# CrowPanel Basic HMI ESP32 Display (3.5 / 2.8 / 2.4-inch)

# ---ESPHome-Tutorial

## 1.course introduction

In this class, we will teach you how to use and operate the CrowPanel Basic HMI ESP32 Display through ESPHome.

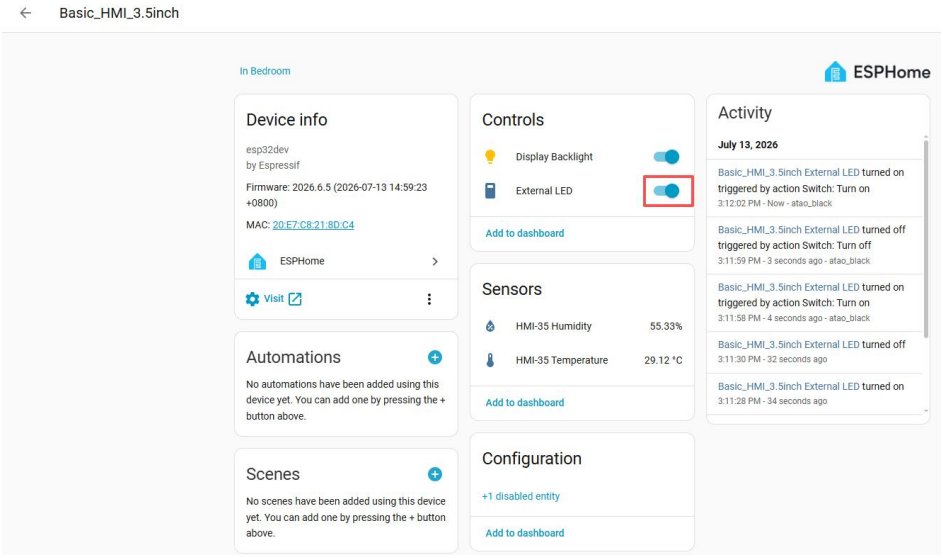
So let's move on to learn how to install the ESPHome environment, allowing you to edit code on it to achieve reading temperature and humidity data and remotely control the on and off of LEDs.

## 2.learning objectives

1. Get a basic understanding of the software required for writing ESPHome code.
2. Learn how to install the necessary software.
3. Learn how to create a new project.
4. Learn to write relevant code to achieve the function of collecting temperature and humidity data, and be able to view the temperature and humidity data remotely on the ESPHome platform.
5. Learn to write relevant code to achieve the function of turning on and off an LED, and be able to control the LED remotely on the ESPHome platform.

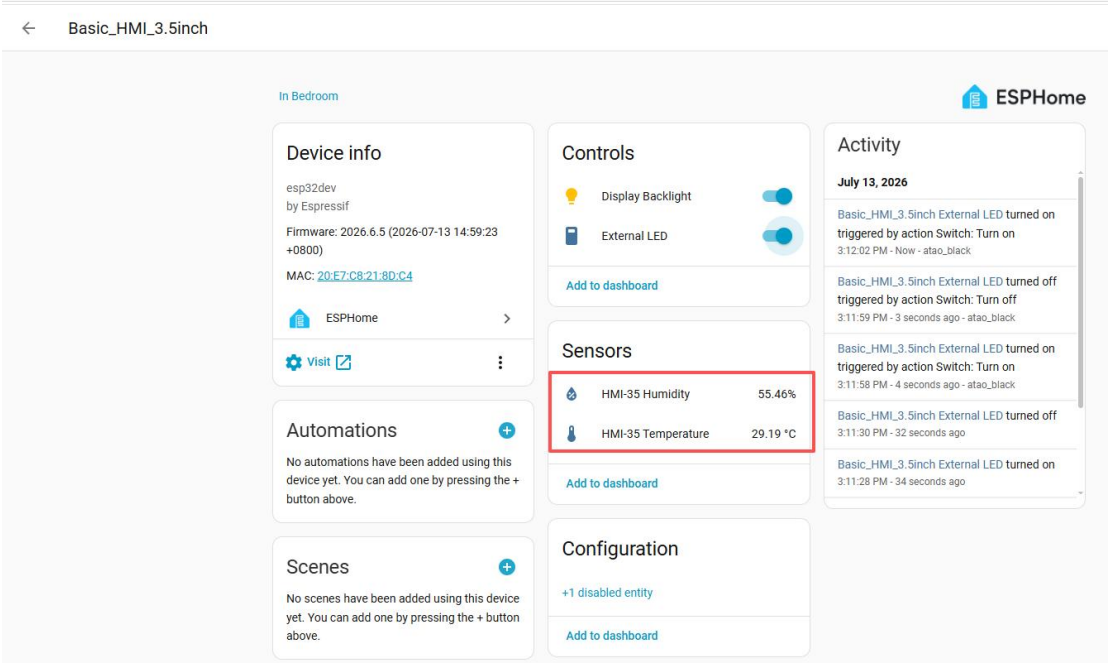
## 3.Project operation effect illustration

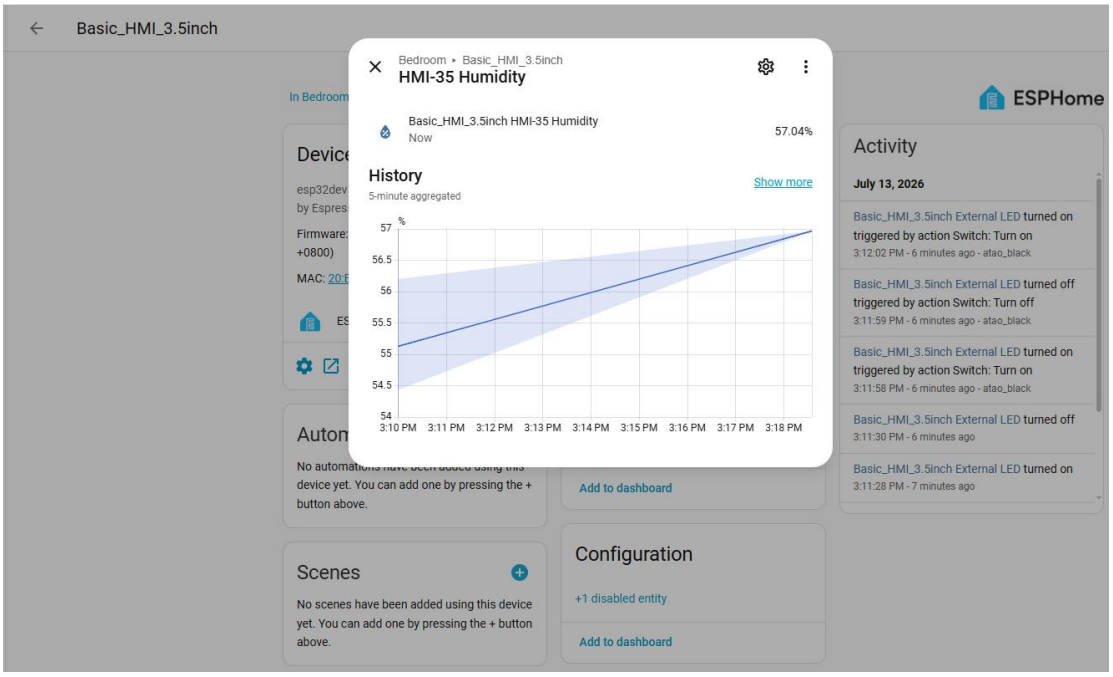
Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

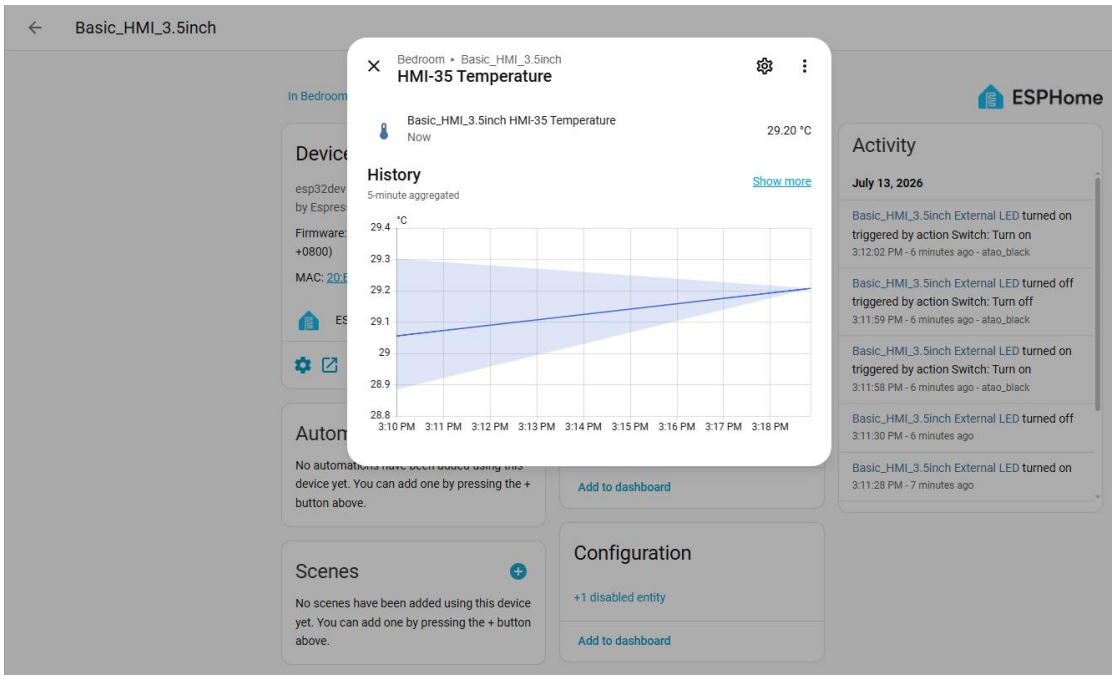


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

And you can also see the historical data of temperature and humidity collection from here.


Basic_HMI_3.5inch
In Bedroom
ESPHome
Device info
esp32dev
by Espressif
Firmware: 2026.6.5 (2026-07-13 14:59:23 +0800)
MAC: 20:E7:C8:21:8D:C4
ESPHome
Visit
Automations
No automations have been added using this device yet. You can add one by pressing the + button above.
Scenes
No scenes have been added using this device yet. You can add one by pressing the + button above.
Controls
Display Backlight
External LED
Add to dashboard
Sensors
HMI-35 Humidity 55.46%
HMI-35 Temperature 29.19 °C
Add to dashboard
Configuration
+1 disabled entity
Add to dashboard
Activity
July 13, 2026
Basic_HMI_3.5inch External LED turned on triggered by action Switch: Turn on
3:12:02 PM - Now - atao_black
Basic_HMI_3.5inch External LED turned off triggered by action Switch: Turn off
3:11:59 PM - 3 seconds ago - atao_black
Basic_HMI_3.5inch External LED turned on triggered by action Switch: Turn on
3:11:58 PM - 4 seconds ago - atao_black
Basic_HMI_3.5inch External LED turned off
3:11:30 PM - 32 seconds ago
Basic_HMI_3.5inch External LED turned on
3:11:28 PM - 34 seconds ago


Basic_HMI_3.5inch
Bedroom ▸ Basic_HMI_3.5inch
HMI-35 Humidity
Basic_HMI_3.5inch HMI-35 Humidity
Now
57.04%
History
5-minute aggregated
Show more
%
57
56.5
56
55.5
55
54.5
54
3:10 PM
3:11 PM
3:12 PM
3:13 PM
3:14 PM
3:15 PM
3:16 PM
3:17 PM
3:18 PM
ESPHome
Activity
July 13, 2026
Basic_HMI_3.5inch External LED turned on triggered by action Switch: Turn on
3:12:02 PM - 6 minutes ago - atao_black
Basic_HMI_3.5inch External LED turned off triggered by action Switch: Turn off
3:11:59 PM - 6 minutes ago - atao_black
Basic_HMI_3.5inch External LED turned on triggered by action Switch: Turn on
3:11:58 PM - 6 minutes ago - atao_black
Basic_HMI_3.5inch External LED turned off
3:11:30 PM - 6 minutes ago
Basic_HMI_3.5inch External LED turned on
3:11:28 PM - 7 minutes ago
Add to dashboard
Configuration
+1 disabled entity
Add to dashboard
Scenes
No scenes have been added using this device yet. You can add one by pressing the + button above.




## 4.Introduce the software

Here's a clear, accessible overview for international customers (no overly technical jargon): Home Assistant (often called "HA" for short) is a free, open-source smart home management platform—think of it as the central "control brain" for all your smart devices.

It works on computers, small servers, or even dedicated smart home hubs, and lets you manage every smart product you own (lights, thermostats, sensors, your rotary screen, etc.) in one unified interface—regardless of the brand or technology the device uses. For your rotary screen, HA acts as the "command center": it can receive inputs from the screen (like knob rotations or touch taps), control other devices based on those inputs (e.g., turn up the lights when you twist the knob), and send data (like room temperature or music volume) to the screen for display.

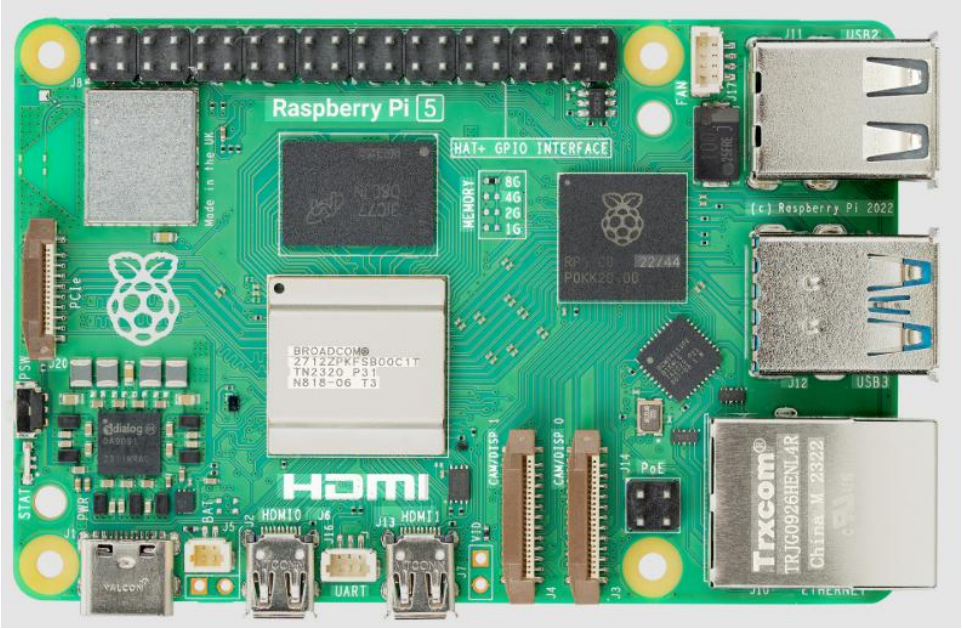
ESPHome is a free, open-source tool built to configure smart hardware (specifically devices powered by ESP32/ESP8266 chips—your rotary screen uses an ESP32). The key benefit? You don't

need to write complex code to make the screen work. Instead, you use simple, plain-text configurations (like filling in a template) to define what the rotary screen does: e.g., "show 'Hello World' on the display," "adjust the thermostat when the knob turns," or "wake the screen when touched." ESPHome takes these configurations, turns them into instructions the ESP32 can understand, and "loads" them onto your rotary screen—while also automatically syncing the screen's status (what it's displaying, how the knob is being used) with Home Assistant.

Together, Home Assistant and ESPHome turn your rotary screen into a flexible, customizable smart device: ESPHome gives the screen its "basic skills" (display, knob/touch response), and HA connects those skills to your entire smart home—so the screen can both control your devices and show you what's happening with them, all without requiring technical expertise.

## 5.How to install software

First, we need to prepare:
a Raspberry Pi 5;




a 64GB SD card;

a CrowPanel Basic HMI ESP32 Display(2.4-inch \ 2.8-inch \ 3.5-inch);
purchase link :
https://www.elecrow.com/esp32-display-3-5-inch-hmi-display-spi-tft-lcd-touch-screen.html
https://www.elecrow.com/esp32-display-2-8-inch-hmi-display-spi-tft-lcd-touch-screen.html
https://www.elecrow.com/esp32-display-2-4-inch-intelligent-spi-tft-lcd-touch-screen-hmi-display.html

A humidity and temperature sensor;

purchase link :

https://www.elecrow.com/crowtail-dht20.html

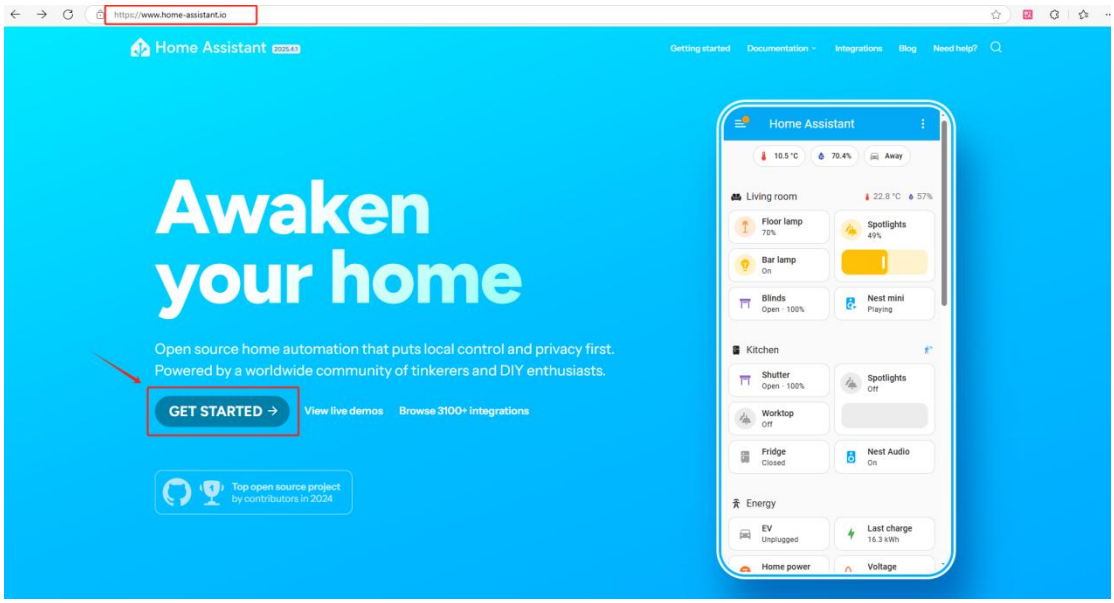
An LED light;

purchase link :

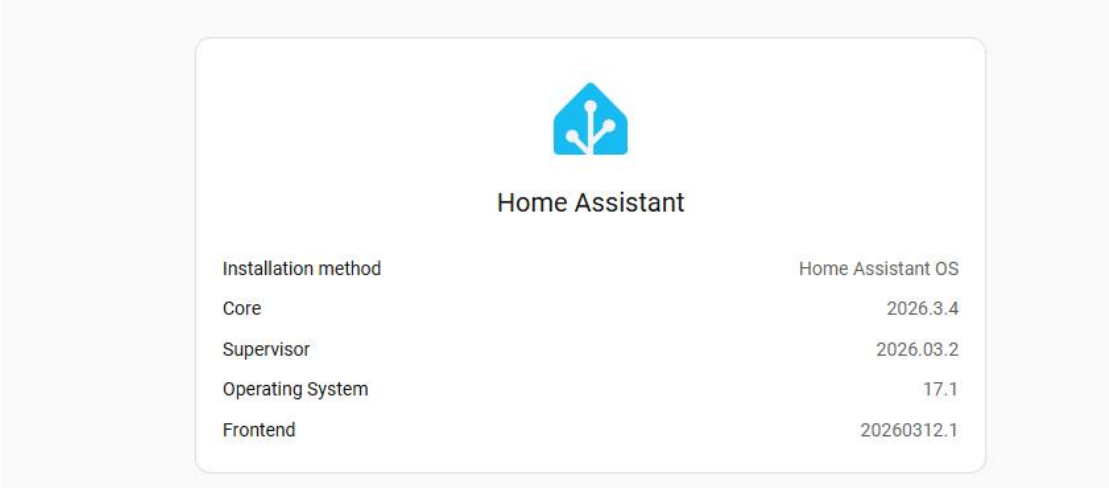
https://www.elecrow.com/crowtail-led-p-1224.html

## Download HomeAssitant

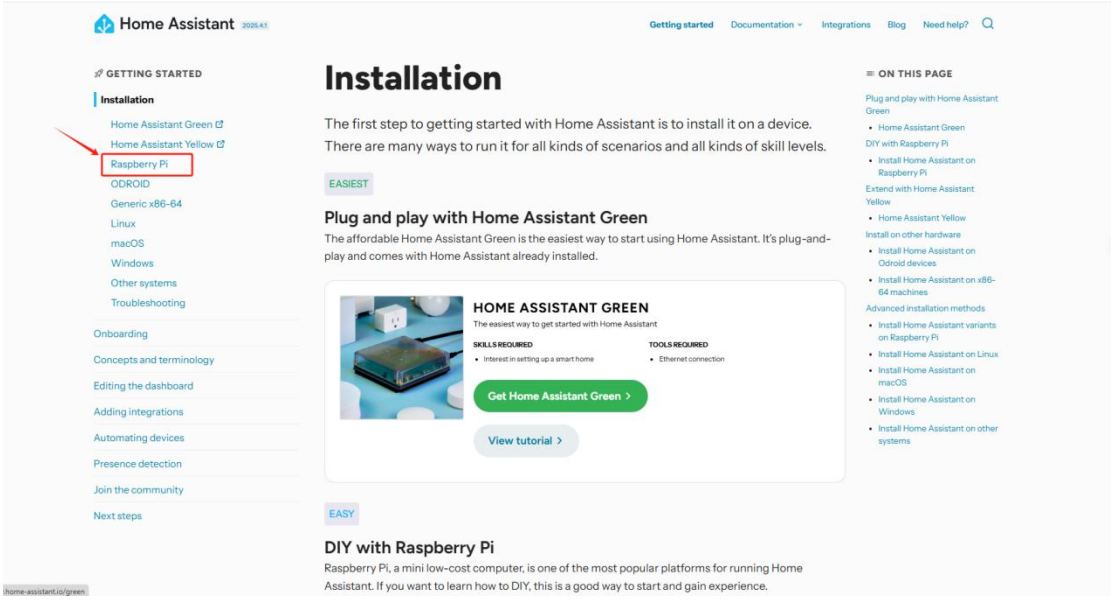
Open the official website of HomeAssitant：https://www.home-assistant.io/




**The version we are using here is 2026/3/4.**




Select a Raspberry Pi and install the Home Assistant system according to this installation guide.(Install according to the official installation guide.)




