UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Smart Home

Implementing an automated window using ESP8266 and MQTT

Lucas de Camargo Souza

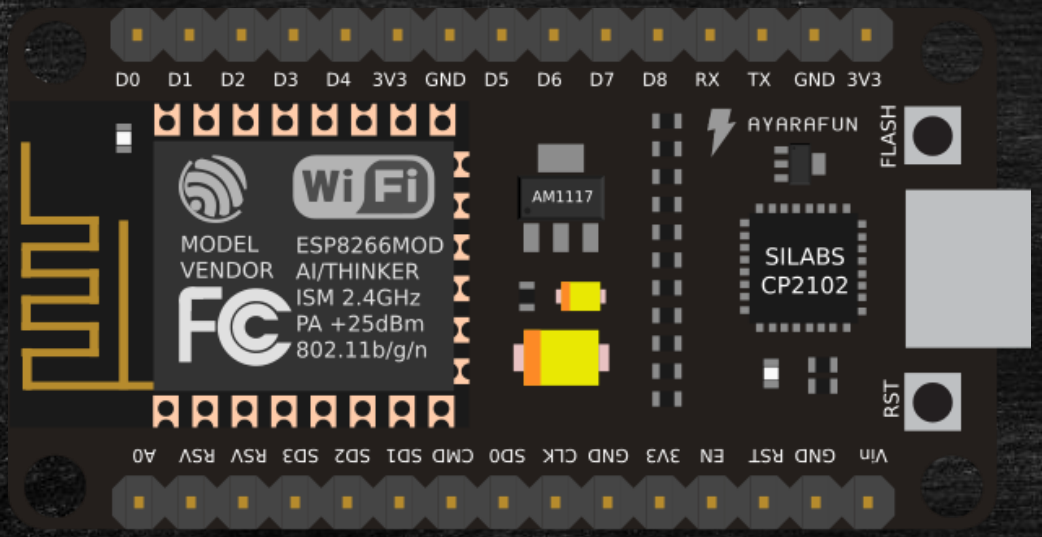
Smart Home

$77.3b

worldwide revenue of the Smart Home market in 2020, and is expected to rise to $175.7b by 2025.

