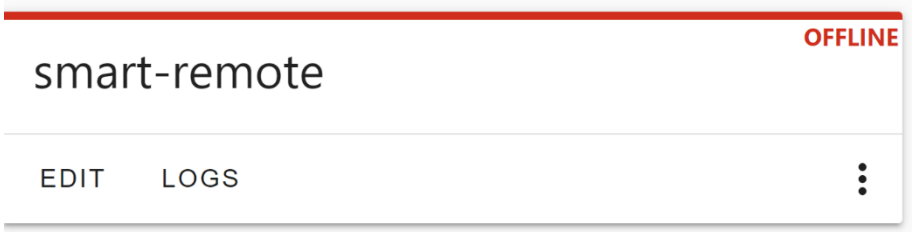
ESPHome Automation

Configuration:

We can configure our ESP32 board using a YML file. The ESPHome framework will translate into cpp code and compile the firmware. All that is left to do is to upload that firmware to the board.



By clicking “Edit” you will gain access to the YML file to edit, and by clicking “Logs” you will see the output logs from your ESP32 board. In order to upload a new firmware to your device press the three dots on the right side and press install. You can install using a few methods, the two main methods are:

- Via usb – you will need to use this option when configuring your board for the first time since it doesn’t know the password to your home network.
- Wirelessly – this can be done after the first configuration when your board knows the wifi password and can establish a wireless connection with Home Assistnant.

Changing Wifi credentials without plugging in:

Your Wi-Fi credentials are stored in the secrets.yml file which can be accessed by clicking the “SECRETS” button on the top right corner in the ESPHome add-on in home assistant. Security wise it is best to store your credentials there and not in clear text in the configuration file. To change your wifi ssid or password just add a new name or password to the secrets.yml file and then set them to be the ssid and password of the fallback hotspot in your configuration file. The fallback hotspot defines the name and password of a network to access in case the first one is unavailable.

Bluetooth Automation:

The ESP32 has a Bluetooth low energy(BLE) device. This can allow automations based on Bluetooth recognition. In order to setup the ble stack on the board and listen to available devices we must add “esp32_ble_tracker” component. We can connect to Bluetooth devices using the “ble_client” component and configure automations by configuring the “on_connect” and “on_disconnect” variables like so:

```
esp32_ble_tracker:
ble_client:
  - mac_address: 11:22:33:44:55:66
    id: my_phone
    on_disconnect:
      then:
        - lambda: |-
            id(off_button).press();
    on_connect:
      then:
        - lambda: |-
            id(on_button).press();
```

Although this is an option it is not a recommended one since it requires establishing a connection between the device and the esp32 board and a normal user who can only establish one connection would rather connect to his speakers or any other device than to his esp32 board.

To overcome this we can buid a binary sensor that detects the presence of Bluetooth devices according to their mac address or their iBeacon UUID.

Detecting a device according to its mac address is problematic since many devices, like iPhones randomize their mac addresses. That means that we should set up our sensor to detect specific iBeacon UUID's, instead of mac addresses. In order to turn your phone into an iBeacon and transmit a specific UUID all you need to do is download one of the many iBeacon apps.

Here is an example of the code to set up an iBeacon binary sensor which detects a specific UUID and then switches the ac on/off according to the state of the binary sensor:

```
38  binary_sensor:
39    # Presence based on iBeacon UUID
40    - platform: ble_presence
41      name: "Phone Tracker"
42      id: phone_tracker
43      ibeacon_uuid: 'de98a2e9-db46-477b-8909-ddfd22c2fcff'
44      ibeacon_major: 123
45      ibeacon_minor: 456
46      on_state:
47        then:
48          - lambda: |-
49              if (x) {
50                id(on_button).press();
51              } else {
52                id(off_button).press();
53              }
```

More on binary sensors and their automations here, and more on the ble_presence platform here.

Temperature and humidity sensor and automation:

We connected the dht11 sensor which provides a temperature and humidity sensor. ESPHome we integrated them into our project using the “sensor” component and choosing a “dht” platform. In the picture below we can see an example of how we configured the sensor, gave it a name, id and configured an automation based on the temperature range:

- If the temperature is above 27 degrees and I’m home -> turn on the AC
- If the temperature is below 26 degrees and I’m home -> turn off the AC

```
sensor:
  # temperature and humidity sensor
  - platform: dht
    pin: GPIO16
    model: DHT11
    temperature:
      name: "Room Temperature"
      id: room_tmp
      on_value_range:
        # if tmp is above 27 and i'm home turn ac on
        - above: 27.0
          then:
            - lambda: |-
                if(id(phone_tracker).state) {
                  id(on_button).press();
                }
        # if tmp is below 26 and i'm home turn ac off
        - below: 26.0
          then:
            - lambda: |-
                if(!id(phone_tracker).state) {
                  id(on_button).press();
                }
    humidity:
      name: "Room Humidity"
      id: room_hmd
    update_interval: 60s # update the sensor gui every minute

```

More about the “sensor” component here, and more about the “dht” platform here.