After the installation is complete, insert the SD card with the Home Assistant system into the Raspberry Pi and connect the Ethernet cable.

**Note:** Make sure that the Wi-Fi network your CrowPanel Basic HMI ESP32 Display(2.4-inch \ 2.8-inch \ 3.5-inch) will connect to is on the same local area network (LAN) as the Raspberry Pi.

The following devices need to be on the same LAN:

① Your computer

②CrowPanel Basic HMI ESP32 Display (2.4-inch \ 2.8-inch \ 3.5-inch);

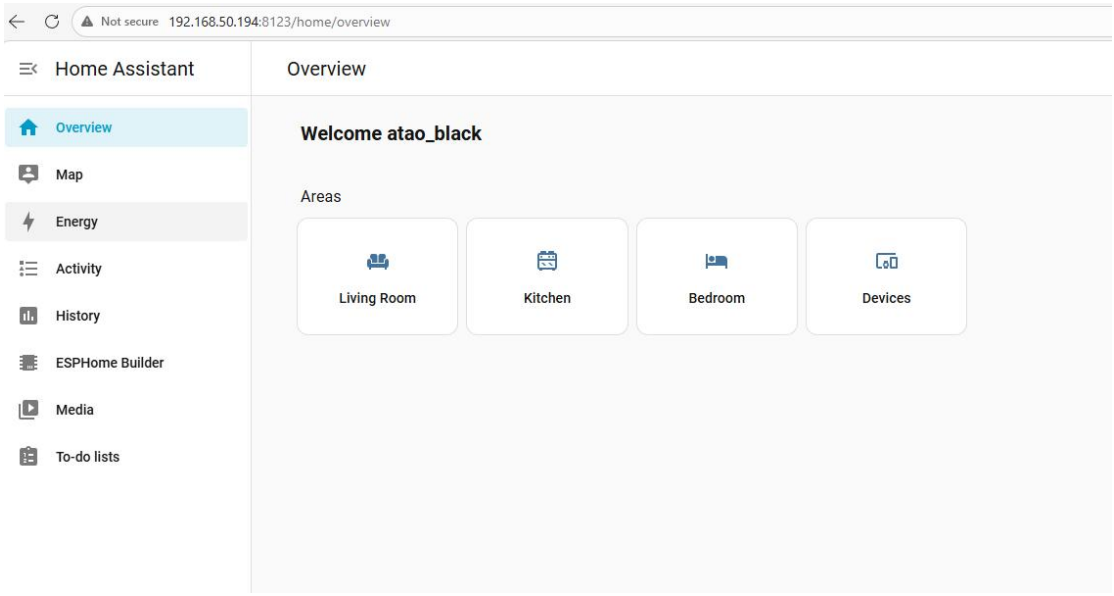
③ Raspberry Pi with the Home Assistant system

**Note:** The Raspberry Pi must be connected to a screen to view information. Once everything is ready, power on the Raspberry Pi and wait for it to load. The screen will display the following image.

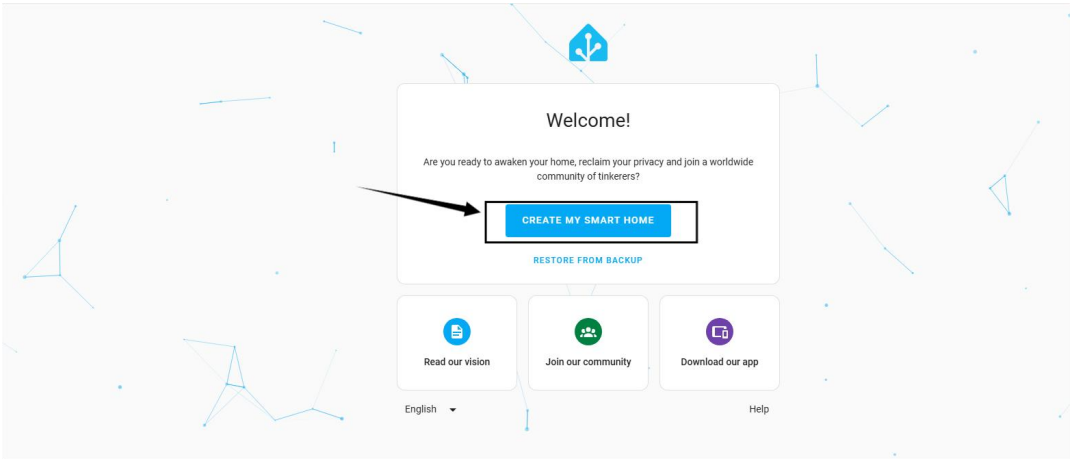
(You need to connect a display screen to the Raspberry Pi in order to see these information.)

Remember the IP address and the Home Assistant URL.

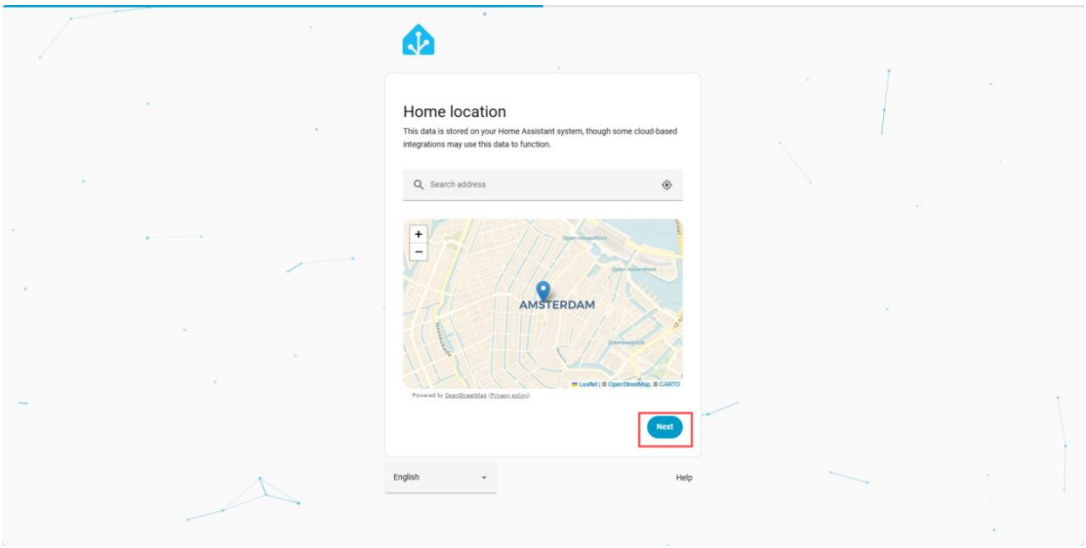
In my case, the IP address is: 192.168.50.194
So I enter **192.168.50.194:8123** in the browser.



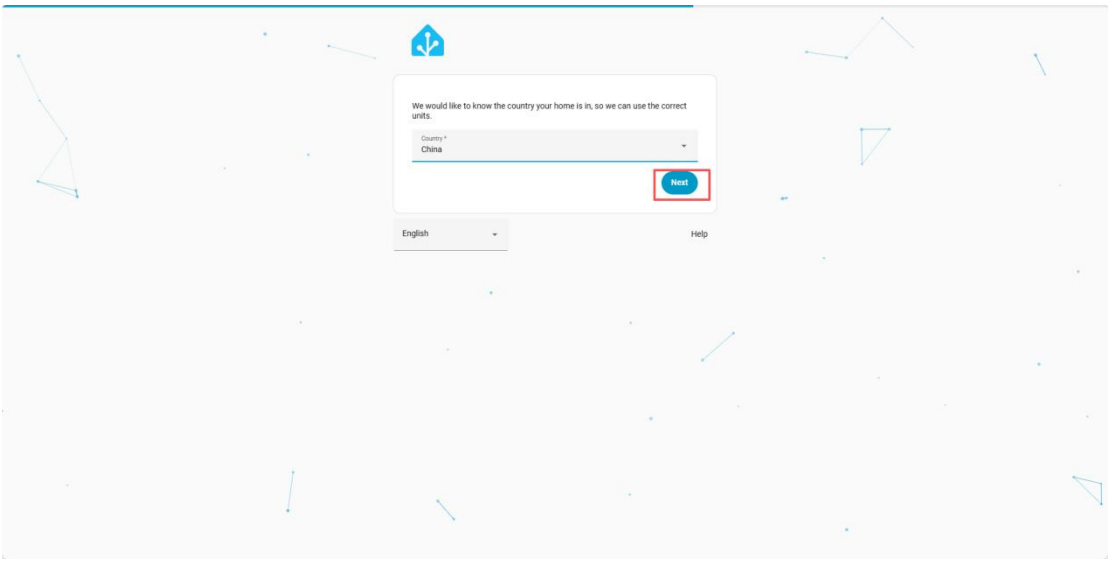

Next, start configuring the Home Assistant system.



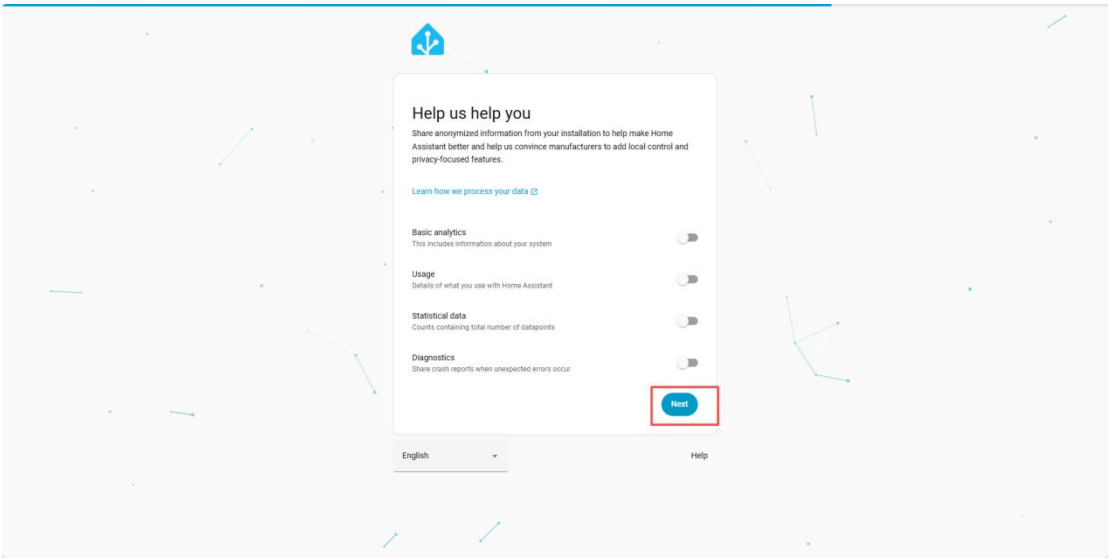

You can either manually set your location or allow it to be detected automatically.



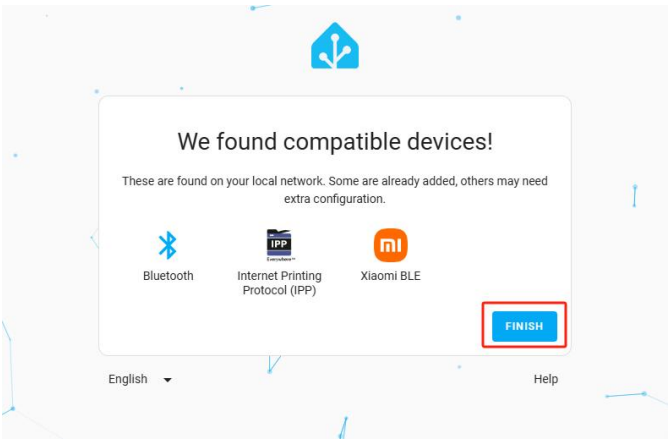

Set the country where you are located.



Keep it as default here.

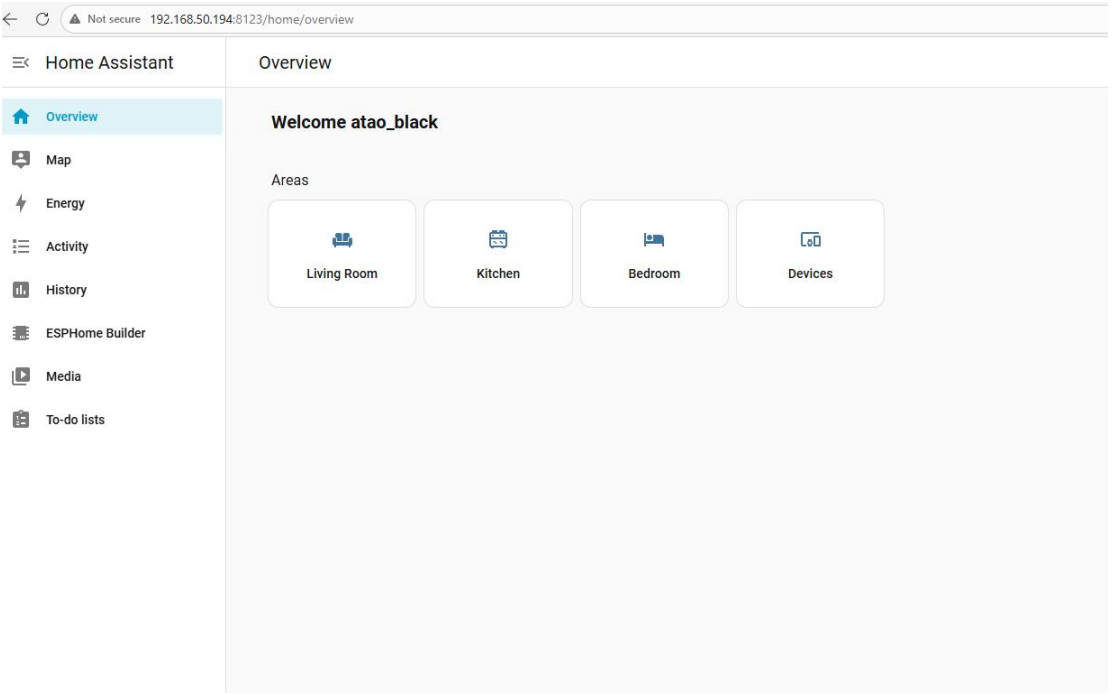



You can either add smart devices now or click **Finish** to add them later.
Here, we will add the devices later.

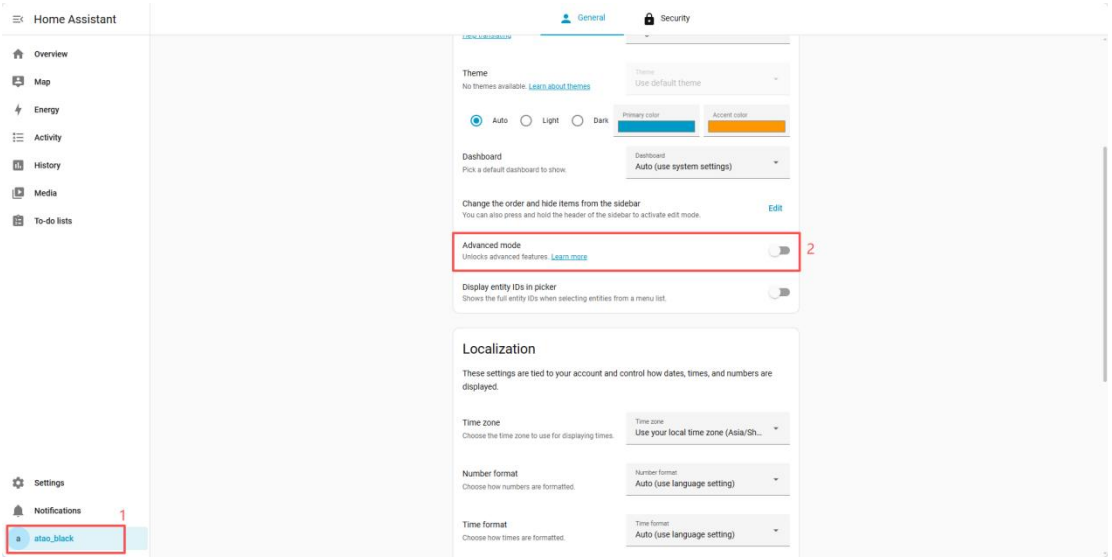



## Add ESPHome

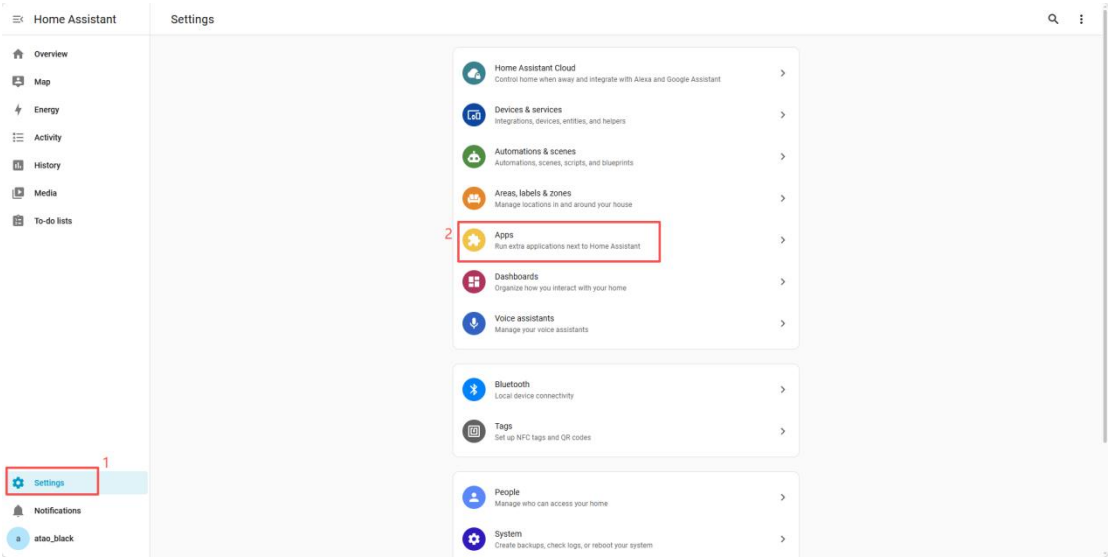
This will bring you to the main interface of Home Assistant.



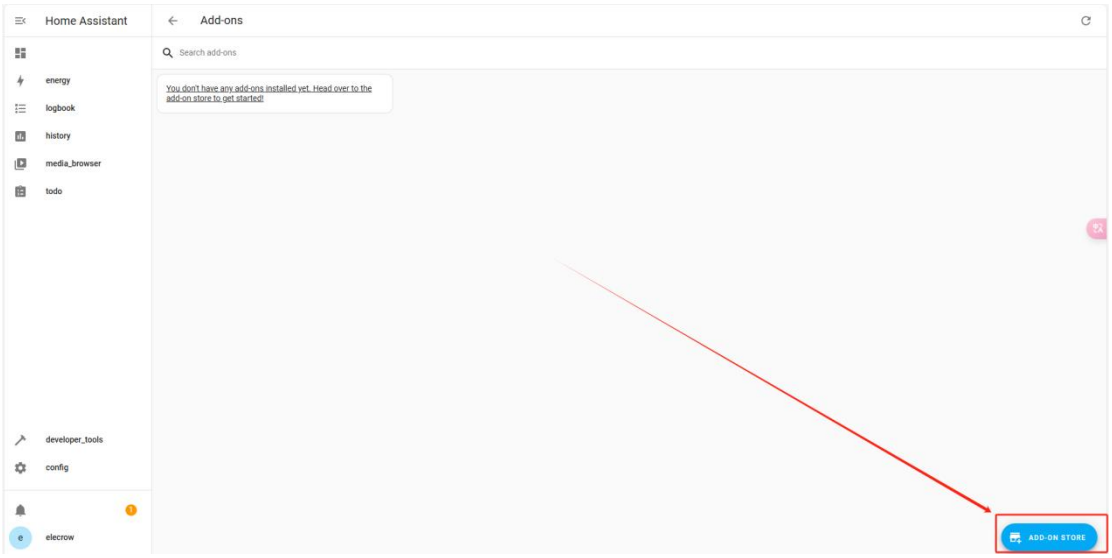

Click on the username and enable "Advanced Mode."

