oct	22-Oct	Time	1 Dias	ok
Entrega 2	24-Oct	24-Oct	Time	1 Dias	ok
Testes de bancada do sistema de comunicação e interface Web do projeto IoT (Estrutura da comunicaç	25-Oct	6-Nov	Time	13 Dias	ok
Testes de Bancada dos sensores IoT de humidade e temperatura	25-Oct	6-Nov	Time	13 Dias	ok
Testes de Bancada do sistema Fotovoltaíco autônomo IoT Off-Grid	25-Oct	6-Nov	Time	13 Dias	ok
Testes integrados de bancada do projeto	6-Nov	6-Nov	Time	1 Dias	ok
Validação dos testes integrados em bancada	6-Nov	6-Nov	Time	1 Dias	ok
Planejamento da prototipação do projeto integrado	6-Nov	12-Nov	Time	7 Dias	ok
Construção do relatório da entrega 3 (Arquivo PDF)	12-Nov	12-Nov	Time	1 Dias	ok
Entrega 3	14-Nov	14-Nov	Time	1 Dias	ok
Prototipação do projeto	14-Nov	21-Nov	Time	8 Dias	ok
Testes do protótipo	21-Nov	21-Nov	Time	1 Dias	ok
Ajustes técnicos	21-Nov	28-Nov	Time	8 Dias	ok
Testes finais	28-Nov	28-Nov	Time	1 Dias	ok
Validação do projeto prototipado	28-Nov	28-Nov	Time	1 Dias	ok
Estruturar relatório de entrega final de Redes	14-Nov	5-Dec	Time	22 Dias	ok
Construção do relatório da entrega final (Arquivo PDF)	28-Nov	5-Dec	Time	8 Dias	ok
Entrega final	5-Dec	5-Dec	Time	1 Dias	ok

Development

- **Project design**

The project was developed through the article "*IoT Based Solar Power Monitoring & Data Logger*", which was written by electrical engineers and researchers from different countries and published on the "*IEEE Xplore*" website. This site, which was recommended by Professor Hamilton Schroder in class, is non-profit and calls itself the world's largest professional technical organization dedicated to the advancement of technology for the benefit of humanity.

This article focuses on the Internet of Things and uses well-known components in this field, such as the ESP32 microcontroller, created and developed by the Chinese company Espressif Systems, an off-grid solar panel system, current, voltage, humidity, temperature and light sensors and a data monitoring platform called *Thingspeak*, which uses communication with the cloud to monitor data in real time.

However, as the focus of this project will be on networks, the number of materials required for this project has been reduced. The exception is the use of the *Ubidots* platform for data monitoring instead of the *Thingspeak* platform, which is not a strategic choice for the project, as both have the same complexity.

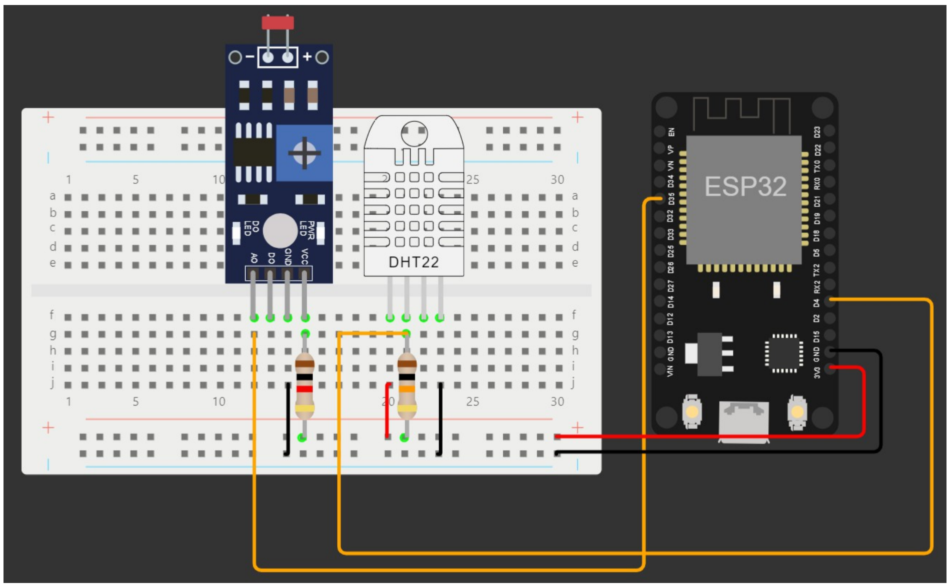
- ## ESP32, simulator assembly and code

In order to control the system by capturing temperature, humidity and luminosity data from the environment, we used the ESP32 microcontroller, which is well known in the Internet of Things world for having - compared to Arduino - superior performance, Wi-Fi and Bluetooth built into its basic configuration and versatility when programming, as well as data security. It will be covered in a more technical way in the networking topic.

With regard to the ESP32 and its IoT technologies, the ESP32 is a low-cost, low-power SoC (system-on-a-chip) designed for IoT applications. It is compatible with a range of wireless protocols, including Wi-Fi, Bluetooth and BLE. ESP32 is also compatible with a variety of IoT protocols, including MQTT, CoAP and HTTP. These protocols are designed to meet the unique requirements of IoT devices, such as low power consumption, low data rates and short-range communication.