Source: Statista Digital Market Outlook

Do it Yourself



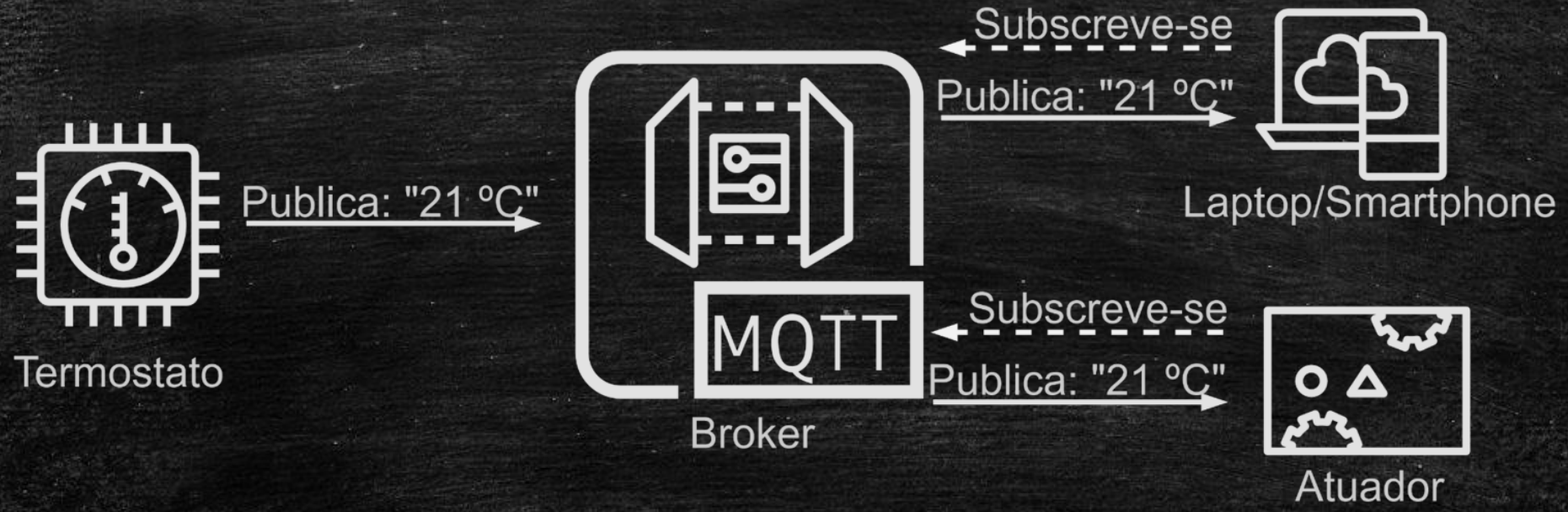
ESP8266

aws

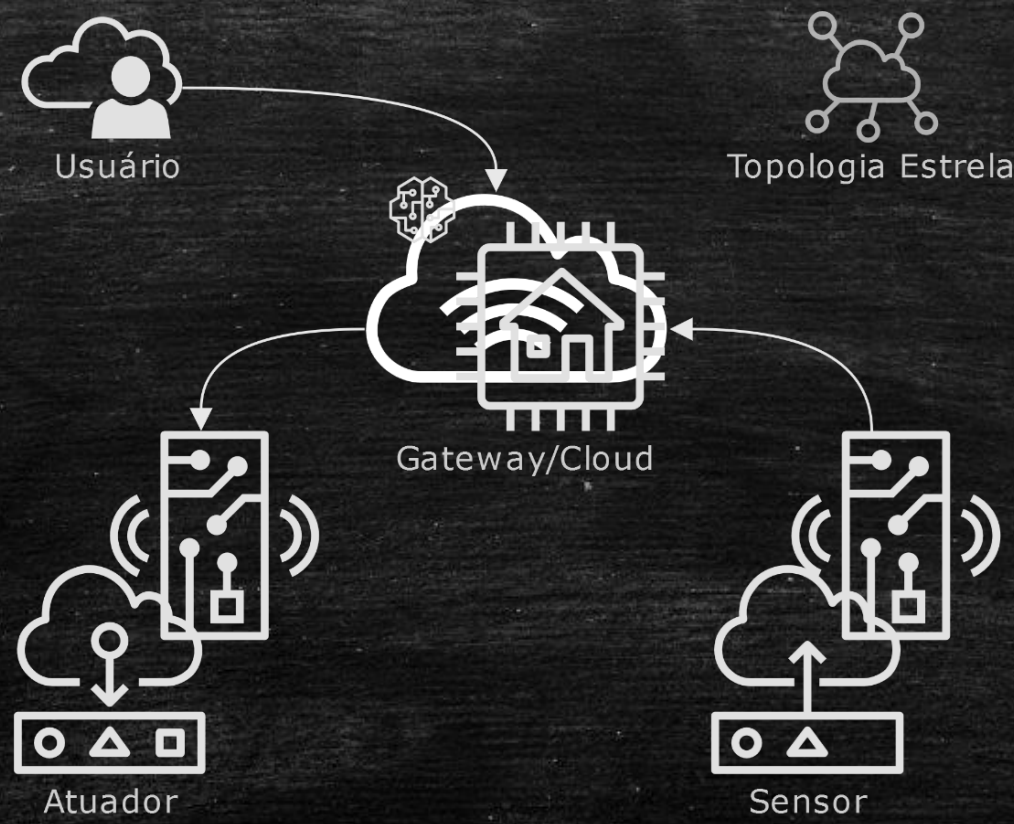
MQTT

ARDUINO

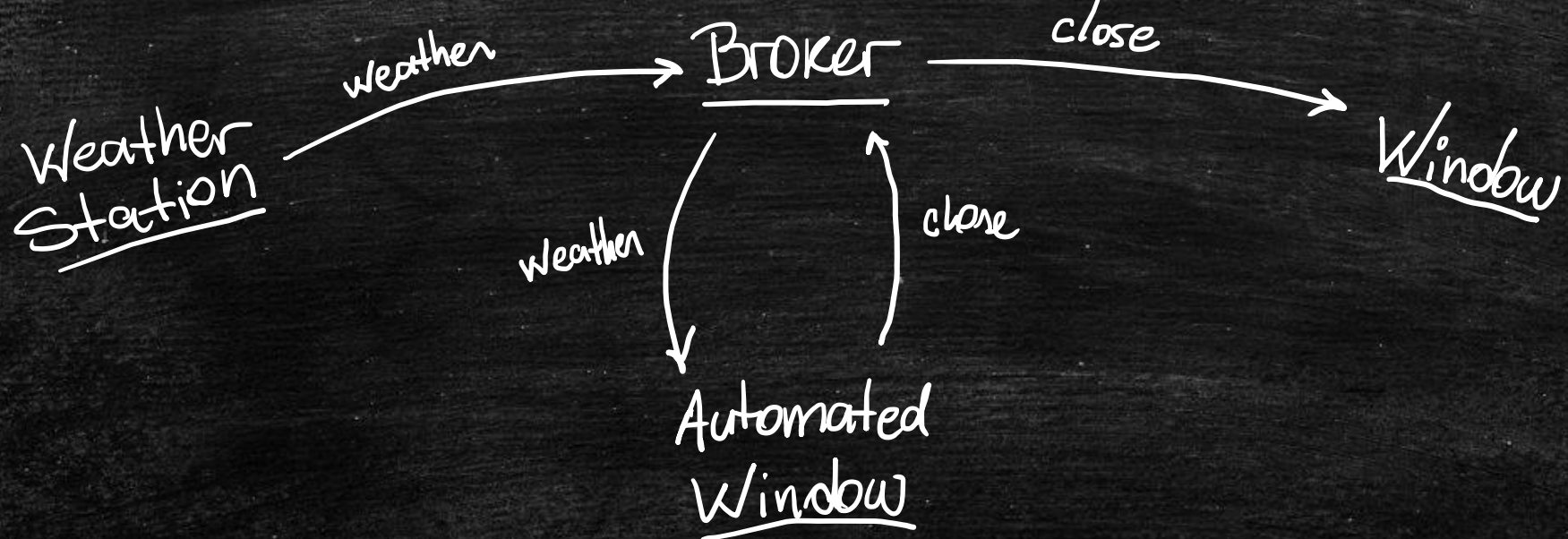
MQTT


Subscreve-se
Publica: "21 ºC"
Laptop/Smartphone
Publica: "21 ºC"
MQTT
Termostato
Broker
Subscreve-se
Publica: "21 ºC"
Atuador