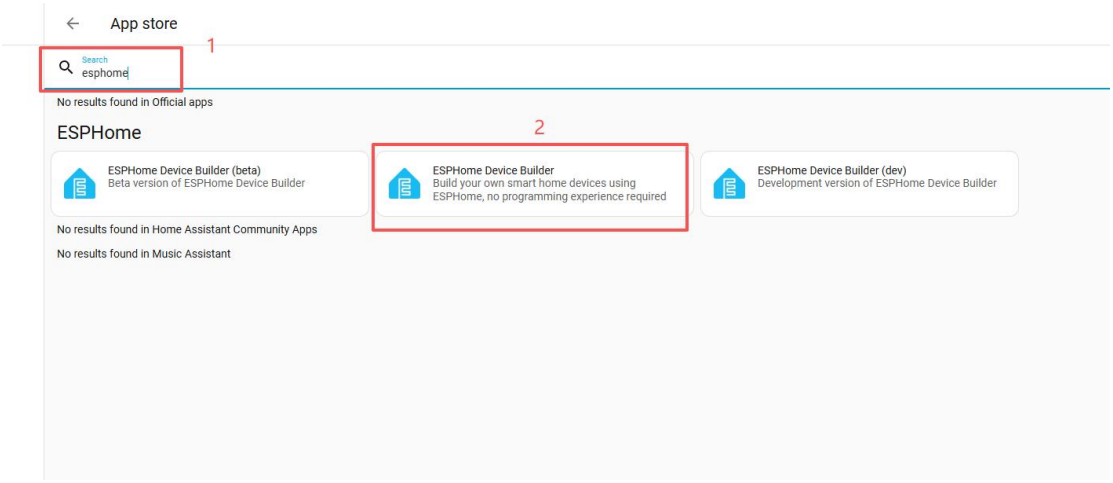



Click on "Settings" and then "Apps."

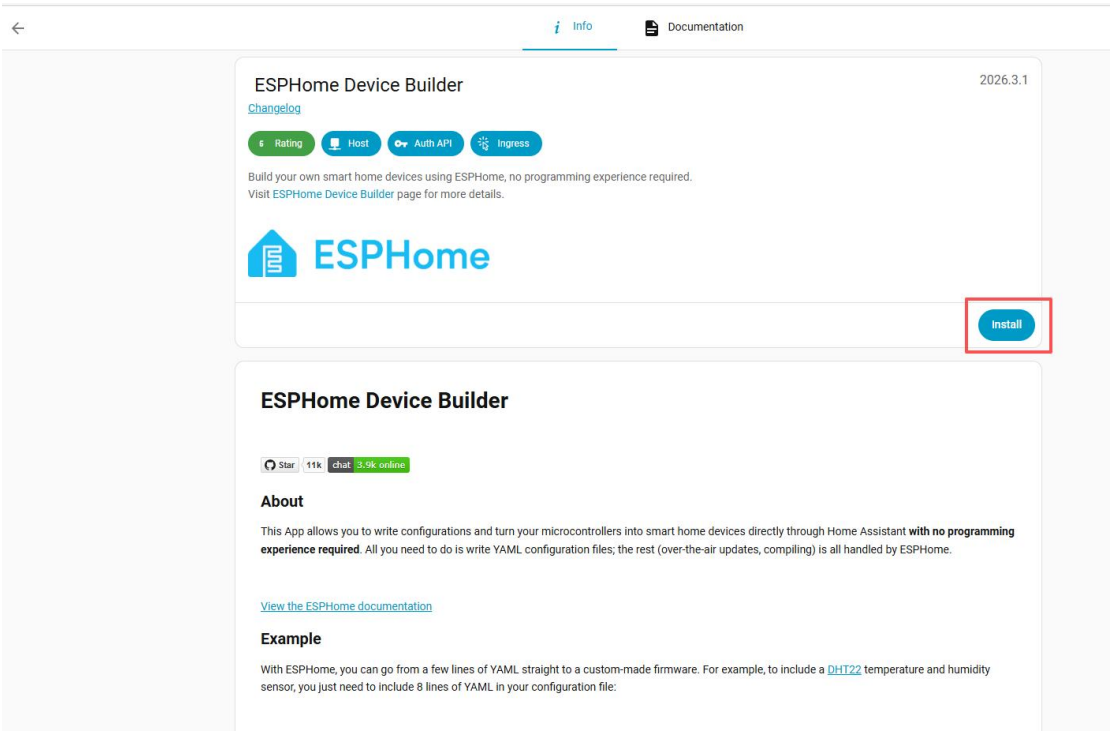


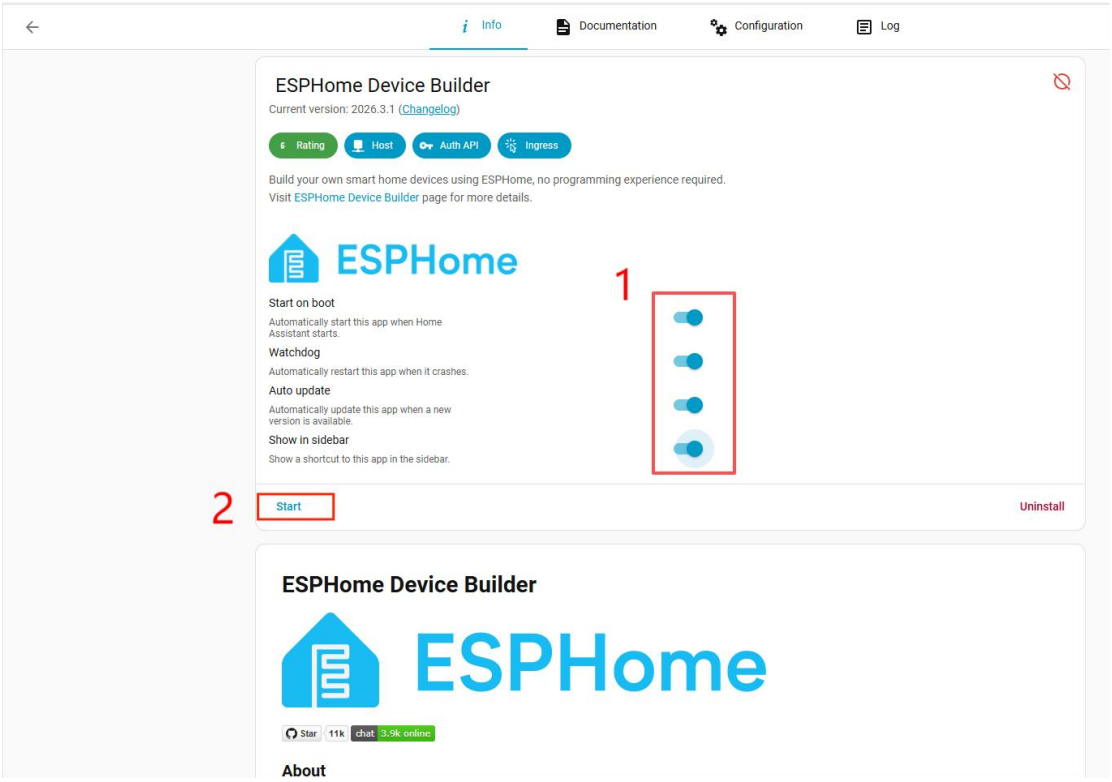

Click on "Apps" and type **ESPHome** to install it.




The installation is in progress, as shown in the image.



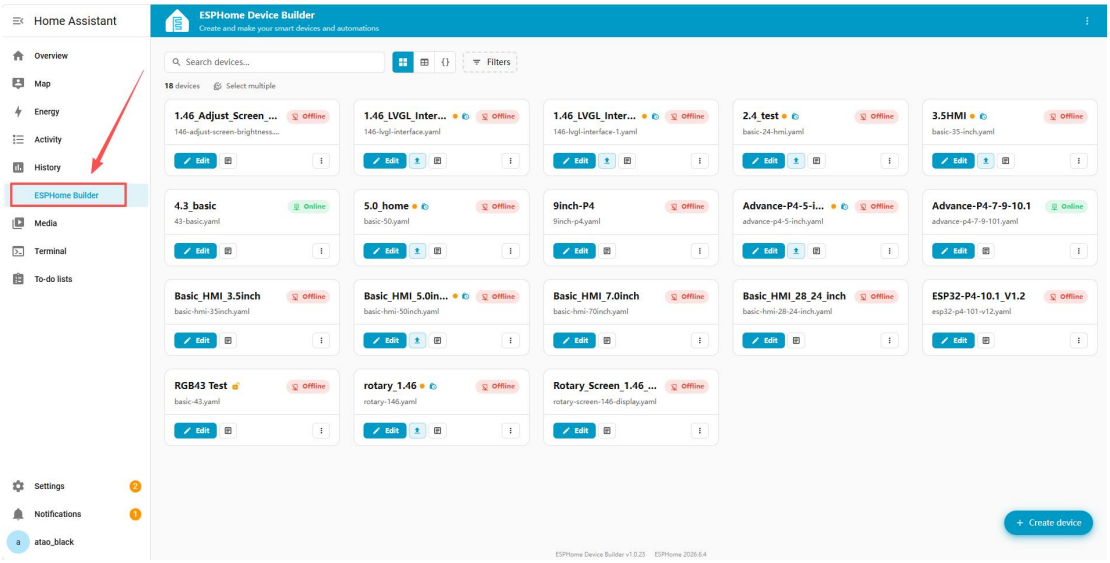




So we have successfully installed ESPHome. From now on, we can use ESPHome to edit the code and achieve the functions we want.

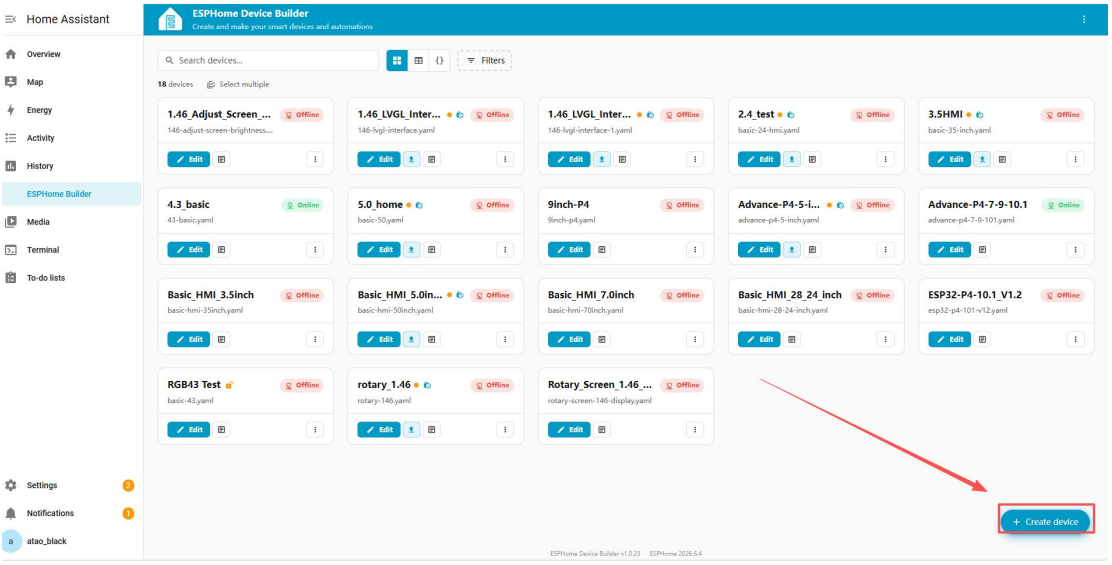
# 6. New Project

Once the installation is complete, we can start adding devices.

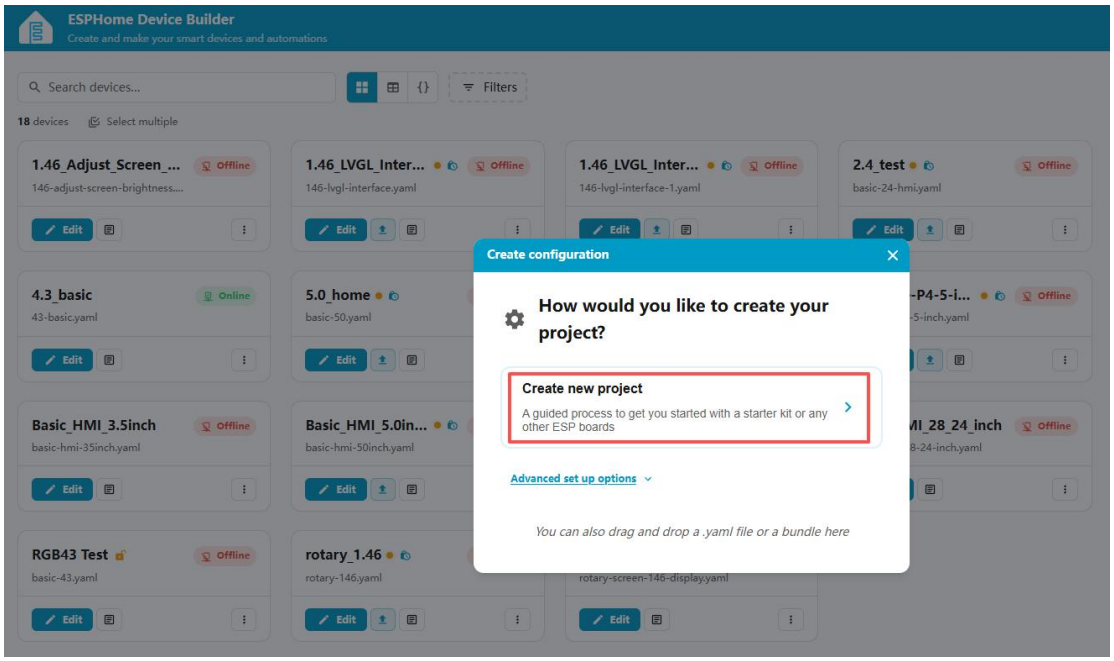
Come to ESPHome Builder



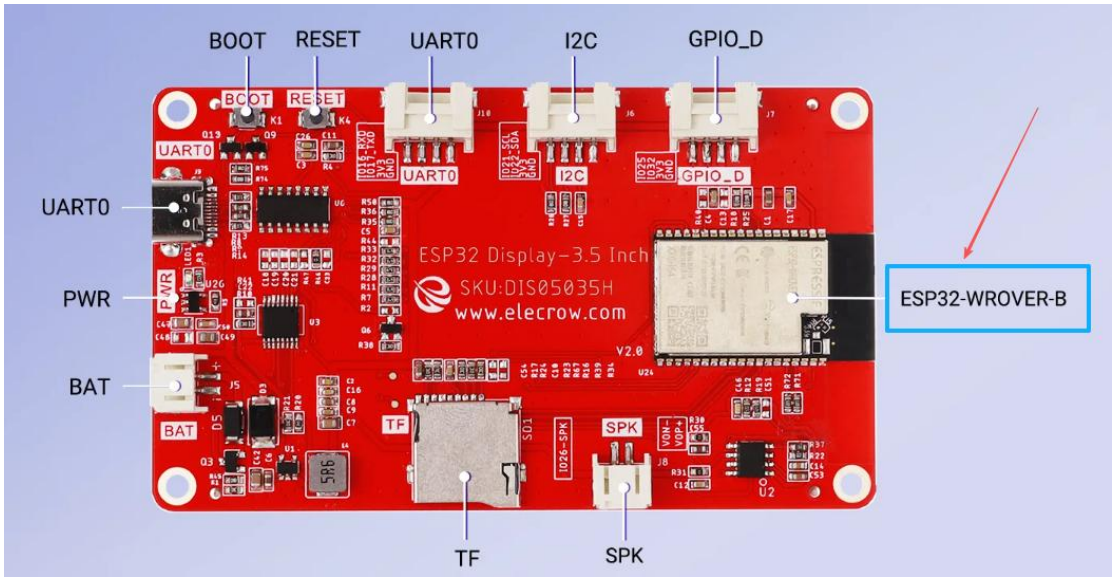

Click to add a new device



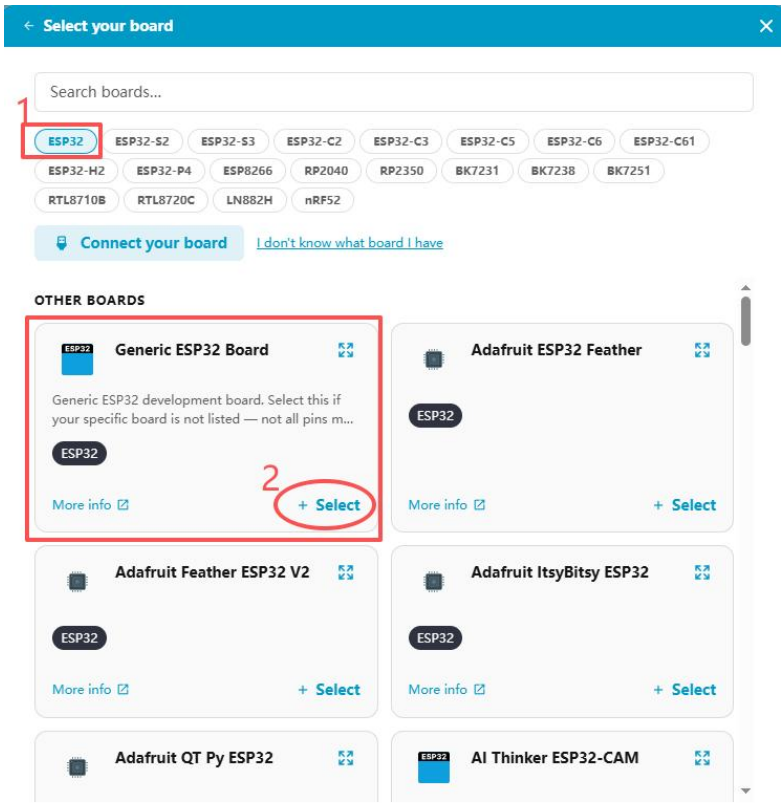

Click to create



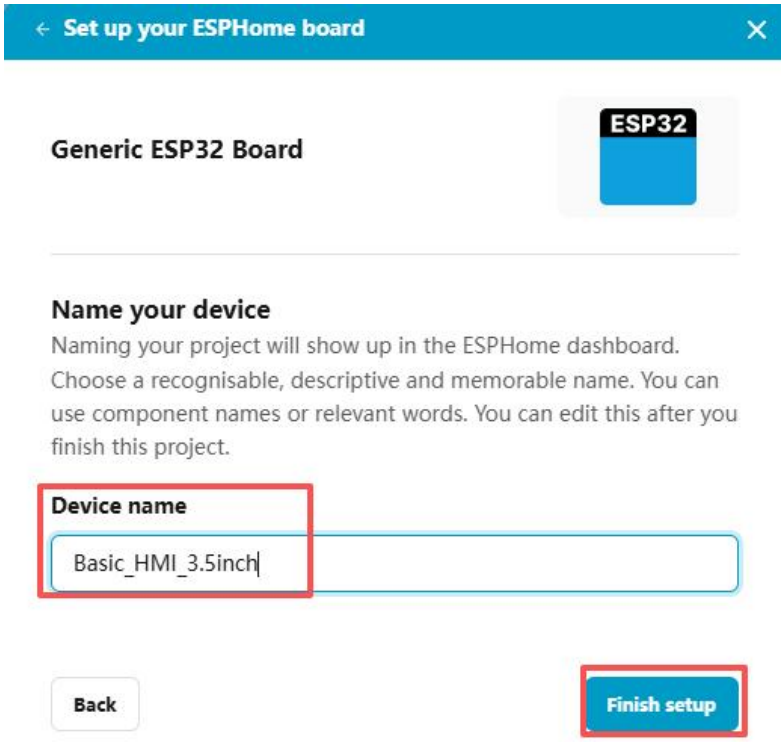

Next, based on the size of the CrowPanel Basic HMI ESP32 Display (2.4-inch / 2.8-inch / 3.5-inch) you are using, select the corresponding ESP32 main control chip.




Choose the ESP32 chip used in our project

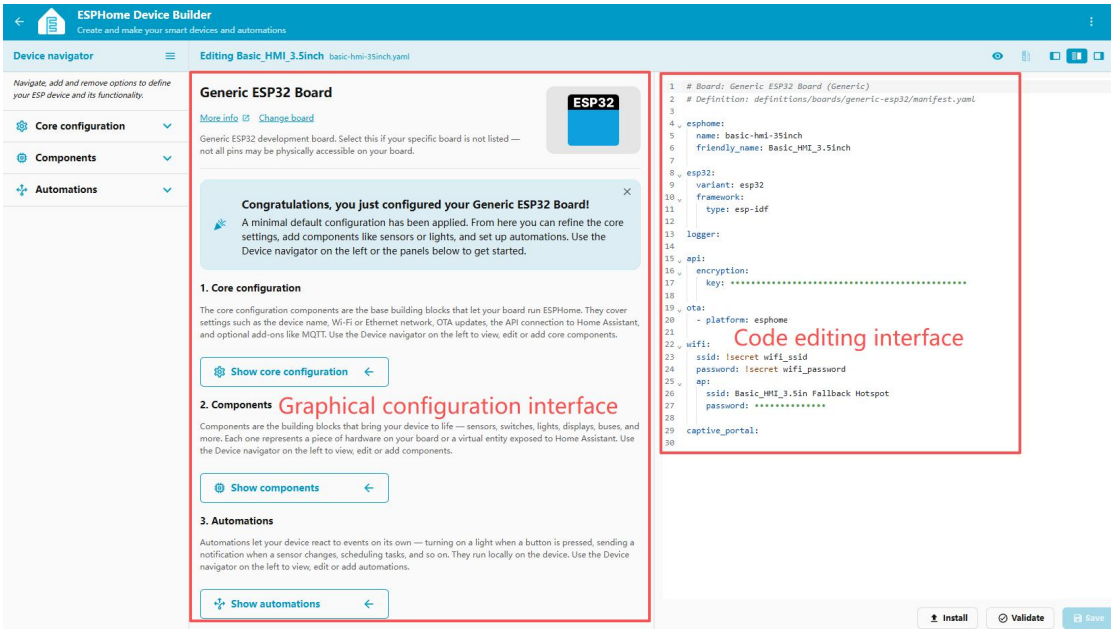



Set a name for this project.