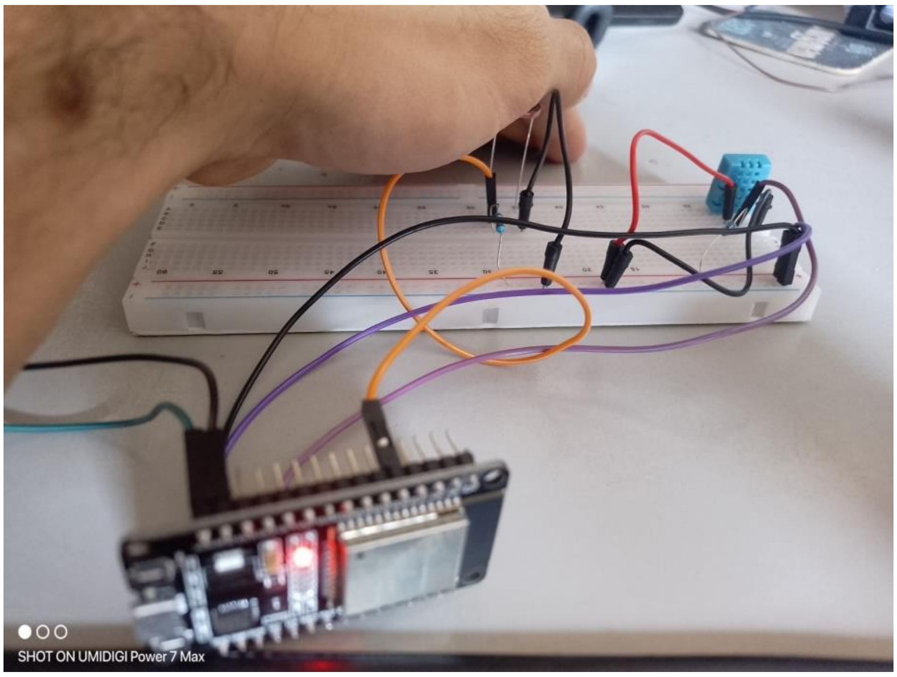
Simulator assembly

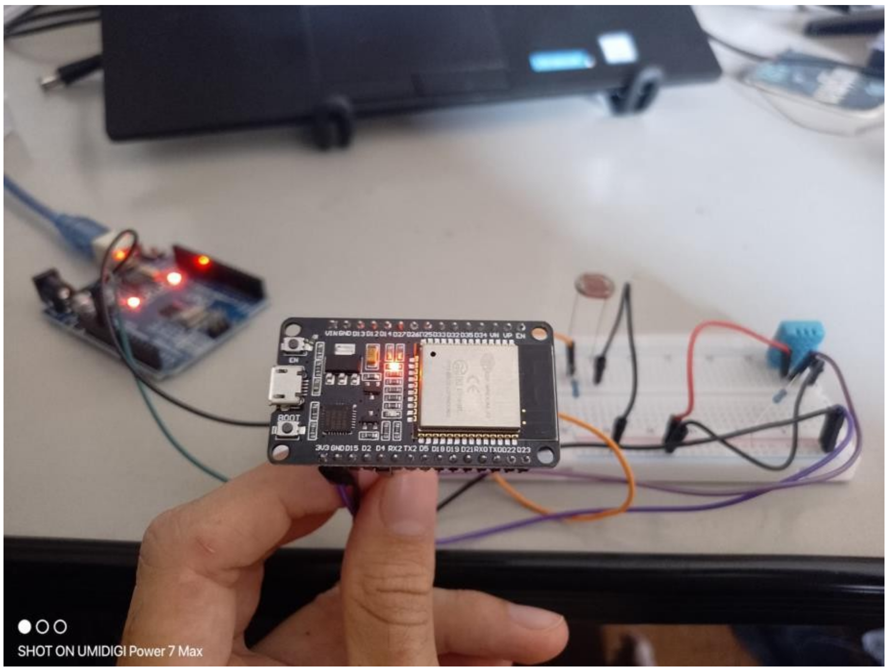
The assembly consists of light (LDR) and temperature and humidity (DHT11) sensors, the ESP32 microcontroller itself and 2 resistors: 1 1K Ohm resistor on the LDR and another 10K Ohm resistor on the DHT11 signal pin. The image below is for illustrative purposes only - please note the connections
- because the materials were not found in the *Wokwi* simulator list.



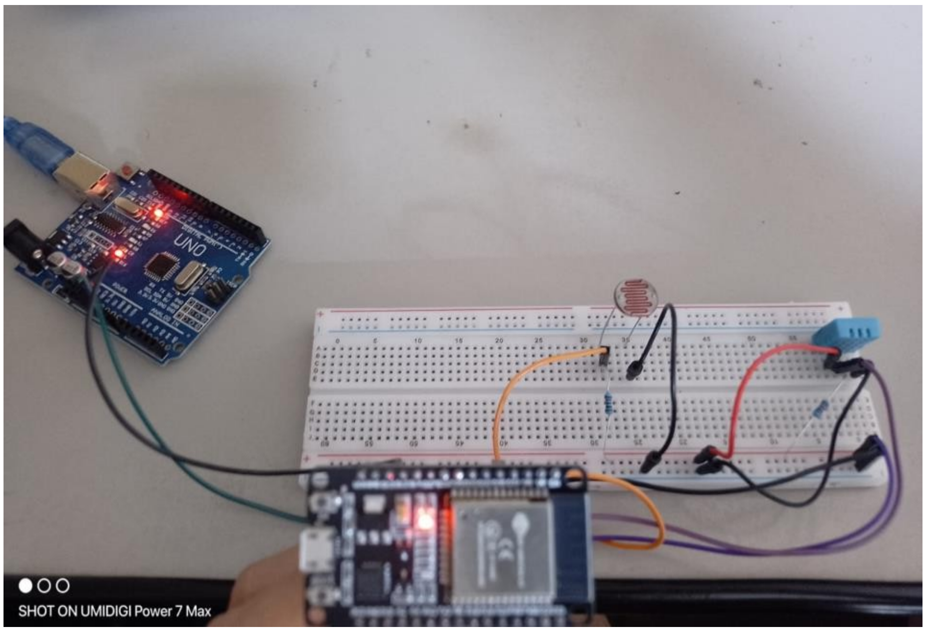
Practical assembly for testing

The practical assembly has the same idea as the simulator assembly, but only a few components are changed in practice, such as the DHT11 instead of the DHT22 and the LDR sensor without a ready-made module. So, to carry out the tests with the ESP32 microcontroller, it was as follows:





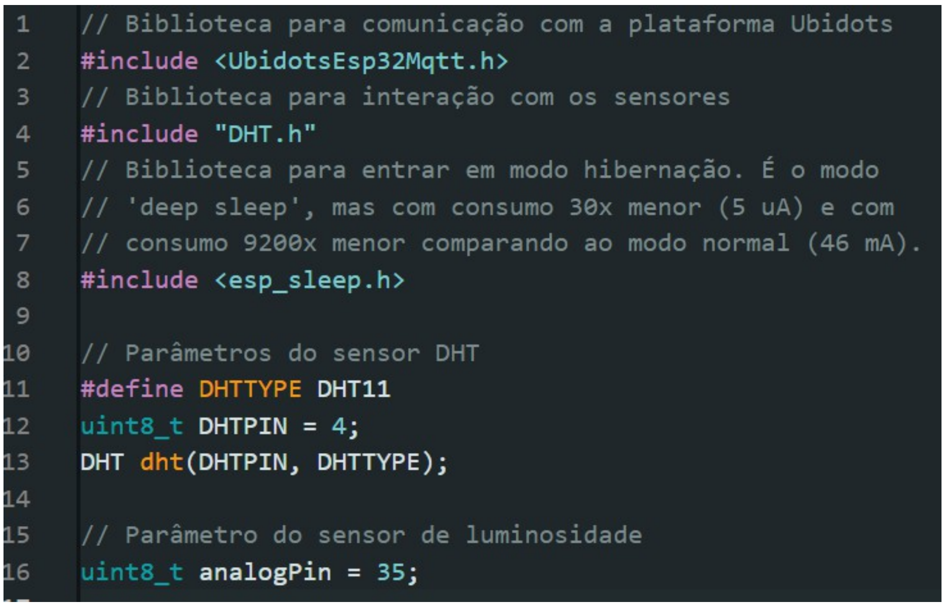
Since the ESP32 is the brain of the circuit, it was possible to use the analog and digital inputs, representing the results received from the signals with the cloud through graphs on the Ubidots platform.



The Arduino UNO was used to simulate the battery and the step-up that injects the ESP32 with the 5 V it needs to function properly.

Code

The initial idea was to comment in Word itself, but someone might want to use the code and it wouldn't have embedded comments. So it has been commented in the code itself to make life easier for anyone who uses it later.

