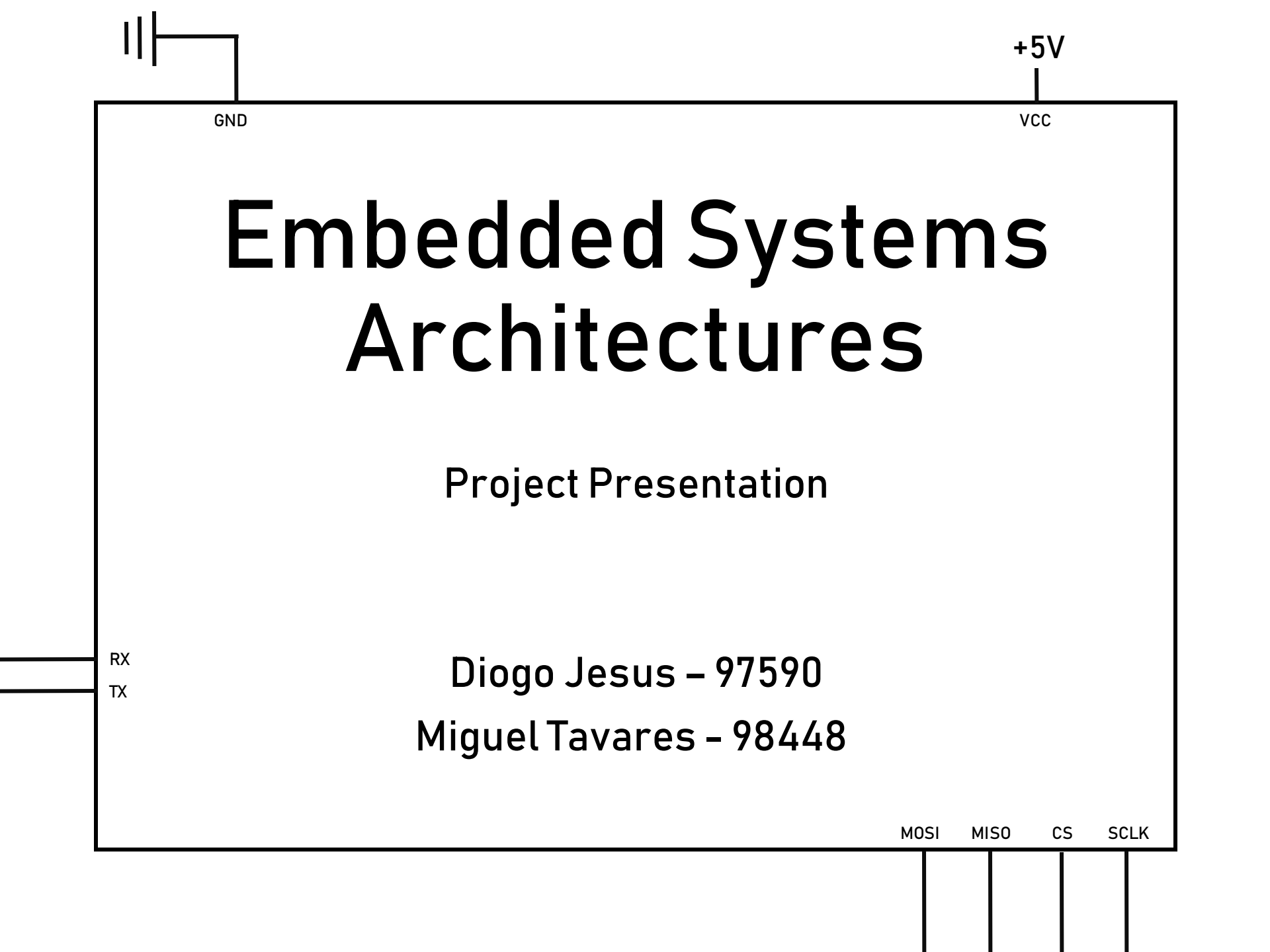


Embedded Systems Architectures

Project Presentation

Diogo Jesus – 97590

Miguel Tavares - 98448

Capacitive Soil Moisture Sensor v1.2

The Capacitive Soil Moisture Sensor v1.2 operates based on capacitance measurement to determine soil moisture levels. Indirectly evaluates humidity by detecting changes in capacitance caused by variations in the dielectric properties of the soil. The sensor measures the ions dissolved in the moisture rather than directly measuring soil moisture.