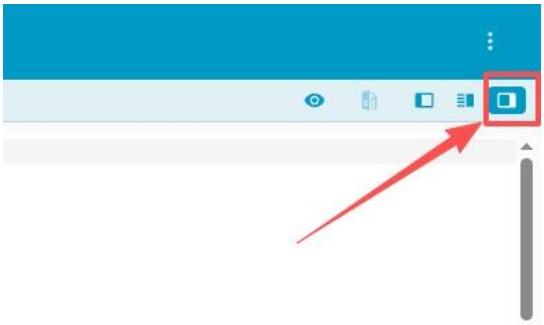



After the configuration is completed, the system will present a graphical configuration interface. This textbook will mainly guide you to learn how to write the programs in the right code column. In the subsequent operations, you will find that the final effect achieved by manually writing the

code and operating through the graphical interface is completely the same.



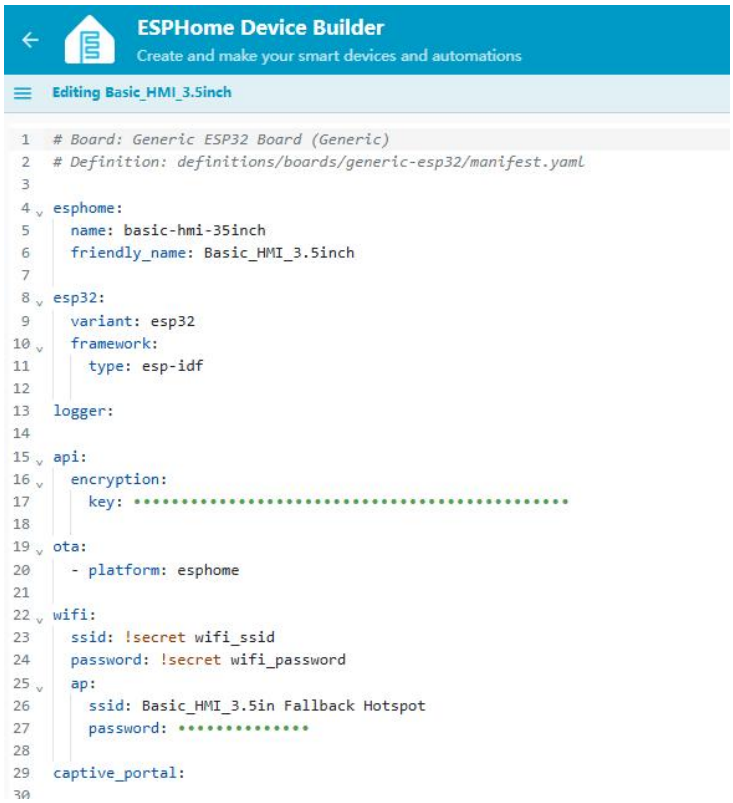

So we simply click on this button at the top right corner and keep only the part for code writing.



## 7. Configuration Code

Then, return to the main interface, find the Basic_HMI_3.5inch you just created, click "EDIT", and enter the code editor.

This code is automatically generated based on the previous steps. Next, we will make replacements in it, which will help optimize the code for more efficient operation.




You can click here to download the code in this course, which will help you achieve the related functions.

3.5-Size Code:
https://github.com/Elecrow-RD/CrowPanel-3.5-HMI-ESP32-Display-480x320/tree/master/example/V2.2/ESPHome

2.8-Size Code:

https://github.com/Elecrow-RD/CrowPanel-2.8-ESP32-HMI-320x240/tree/master/example/V2.1/ESPHome

2.4-Size Code:
https://github.com/Elecrow-RD/CrowPanel-2.4-ESP32-HMI-320x240/tree/master/example/V2.1/ESPHome

**Please Note: The core logic of this lesson is demonstrated using the 3.5-inch screen code as an example. If you are using a different screen size, the source code will vary due to differences in hardware configurations and display drivers.**

**Please be sure to refer to the matching code for your specific screen size as you follow along.**

Next, you can replace the relevant content in ESP32 as needed. You may also add the corresponding ESPHome configuration here, switch the ESP32 platform as required, and enable

PSRAM support.

**basic-hmi-35inch.yaml**

```yaml
esphome:
  name: basic-hmi-35inch
  platformio_options:
    build_flags: "-DBOARD_HAS_PSRAM"
    board_build.esp-idf.memory_type: qio_opi
    board_build.flash_mode: dio
  friendly_name: Basic_HMI_3.5inch

esp32:
  board: esp32dev
  framework:
    type: arduino

psram:
  mode: quad
  speed: 80MHz

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "a/GlTK39FHKhf3WQqe0FIbrHIRQ+HRMYRQaRKIj7pIQ="

ota:
  - platform: esphome
    password: "e468a4ee6c34878f77f04e0850afb464"

wifi:
  ssid: "ThinkNode-G4_2a2d82"
  password: "elecrow.com"

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-35Inch"
    password: "S8EOSGdtn8ZT"

captive_portal:
```

Here, I have modified the ESP32 section to make it compatible with the ESP32 main controller. Remember to replace your own Wi-Fi name and password.

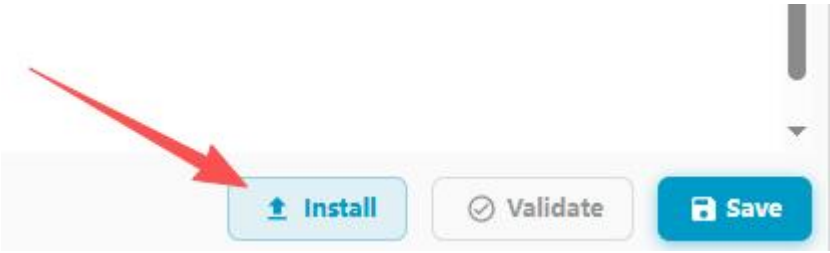
Keep them all within the same local area network.

**basic-hmi-35inch.yaml**

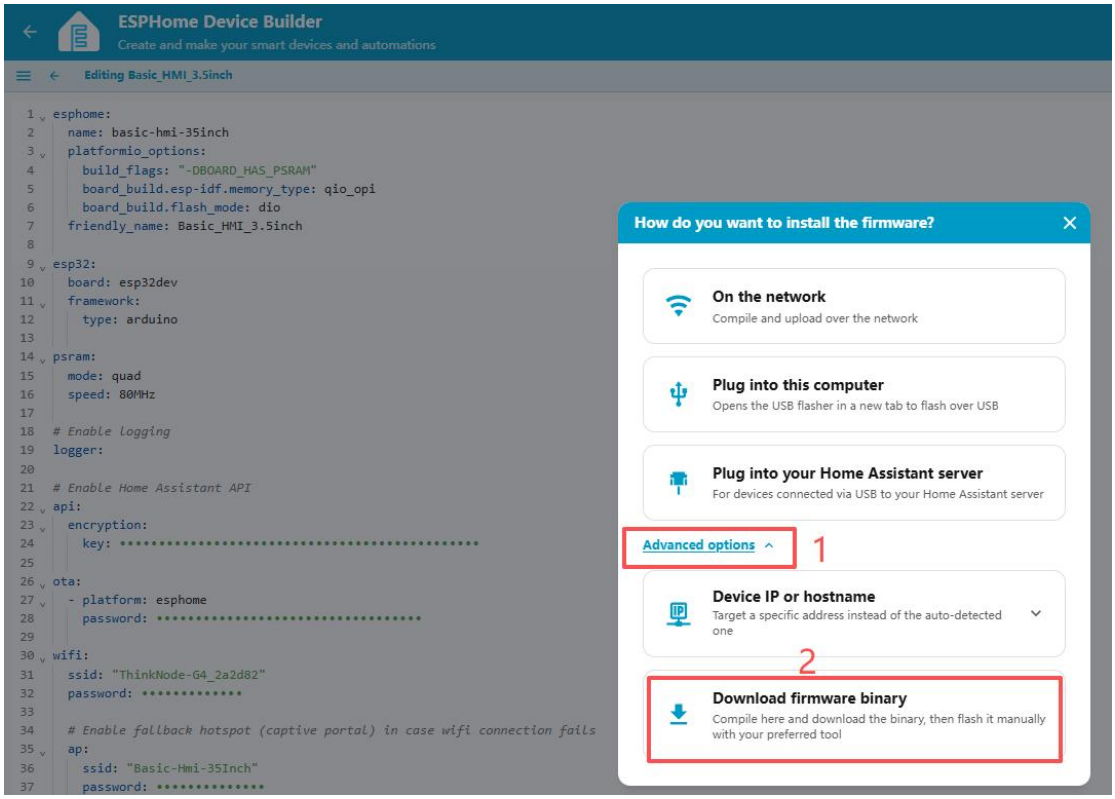
```yaml
esphome:
  name: basic-hmi-35inch
  platformio_options:
    build_flags: "-DBOARD_HAS_PSRAM"
    board_build.esp-idf.memory_type: qio_opi
    board_build.flash_mode: dio
  friendly_name: Basic_HMI_3.5inch

esp32:
  board: esp32dev
  framework:
    type: arduino

psram:
  mode: quad
  speed: 80MHz

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "a/GlTK39FHKhf3WQqe0FIbrHIRQ+HRMYRQaRKIj7pIQ="

ota:
  - platform: esphome
    password: "e468a4ee6c34878f77f04e0850afb464"

wifi:
  ssid: "ThinkNode-G4_2a2d82"
  password: "elecrow.com"

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-35Inch"
    password: "S8EOSGdtn8ZT"

captive_portal:
```

## 8. First upload of code

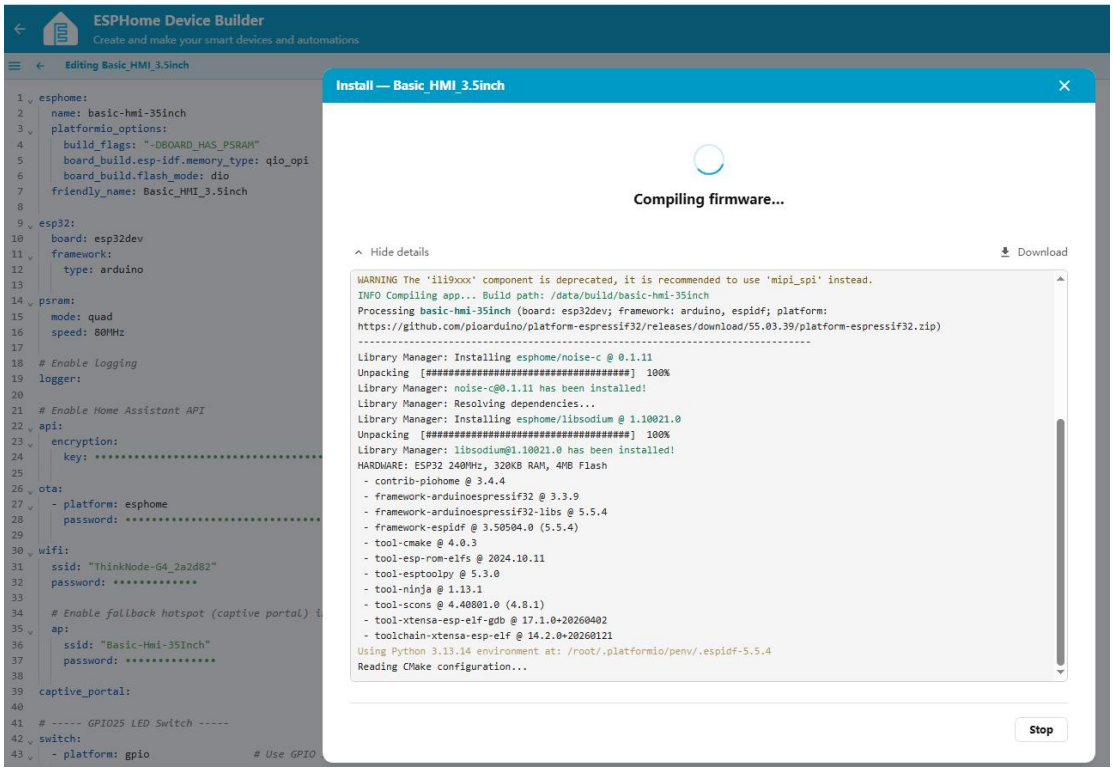
After the code replacement is completed, click the "Install" button at the bottom right corner.



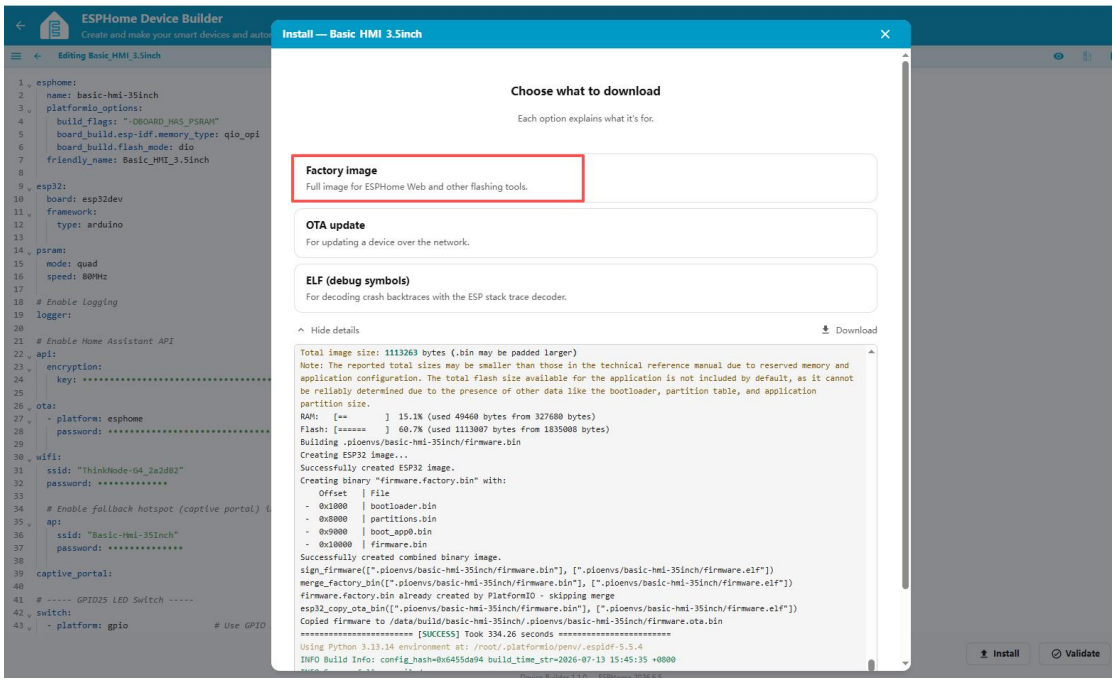

Subsequently, select the appropriate firmware burning method and carry out the first code upload.



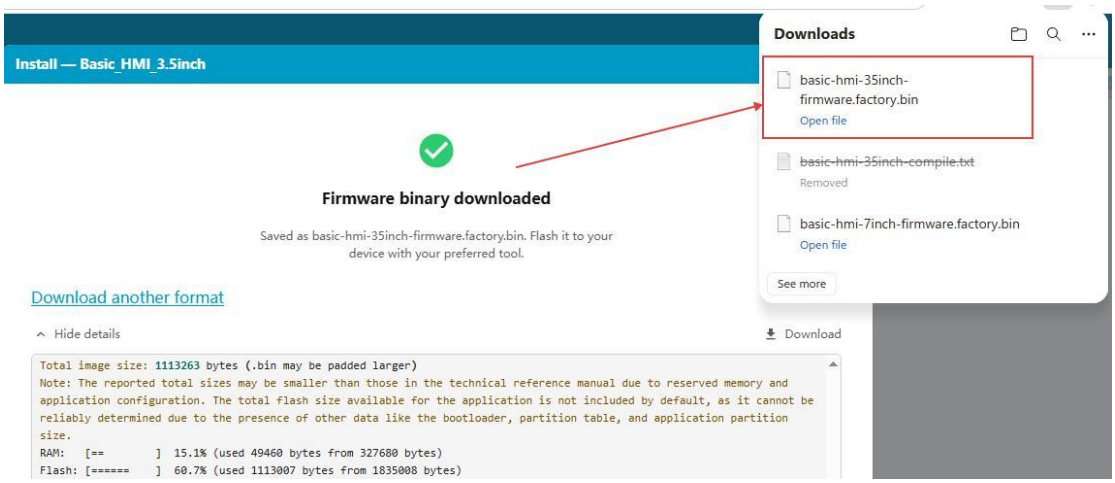

Wait for a few minutes until the installation is complete.The first installation and download process may take some time as it involves downloading necessary tools and libraries and compiling the code.



Then, after the download is completed, select "Factory image".




Once the download is complete, you will see the .bin file.




Remember the path of this .bin file.

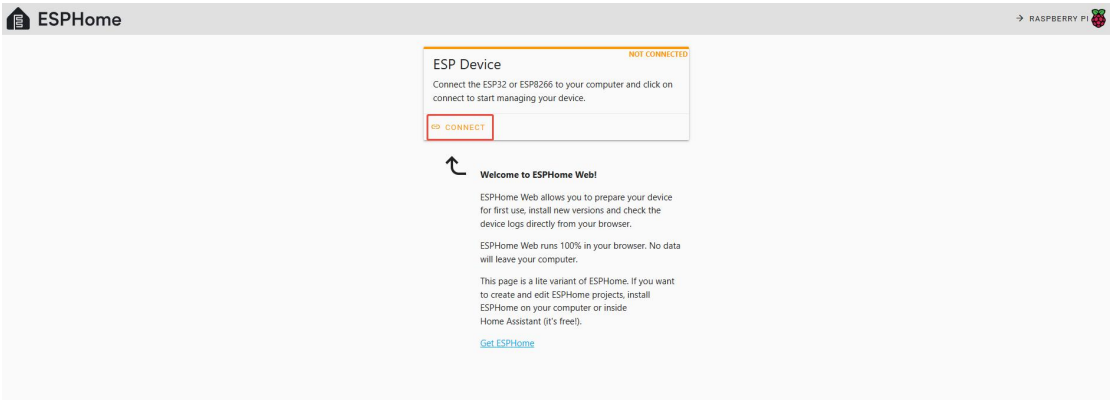
Open the following website:

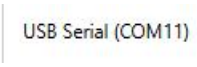
https://web.esphome.io/?dashboard_wizard

Next, we will flash this .bin file into the CrowPanel Basic HMI ESP32 Display (2.4-inch / 2.8-inch / 3.5-inch) .

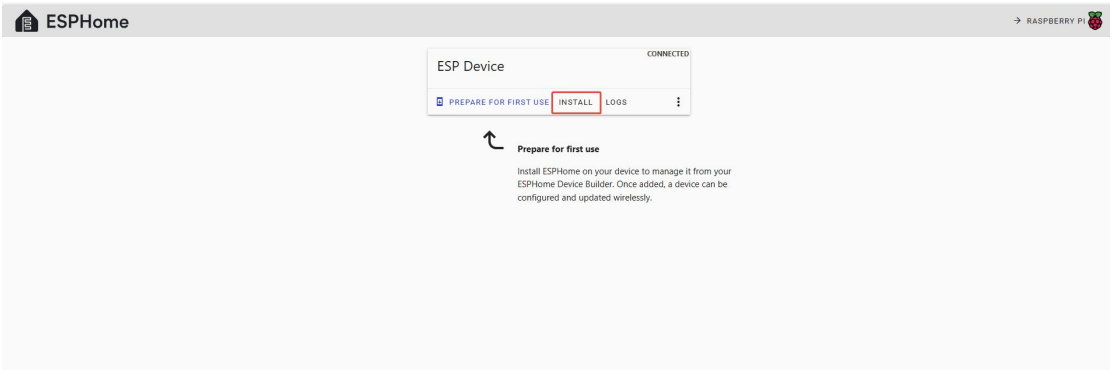
Connect the CrowPanel Basic HMI ESP32 Display (2.4-inch / 2.8-inch / 3.5-inch) to your computer.

Click "Connect", select the COM port, and connect it.