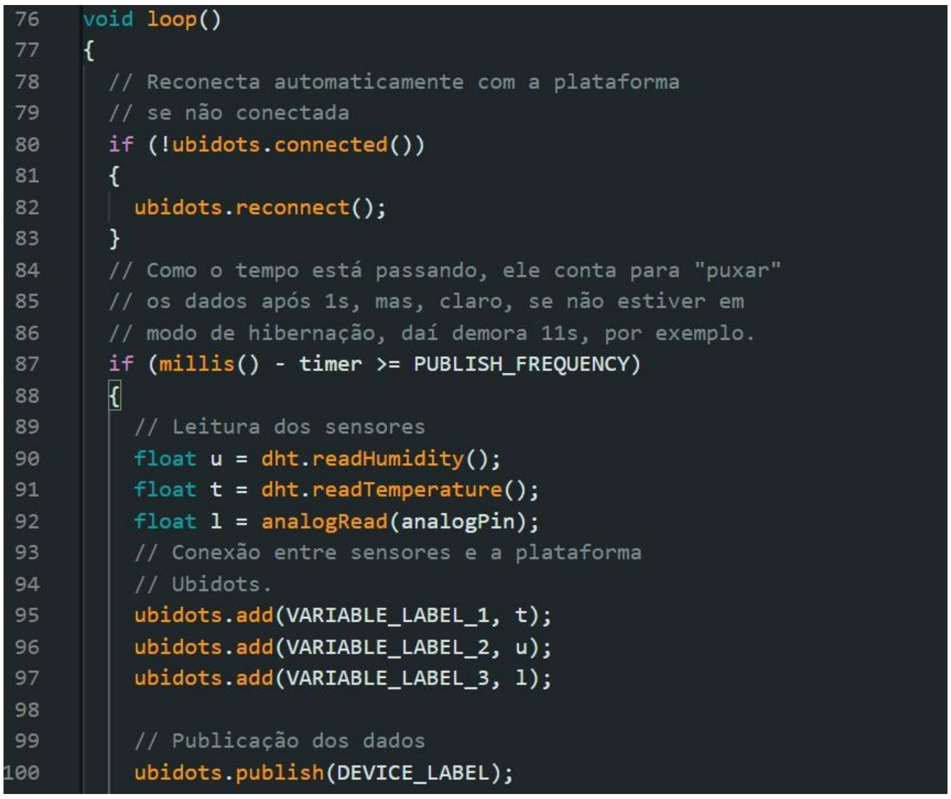


```
// Biblioteca para comunicação com a plataforma Ubidots
#include <UbidotsEsp32Mqtt.h>
// Biblioteca para interação com os sensores
#include "DHT.h"
// Biblioteca para entrar em modo hibernação. É o modo
// 'deep sleep', mas com consumo 30x menor (5 uA) e com
// consumo 9200x menor comparando ao modo normal (46 mA).
#include <esp_sleep.h>

// Parâmetros do sensor DHT
#define DHTTYPE DHT11
uint8_t DHTPIN = 4;
DHT dht(DHTPIN, DHTTYPE);

// Parâmetro do sensor de luminosidade
uint8_t analogPin = 35;
```

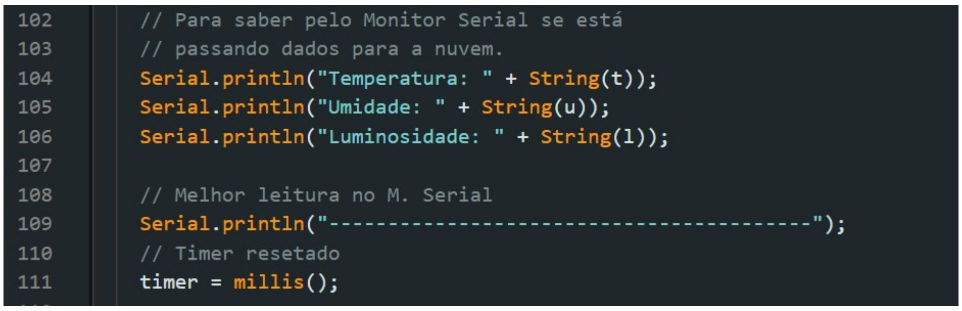
```
l'I_S2I3 =                    ;
```

JïIG'?T?_*'ï:..I.

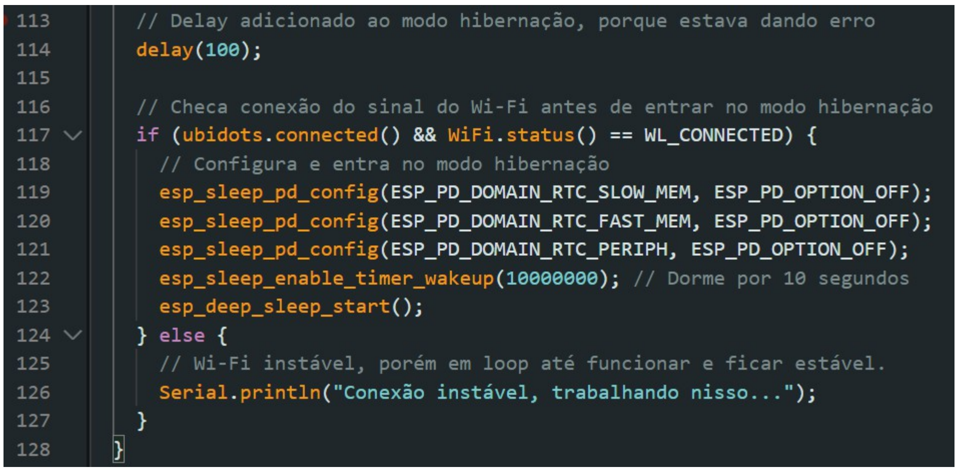
'.IFI_?SID, . .'IFI_F-?S '.