The sensor in our possession did not work correctly due to various factors. It does not have a voltage regulator; the timer chip is different; Has a missing resistor connection.

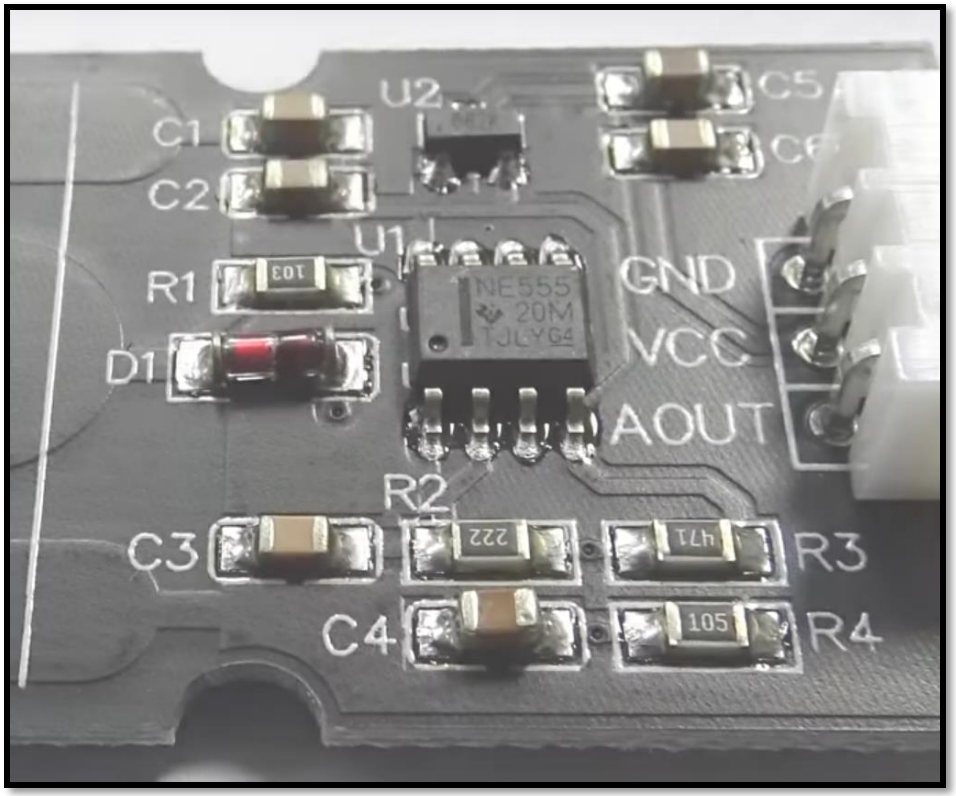


Figure 1: Non-faulty sensor

Project Overview

- Read the current soil moisture detected by the sensor, convert it to percentage and make it available in a dashboard.
- 2 options of readings are available. Timed readings are enabled by default and updates the dashboard if the value changed. Manual readings triggered by the user are also available to get the most recent moisture level.
- Store the detected moistures on the EEPROM and make it available to the user.
- Send a notification to user when the moisture level reaches a certain value.
- Create a graph on the dashboard with the history of values read by the sensor and the respective timestamps.
- 2 options of watering are also available. Automatic watering checks the value read by the sensor each time it is triggered. If the moisture value is less than a predefined value, a water pump is enabled a specific amount of time, calculated based on the moisture. Manual watering is also available on the dashboard. The user defines the watering time.
- The automatic watering can be enabled/disabled.
- The manual detection and the automatic watering can be triggered/set using the terminal. The character 'r' or 'w', respectively, must be sent to the terminal.
- The history of moisture detections can also be checked on terminal, by sending the character 'h'.
- OTA firmware updates are available, using the rainmaker dashboard.