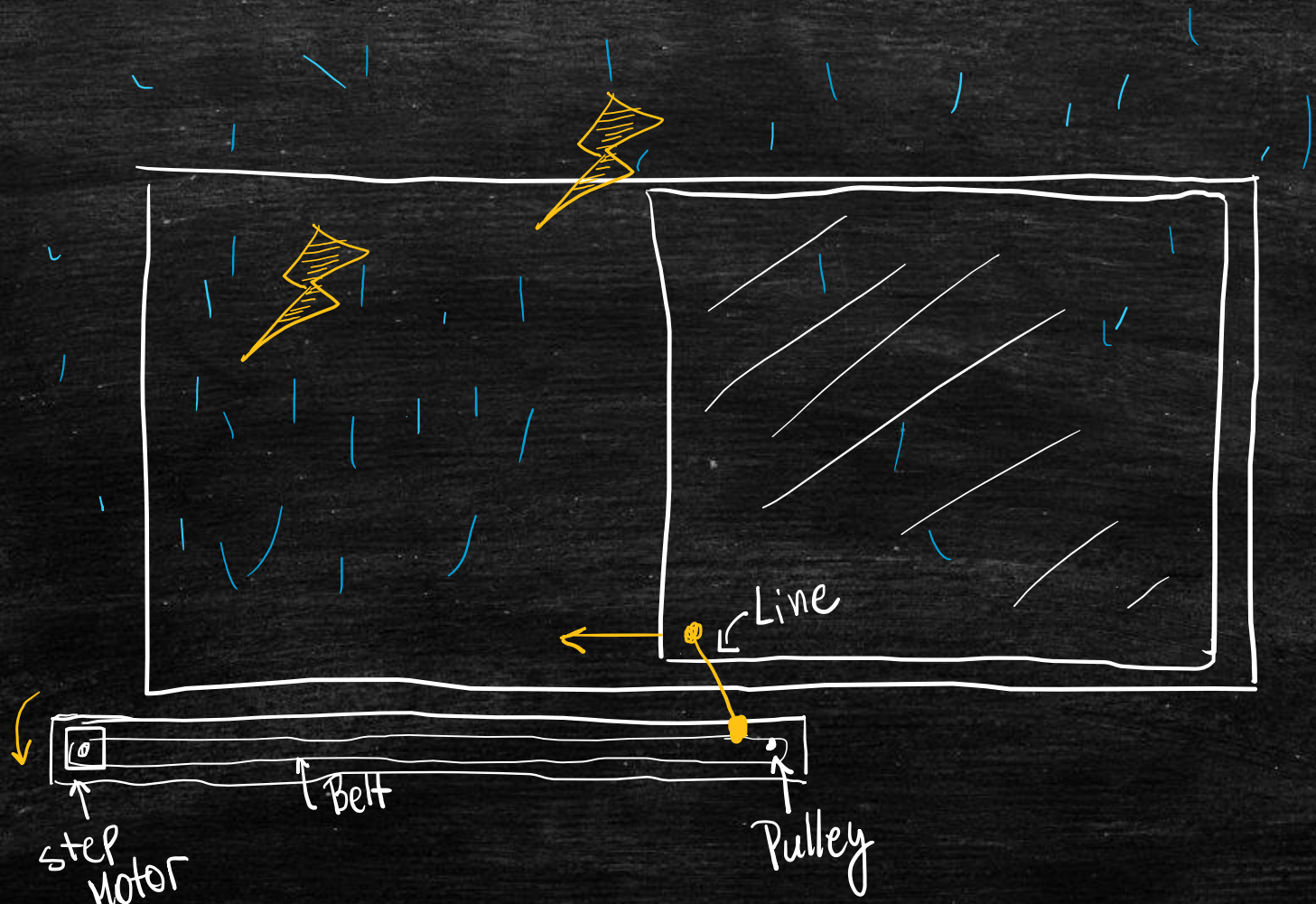
Architecture



Architecture

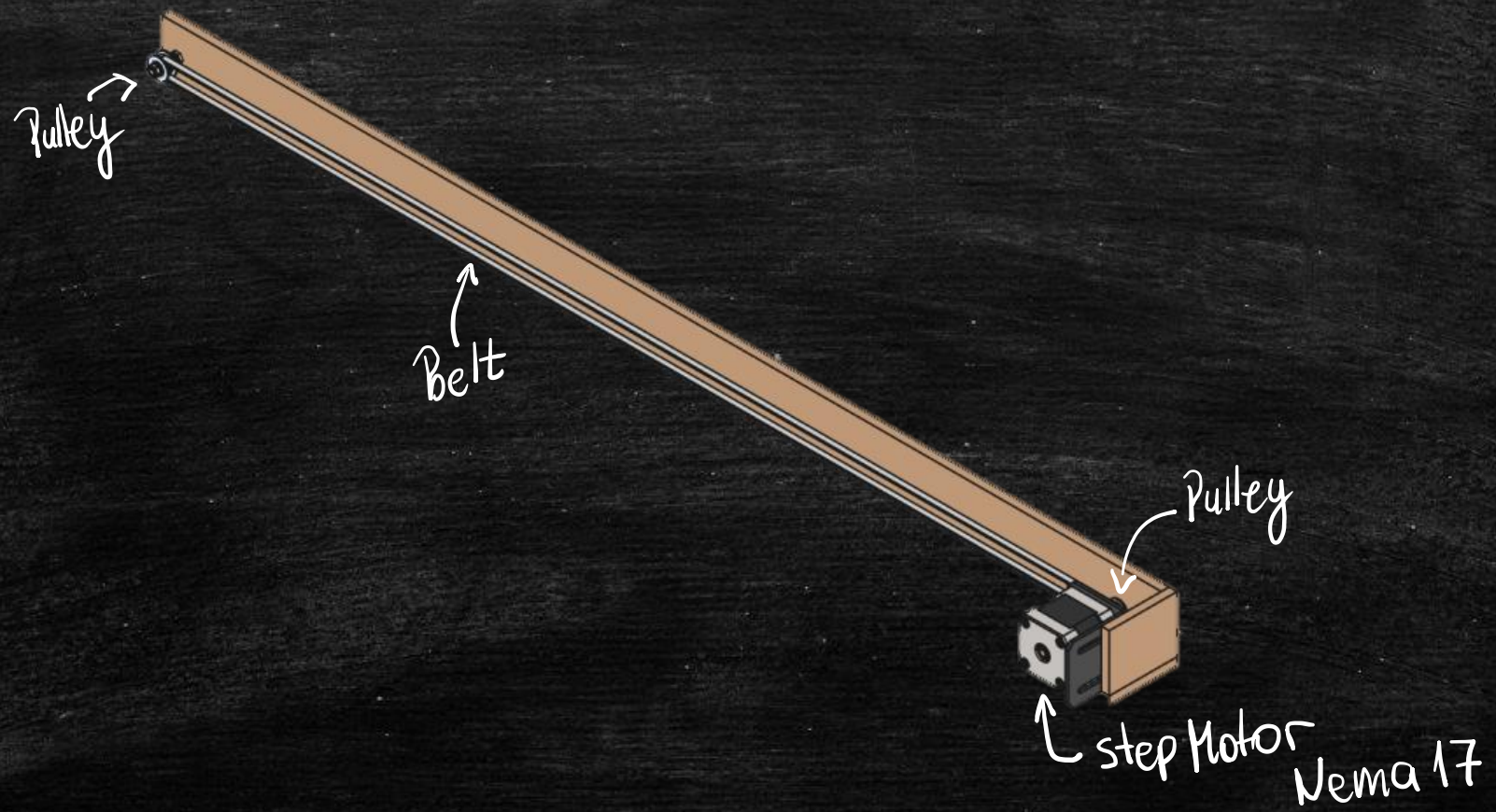


Window Automation



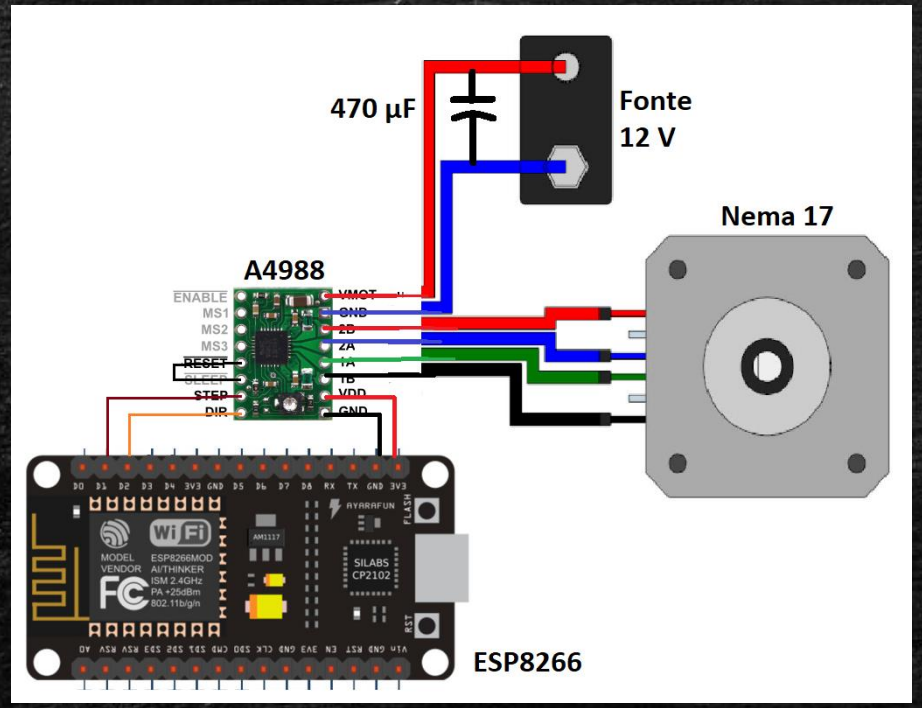
Window Automation

- The Actuator

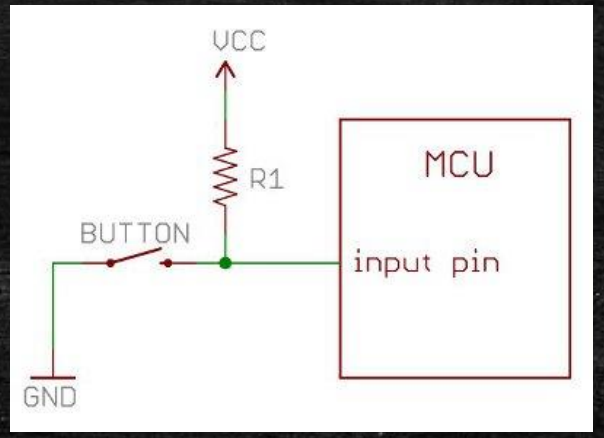


Window Automation