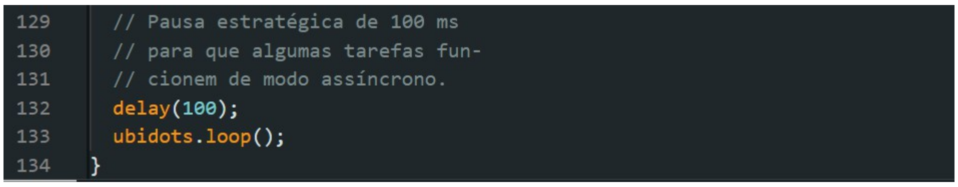
```
void loop()
{
  // Reconecta automaticamente com a plataforma
  // se não conectada
  if (!ubidots.connected())
  {
    ubidots.reconnect();
  }
  // Como o tempo está passando, ele conta para "puxar"
  // os dados após 1s, mas, claro, se não estiver em
  // modo de hibernação, daí demora 11s, por exemplo.
  if (millis() - timer >= PUBLISH_FREQUENCY)
  {
    // Leitura dos sensores
    float u = dht.readHumidity();
    float t = dht.readTemperature();
    float l = analogRead(analogPin);
    // Conexão entre sensores e a plataforma
    // Ubidots.
    ubidots.add(VARIABLE_LABEL_1, t);
    ubidots.add(VARIABLE_LABEL_2, u);
    ubidots.add(VARIABLE_LABEL_3, l);

    // Publicação dos dados
    ubidots.publish(DEVICE_LABEL);
```



```
    // Para saber pelo Monitor Serial se está
    // passando dados para a nuvem.
    Serial.println("Temperatura: " + String(t));
    Serial.println("Umidade: " + String(u));
    Serial.println("Luminosidade: " + String(l));

    // Melhor leitura no M. Serial
    Serial.println("----------------------------------------");
    // Timer resetado
    timer = millis();
```



```
    // Delay adicionado ao modo hibernação, porque estava dando erro
    delay(100);

    // Checa conexão do sinal do Wi-Fi antes de entrar no modo hibernação
    if (ubidots.connected() && WiFi.status() == WL_CONNECTED) {
      // Configura e entra no modo hibernação
      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM, ESP_PD_OPTION_OFF);
      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM, ESP_PD_OPTION_OFF);
      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
      esp_sleep_enable_timer_wakeup(10000000); // Dorme por 10 segundos
      esp_deep_sleep_start();
    } else {
      // Wi-Fi instável, porém em loop até funcionar e ficar estável.
      Serial.println("Conexão instável, trabalhando nisso...");
    }
  }
```

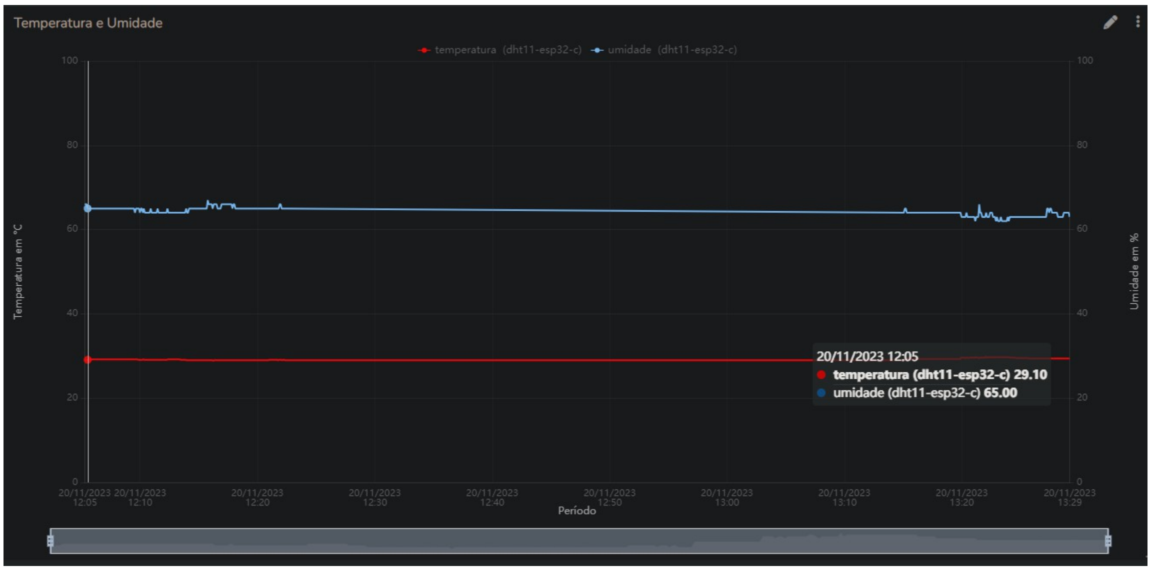


```
  // Pausa estratégica de 100 ms
  // para que algumas tarefas fun-
  // cionem de modo assíncrono.
  delay(100);
  ubidots.loop();
}
```

In other words, the code above implements a temperature, humidity and light monitoring system using ESP32. To do this, the *Ubidots* platform is used to send data collected by sensors to the cloud. The main steps include configuring sensor pins, defining connection information and setting a data publishing frequency of between 11 and 14 seconds. This period was configured with ESP32's own sleep mode. The code monitors the sensors, sends the data to the *Ubidots* platform and displays the readings on the serial monitor for *debugging*. It was important to implement the last item mentioned because it was discovered that the light sensor was not working properly, so it was not a problem with the connection to the cloud. In this case, the LDR sensor was replaced and the data could be displayed on the platform properly.

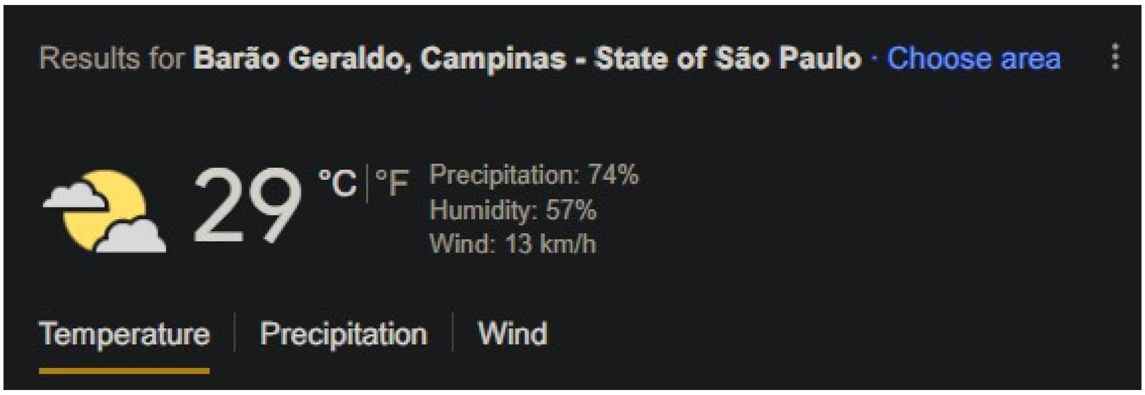
In addition, the *deep sleep* mode that is an option of the ESP32 was configured, thus guaranteeing a longer battery life for the project. As explained in one of the comments in the code, this saving is 9200 times smaller when compared to normal use of the ESP32. In this case, *deep sleep* mode was not implemented, but rather *hibernation mode*, as it uses only 5 µA each time the microcontroller returns to collect the data and send it to the Ubidots platform instead of 46 mA, for example, in *deep sleep* mode. This guarantees significant savings for projects that use batteries in remote areas, as well as making them commercially available with interesting scalability.