Terminal Interaction (Forgot to be shown in presentation)

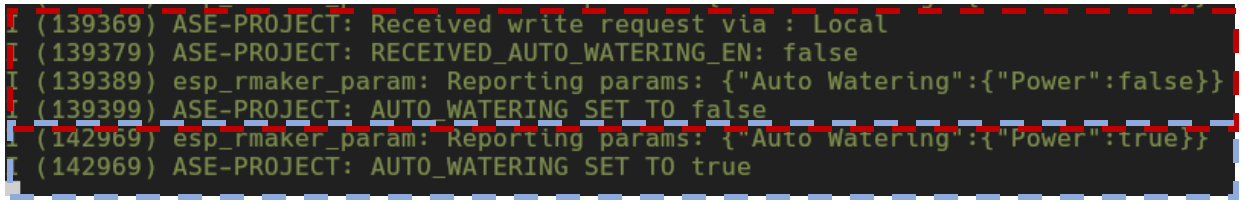


Figure 2: example of auto watering being set via the application (in red; Note: request was received via local because the devices are reachable on WLAN.), and via the terminal (in blue).

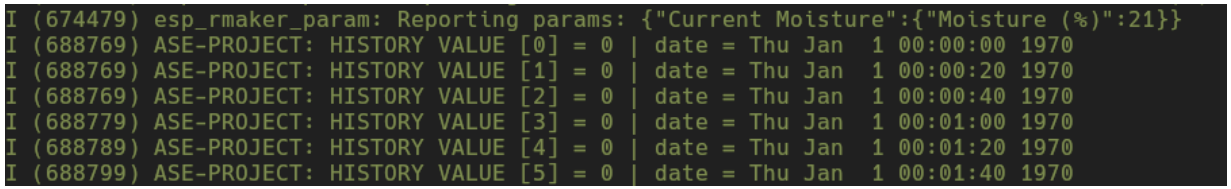


Figure 3: The history could not be shown correctly due to the delivery of the EEPROM to the professor on the day of the presentation. In normal execution, the value of the moisture and the current date is stored. In the print only 0's are read from the pins because of the missing EEPROM. However, the output of the 'h' char on the terminal can be observed.

OTA Pt.1 (Forgot to be shown in presentation)