User Input Based Configuration:

In our product we would like to enable the user to configure some settings. These settings can be trivial ones, like the type of air condition they use so that the correct IR signals will be sent, or more advanced ones like the temperature that will trigger the automation of powering on the AC.

There are many ways to receive user input:

1. We can create switch sensors via ESPHome, every time we perform and automation we can check the state of that switch and run code according to it. For example in our code we created a switch for various types of popular AC models. Every time the user presses the on button, inside a lambda we check the state of every switch and if the state of a specific switch is on than we send the “On” IR signal specific to AC manufacturer represented by that switch.

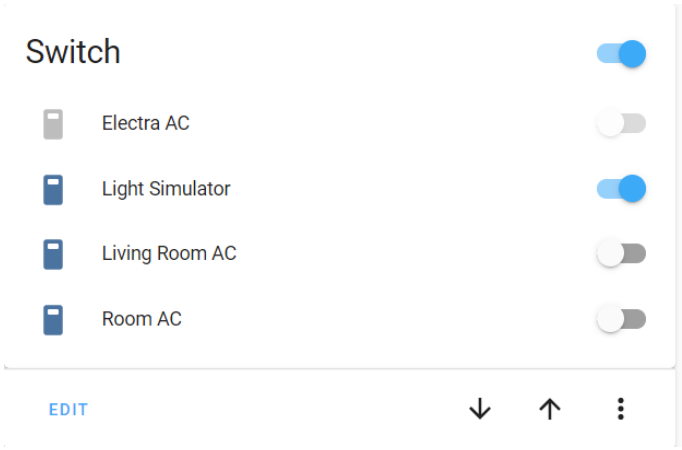
Example of creating such switches on ESPHome:

```yaml
switch:
  # these switches are user input that define what types of AC they have
  # that way we can send the IR signals according to their AC type
  - platform: template
    name: "Room AC"
    id: room_ac
    optimistic: true

  - platform: template
    name: "Light Simulator"
    id: light_ac
    optimistic: true

  - platform: template
    name: "Living Room AC"
    id: living_room_ac
    optimistic: true
```

How the switches look on the Home Assistant dashboard:



Example for automation based on these switches:

```yaml
on_press:
  - lambda: |-
      if(id(room_ac).state) {
        auto call = id(my_transmitter).transmit();
        esphome::remote_base::ProntoData data = { "0000 006D 0058 0002 0078 0097
        esphome::remote_base::ProntoProtocol().encode(call.get_data(), data);
        call.set_send_times(1);
        call.perform();
      }
      if(id(light_ac).state) {
        auto call = id(my_transmitter).transmit();
        esphome::remote_base::ProntoData data = { "0000 006D 0022 0000 0161 00AD
        esphome::remote_base::ProntoProtocol().encode(call.get_data(), data);
        call.set_send_times(1);
        call.perform();
      }
      if(id(living_room_ac).state) {
        auto call = id(my_transmitter).transmit();
        esphome::remote_base::CoolixData data = { 0xB2BF20 };
        esphome::remote_base::CoolixProtocol().encode(call.get_data(), data);
        call.set_send_times(1);
        call.perform();
      }
```

Specifically for sending the on/off IR signals based on user input inside lambdas, we used the "RemoteTransmitter" API, which is documented here. For each manufacturer the API specifies a data structure for the data to be transmitted and a protocol object. By searching the API for a specific manufacturer you can determine how to construct the data object and how to define the transmition protocol.

2. We can create input sensors via the home assistant dashboard using the Home Assistant configuration.yml. This has a lot of advantages since we can receive user input in many different formats such as bool, int or even strings. The official documentation on how to access, modify and reload the Home Assistant configuration.yml file is here. Using this configuration file you can create and add the input interfaces to the home assistant dashboard. In order to access these user inputs, for each input you will need to define a sensor in the ESPHome yaml file like so(the entity_id is the one defined in the Home Assistant configuration.yml file):