- The Actuator

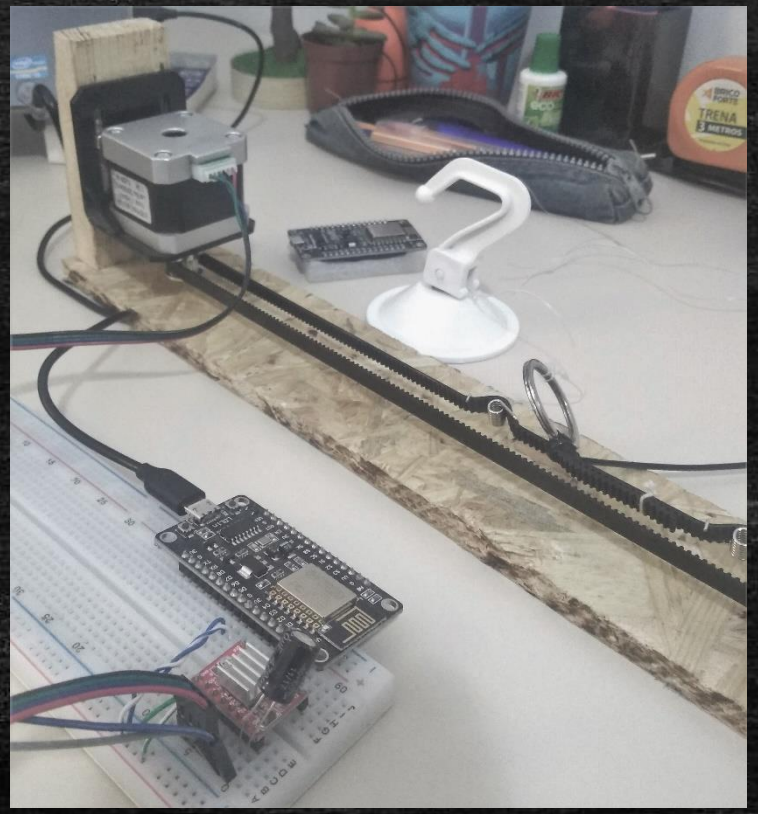


Limit Sensor



Window Automation

- The Actuator



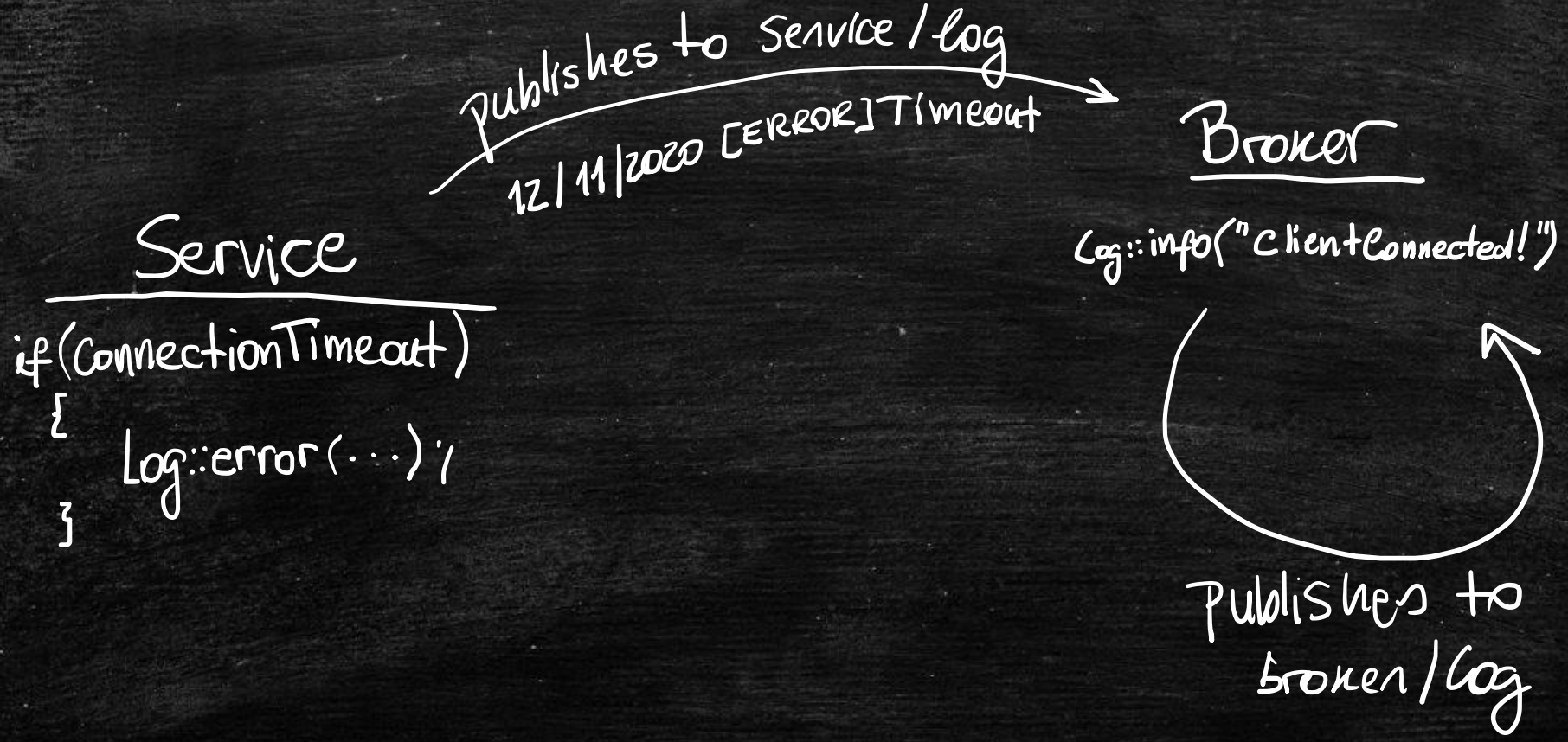
Let’s Start Coding!

- Common Dependencies

- All services run on a NodeMCU ESP8266 microcontroller, so we need to pre-configure our Arduino environment. I use the board config NodeMCU 1.0 (ESP-12E Module).

- Arduino Libraries:
  - NTPClient https://github.com/arduino-libraries/NTPClient
  - PubSubClient https://github.com/knolleary/pubsubclient
  - ArduinoJson https://arduinojson.org/