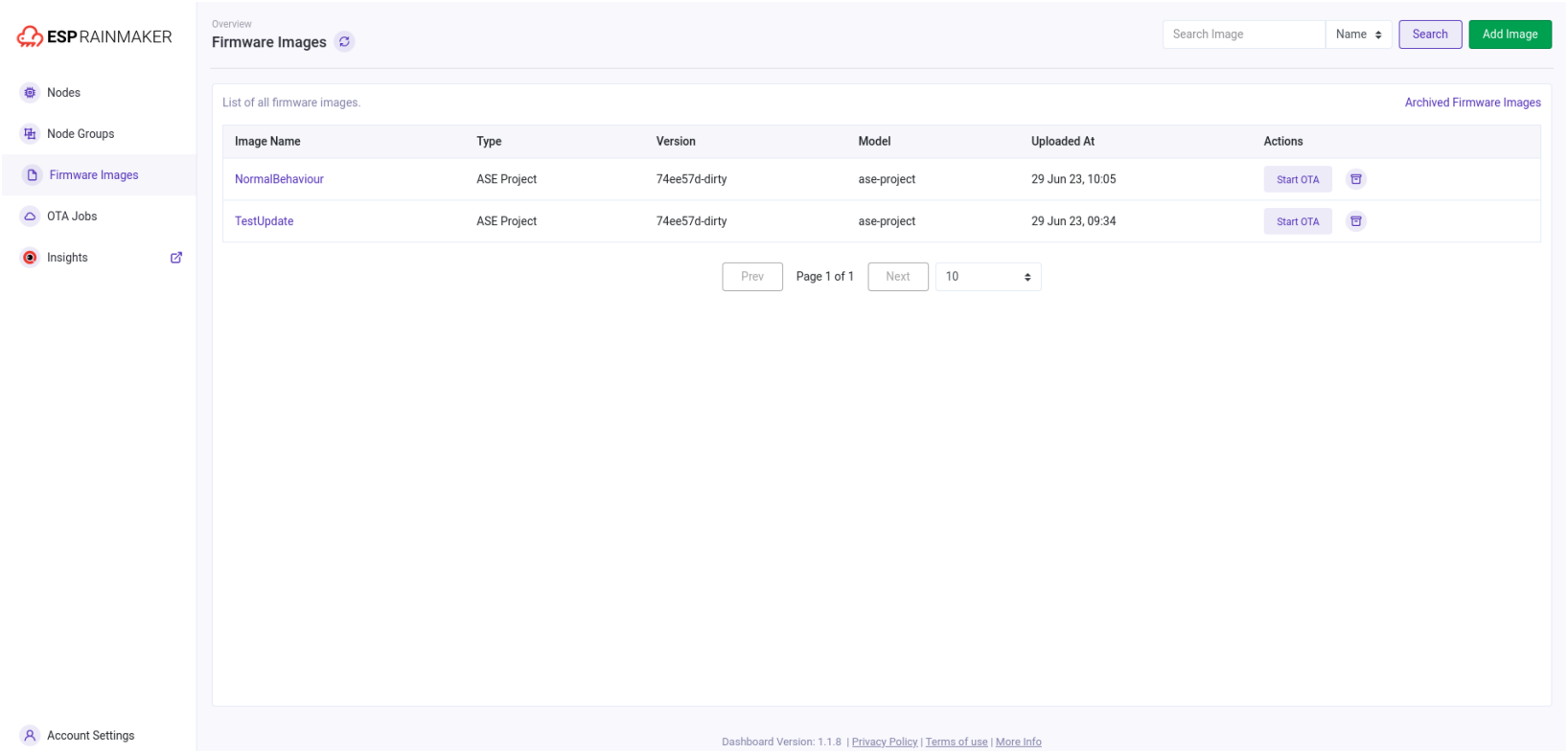


Figure 4: Upload the firmware images to the Rainmaker cloud.

OTA Pt.2 (Forgot to be shown in presentation)