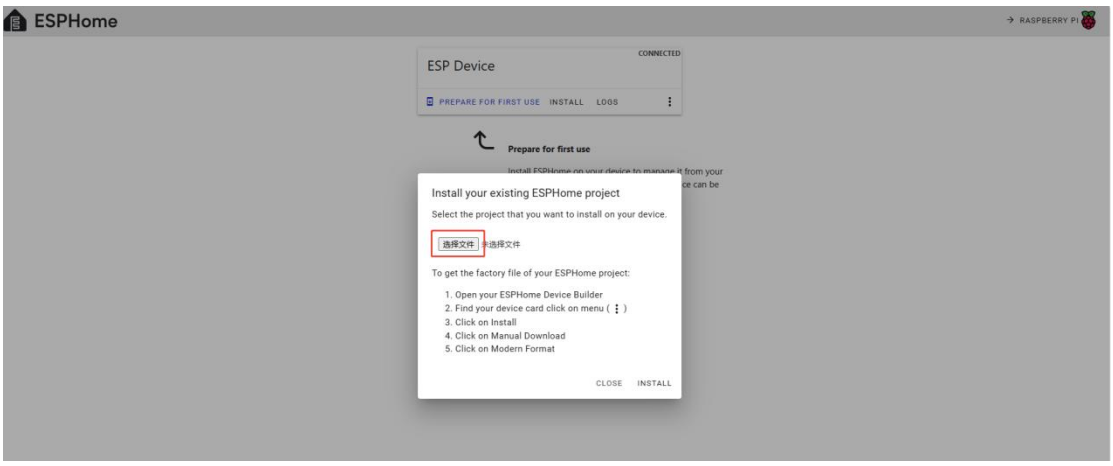


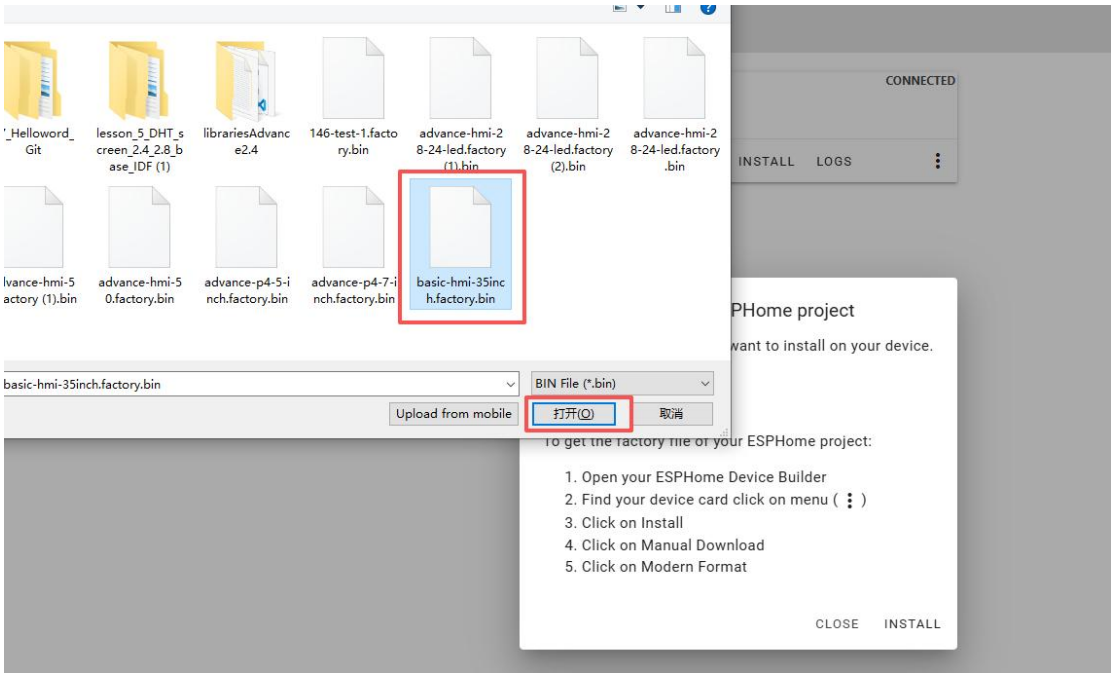



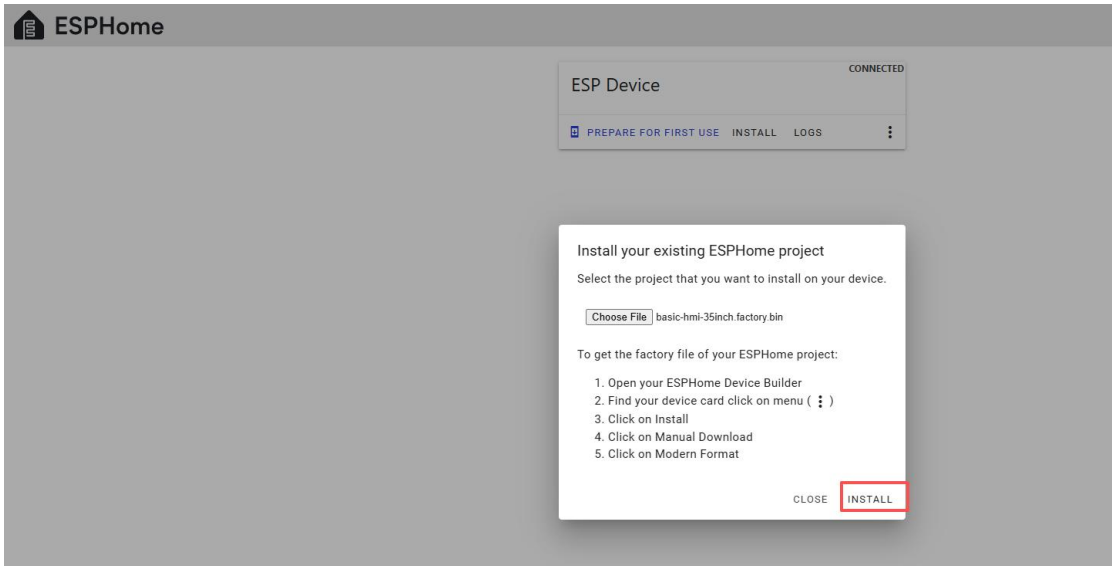

After connecting the CrowPanel Basic HMI ESP32 Display (2.4-inch / 2.8-inch / 3.5-inch) ,click "Install".



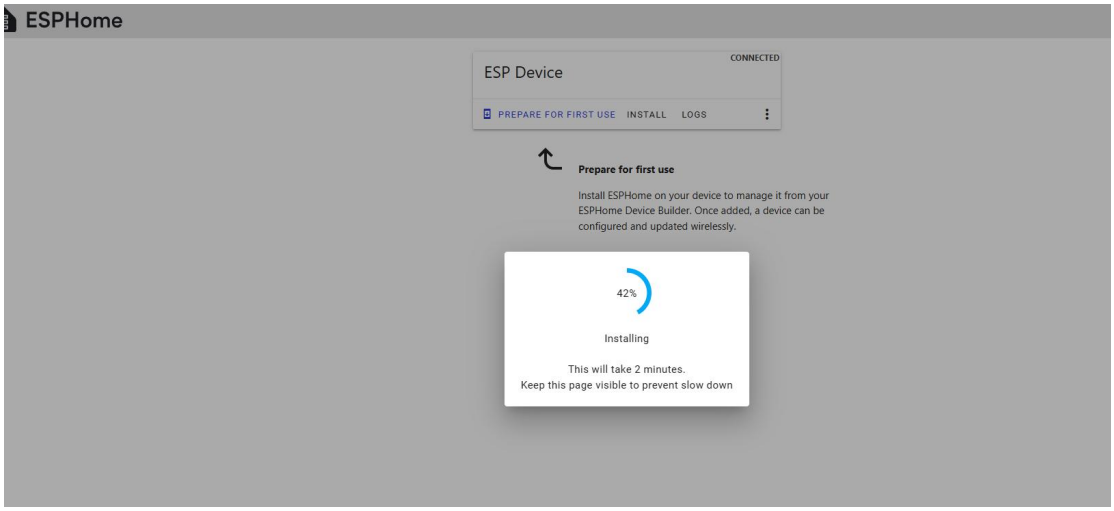

Add the .bin file you just downloaded, then click "Install".



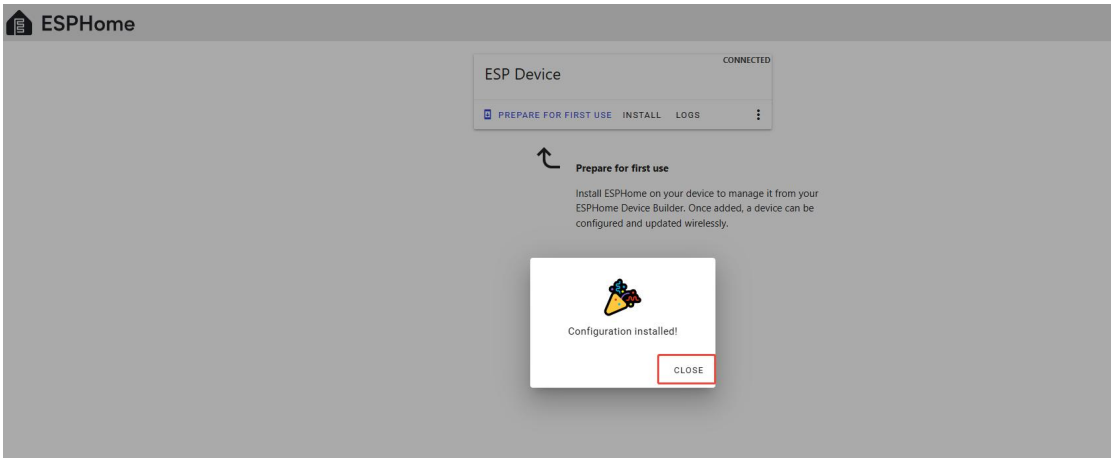



Wait for a few minutes.

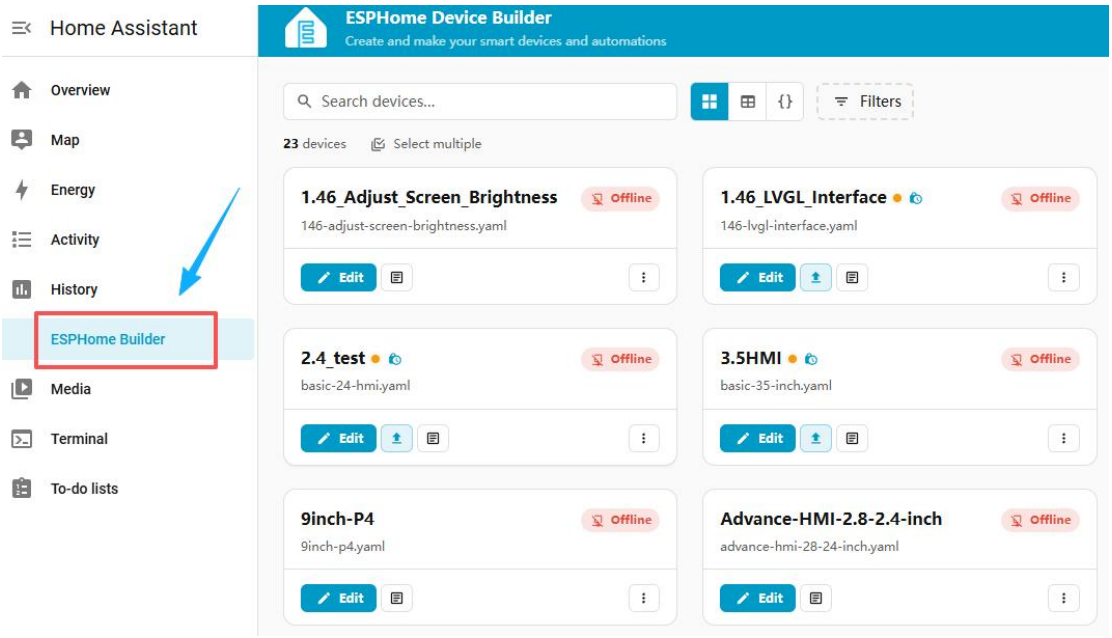



After the installation is complete, click "Close".

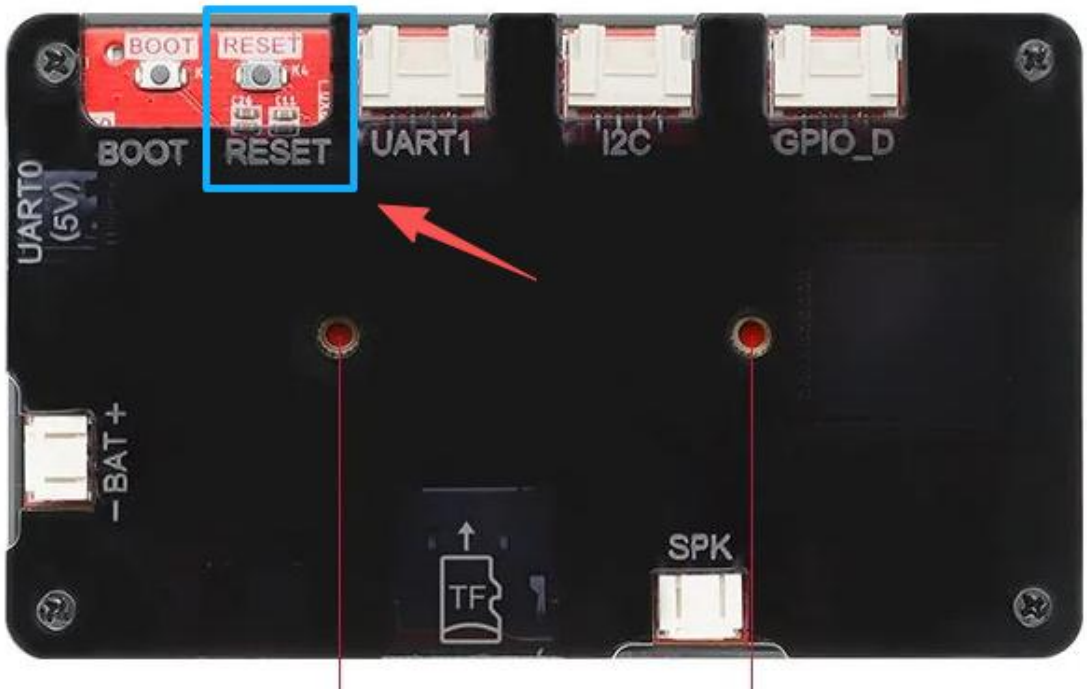



## 9. Observe the Wi-Fi connection status of the equipment

After successfully flashing the .bin file, return to the ESPHome page in Home Assistant.




Press the RESET button on the back of the product to reset it once.




And you should see the device you created earlier show as ONLINE in the top right corner.

If the "ONLINE" status is not displayed, please make sure that your Raspberry Pi 5, CrowPanel Basic HMI ESP32 Display (2.4-inch / 2.8-inch / 3.5-inch), and the WIFI you are using are all within the same local network.

Once your device is activated, you can begin writing code to implement your features.

First, this session focuses on collecting temperature and humidity data, which can be viewed historically in the ESPHome backend. It also adds the function of remotely controlling the light.

We hope the features in this session will help you better understand the convenience of ESPHome in smart homes.

Next, let's get started.

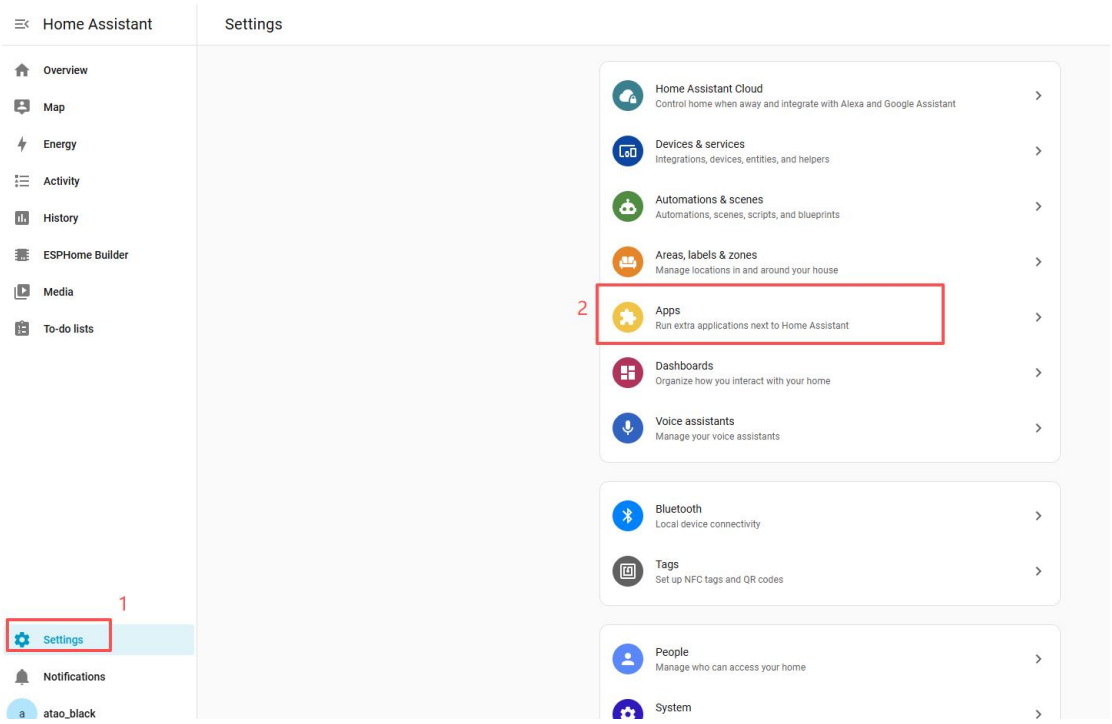
## 10.upload pictures

You can see the materials on the screen, which we need to use on the ESPHome platform, so they need to be uploaded to the ESPHome platform.

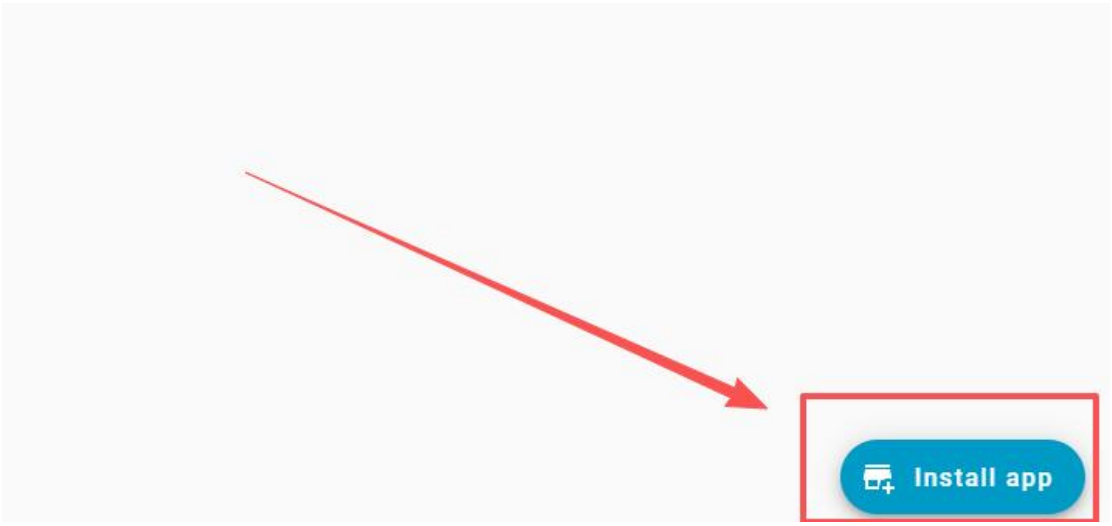
Click here to download the materials shown on our screen.
https://github.com/Elecrow-RD/CrowPanel-3.5-HMI-ESP32-Display-480x320/tree/master/example/V2.2/ESPHome/Materials

After downloading the materials we provide, you need to download a tool to upload these materials to the ESPHome platform.

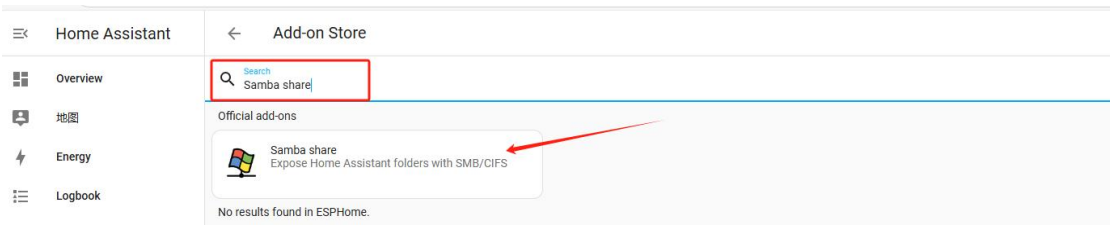
Click Settings and select Apps



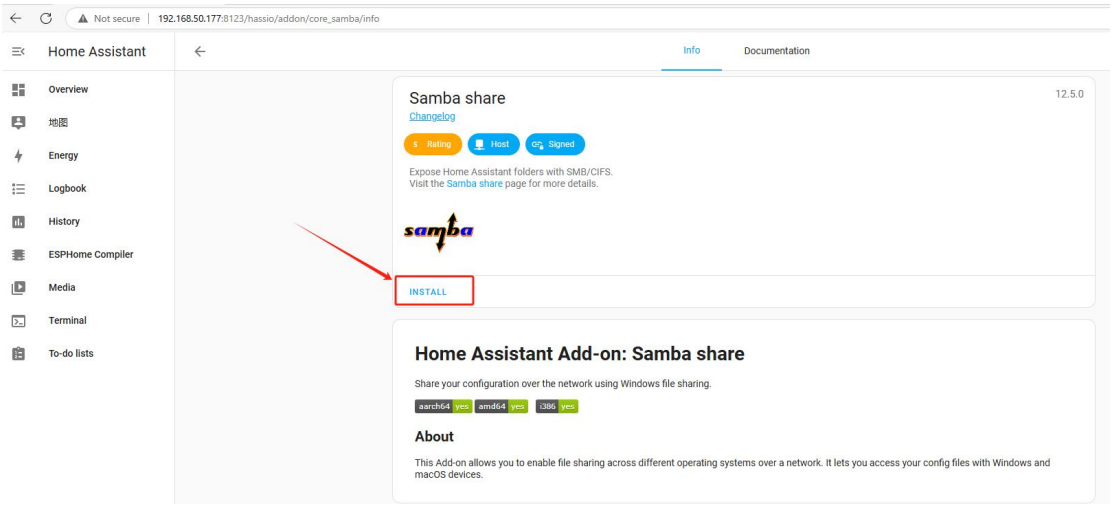

Then, click the “Install app” at the bottom right corner.