The results are as follows:



20/11/2023 12:05
● temperatura (dht11-esp32-c) 29.10
● umidade (dht11-esp32-c) 65.00

Measuring temperature and humidity with very low margins of error as can be seen in the screenshot below according to the location of one of the team members (Barão Geraldo, Campinas):

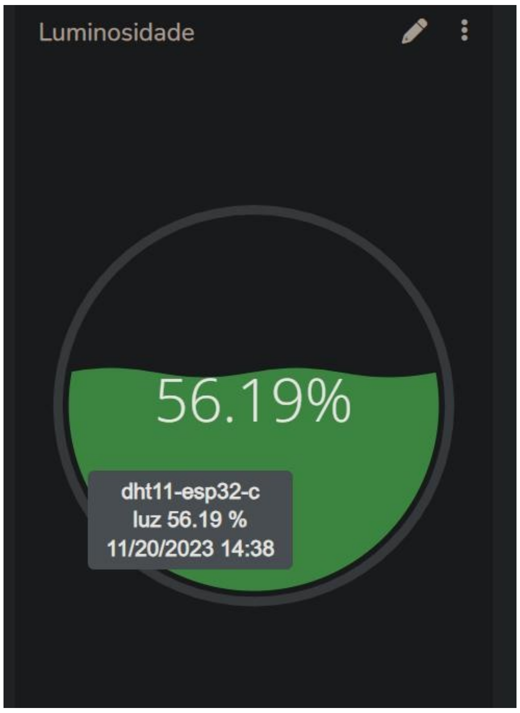


Results for **Barão Geraldo, Campinas - State of São Paulo** · Choose area

29 °C | °F

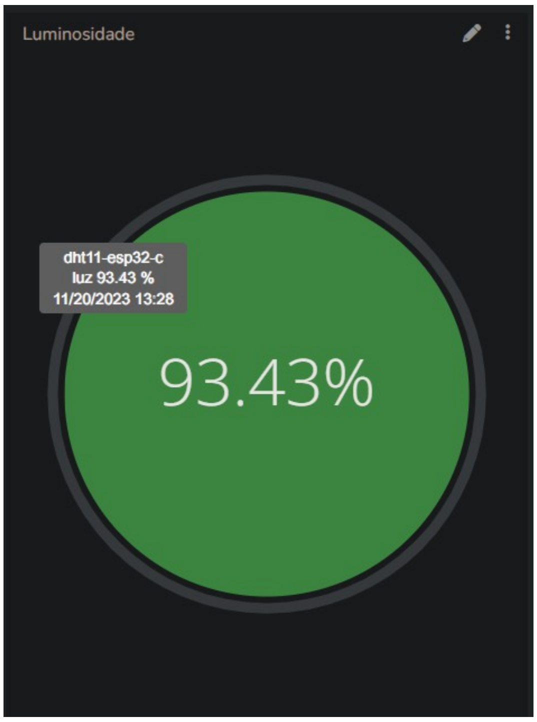
Precipitation: 74%
Humidity: 57%
Wind: 13 km/h

Temperature | Precipitation | Wind



dht11-esp32-c
luz 56.19 %
11/20/2023 14:38

Measuring brightness with the 12mm LDR sensor. Result with cloudy day (20/11/2023), representing 43.81% of brightness in contact with luminosity.

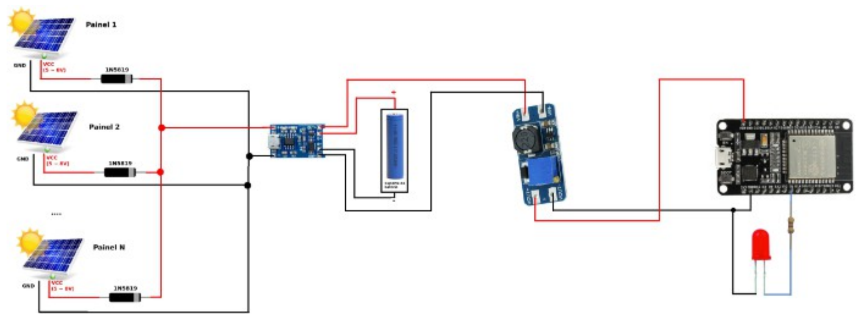


dht11-esp32-c
luz 93.43 %
11/20/2023 13:28

In this case, the percentage represents 93.43% darkness or 6.57% brightness by preventing light from coming into direct contact with the sensor.

- ## The prototype

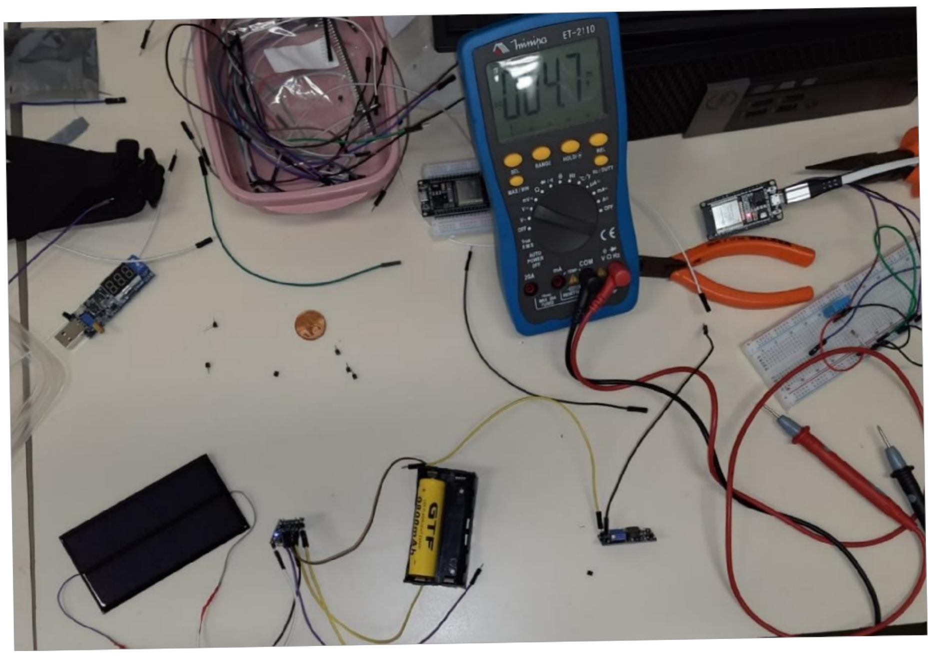
To supply the necessary voltage (4-12V) to the ESP32 microcontroller, an off-grid system is used, consisting of a 6V, 1W solar panel, a TP4056 battery charger module, a battery of 9800 mAh e a step-up step-up voltage which should supply approximately 5V to the ESP32.