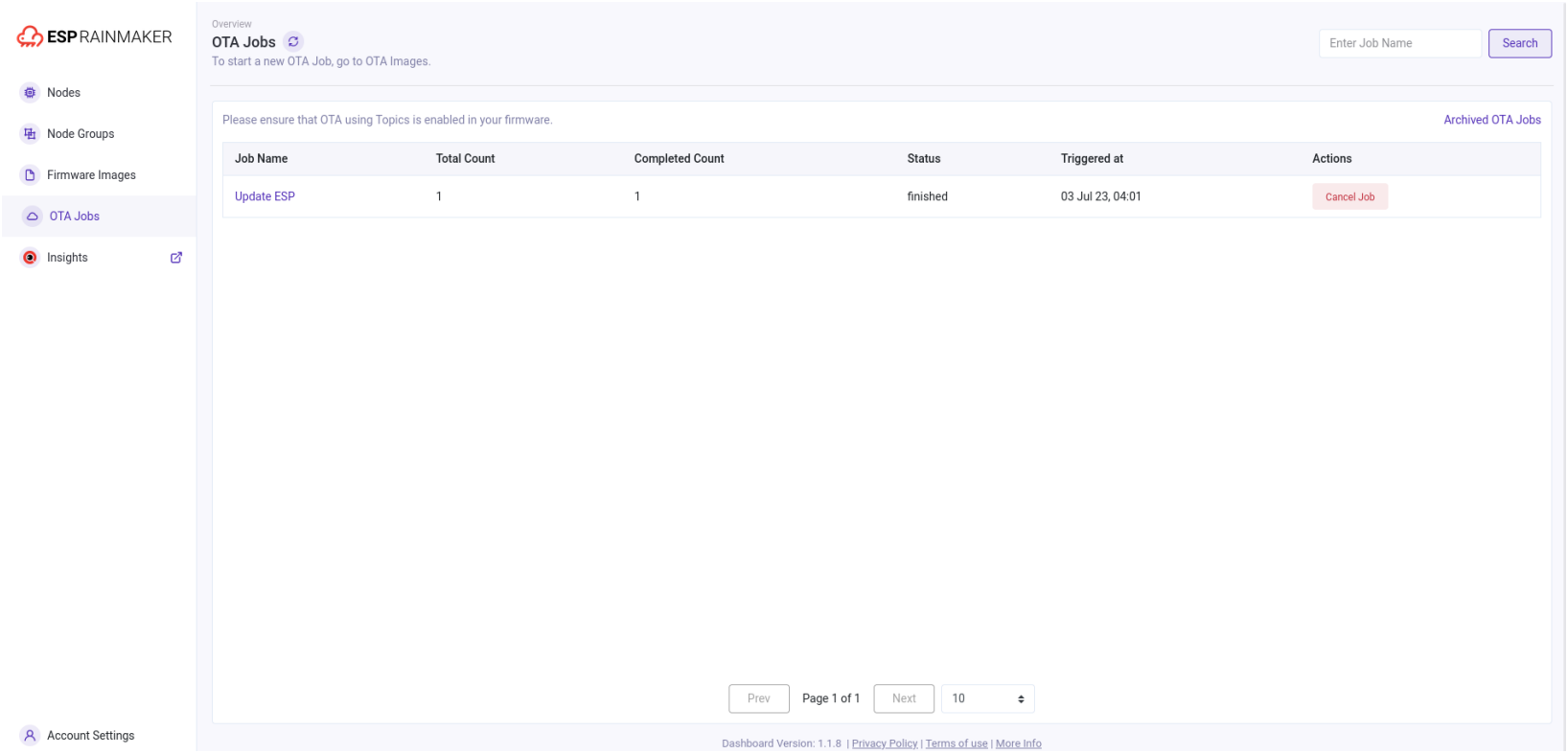


Figure 5: OTA Job created to update ESP.

OTA Pt.2 (Forgot to be shown in presentation)