Logging Messages



class Logger

- **Definition**
  - `template<class T> class Logger;`
  - This is a **static class** implemented as an alternative to the conventional `Arduino.Serial` output method. This class encapsulates output operations such as printing to serial port and logging to the MQTT broker.
- **Usage**
  - You must redefine the class type passing the correct class template `T`. This template should be a valid MQTT Client class like `PubSubClient` or `uMQTTBroker`. Use `typedef` to simplify its definition at the global scope of your code:
    `typedef Logger<PubSubClient> Log;`

class Logger

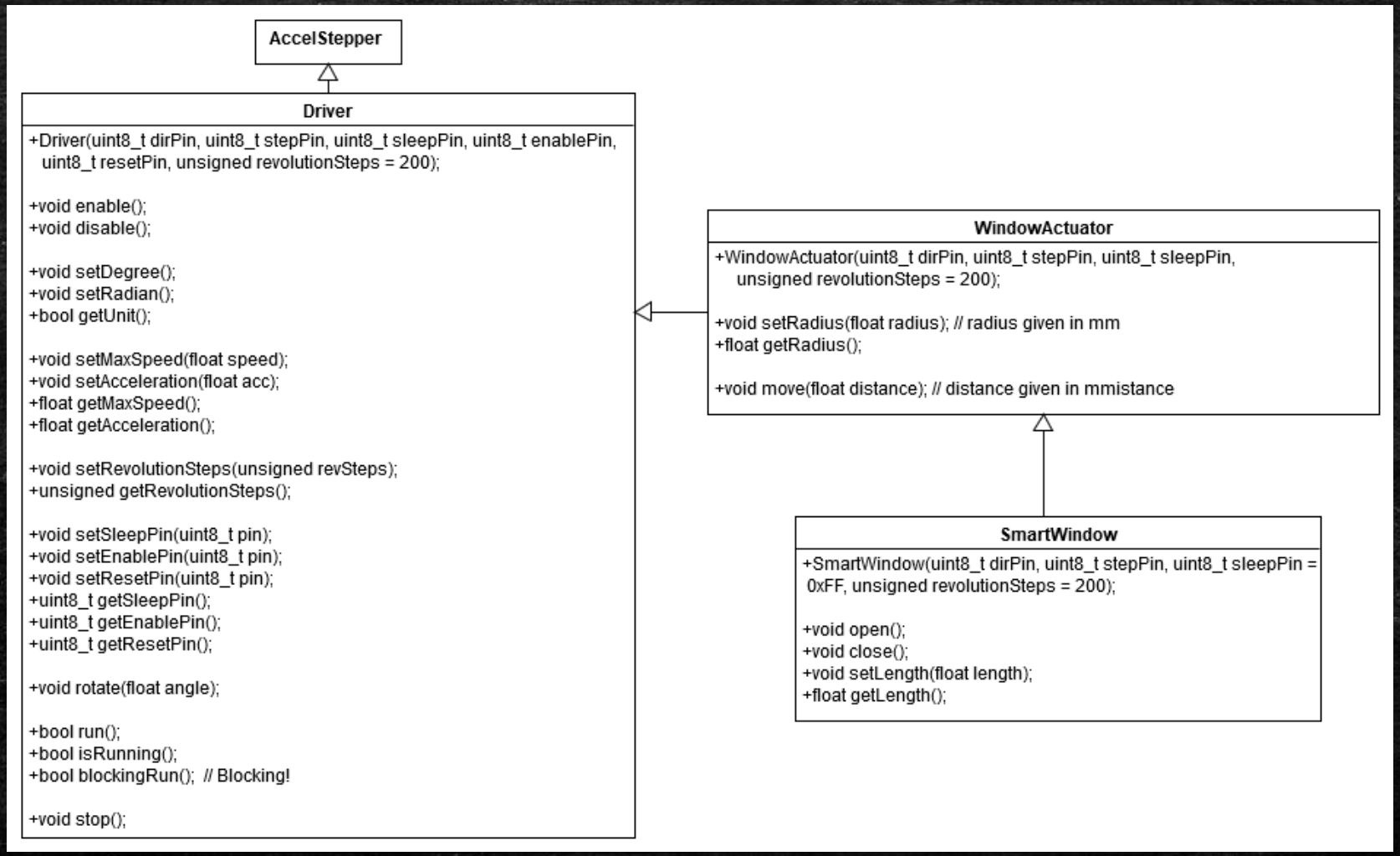
- **Macros**
  - LOG_LEVEL_SILENT
  - LOG_LEVEL_ERROR
  - LOG_LEVEL_WARNING (default)
  - LOG_LEVEL_INFO

- **Static Methods**
  - static void setSerial(Print * serial)
  - static void setMQTT(T * mqtt, String topic = "log")
  - static void setNTP(NTPClient * ntp)
  - static void setPrefix(String pref)
  - static void setLevel(unsigned level)
  - static unsigned getLevel()
  - static void error(String str)
  - static void warning(String str)
  - static void info(String str)

Smart Window

- This service was made for a window actuator that receives commands via MQTT to open, close or setting up operational parameters.
- It cooperates with another service called `AutomatedWindow` that bridges information coming from the weather station to the window through a decision maker that closes or opens the windows based on the external weather or forecast.
- It was based on the `AccelStepper` library:
  http://www.airspayce.com/mikem/arduino/AccelStepper/index.html

Smart Window



Smart Window

- **MQTT API**
  - **First note:** every `/read` topic receives as argument another topic where the response should be published to.
  - A base topic is always set for pre-configurations. Note that `SWALPHA01` will always be set as configuration topic. If more smart windows are to be connected, make sure to change `DEVICE_ID` under `definitions.h`.

Smart Window

- **MQTT API**

1. `SWALPHA01/topic/read`
   Returns the current root topic other than `SWALPHA01`.

2. `SWALPHA01/topic/write`
   Receives the new root topic as argument.

3. `SWALPHA01/reset`
   Resets all configurations parameters.

After defining the new root topic, the API is given. Consider including the root topic before every command.

Smart Window

- **MQTT API**

1. `/log`
   Logging messages destination.

2. `/open`
   Opens the window. No parameters needed.

3. `/close`
   Closes the window. No parameters needed.

4. `/config/read`
   Returns current configurations in JSON format.

5. `/config/write`
   Receives new configuration parameters in JSON format. Not all parameters must be set. You can also just write a new acceleration for example.

Note: changing pins will only take effect after saving configurations and reinitializing the microcontroller.

Smart Window

- **MQTT API**

6. `/config/save`
   Saves the current configurations in the static memory that are loaded in every initialization.

7. `/config/load`
   Loads last saved configuration parameters.