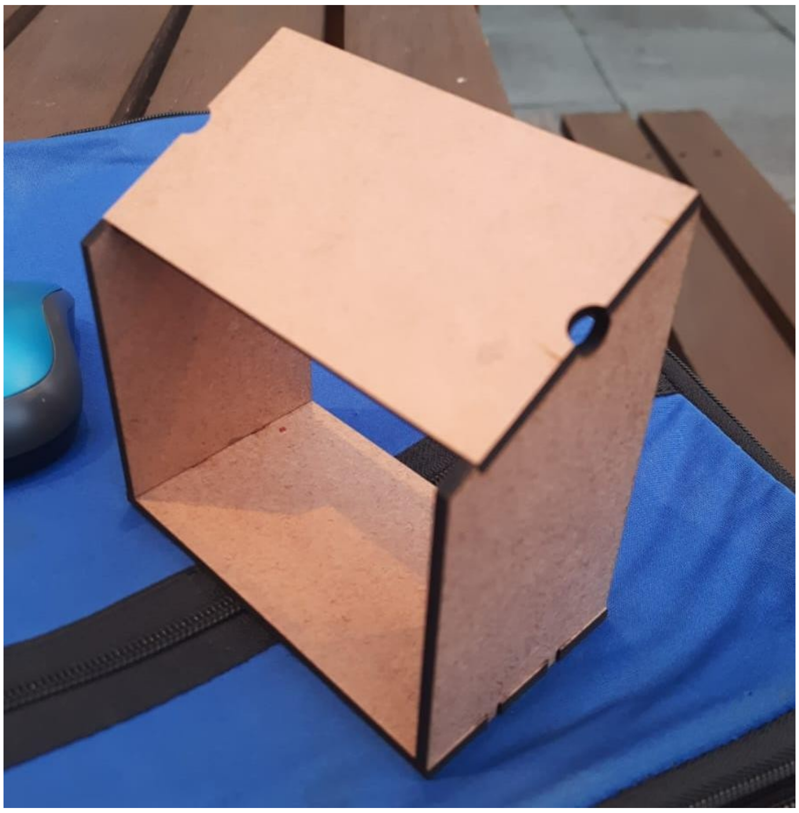
With the exception of not using 3 solar panels (only 1 used) and 3 rectifier diodes (none used), the rest of the circuit remains faithful to ours. Basically, the solar panel transforms solar energy into electrical energy. This electrical energy is then applied to the TP4056 lithium battery charger module, which will charge the Li-Ion battery connected to it which, in turn, feeds an adjustable Step Up DC boost converter, transforming the battery voltage (which can go up to 3.7V without being charged or 4.2V when charged) into 5.0V. This generated voltage is finally applied to the ESP32 (Vin input of the ESP32 Bluetooth WiFi Module), making it work.

The step-up adjustable DC booster converter didn't behave as desired, as it didn't adjust the output voltage (2-28V). As a result, the circuit would need an amplifier at the output to generate more than 4V (preferably 4.5V+) to supply the ESP32. The problem was solved by purchasing another step-up device, thus arriving at the expected 5V output to power the ESP32.

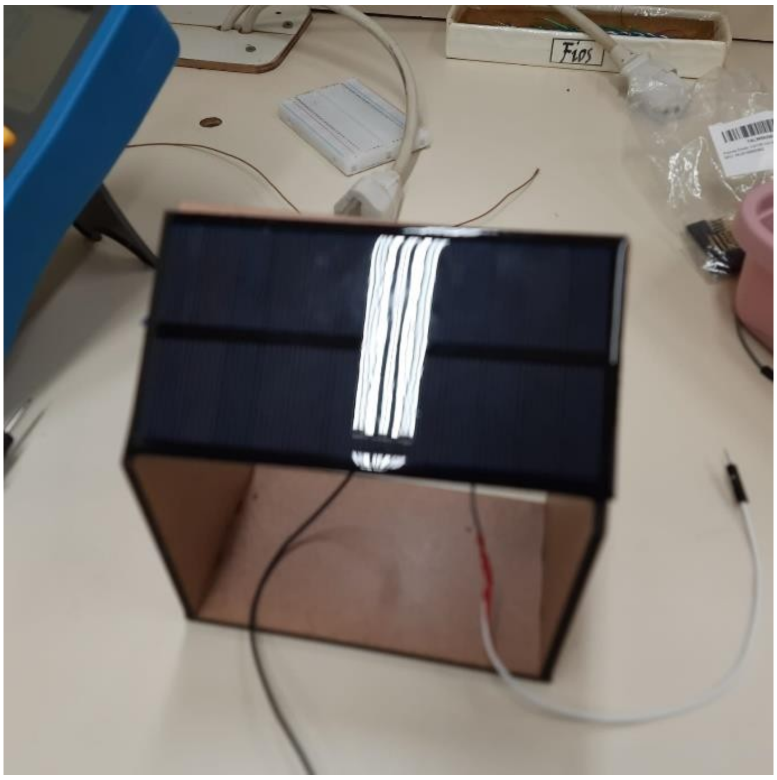
Thus, the circuit, which is still being tested with new physical components, looks like this:



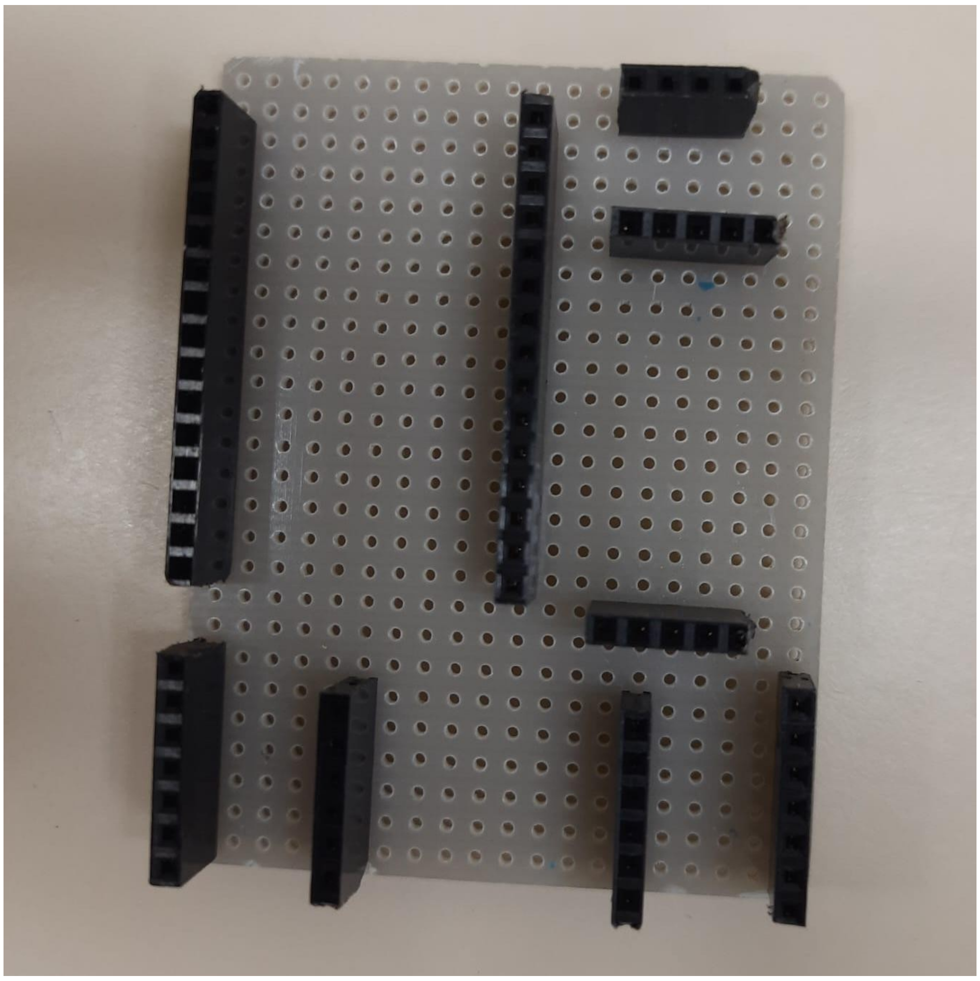
After adjusting each item separately, it was time to assemble them into a single prototype. The .dwg was created by the team and laser-cut by the person in charge of printing and cutting at PUC-Campinas, as follows:



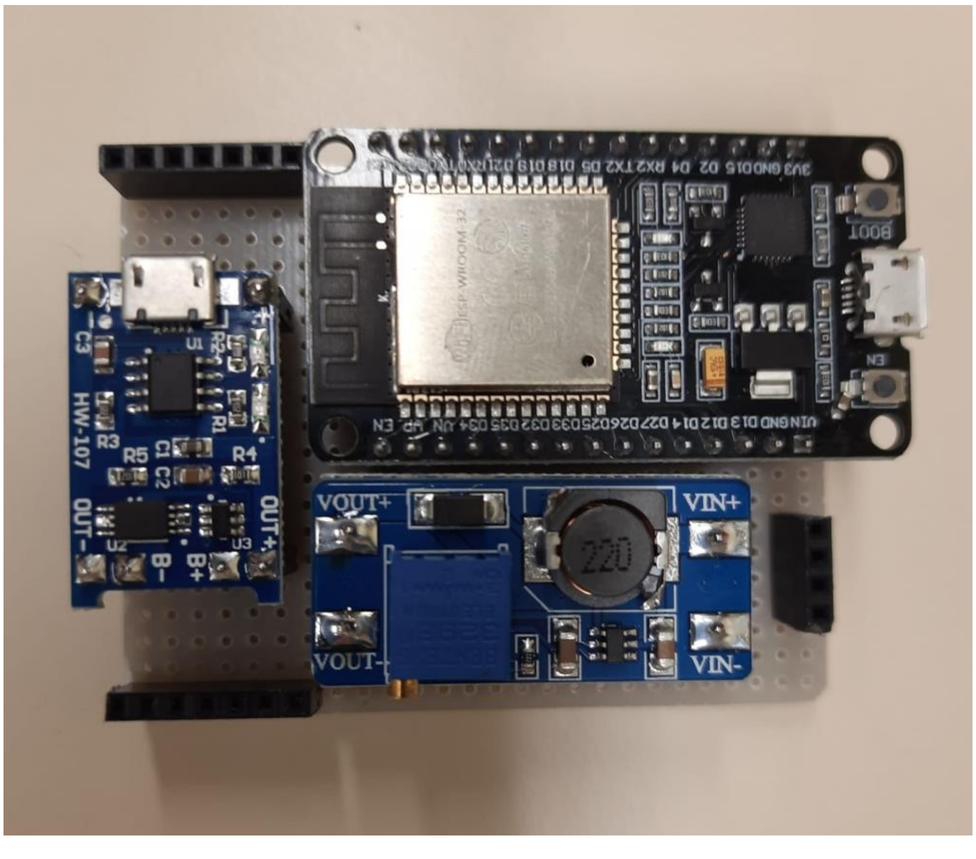
Both holes are for the polarized wires from the solar panel to be passed through and connected to the circuit as shown in the image below:



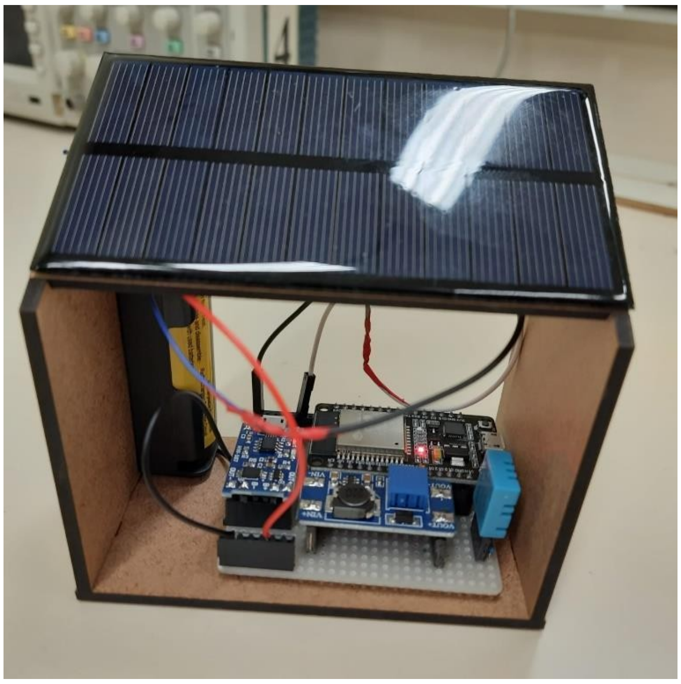
So, using a phenolite plate, the necessary solders were made so that the components could be plugged in:



Components connected to each other:

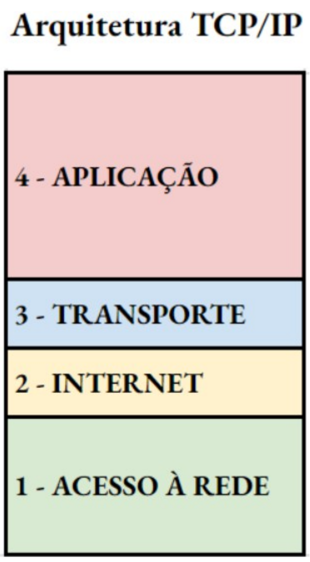


Complete prototype:



- ## The network (connectivity/protocols/topology)

Almost invisibly, and in the context of the project involving the ESP32 and the connection to the cloud platform via Wi-Fi, the protocols used conform to the OSI and TCP/IP models. The model used in this work follows the TCP/IP Architecture model, as in the image below:



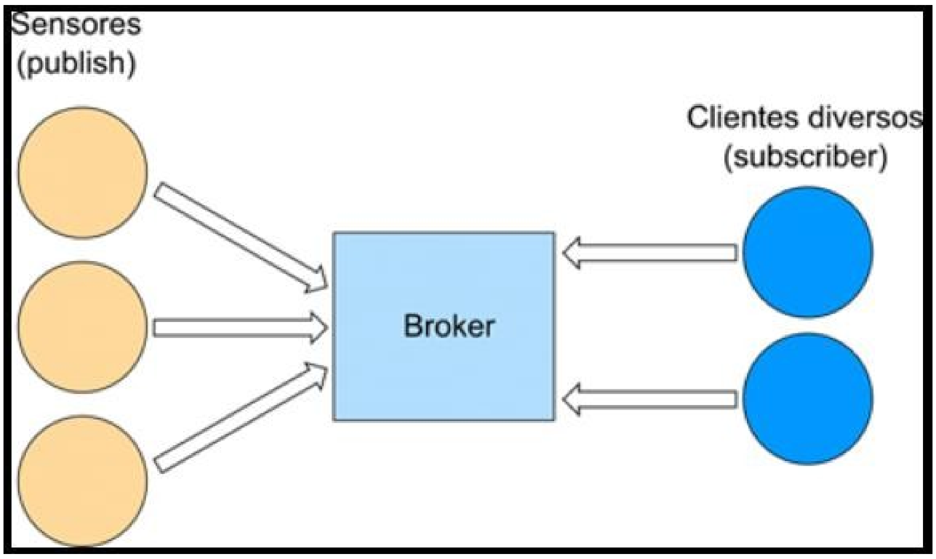
We then start with the data link layer, which plays a certain role in building the ESP32's network interface. For wired networks, the Ethernet protocol is used, while wireless networks use the Wi-Fi protocol, specifically the IEEE 802.11 standards. These protocols make it possible to connect the ESP32 to the local network (such as the one at home or at PUC-Campinas), allowing data to be transmitted and received.