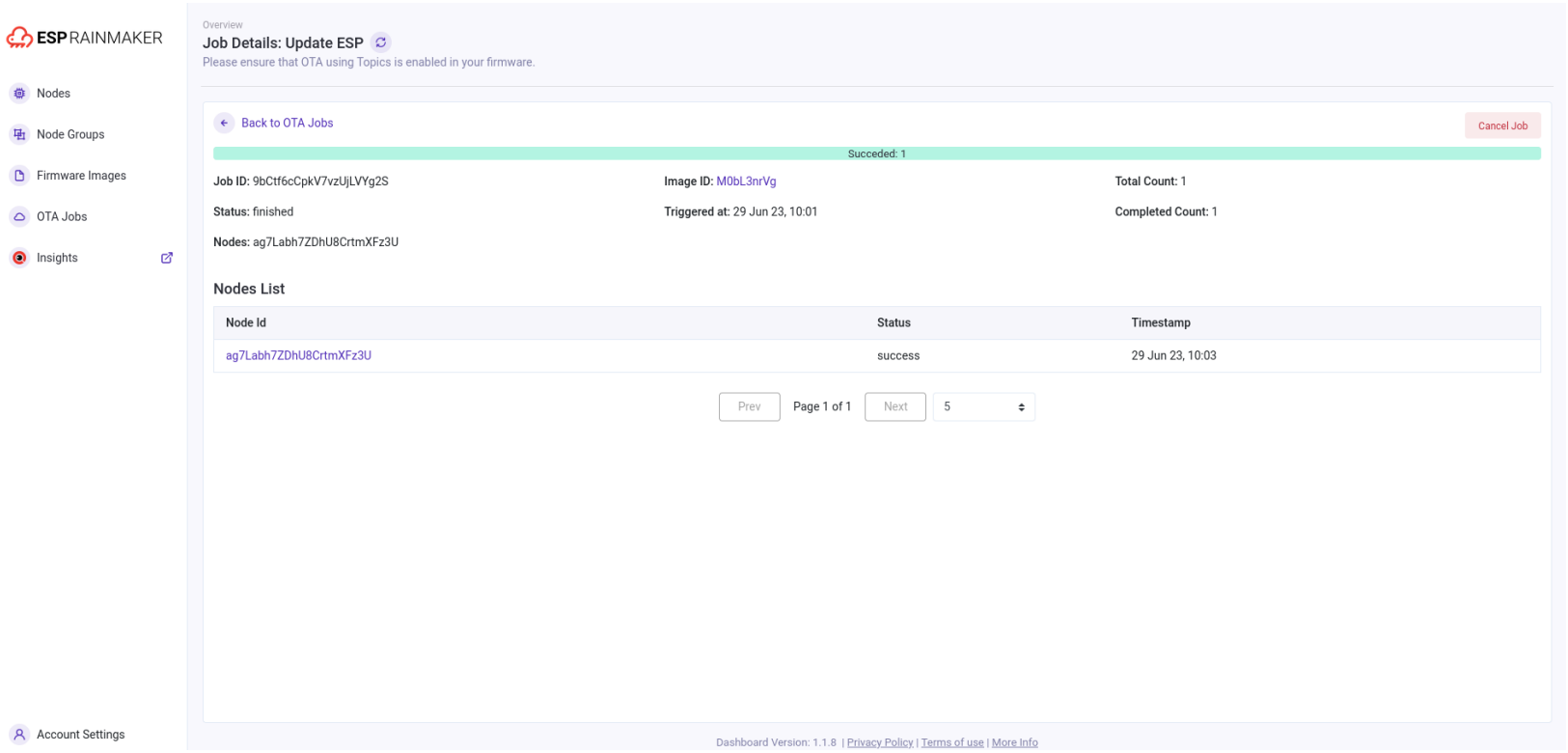


Figure 6: OTA Job details.

ESP-Rainmaker

End-to-end solution, developed by Espressif, to enable remote control and monitoring of ESP32 based products.

It is able to claim devices and associate them with the user account in the service.

It has a agent module that is available to develop firmware that works with this solution.

Has a cloud service available to offer remote connectivity to the devices (OTA updates).