8. `/config/reset`
   Resets all configuration parameters to their default value.

Smart Window

- **MQTT API**
- JSON Format

```
{
   "dirPin":4,
   "stepPin":5,
   "slpPin":16,
   "revSteps":200,
   "radius":6.359439,
   "length":500,
   "maxSpeed":1080,
   "acc":360,
   "limOpenSwitch":0,
   "limCloseSwitch":0,
   "timeUTC":-3,
   "serialOutput":true,
   "mqttTopicRoot":"SWALPHA01",
   "logLevel":3
}
```

Units!

mm

°

s

e.g.

acc °/s²

Weather Station

- Every smart home you think must include some kind of weather station. Weather and forecast are really important decision factors to take actions inside a house.
- In order to develop a demonstrator model, I wrote this station service that can be attached to other applications within the ESP8266 or even in one single controller.
- Weather data are periodically collected from **OpenWeather** through its API. In order for that to work, you must create an account and **generate your free API key** for the HTTP requests. Once you have the key, you can set it by using the MQTT API.

Weather Station

Weather Station

- **MQTT API**

- **First note:** every `/get` topic receives as argument another topic where the response should be published to.

- It consists of getters and setters for its parameters as follows. The **standard root topic** is `weather`

1. `city/get`
   Returns the current set city from where weather data is collected and its country code as `<city>,<country code>`, *e.g.* `Berlin,DE`.
2. `city/set`
   Sets city from where weather data is collected and its country code as `<city>,<country code>`, *e.g.* `Berlin,DE`.

Weather Station

- **MQTT API**

3. `npredictions/get`
   Returns the number of predictions/forecast data.
4. `npredictions/set`
   Sets the number of predictions/forecast data. Maximum value tested is two.
5. `topic/get`
   Returns the current root topic.
6. `topic/set`
   Sets a new root topic.
7. `apiKey/get`
   Returns the current API key from OpenWeather.
8. `apiKey/set`
   Sets an API key from OpenWeather. Consider creating an account and generating one. Otherwise you won't get any response from the weather server.

Weather Station

- **MQTT API**

9. `period/get`
   Returns the current data request period in seconds.
10. `period/set`
    Sets a new data request period in seconds. Consider that OpenWeather lets you make a maximum of 1,000,000 calls per month with a free account! This implies in 60 calls per minute.
11. `save`
    Saves current configuration parameters in the static EEPROM memory.
12. `load`
    Loads last saved configuration parameters from the static EEPROM memory.

Weather Station

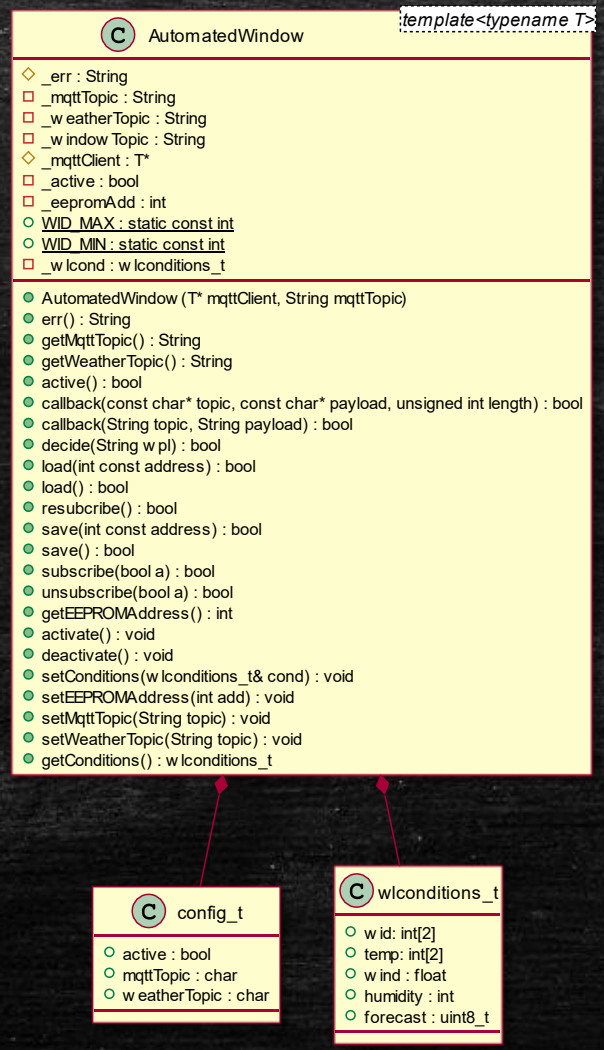
- **MQTT API**
- Output JSON format


```
{
    "weather":
    [
        {
            "id": 801,
            "main": "Cloudy",
            "description": "Pigs are falling from sky!",
            "temp": 23.33,
            "feels_like": 25.66,
            "humidity": 50,
            "wind": 0.21,
            "dt": 1603167280
        },
        {
            "id": 600,
            "main": "Cloudy",
            "description": "Aliens will be seen on sky!",
            "temp": 23.33,
            "feels_like": 25.66,
            "humidity": 50,
            "wind": 0.21,
            "dt": 1603169000
        }
    ]
}
```

Automated Window

- This service is a decision maker and it subscribes to the weather client and publishes to the smart window depending on the user preferences. It consists of closing/opening the window according to weather conditions.
- It is encapsulated in `AutomatedClient.h` and therefore can be implemented in other applications as well rather than in the broker specifically.
- **Note** that the user must set the conditions for the window to be **open**! If these conditions do not match, then it will call the operation to close the window.

Automated Window



Automated Window

- **MQTT API**
- **First note:** every /get topic receives as argument another topic where the response should be published to.
- It consists of getters and setters for its parameters as follows. The **standard root topic** is automatedWindow.
- Intervals are given in **JSON format** like: `{"min": <value>, "max": <value>}`

Automated Window

- **MQTT API**

1. `/wid/get`
   Returns the current weather ID interval set. Output is JSON.