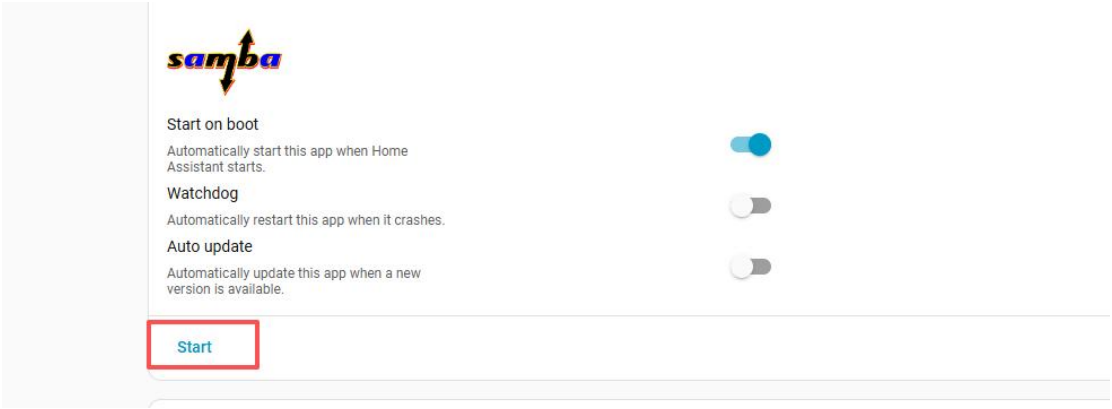

Search for Samba share at the top



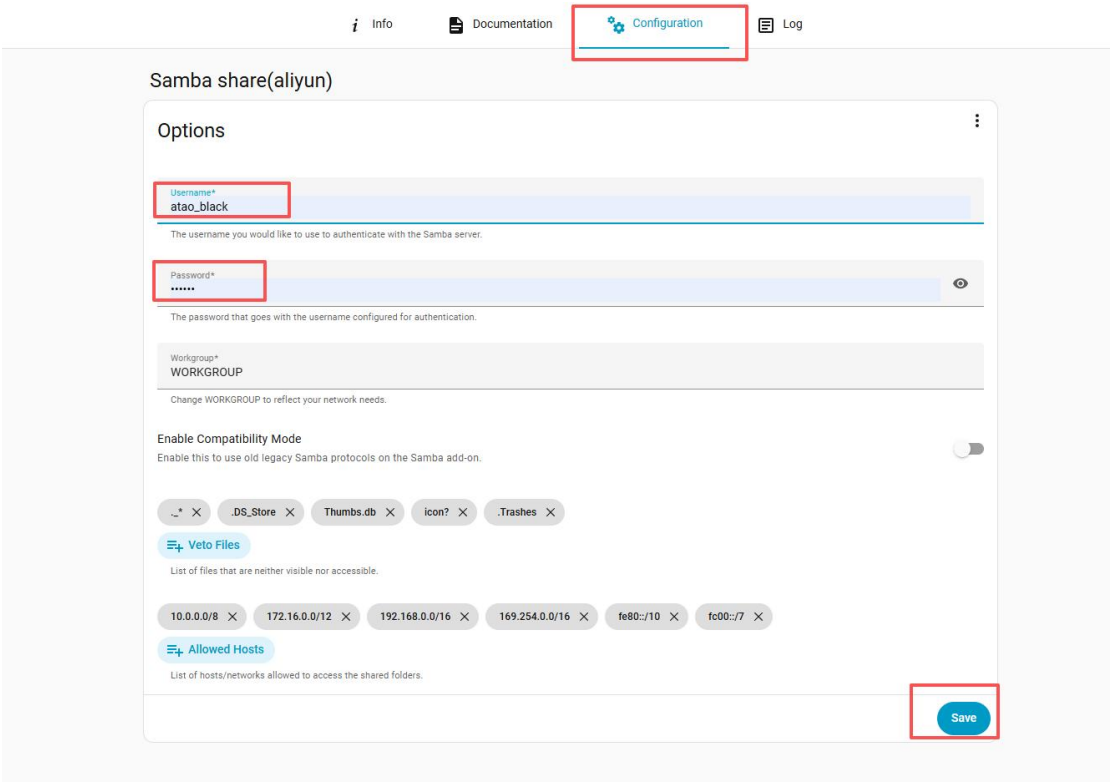

Click INSTALL to install it



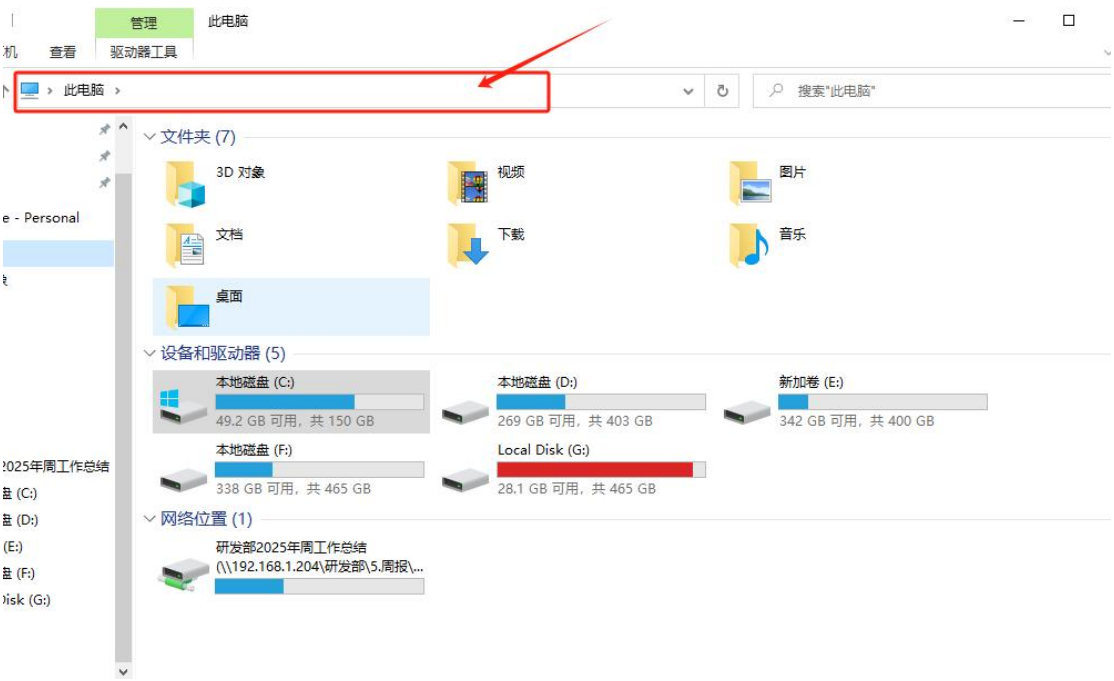

After the installation is completed, remember to start it.



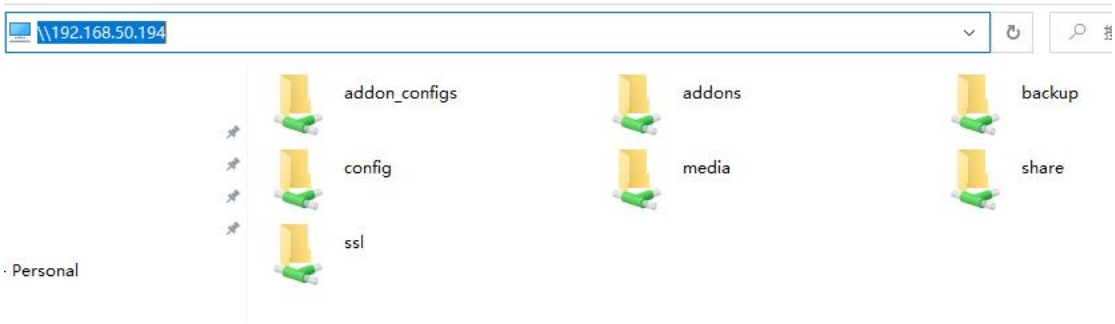

After installation, configure your Samba share with your own username and password—please remember them!



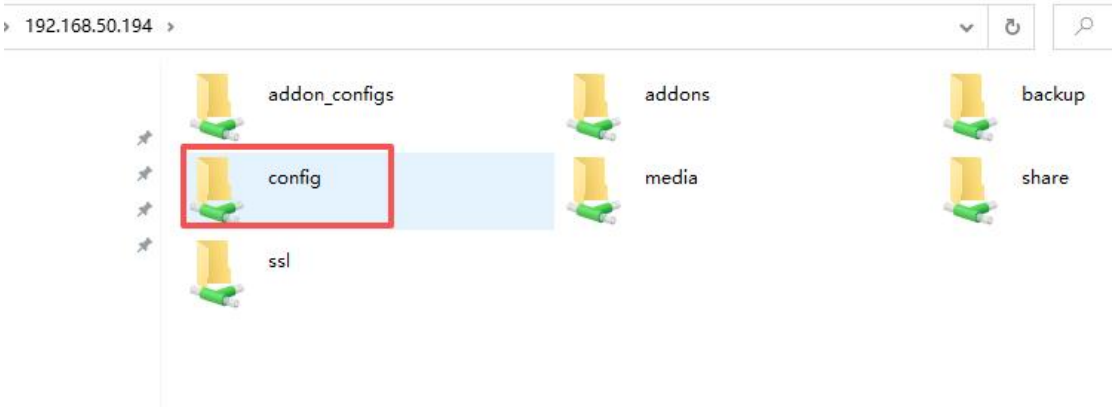

After saving, go to the File Explorer on your computer



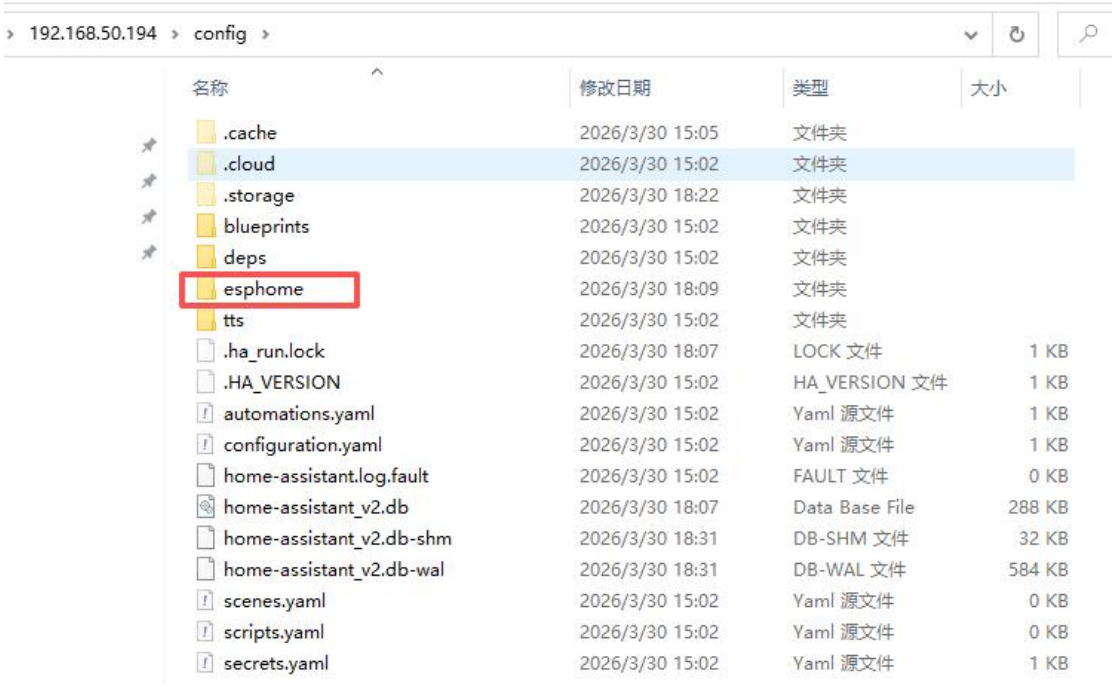

Enter “\\ + your Home Assistant IP address”




Go to config




Then choose esphome




Put the materials you just downloaded into the esphome folder

That completes the mission of Samba share.

## 11. Edit the code and complete the functionality

Next, you can edit your code to implement the simple smart home functions we mentioned.

Click "EDIT" to start writing the code.

You can use the code we provided earlier.

After downloading the code, you can copy it into your project.

Next, let’s talk about things to pay attention to while using the code.

Let's take a look at the code.

## 1.The ESPHome section

The "name" within the top block of EspHome is the unique identifier of the device, used for WiFi networking, OTA upgrades, Home Assistant device identification, and log printing. It must remain unique throughout the process; "friendly_name" is the intuitive device name displayed on the smart home panel, facilitating the distinction between multiple screen devices.

The "platformio_options" inside is the custom compilation instructions directly passed to the underlying PlatformIO compilation framework: build_flags: "-DBOARD_HAS_PSRAM" adds global compilation macros to inform the Arduino underlying driver chip that it is equipped with external PSRAM, allowing all screen graphics caching, large-sized images, multiple font sizes, and touch frame buffers to be stored in the external large-capacity PSRAM, avoiding screen rendering at 480×320 high resolution on the 480×320 high-resolution color screen when occupying a small amount of RAM in the ESP32, resulting in screen flickering, system crashes, and repeated restarts; board_build.esp-idf.memory_type: qio_opi configures Flash to use the four-line high-speed QIO_OPI read-write mode, significantly improving the speed of firmware reading and image material loading; board_build.flash_mode: dio sets the Flash dual-line drive mode, adapting to the mainstream ESP32 development board hardware Flash circuit, preventing the inability to start normally after flashing, or failure to read material files after flashing, and the entire parameter combination is an indispensable stabilization configuration for the 3.5-inch large-screen human-machine interface driver.

× basic-hmi-35inch.yaml

```yaml
esphome:
  name: basic-hmi-35inch
  platformio_options:
    build_flags: "-DBOARD_HAS_PSRAM"
    board_build.esp-idf.memory_type: qio_opi
    board_build.flash_mode: dio
  friendly_name: Basic_HMI_3.5inch

```

## 2.esp32

This configuration is used to declare that the main control chip of the current project is ESP32, specify the hardware development board model as the universal esp32dev development board adaptation configuration, and select the firmware underlying operating framework as the Arduino framework. The esp32dev is equipped with general ESP32 pin definitions, Flash partitioning, clock and other basic hardware parameters, which can be compatible with most standard ESP32 development boards. There is no need to customize pins and hardware underlying parameters separately. Choosing the Arduino framework is more concise in syntax compared to the native ESP-IDF framework, and the graphics and peripheral driver library are more compatible. It has better compatibility with external peripherals such as ILI9488 display, XPT2046 touch, and various sensors, reducing the development and debugging difficulty of large-screen HMI interfaces.

```yaml
esp32:
  board: esp32dev
  framework:
    type: arduino
```

## 3.psram + logger + api + ota

This section sequentially completes four basic supporting configurations: external PSRAM hardware driver, system log, HomeAssistant encrypted communication, and wireless OTA upgrade: The PSRAM block enables four-way parallel external memory operation at 80 MHz high frequency. Combined with the previous compilation macros, it stores the large screen frame cache and image font resources in the external memory, solving the memory overflow and screen flickering restart problems of the 480×320 color screen; logger enables serial port log output, printing sensor, touch, screen, and network operation information, facilitating hardware interaction and program abnormal debugging; the API block enables the HA local communication interface and configures the exclusive encryption key to achieve secure two-way data communication between the device and the smart home platform, supporting panel temperature and humidity data reporting, LED and backlight remote control; the OTA configuration uses the native wireless upgrade channel of ESPHome and sets the access password. Without the USB cable, it allows for remote flashing and firmware update, and the password is used to prevent unauthorized devices from randomly updating the device program.

× **basic-hmi-35inch.yaml**

```yaml
esphome:
  platformio_options:
    board_build.flash_mode: dio
  friendly_name: Basic_HMI_3.5inch

esp32:
  board: esp32dev
  framework:
    type: arduino

psram:
  mode: quad
  speed: 80MHz

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "a/GlTK39FHKhf3WQqe0FIbrHIRQ+HRMYRQaRKIj7pIQ="

ota:
  - platform: esphome
    password: "e468a4ee6c34878f77f04e0850afb464"
```

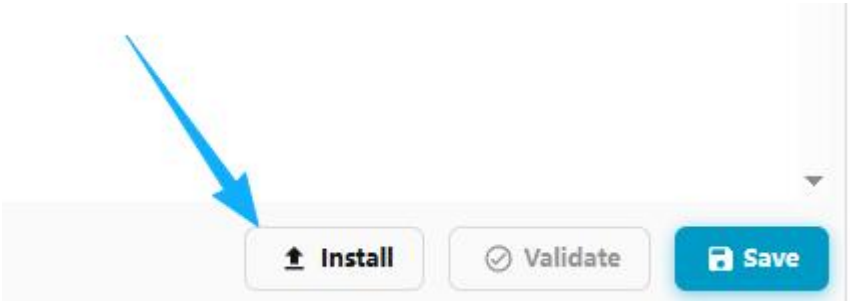
Finally, it's ota:

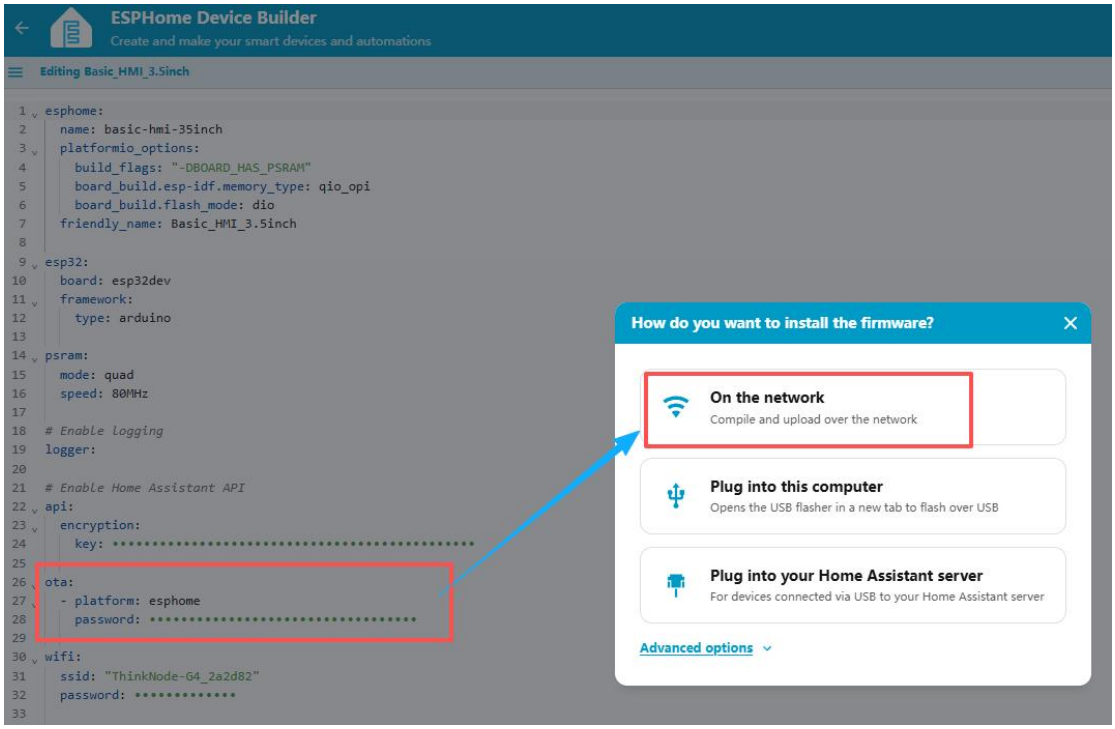
```yaml
ota:
  - platform: esphome
    password: "e468a4ee6c34878f77f04e0850afb464"
```

This is the wireless upgrade feature (OTA), which is what you will be able to use in the future.

A password has been set, indicating that you will need to enter the password for firmware updates in the future to ensure that others cannot tamper with your device.

With this feature, you won't need to plug in a USB cable anymore. Instead, you can directly update the program via WiFi.






(The prerequisite is that your computer and the Wi-Fi are in the same local network, and your device is in the ONLINE state, indicating that the Wi-Fi connection is successful.)

## 4.wifi

This section describes the complete fault-tolerant configuration for the device's wireless network. First, fill in the name and access password of the main router WiFi. After powering on the device, it will prioritize connecting to this local area network to enable communication with Home Assistant, OTA upgrades, and data reporting. When the main WiFi signal is lost, the password is incorrect, or the network is disconnected, it will automatically activate the built-in AP hotspot mode, generate an independent hotspot, and set an exclusive connection password. Combined with the captive_portal component below, it can allow mobile phones to connect to the hotspot and automatically pop up the network configuration webpage. Without a serial port or a computer, it can re-modify the WiFi parameters. The entire mechanism ensures that the device can still provide a convenient re-networking entry when the network is abnormal, avoiding the situation where the device loses connection completely and cannot be managed after the network disconnection.

```yaml
wifi:
  ssid: "ThinkNode-G4_2a2d82"
  password: "elecrow.com"

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Basic-Hmi-35Inch"
    password: "S8EOSGdtn8ZT"
```

## 5. External LED

This configuration defines an external LED control channel through the GPIO switch component of ESPHome. The specified hardware pin is GPIO25.
The `id` is an internal unique identifier within the program, which can be used for reading or switching the status of the light by the touch logic, screen rendering, and HA automation.
The `name` is the entity name displayed on the Home Assistant panel.
`restore_mode: RESTORE_DEFAULT_OFF` indicates that the device will not remember the last LED status after power loss and restart. It is uniformly set to default off, enabling both local screen touch to switch the light and remote control via the smart home platform, achieving standardized smart home entity encapsulation for digital output IO.

```yaml
# ----- GPIO25 LED Switch -----
switch:
  - platform: gpio                # Use GPIO switch platform
    pin: GPIO25                   # LED connected to GPIO25
    id: led_switch                # Internal ID
    name: "External LED"          # Display name in ESPHome
    restore_mode: RESTORE_DEFAULT_OFF  # Restore to OFF after reboot
```

## 6. Screen backlight adjustment

This configuration utilizes the dedicated LEDC hardware PWM peripheral of the ESP32. It generates a pulse signal with adjustable duty cycle at the GPIO27 pin for screen backlight adjustment. Setting inverted to False indicates that the output level logic is not inverted, and a high level corresponds to brighter backlight. The id is set to "backlight_pwm" as the unique internal identifier. It is provided for the binding and invocation of the following light component, relying on the hardware PWM to achieve non-flashback and high-precision backlight brightness adjustment without occupying CPU resources. It is the underlying output channel definition for screen backlight dimming.