Disposes of a phone application to be used by clients for remote access, or as a dashboard.

It is possible to define own devices and parameters in the firmware.

The phone applications dynamically render the UI according to the device information.

The agent module works on top of esp-idf.

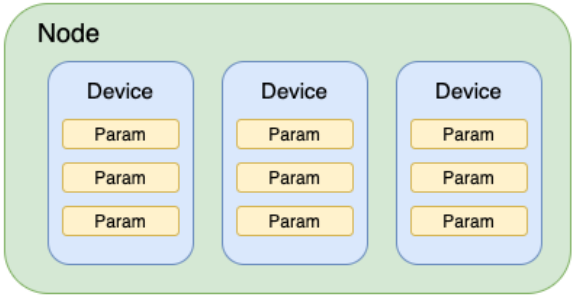
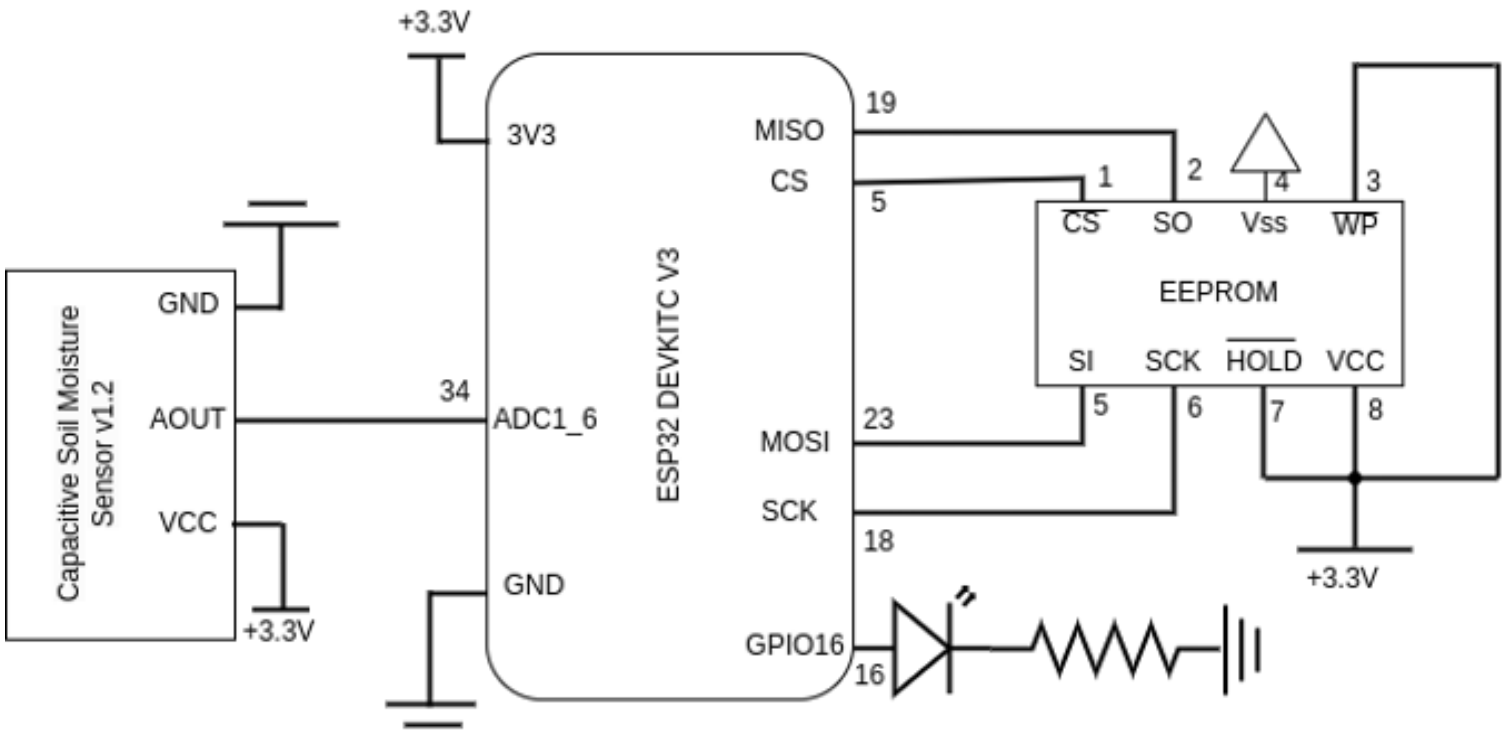
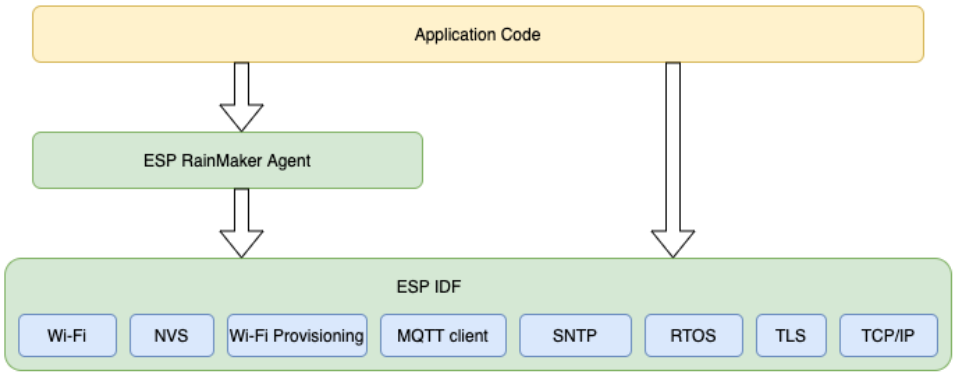