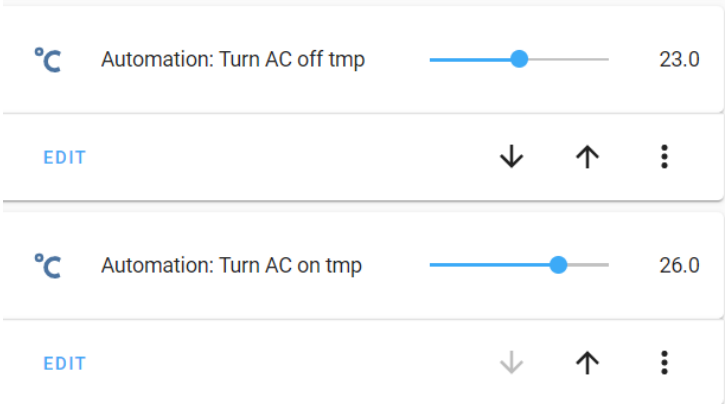
```
  - platform: homeassistant
    name: "Turn AC off tmp"
    entity_id: input_number.automation_turn_ac_off_tmp

  - platform: homeassistant
    name: "Turn AC on tmp"
    entity_id: input_number.automation_turn_ac_on_tmp
```

The data from the sensor can be accessed in lambda automations like so:

```
sensor:
  # temperature and humidity sensor
  - platform: dht
    pin: GPIO16
    model: DHT11
    temperature:
      name: "Room Temperature"
      id: room_tmp
      on_raw_value:
        then:
          # Automation to turn AC on/off
          # Turn the AC on if current temp higher than user input
          # Turn the AC off if current temp lower than user input
          - lambda: |-
              if(id("Turn AC on tmp") <= x) {
                id(on_button).press();
              }
              if(id("Turn AC off tmp") > x) {
                id(off_button).press();
              }
```

For the example above, the user will be able to configure the on/off temperature from the Home Assistant dashboard like so:



An example of how to configure user input here.

Notifications:

The topic of our seminar was gamification. While researching gamification we discovered what a powerful tool it is to retain and attract users. We thought that it could be a good idea not throw that knowledge away and implement it in our product in order to enhance our user experience.

Like we stated earlier, our product has a bluetooth automation, that turns the AC on when we arrive home and turns it off when we leave.

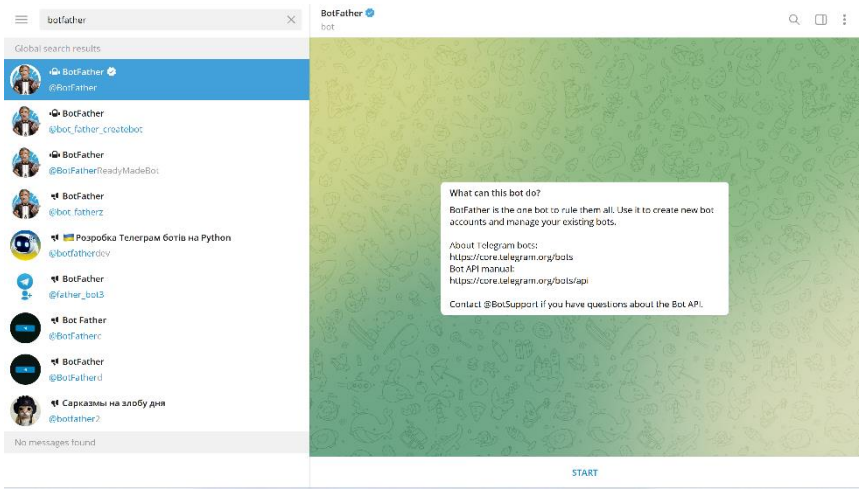
We designed our gamification feature such that the home assistant server will send a telegram message to the user every time the AC is automatically turned off when he leaves home. The message will have celebration emojis to make the user feel as if he won a game, and present him with data that will convince him that his achievement is worthwhile, for example:

1. The amount of money he saved until today with the help of this feature.
2. The amount of trees he saved until today with the help of this feature.

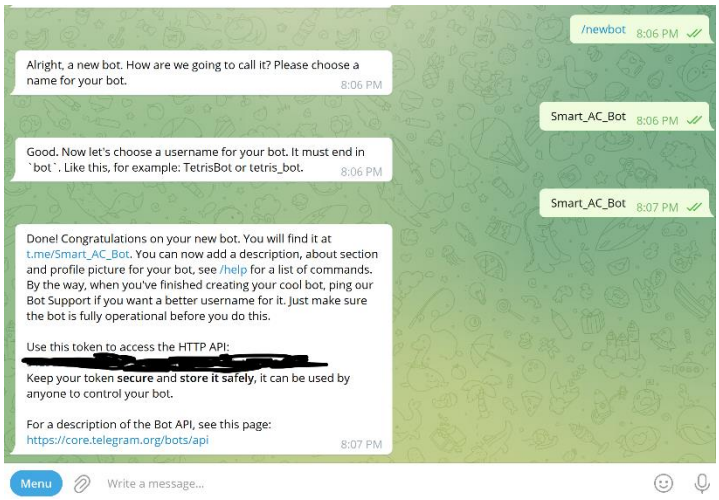
This section while explain the notification API in Home Assistant, how to setup a Telegram bot and how to integrate the two together in order to send automated messages. Just before we begin, let us state that Home Assistant offers much more notification APIs such as native push notifications and emails. A very well documented video regarding those notification automations can be found here.