The "light" single-color lighting component is bound to the previous PWM output channel. It is encapsulated into a lighting entity recognizable by Home Assistant, with the external display name and internal program call ID set as "back_light".
At the same time, restore_mode:ALWAYS_ON is configured. When the device is powered on, the screen backlight will automatically light up. Eventually, it supports stepless adjustment of the backlight brightness on the smart home page.

```yaml
# ----- PWM Output (Backlight) -----
output:
  - platform: ledc                # Use ESP32 LEDC PWM output
    pin: GPIO27                   # Backlight PWM output pin
    inverted: False               # Normal output (not inverted)
    id: backlight_pwm             # PWM output ID

# ----- Display Backlight -----
light:
  - platform: monochromatic       # Monochrome light component
    output: backlight_pwm         # Bind to PWM output
    name: "Display Backlight"     # Display name
    id: back_light                # Internal ID
    restore_mode: ALWAYS_ON       # Turn on automatically after reboot
```

## 7.font

This configuration preloads the Google Roboto sans-serif vector font built into ESPHome, generating 4 different-sized text resources at once and assigning them independent internal IDs, namely font4 (48), font3 (36), font2 (24), and font1 (12). The font resources are uniformly parsed and cached at device startup. During subsequent screen drawing and rendering, they can directly reference the corresponding IDs to print titles, numerical values, and small text explanations. There is no need to repeatedly load the font each time for text drawing, which saves computing power and memory. The unified vector font ensures clear and smooth text on the screen without jagged edges, and it meets the hierarchical display requirements for distinguishing titles, data, and auxiliary small text in large-screen UIs.

```yaml
# ----- Fonts -----
font:
  - file: "gfonts://Roboto"       # Google Roboto font
    id: font4                     # Font ID
    size: 48                      # Font size
  - file: "gfonts://Roboto"
    id: font3
    size: 36
  - file: "gfonts://Roboto"
    id: font2
    size: 24
  - file: "gfonts://Roboto"
    id: font1
    size: 12
```

## 8.DHT20

This section is divided into two parts: I2C hardware bus initialization and AHT20 temperature and humidity sensor acquisition. First, define the hardware I2C bus, specify data pin SDA as GPIO22 and clock pin SCL as GPIO21, and enable the automatic boot scan function to check if the I2C-mounted devices are properly recognized; then, based on the general driver of aht10, adapt

the AHT20 temperature and humidity chip, configure the standard I2C communication address of the chip as 0x38, split the two independent sensing data of temperature and humidity, respectively set the display names for Home Assistant and the internal IDs for program calls, set to collect and refresh the temperature and humidity values every 5 seconds, and the bus and sensor cooperate to achieve hardware data reading. The collected temperature and humidity data will be synchronously reported to the smart home platform.

```
# ----- I2C Bus (DHT20) -----
i2c:
  sda: GPIO22                    # SDA pin
  scl: GPIO21                    # SCL pin
  scan: true                     # Scan for I2C devices during boot

# ----- Sensor -----
sensor:
  - platform: aht10              # Use AHT10/AHT20 platform
    variant: AHT20               # Specify AHT20 sensor
    address: 0x38                # I2C address
    temperature:
      name: "HMI-35 Temperature" # Temperature sensor name
      id: temperature35          # Temperature sensor ID
    humidity:
      name: "HMI-35 Humidity"    # Humidity sensor name
      id: humidity35             # Humidity sensor ID
    update_interval: 5s          # Update every 5 seconds
```

## 9. touch screen

This section is divided into two main parts: initialization of the shared hardware SPI bus and the driver for the XPT2046 resistive touch screen. This is to achieve time-sharing and multiplexing of the SPI pins for the screen and touch chip, thereby saving GPIO resources: Firstly, define the hardware SPI bus and name it "spi_bus". Allocate the clock as CLK = GPIO14, the host output as MOSI = GPIO13, and the host input as MISO = GPIO33. Select the ESP32 hardware SPI interface to ensure high-speed communication and support for large-screen and large amount of image data transmission.

Subsequently, load the XPT2046 touch driver and bind it to the shared SPI bus. The touch chip select (CS) pin is connected to GPIO12. Comment out the optional interrupt pins, set a 50-millisecond scan interval for the touch signal, a 400-pressure threshold for filtering minor false touches, fill in the dedicated raw ADC coordinate calibration range for the screen, and enable X-axis mirroring correction to address the problem of the touch point position flipping left and right. In the on_touch touch callback, first print the converted coordinates and the original sampling value using logs for easy debugging, and then add logic for coordinate area judgment.

When the touch point falls within the rectangular area of X280~310 and Y235~265, automatically flip the GPIO25 external LED switch. The entire configuration completes the touch signal acquisition, coordinate calibration, and local interactive key functions. The touch coordinate data can also be used for screen UI linkage and Home Assistant automation calls.

```yaml
# ----- SPI Bus (Shared by Display and Touchscreen) -----
spi:
  - id: spi_bus                    # SPI bus ID
    clk_pin: GPIO14                # SPI clock pin
    mosi_pin: GPIO13               # SPI MOSI pin
    miso_pin: GPIO33               # SPI MISO pin
    interface: hardware            # Use hardware SPI

# ----- Touchscreen (XPT2046) -----
touchscreen:
  platform: xpt2046                # XPT2046 touch controller
  id: my_touch                     # Touchscreen ID
  spi_id: spi_bus                  # Use shared SPI bus
  cs_pin: GPIO12                   # Chip Select pin (modify according to your wiring)
  #interrupt_pin: GPIO36           # Optional T_IRQ pin for faster response
  update_interval: 50ms            # Touch scan interval
  threshold: 400                   # Touch pressure threshold
  calibration:                     # Touch calibration parameters
    x_min: 280                     # Raw X minimum
    x_max: 3860                    # Raw X maximum
    y_min: 340                     # Raw Y minimum
    y_max: 3860                    # Raw Y maximum
  transform:                       # Coordinate transformation
    mirror_x: true                 # Mirror X-axis
    mirror_y: false                # Do not mirror Y-axis
    swap_xy: false                 # Do not swap X and Y axes
  on_touch:                        # Trigger when screen is touched
    - lambda: |-
        ESP_LOGI("touch", "x=%d, y=%d, x_raw=%d, y_raw=%d",
                 touch.x,
                 touch.y,
                 touch.x_raw,
                 touch.y_raw);
    - if:
        condition:
          and:
            - lambda: 'return (touch.x >= 280 && touch.x <= 310);'   # X coordinate range
            - lambda: 'return (touch.y >= 235 && touch.y <= 265);'   # Y coordinate range
        then:
          - switch.toggle: led_switch    # Toggle LED switch
```

## 10.image

This section provides six PNG icon materials required for batch preloading of the UI, corresponding to six interface icons for LED lighting, humidity, device logo, LED off, temperature, and turning off the lights. Each item is specified by the file with the local image file name, an unique call id within the program, and the images are resized in advance during the compilation stage to a fixed pixel size. This avoids the consumption of computing power when the screen is running due to real-time resizing. The type is uniformly set to RGB565, which is the native 16-bit color standard of the ILI9488 color screen and has a fast rendering speed and is compatible with the PSRAM cache. After the device is powered on, all images will be parsed and stored in the external PSRAM at once. Subsequent screen rendering code can directly retrieve and draw through the id, achieving unified management of static resources for interface icons, providing graphic material support for the visual display of temperature, humidity, and LED status.

```yaml
# ----- Image Resources -----
image:
  - file: "light.png"             # LED ON icon
    id: on_light                  # Image ID
    resize: 50x50                 # Resize image
    type: RGB565                  # Image color format
  - file: "small_hum.png"         # Humidity icon
    id: small_hum
    resize: 50x75
    type: RGB565
  - file: "small_logo.png"        # Logo image
    id: small_logo
    resize: 250x52
    type: RGB565
  - file: "small_off.png"         # LED OFF icon
    id: small_off
    resize: 50x75
    type: RGB565
  - file: "small_temp.png"        # Temperature icon
    id: small_temp
    resize: 50x75
    type: RGB565
  - file: "no_light.png"          # Light OFF image
    id: no_light
    resize: 50x50
    type: RGB565
```

## 11. display

This section describes the complete driver for the 3.5-inch color screen and the dynamic UI rendering logic. The hardware layer is bound to the shared hardware SPI bus "spi_bus", with the specified driver model being ILI9488_A, a resolution of 480×320, configuring the SPI communication rate at 20 MHz and the timing mode as 0.

The chip select GPIO15, data command GPIO2, and hardware reset GPIO4 are allocated. The color inversion is disabled and the BGR color channel is adapted to the screen hardware. Lambda internally contains the automatic drawing code for each frame, using "it" as the screen drawing object. First, the entire screen is filled with a white background color. Then, at the coordinate (100,10), the device logo is drawn.

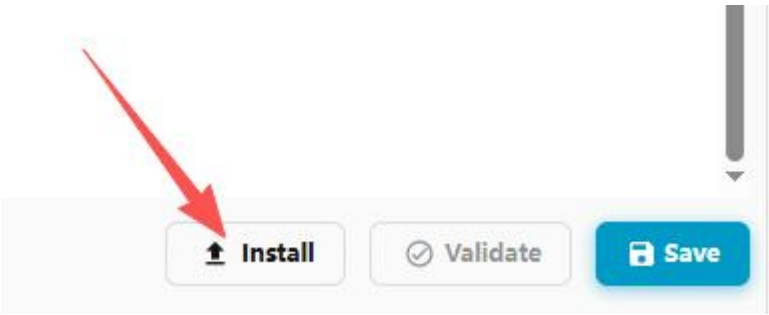
Next, the state of the led_switch switch is read for branch judgment. When the LED is on, a lit icon is drawn and the text "ON" is printed. When it is off, the extinguished icon and the text "OFF" are switched. Subsequently, temperature and humidity icons are drawn at corresponding coordinates, and the real-time values of the AHT20 sensor are retrieved, rounded to one decimal place, and printed below the icons.

Throughout, the previously defined images, fonts, and color IDs are reused to achieve an on-board visual human-machine interface that automatically refreshes upon power-on and synchronizes the status in real time.

```
# ----- Display -----
display:
  - platform: ili9xxx             # ILI9XXX display driver
    id: lcd_display               # Display ID
    model: ILI9488_A              # Display controller model
    spi_id: spi_bus               # Use shared SPI bus
    data_rate: 20MHz              # SPI communication speed
    spi_mode: mode0               # SPI mode
    cs_pin: GPIO15                # Chip Select pin
    dc_pin: GPIO2                 # Data/Command pin
    reset_pin: GPIO4              # Important! ILI9488 usually requires a reset pin
    dimensions: 480x320           # Display resolution
    invert_colors: false          # Do not invert colors
    color_order: bgr              # Color order
    lambda: |-
      it.fill(id(my_white));                          // Fill background with white
      it.image(100, 10, id(small_logo));              // Draw logo

      // Switch the icon according to the LED state
      if (id(led_switch).state) {
        it.image(85, 180, id(on_light));              // Draw ON icon
        it.printf(90, 240, id(font2), id(my_black), "ON");   // Display ON text
      } else {
        it.image(85, 180, id(no_light));              // Draw OFF icon
        it.printf(90, 240, id(font2), id(my_black), "OFF");  // Display OFF text
      }

      it.image(240, 160, id(small_temp));             // Draw temperature icon
      it.image(380, 160, id(small_hum));              // Draw humidity icon
      it.printf(245, 240, id(font2), id(my_black), "%.1f", id(temperature35).state);  // Display temperature
      it.printf(385, 240, id(font2), id(my_black), "%.1f", id(humidity35).state);     // Display humidity
```

## 12.Upload the complete code

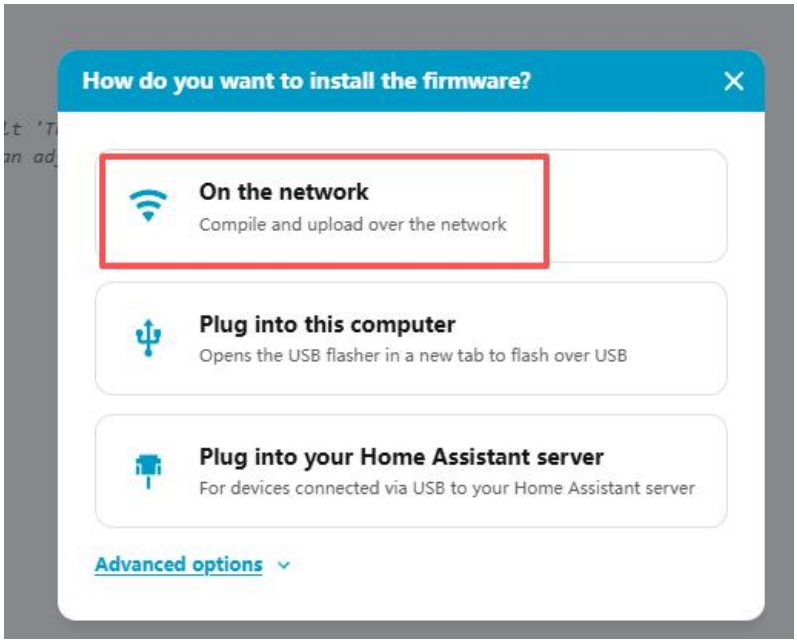
Now, all the codes have been prepared. Next, we will upload the codes.



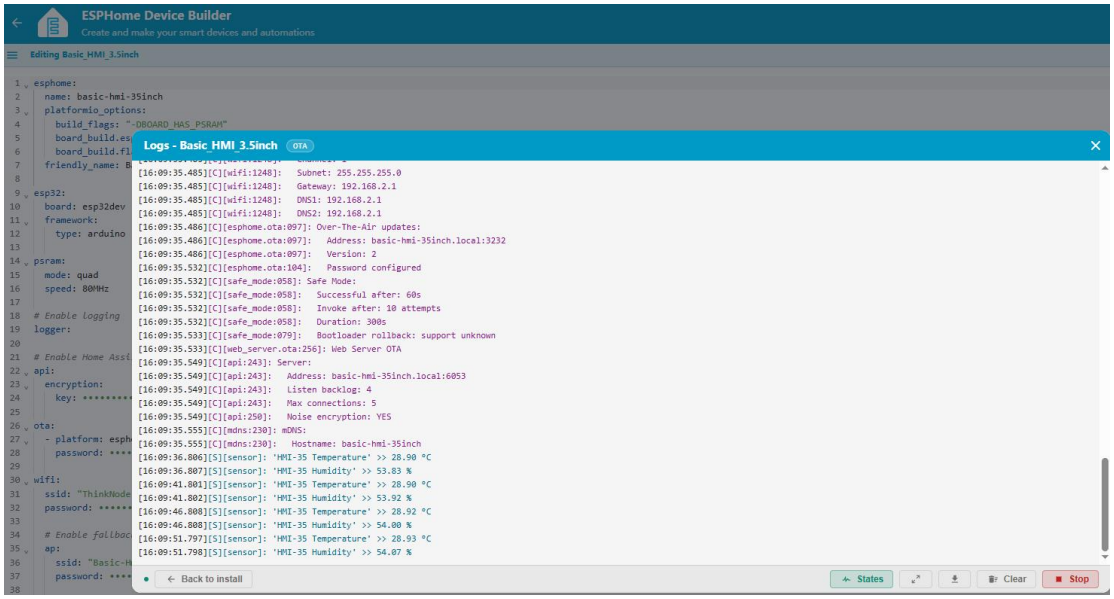

Then select "On the network"

From now on, as long as the network and OTA are functioning properly, just upload the code in this way.

( If this method doesn't work, then use the installation method described in "8. First upload of code". )

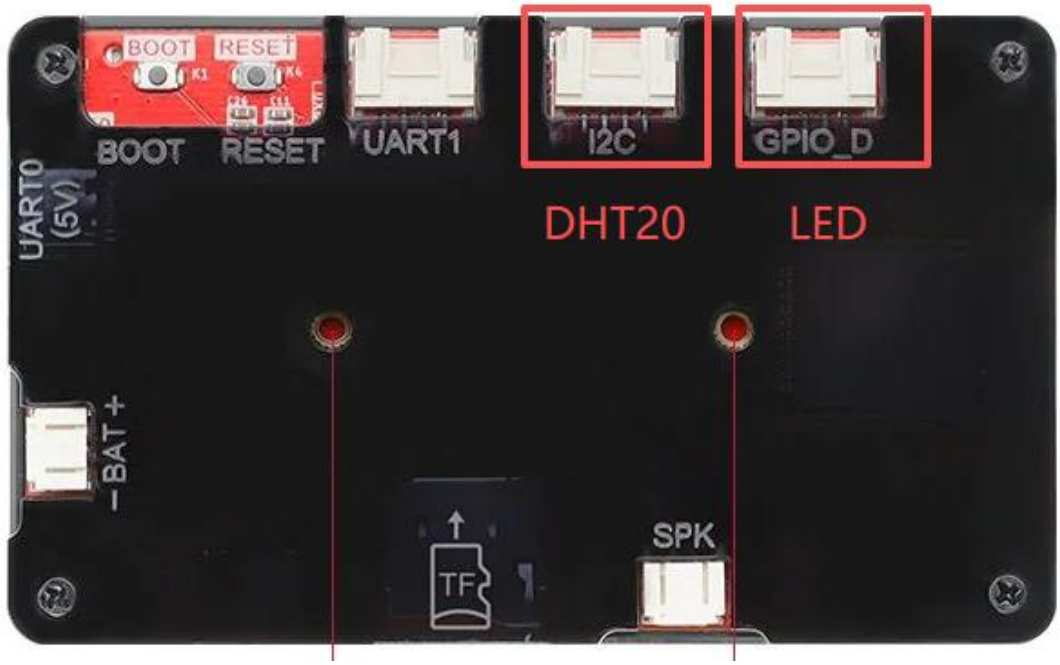



After the code is uploaded, the temperature and humidity values connected to the product will start to be printed.




Following the previous steps for "8. First upload of code", the code was successfully uploaded. After uploading the code, you will be able to see the target interface displayed on the screen.

Since you have used the DHT20 temperature and humidity sensor and the LED light, you need to connect these two sensors on the back.

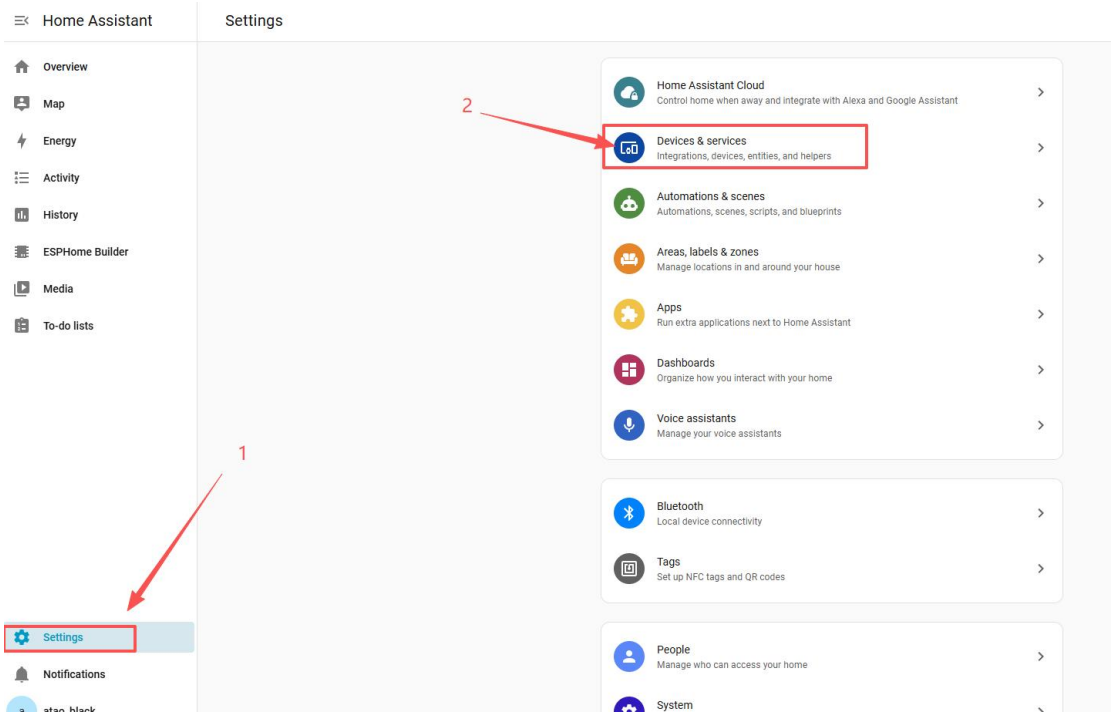



Then you will be able to see the real-time display of temperature and humidity data on the screen.