Figure 7: Node structure

Hardware Architecture

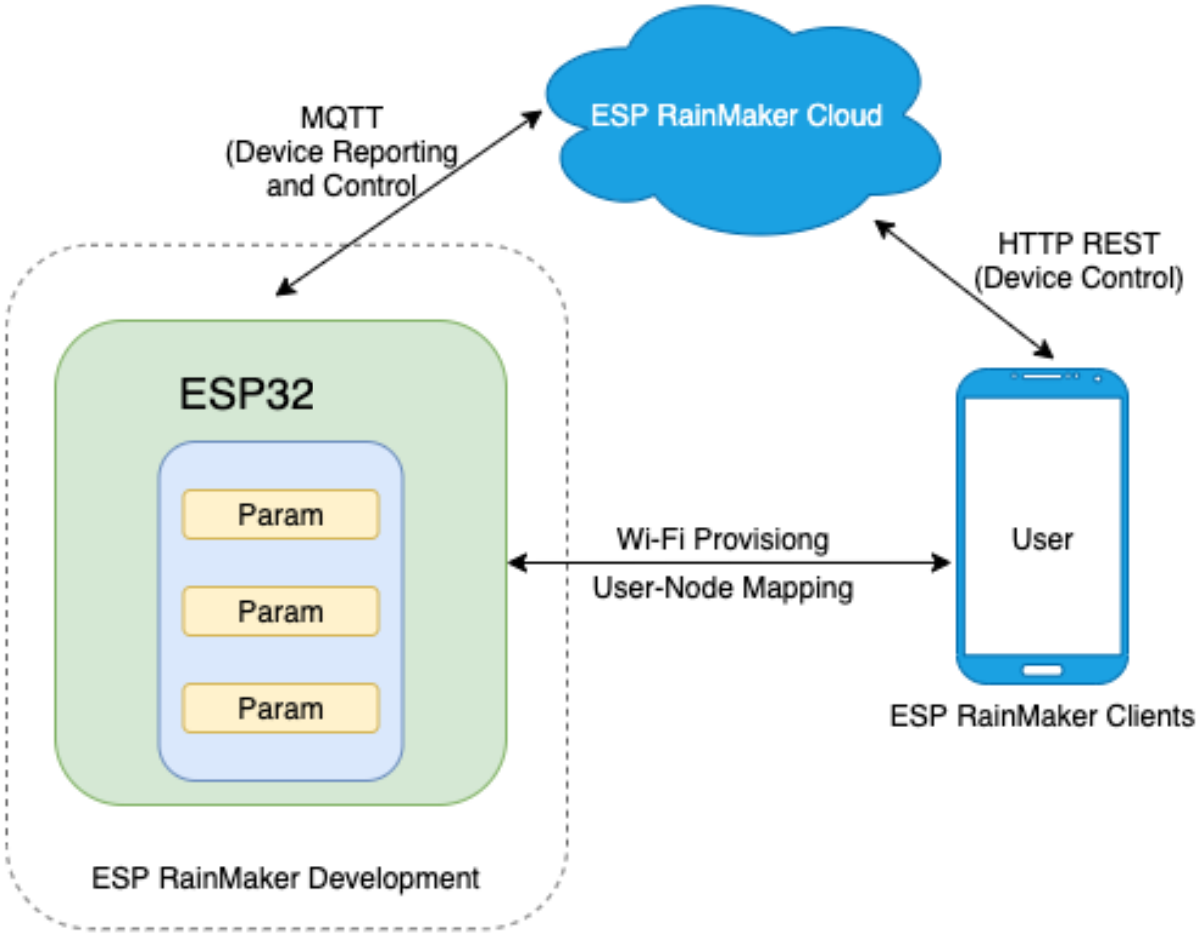


Software Architecture




```
.
├── CMakeLists.txt
├── main
│   ├── app_adc.c
│   ├── app_adc.h
│   ├── app_eeprom.c
│   ├── app_eeprom.h
│   ├── app_gptimer.c
│   ├── app_gptimer.h
│   ├── app_main.c
│   ├── app_pwm.c
│   ├── app_pwm.h
│   ├── app_rmaker.c
│   ├── app_rmaker.h
│   ├── CMakeLists.txt
│   ├── spi_25LC040A_eeprom.c
│   └── spi_25LC040A_eeprom.h
├── partitions.csv
└── sdkconfig.defaults
```

Communication Architecture



Checklist

A aplicação a executar no kit ESP32DevKitC deve ser desenvolvida em C/C++ e tirar partido do FreeRTOS	☑
Devem ser explorados os periféricos do ESP32 que fizerem sentido no contexto do projeto, incluindo aspetos de interrupções e DMA	☑
Os dados recolhidos do sensor e processados no ESP32 devem ser apresentados num dashboard remoto, sendo para tal necessária conectividade de rede (WiFi / BT)	☑
Deve ser disponibilizada uma ligação por Terminal; independente do dashboard remoto	☑
Devem ser exploradas as várias funcionalidades das ferramentas de desenvolvimento.	☑
Podem ser explorados os modos de baixo consumo energético do ESP32	✕
Podem ser suportadas atualizações remotas (Over-the-Air) do sistema	☑
Pode ser incluído algum tipo de atuador cuja utilização faça sentido com o sensor usado (de forma a criar um loop de controlo; ou que seja controlado através do dashboard)	½
Pode ser suportado um sistema de ficheiros para armazenar dados localmente	✕

Code Baseline

All the code was developed by us, with the help of the following examples:

- Analog to Digital Converter (ADC) Oneshot Mode Driver
- LED Control (LEDC) for PWM implementation
- https://github.com/espressif/esp-rainmaker/tree/master/examples
- Code developed in class for the use of EEPROM memory 25LC040

Documentation

- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html
- https://rainmaker.espressif.com/docs/get-started
- https://www.youtube.com/watch?v=IGP38bz-K48

Contribution

Diogo Jesus – 50%

Miguel Tavares – 50%