Moving on to the network layer, there is the famous Internet Protocol

(IP), which is essential for routing data packets. This layer is responsible for directing data from its source to its destination on the network. Depending on the configuration and requirements of the project, the ESP32 can be configured to use the IP protocol in the form of IPv4 or IPv6. IPv6 was used as the test home's local network, even though this is a simpler project with less scalability for the time being. IPv4 or version 4 of the IP protocol could also be used for this application, as there are fewer IoT devices connected.

The transport layer, located next, incorporates the Transmission Control Protocol (TCP) - reliable transmission of data, organizing it into packets and ensuring that these packets are delivered in the correct order, without loss or corruption - unlike UDP.

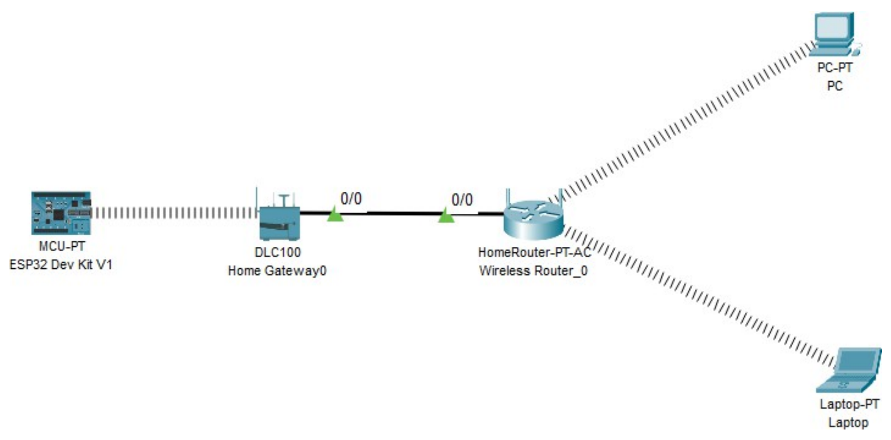
At the application layer, there are specific protocols for Internet of Things (IoT) applications such as MQTT (Message Queuing Telemetry Transport) and HTTP (Hypertext Transfer Protocol). MQTT is often adopted in IoT projects due to its efficiency and lightness, while HTTP is more common in web applications. Here, we use MQTT as the standard, which operates under the TCP/IP model and not the OSI (Open Systems Interconnection) model, which is important to mention even though they are not dependent on each other, as they are often confused.



The differences between them are that, while the OSI model uses seven layers to define data transmission in a network, the TCP/IP model employs four layers for communication on the Internet. One notable difference lies in the perception layer, equivalent to the physical layer in TCP/IP, which is responsible for converting data into a format that can be transmitted on an IoT network. The transport layer in the IoT corresponds to the middleware layer, facilitating communication between devices such as the ESP32, differing from the transport layer in TCP/IP, which guarantees reliable data transmission. In terms of specific IoT protocols, IEEE 802.15.4 stands out as a standard for LR-WPAN, enabling efficient data transmission in industrial environments, such as factories, where sensors can operate simultaneously on 16 different channels, contributing to reliable real-time data collection

In this case, as we used the Ubidots platform, MQTT, implemented over TCP/IP, was the protocol used as a standard to help send sensor data to the cloud platform. To send temperature and humidity data via specific topics, an MQTT server (Broker) facilitated communication between the ESP32s and Ubidots, forwarding the messages to the platform. Ubidots, acting as an MQTT subscriber, received and processed the data in real time.

Below is the model of the IoT-based network system developed by the authors, using Cisco Packet Tracer software to support the development and understanding of the necessary connections.



In the image above, the "HomeRouter-PT-AC" wireless router and the "MCU- PT ESP32 Dev Kit V1" establish a Wi-Fi connection with each other via the "DLC100" Gateway, which communicates between them by converting protocols, managing IP addresses etc. The wireless router then provides wireless connectivity for both the "PC-PT" and the "Laptop-PC". In this way, the ESP32 is integrated into the local network via this structure, allowing wireless communication with the end devices, the PC and the Laptop, via the wireless router, which achieves the goal of creating an interconnected network that facilitates data transmission between the ESP32 and the end devices on the local network, i.e. our computer.

Roughly speaking, this whole setup involves a functional structure using the MQTT protocol to facilitate communication between the ESP32 and the Ubidots. The ESP32 acts as a central node, connecting via Wi-Fi and using MQTT to send specific data such as temperature and humidity via designated topics. This data is transmitted to Ubidots via the Broker, which manages the transmission. The topology adopted is a star, where the ESP32 is at the center and is connected by MQTT. This architecture results in the effective integration of the ESP32 into the local network, establishing an interconnected mesh that enables wireless communication between the ESP32 and the end devices, represented by the PC or Laptop according to the image above in the Cisco software. This makes it possible to send data and adjust it according to our specific needs, ultimately guaranteeing low energy consumption, scalability potential and low latency.

References

1. Bertoleti, Pedro. **Learn how to power your ESP32 with a solar-charged battery.** Available at: Powering an ESP32 with a battery: Learn how to do it - MakerHero. Accessed on: October 22, 2023.

2. H. Tanimun, M. Sultana Shompa, M. Abdur Rahman, M. Abu Rasel, M. Rahim Hossain Apu and M. Arifur Rahman, **"IoT Based Solar Power Monitoring & Data Logger System."** 2022 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), Naya Raipur, India, 2022, pp. 182-187, doi: 10.1109/WIECON- ECE57977.2022.10150511.

3. NAGAR, Nehru. **ESP32 Interfacing With LDR Sensor**. Maharashtra: Prateeks, 2022. Available at: https://www.prateeks.in/2022/09/esp32-interfacing-with-ldr-sensor.html. Accessed on: October 1, 2023.

4. SANTOS, Sara. **ESP32 Deep Sleep with Arduino IDE and Wake Up Sources**. Lisboa: Random Nerd Tutorials, 2019. Available at: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/. Accessed on: 01 Oct. 2023.

5. RODRIGUEZ, Gabriel. **Ubidots Basics: Devices, Variables, Dashboards, and Alerts**. Colombia: Ubidots, 2013. Available at: https://help.ubidots.com/en/articles/854333-ubidots-basics-devices- variables-dashboards-and-alerts. Accessed on: September 30, 2023.