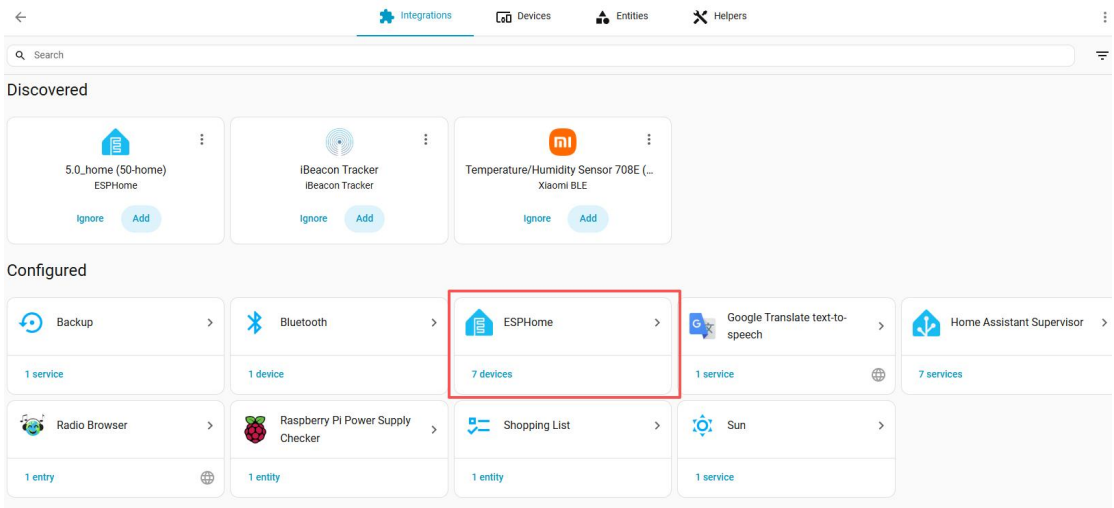
## 13.Remote observation and control

This is the local data we have observed. Next, let's take a look at how to remotely view the temperature and humidity data on the esphome platform and how to remotely control the LEDs.

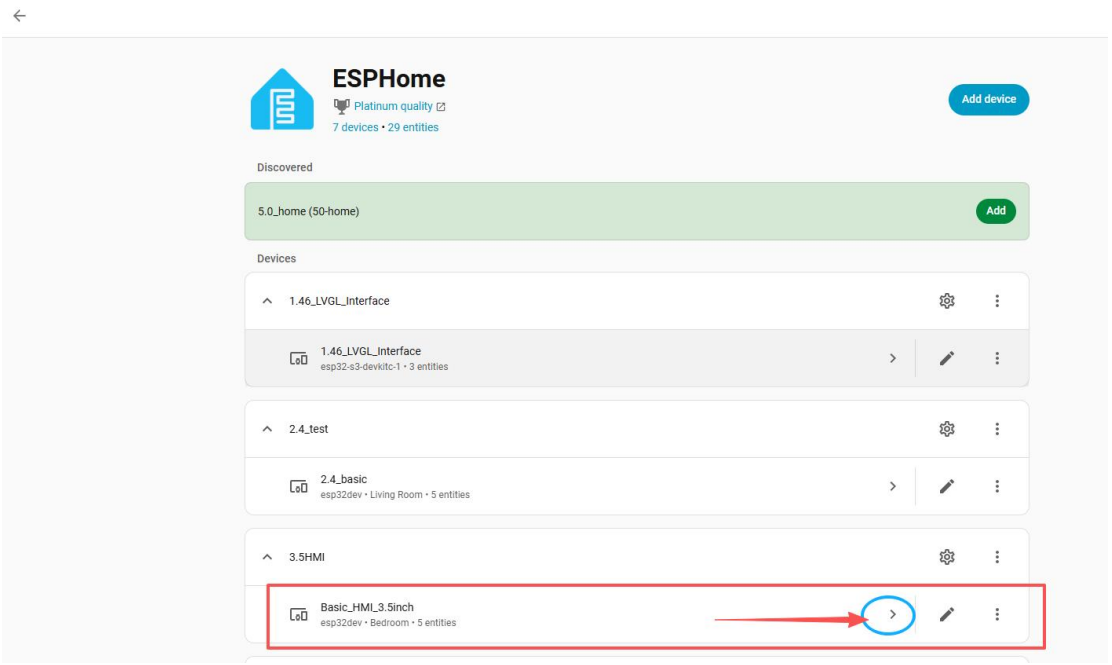
go to settings and select devices

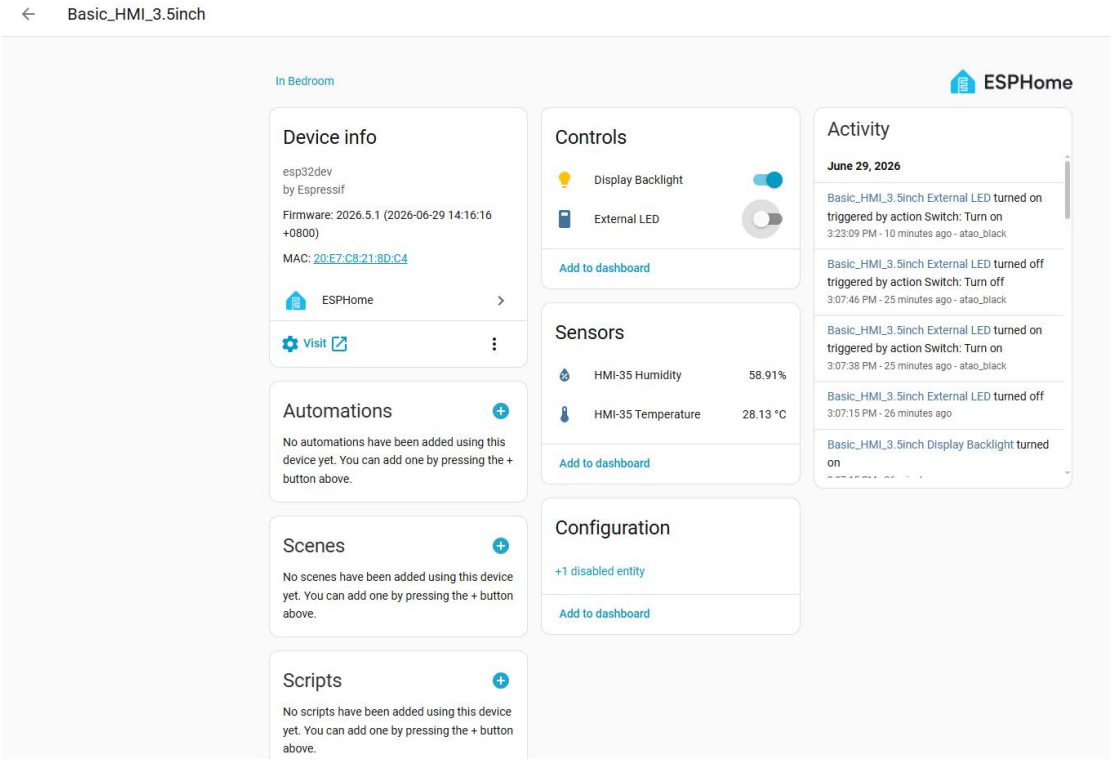



Choose ESPHome

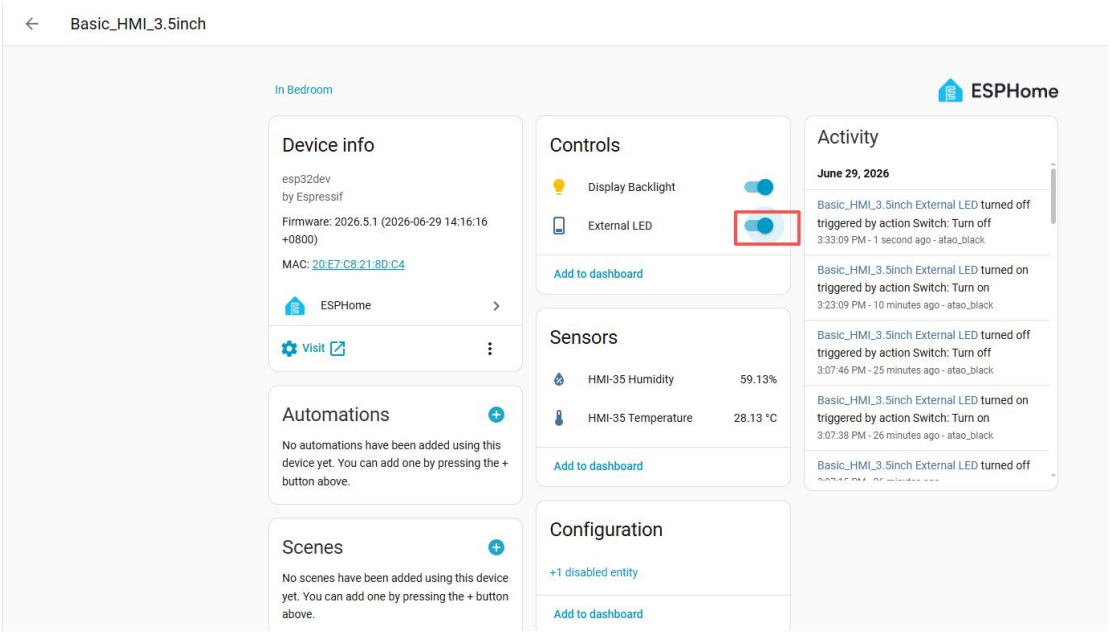



Here you can see the specific details of the equipment we just added.



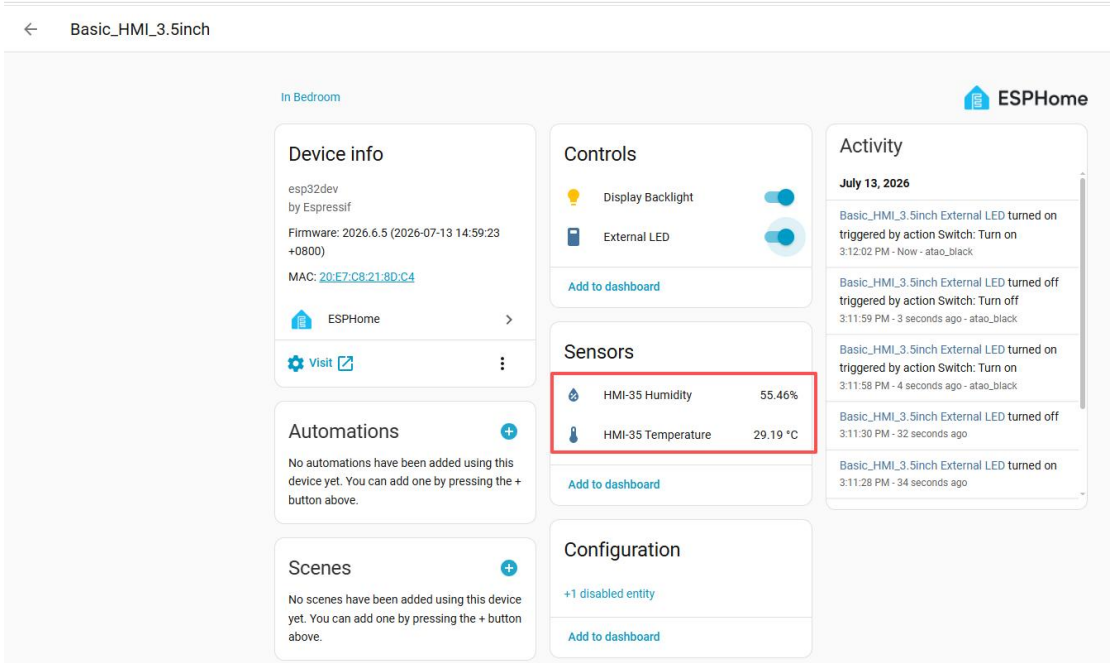



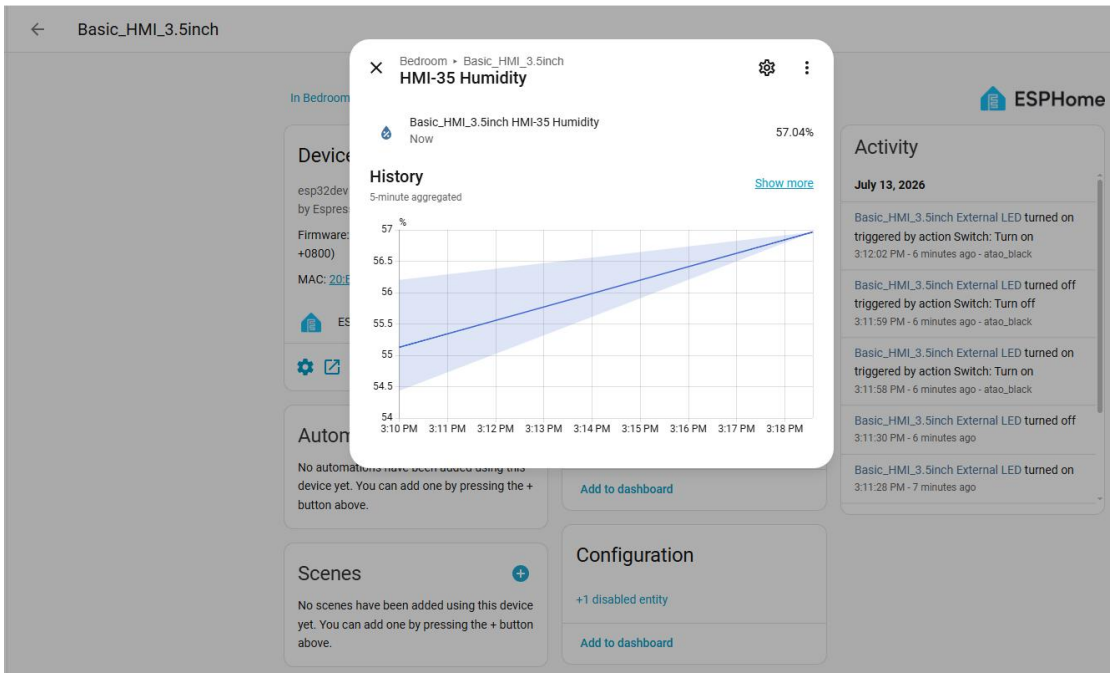

Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

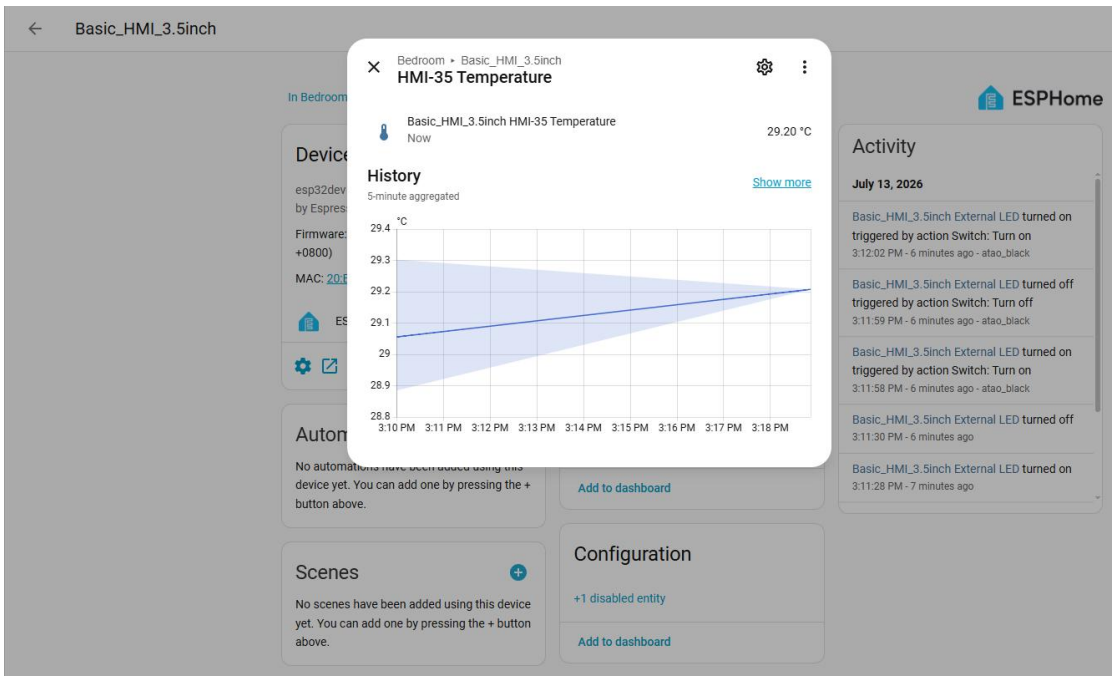


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

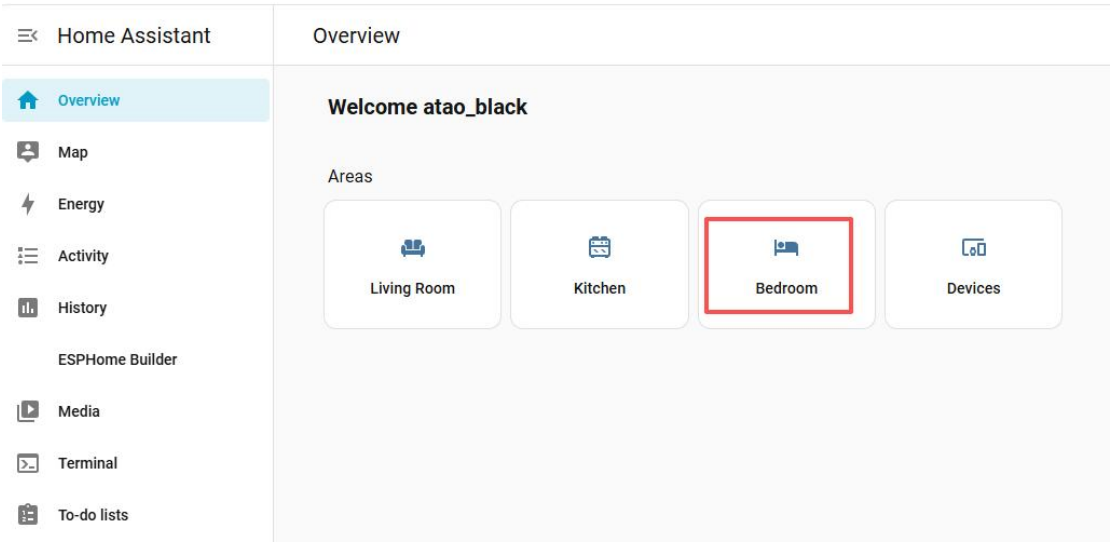
And you can also see the historical data of temperature and humidity collection from here.

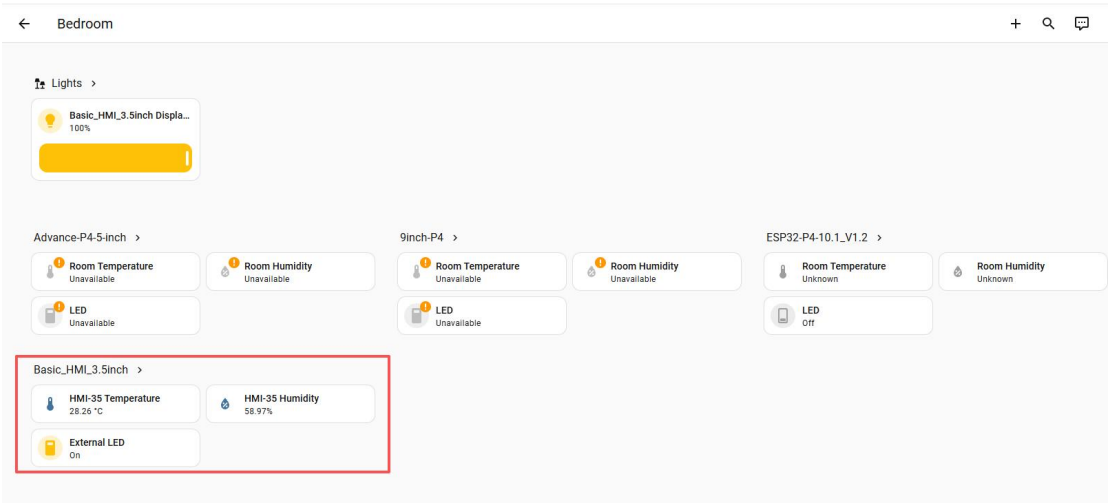
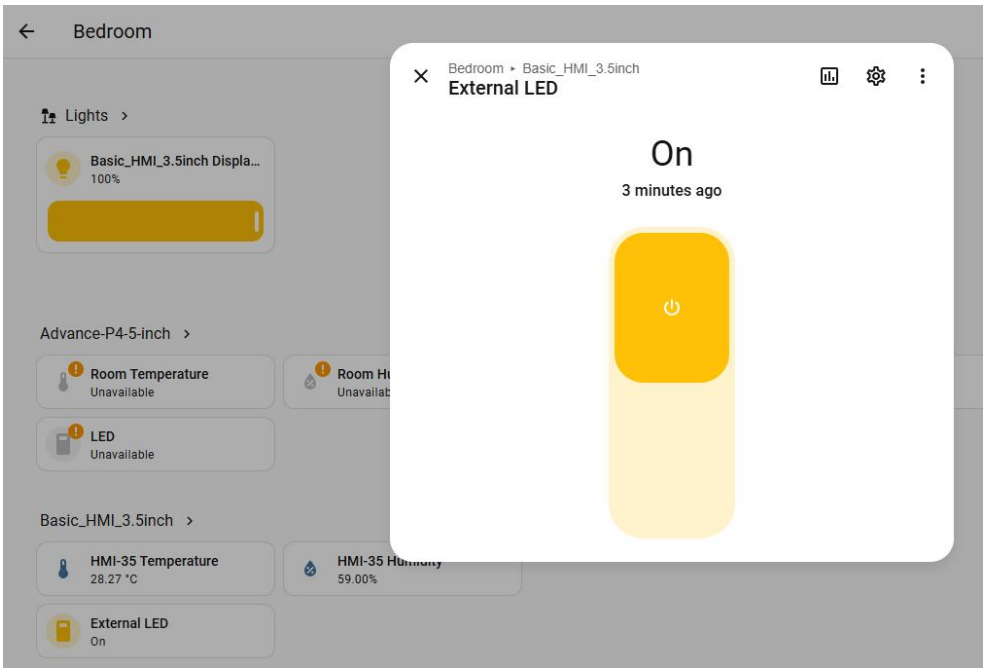





And then you go back to the main interface and arrive at the Bedroom.(You can freely choose which area you want to store it in.)




It can control the LED and also display the temperature and humidity data.