2. `/wid/set`
   Sets a new weather ID interval. To avoid letting your window open during a thunderstorm, the interval is limited to [800, 804]. Input is JSON. Default: `{"min": 800, "max": 804}`

3. `/temp/get`
   Returns the current temperature interval set. Output is JSON.

4. `/temp/set`
   Sets a new temperature interval. Input is JSON. Default: `{"min": 16, "max": 50}`

Automated Window

- **MQTT API**

5. `/wind/get`
   Returns the current maximum wind speed condition set in m/s.

6. `/wind/set`
   Sets a maximum wind speed condition in m/s. Default: `5.5`.

7. `/humidity/get`
   Returns the current maximum humidity condition set in %.

8. `/humidity/set`
   Sets a maximum humidity condition in %. Default: `40`.

Automated Window

- **MQTT API**

9. `/forecast/get`
   Returns the number of forecasts to consider for taking decision. For example, if it is about to rain in, it closes the window before it even starts to rain.

10. `/forecast/set`
    Sets the number of forecasts to consider for taking decision. Default: `1`.

11. `/topic/get`
    Returns the current root topic.

12. `/topic/set`
    Sets a new root topic. Default: `automatedWindow`.

Automated Window

- **MQTT API**

13. `/activate`
    Activates the automation client. Active by default.

14. `/deactivate`
    Deactivates the automation client. (Watch out!)

15. `/save`
    Saves current configuration parameters.

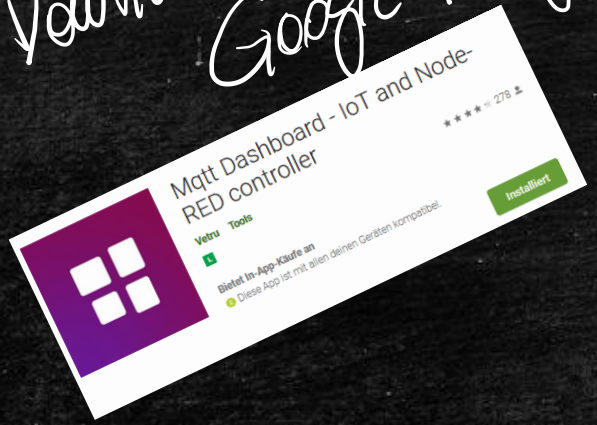
16. `/load`
    Loads last saved configuration parameters.

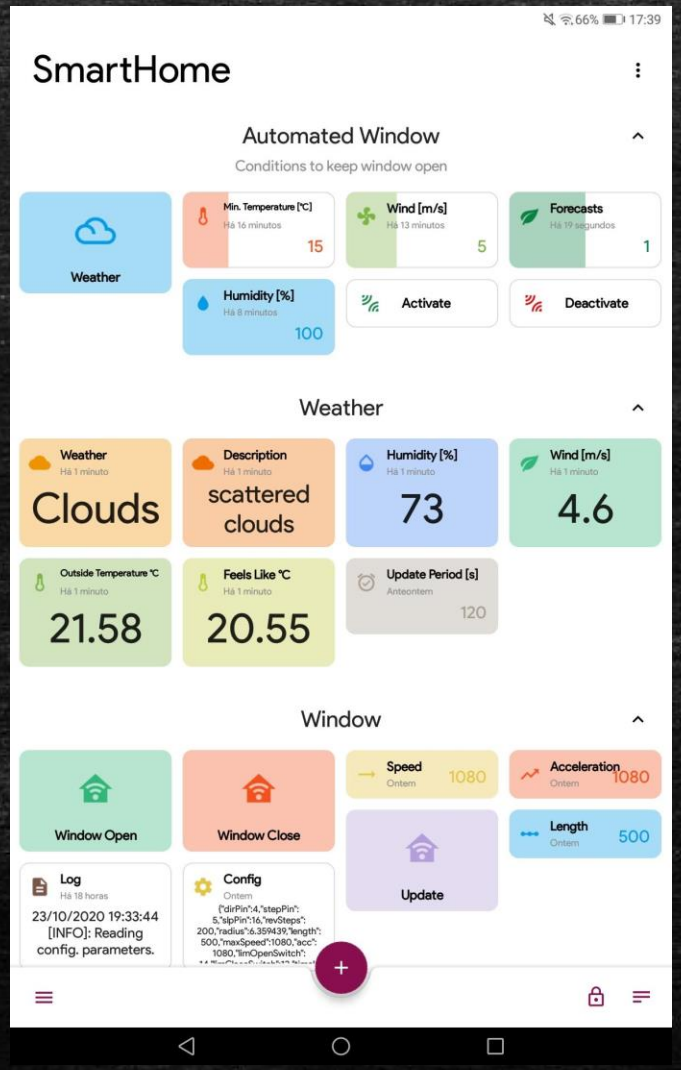
Broker

- It just implements the MQTT broker library uMQTTBroker. Mind the connection limitations described in their page! For example, it does not support QoS greater than zero and neither too many clients connected simultaneously.
- Configure your connection parameters in `definitions.h`. There you can set your static IP address for the broker as well as your Wi-Fi settings.
- This application also implements the Automation Client for the Smart Window.
- Before uploading the code to the ESP8266, go to your Arduino Board Configuration and **set IwIP Variant to v1.4 Higher Bandwidth**.

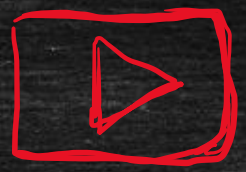
Results

Download it from Google Play





Results



Check out the Video on YouTube

https://youtu.be/L8XZHv6YP_w


That's All