Setting up the Telegram bot:

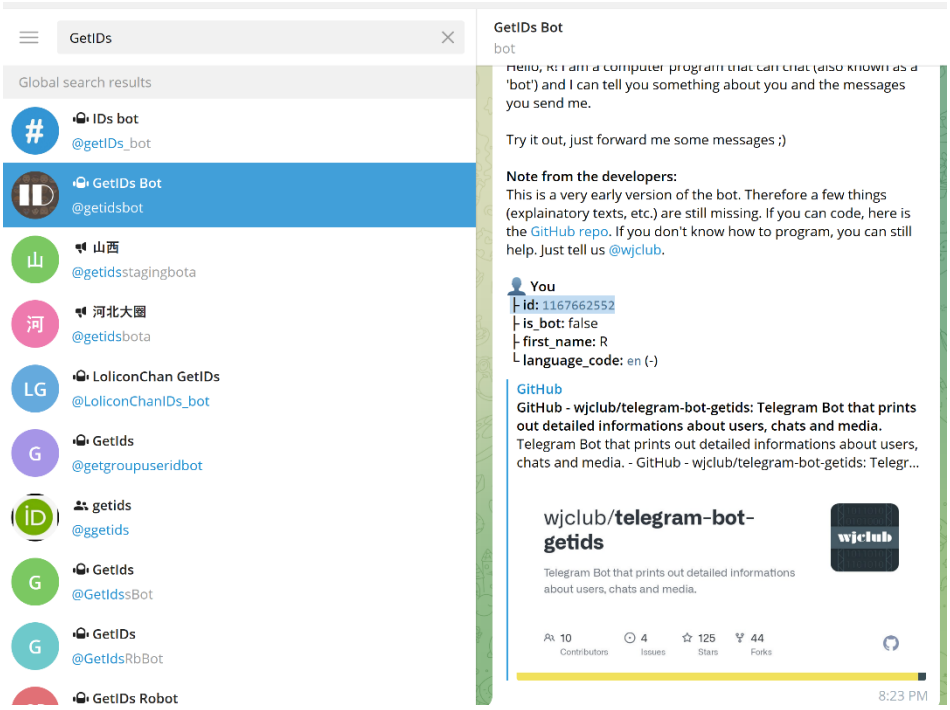
1. Install the Telegram desktop app from here.
2. Search for “botfather” and hit start(choose the BotFather with the official v):



3. Type the command “/newbot”, then choose your bot name and username. The blacked out part the the API token. Copy it and save it in the /config/secrets.yml file to securely save and use it.



4. Search for the “getidsbot”, enter “/start” to get your chat id:



5. Search for the bot you just created and enter the command “/start”

Integrating the bot with Home Assistant:

1. Initiate the bot in the “/config/configuration.yaml” file like so:

```
14 telegram_bot:
15   - platform: polling
16     api_key: !secret telegram_api_token
17     allowed_chat_ids:
18       - !secret telegram_chat_id
```

2. Now we will use this bot in an automation. We will send an achievement message everytime the Bluetooth binary sensor turns off, meaning that the user left his room. To do this we added the following automation in the “/config/automations.yaml” file:

```yaml
- id: '6634902365988'
  alias: Leave Home Notification
  trigger:
  - entity_id: binary_sensor.am_i_home
    platform: state
    to: 'off'
  condition: []
  action:
    service: telegram_bot.send_message
    data:
      title: Your AC was autmatically turned off
      message: 'Congrats!!! Our automation turned off your AC since we detected that you left your room. You just saved money and reduced air pollution'
```

The official document for the Home Assistant Telegram API is here. Let us note that the options listed in the documentation are endless, you can send photos, files or even videos. In case we want to create a product that is fully automated out of the box and don’t want the user to start configuring yaml files, we can generate an installation script that creats an api token and pulls it and the chat id using a get request, then it can automatically patch the configuration.yaml, automations.yaml and secrets.yaml files with the relevant information. All that is needed for this is listed in the official documentation we linked above.

An example of the achievement notification message sent to Telegram: